

Índice

<i>Apresentação</i>	4
<i>MATLAB – v. 5.3</i>	5
1. Introdução	5
2. Noções Básicas	6
2.1. Ajuda on-line	7
3. Lições Tutoriais	8
3.1. Lição 1: Criando e Trabalhando com Vetores	8
3.2. Lição 2: Plotando Gráficos Simples	10
3.3. Lição 3: Criando, Salvando e Executando Procedimentos	11
3.4. Lição 4: Criando e Executando uma Função	12
4. Computação Interativa	14
4.1. Matrizes e Vetores	14
4.1.1. Entrada	14
4.1.2. Índices	14
4.1.3. Manipulação de Matrizes	16
4.1.4. Criando Vetores	19
4.2. Operações com Matrizes	20
4.2.1. Operações Aritméticas	20
4.2.2. Operações Relacionais	22
4.2.3. Operações Lógicas	22
4.3. Funções Matemáticas Elementares	23
4.4. Funções de Matrizes	24
4.5. Caracteres de “Strings”	25
5. Gráficos	28
5.1. Gráficos Básicos em 2-D	28
5.2. Opções de Estilo	28
5.3. Rótulos, Título, Legenda e Outros objetos de texto	28
5.1 Controle dos Eixos e de Zoom	29
5.4. Gráficos sobrepostos	30
5.5. Gráficos específicos em 2-D	32
5.6. Usando o comando subplot	37
5.7. Gráficos 3D	38
5.8. Comando View	38

5.9. Rotacionar a vista	40
5.10. Gráficos de malha e superfície	40
5.10.1. Gráficos de Superfície Interpolados	45
5.10.2. Manejar com Gráficos	46
5.10.3. A Hierarquia dos Objetos	47
5.10.4. Objetos Handles	47
5.10.5. Propriedades dos objetos	48
5.10.6. Modificando um gráfico existente	52
5.10.7. Controle completo sobre o plano gráfico (layout)	52
5.10.8. Salvando e Imprimindo Gráficos	53
5.10.9. Animação	55
6. Programando no MATLAB: Funções e Procedimentos	61
6.1. Procedimentos	61
6.2. Funções	61
6.2.1. Executando uma função	62
6.2.2. Mais sobre funções	64
6.2.3. Sub-funções	66
6.2.4. Funções compiladas (Analisadas): P-Code	66
6.2.5. O Profiler	66
6.3. Características Específicas da Linguagem	67
6.3.1. O uso de comentário para criar ajuda on-line	67
6.3.2. Continuação	68
6.3.3. Variáveis Globais	68
6.3.4. Laços, ramificações e controle de fluxo	69
6.3.5. Entrada interativa	72
6.3.6. Recursion	74
6.3.7. Entrada/Saída	74
6.4. Objetos de Dados Avançados	77
6.4.1. Matrizes Multidimensionais	77
6.4.2. Estruturas ou Registros	78
6.4.3. Células	82
7. Erros	84
<i>Apêndice (Respostas dos exercícios do tutorial)</i>	88

Apresentação

Esta apostila foi confeccionada pelos bolsistas do Grupo Pet de Engenharia Elétrica Diogo Chadud Milagres, Hudson Paz da Silva, Igor Gavilon e Luigi Galotto Júnior, baseada no livro de Rudra Pratap, com o objetivo de ensinar desde de os princípios básicos até os comandos mais avançados do Matlab.

Meses de trabalho não foram suficientes para que se dominasse toda a ferramenta MATLAB, uma vez que o trabalho realizado visou a preparação de um material apropriado para o ensino em nível acadêmico. Esta obra tem caráter exclusivamente científico e sua visão expressa por acadêmicos que buscam na ferramenta seu melhor desempenho do ponto de vista experimental. Portanto, a aplicação específica para algumas extensões que existem do MATLAB no mercado hoje em dia – como confecção de mapas, estradas, lay-outs, redes neurais, logica fuzzy entre outros – não será abordada nesta apostila.

Esperamos que esta apostila seja de grande utilidade para todos, não sendo usada apenas durante o aprendizado, mas que seja utilizada como uma fonte de consulta.

MATLAB – v. 5.3

1. INTRODUÇÃO

Nesta apostila, apresentaremos um pouco sobre a potencialidade do MATLAB. Precisariamos de muito tempo para mostrar tudo aquilo de que o MATLAB é capaz. Você provavelmente sentirá a necessidade de estudar este tutorial executando o MATLAB ao mesmo tempo. Dessa forma, poderá introduzir os comandos conforme descrito aqui, confirmar os resultados apresentados e desenvolver uma compreensão prática do MATLAB.

Talvez a maneira mais fácil de visualizar o MATLAB seja pensar nele como uma calculadora científica completa. Assim como em uma calculadora básica, ele faz operações matemáticas elementares como a adição, subtração, multiplicação e divisão. Tal como uma calculadora científica, ele opera com números complexos, raízes quadradas e potenciações, logaritmos e operações trigonométricas, tais como seno, cosseno e tangente. Da mesma forma que uma calculadora programável, é possível armazenar e recuperar dados, criar, executar e armazenar seqüências de comandos para automatizar o cálculo de equações importantes, e fazer comparações lógicas e controlar a ordem na qual os comandos são executados. Assim como as calculadoras mais poderosas disponíveis, ele lhe permite plotar os dados de diversas maneiras, executar álgebra matricial, manipular polinômios, integrar funções, manipular as funções simbolicamente etc.

Na realidade, o MATLAB oferece muitas características e é muito mais versátil do que qualquer calculadora: é uma ferramenta para fazer cálculos matemáticos; é uma linguagem de programação com características mais avançadas e muito mais fácil de usar que linguagem de programação tais como BASIC, Pascal ou C; apresenta um ambiente rico para a visualização de dados graças a sua poderosa capacidade gráfica; é uma plataforma de desenvolvimento de aplicações, na qual conjuntos de ferramentas inteligentes de resolução de problemas para aplicações específicas, geralmente denominados toolboxes, podem ser desenvolvidos de forma relativamente fácil.

2. NOÇÕES BÁSICAS

Como é possível fazer operações aritméticas básicas utilizando o MATLAB?

Os operadores são:

+	Adição,
-	Subtração,
*	Multiplicação,
/	Divisão,
^	Exponenciação.

Os cálculos efetuados em uma fórmula obedecerão à hierarquia matemática, ou seja, primeiro será calculado a exponenciação, depois a multiplicação e divisão e por último a soma e a subtração. Caso você tenha a necessidade de obter prioridade de cálculo em outra ordem, lembre-se de utilizar os caracteres parênteses.

Se você digitar:

```
» 5 + 5
```

A resposta será:

```
ans =  
    10
```

Este resultado significa que a sua resposta foi armazenada na variável ‘ans’, pois todos os resultados devem ser armazenados em uma variável.

No MATLAB, as variáveis são declaradas automaticamente, portanto basta fazer uma atribuição. Por exemplo:

```
» a = 5  
a =  
    5
```

Neste caso, a variável ‘a’ recebe o valor 5, assim o mesmo cálculo anterior pode ser:

```
» b = a + a  
b =  
    10
```

Obs.: Colocando ‘;’ no final de cada sentença, o MATLAB não retorna a resposta.

Se não conseguir se lembrar do nome de uma variável, pode pedir para o MATLAB apresentar uma lista das variáveis que ele conhece, utilizando o comando who:

```
» who
```

Your variables are:

```
a      ans      b
```

Observe que o MATLAB não informa o valor das variáveis, mas somente lista seus nomes. Para descobrir seus valores, basta introduzir seus nomes após o prompt do MATLAB.

Para chamar os comandos previamente utilizados, o MATLAB utiliza as teclas de cursor (\leftarrow , \rightarrow , \uparrow , \downarrow) do seu teclado. Por exemplo, ao pressionar uma tecla \uparrow uma vez, chamamos o comando mais recente do prompt do MATLAB. Pressionando-se a tecla repetidamente chamamos os comandos anteriores, um de cada vez. De forma semelhante, pressionando-se a tecla \downarrow , chamam os seus comandos posteriores.

2.1. AJUDA ON-LINE

O comando help é a maneira mais simples de se conseguir ajuda caso você saiba exatamente o tópico a respeito do qual você necessita de informações. Ao digitar help <tópico>, a tela apresenta ajuda sobre o tópico, caso ela exista. Por exemplo:

```
» help sqrt
```

SQRT Square root.

SQRT(X) is the square root of the elements of X. Complex results are produced if X is not positive.

See also SQRTM.

Essa foi a resposta à ajuda sobre o tópico SQRT.

Caso você não sabe o nome da função, é possível utilizar o comando lookfor.

O comando lookfor fornece ajuda fazendo uma busca em toda a primeira linha dos tópicos de ajuda e retornando aqueles que contêm a palavra-chave que você especificou. Por exemplo:

```
» lookfor complex
```

CONJ Complex conjugate.

CPLXPAIR Sort numbers into complex conjugate pairs.

IMAG Complex imaginary part.

REAL Complex real part.

CDF2RDF Complex diagonal form to real block diagonal form.

A palavra-chave complex não é um comando MATLAB, mas o comando localizou nas descrições de ajuda alguns comandos do MATLAB. A partir dessa informação, o comando help pode ser usado para buscar ajuda sobre um comando específico.

3. LIÇÕES TUTORIAIS

As seguintes lições têm o objetivo de auxiliar você a conhecer o MATLAB rapidamente. As lições devem durar de 10 a 15 minutos e aconselhamos que você faça os exercícios no final de cada lição.

3.1. LIÇÃO 1: CRIANDO E TRABALHANDO COM VETORES

O que você aprenderá:

- Como criar matriz linha e coluna.
- Como criar um vetor com n elementos linearmente (igualmente) espaçados entre dois números a e b .
- Como fazer operações aritméticas simples com vetores.
- Como fazer operações termo por termo (usando: `.*`, `./`, `.^`, etc).
- Como usar funções trigonométricas com vetores.
- Como usar funções matemáticas elementares como raiz quadrada, exponenciais e logaritmos com vetores.

Método: Tente executar os comandos a seguir e tente entender os resultados apresentados.

```
» x = [1 2 3]
```

x é uma matriz linha com 3 elementos.

```
x =  
    1    2    3
```

```
» y = [2; 1; 5]
```

y é uma matriz coluna com 3 elementos.

```
y =  
    2  
    1  
    5
```

```
» z = [2 1 0];  
» a = x + z
```

Você pode somar (ou subtrair) dois vetores **com o mesmo comprimento**.

```
a =  
    3    3    3
```

```
» b = x + y  
??? Error using ==> +  
Matrix dimensions must agree.
```

Mas você não pode somar (ou subtrair) uma matriz linha com uma coluna.

» $a = x.*z$

$a =$
2 2 0

Você pode multiplicar (ou dividir) os elementos de dois vetores de mesmo tamanho termo por termo, utilizando um operador especial ($.*$, $./$, etc.).

» $b = 2*a$

$b =$
4 4 0

Não é necessário um operador especial para multiplicar um escalar com um vetor.

» $x = \text{linspace}(0,10,5)$

$x =$
0 2.5000 5.0000 7.5000 10.0000

Cria um vetor x com 5 elementos linearmente espaçados entre 0 e 10.

» $y = \sin(x);$
» $z = \text{sqrt}(x).*y$

$z =$

0 0.9463 -2.1442 2.5688 -1.7203

Funções trigonométricas \sin , \cos , etc., assim como funções matemáticas elementares sqrt , \exp , \log , etc., operam com vetores termo por termo.

EXERCÍCIOS

1. A equação de uma reta é $y = mx - c$, onde m e c são constantes. Compute os valores de y para os seguintes valores de x :

$x = 0, 1.5, 3, 4, 5, 7, 9$ e 10 .

2. Crie um vetor t com 10 elementos: 1, 2, 3, ..., 10. Depois compute os seguintes valores:

- $x = t \cdot \sin(t)$.
- $y = (t - 1)/(t + 1)$.
- $z = \sin(t^2)/t^2$.

3. Todos os pontos com coordenadas $x = r \cos \theta$ e $y = r \sin \theta$, sendo r uma constante, representam um círculo de raio r , que satisfazem a equação $x^2 + y^2 = r^2$. Crie um vetor coluna para θ com valores 0, $\pi/4$, $\pi/2$, $3\pi/4$, π e $5\pi/4$.

Considere $r = 2$ e compute os vetores coluna x e y , e verifique se eles satisfazem a equação do círculo encontrando o raio $r = \sqrt{x^2 + y^2}$.

4. A soma de uma série geométrica $1 + r + r^2 + r^3 + \dots + r^n$ possui um limite $1/(1-r)$ para $r < 1$ e $n \rightarrow \infty$. Crie um vetor n de 11 elementos de 0 a 10. Considere $r = 0.5$ e crie outro vetor $x = [r^0 \ r^1 \ \dots \ r^n]$. Compute a soma deste vetor com o comando $s = \text{sum}(x)$ (s é soma da série atual). Calcule o limite $1/(1-r)$ e compare com s . Repita o procedimento para n de 0 a 50 e depois de 0 a 100.

3.2. LIÇÃO 2: PLOTANDO GRÁFICOS SIMPLES

O que você irá aprender:

- Como criar as coordenadas x e y de uma circunferência de raio unitário.
- Como plotar o gráfico x x y, visualizando a circunferência.
- Como escolher os eixos x e y em escalas iguais, para que a circunferência não pareça uma elipse.
- Como nomear os eixos e o gráfico.
- Como imprimir o gráfico.

Os comandos usados do MATLAB são: axis, xlabel, ylabel, title e print.

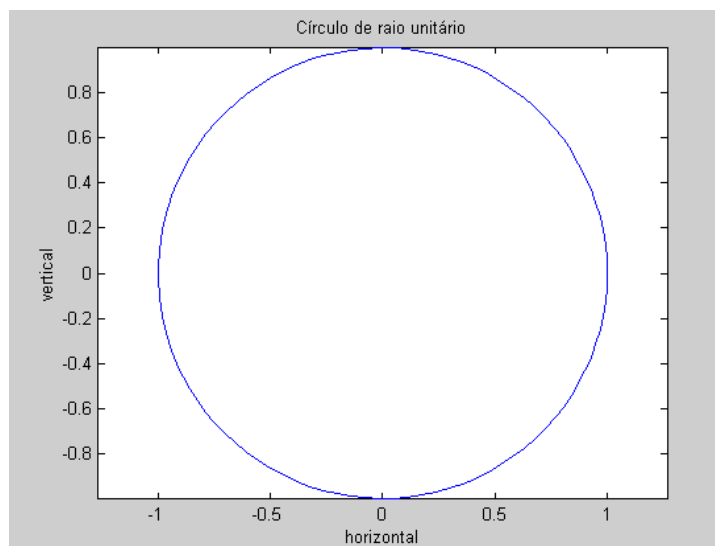
Método: Você irá desenhar um círculo de raio unitário. Para fazê-lo, primeiro criam-se os dados (coordenadas x e y) então plotam-se os dados e finalmente imprima o gráfico.

Para criar os dados, use as equações paramétricas de um círculo unitário:

$$x = \cos \theta \quad y = \sin \theta \quad 0 \leq \theta \leq 2\pi$$

Nos procedimentos abaixo só serão mostrados os

» teta = linspace(0,2*pi,100);	Cria um vetor teta de 100 elementos linearmente espaçados.
» x = cos(teta);	Calcula as coordenadas x e y.
» y = sin(teta);	
» plot(x,y)	Plota o gráfico x x y.
» axis('equal');	Iguala a escala dos eixos.
» xlabel('horizontal')	Nomeia os eixos x e y.
» ylabel('vertical')	
» title('Círculo de raio unitário')	Insere um título para o gráfico.
» print	Imprime.



EXERCÍCIOS

1. Plote $y = e^{-0.4x} \sin x$, $0 \leq x \leq 4\pi$. Tomando 10, 50 e depois 100 pontos no intervalo.
2. Use o comando `plot3(x,y,z)` para plotar a hélice circular $x(t) = \sin t$, $y(t) = \cos t$ e $z(t) = t$, $0 \leq t \leq 20$.
3. Os comandos de plotagem `semilogx`, `semilogy`, e `loglog` serão usados neste exercício. Plote os valores de x , os valores de y e ambos no gráfico na escala \log_{10} respectivamente. Crie o vetor $x = 0:10:1000$. Plote x e $y = x^3$, usando as três escalas logarítmicas descritas no início.

3.3. LIÇÃO 3: CRIANDO, SALVANDO E EXECUTANDO PROCEDIMENTOS

O que você irá aprender:

- Como criar, escrever e salvar o procedimento.
- Como executar o procedimento no MATLAB.

Método: Escrever um procedimento para desenhar o círculo unitário da lição 2. Você essencialmente escreverá os comandos mostrados na lição anterior, irá salvá-los, nomeá-los e executá-los no MATLAB. Siga as instruções abaixo:

1. Escolha a opção Novo (New) no menu Arquivo (File) do MATLAB e selecione a opção M-file.
2. Escreva as linhas a seguir. As linhas que começam com o caracter % são interpretadas como comentários pelo MATLAB e são ignoradas.

% CIRCLE – Procedimento que desenha um círculo unitário.

% Arquivo escrito pelo Grupo Pet.

% -----

```
teta = linspace(0,2*pi,100);    % Cria o vetor teta.
x = cos(teta);                  % Gera coordenadas x.
y = sin(teta);                  % Gera coordenadas y.
plot (x, y);                    % Plota o círculo.
axis ('equal');                 % Iguala a escala dos eixos.
title('Círculo de raio unitário') % Põe um título.
```

3. Depois de escrito, salve o arquivo como `circle.m`.
4. Volte ao MATLAB e verifique se é possível executar o seu arquivo, da forma a seguir:

»help circle

CIRCLE – Procedimento que desenha um círculo unitário.

Arquivo escrito pelo Grupo Pet.

»circle

Você deverá ver o mesmo círculo do exemplo anterior.

EXERCÍCIOS

1. Modifique o arquivo `circle.m` para mostrar o centro do círculo também. Marque o ponto central '+’.

2. Modifique o arquivo `circle.m` para formar um círculo de raio qualquer.

- Use o comando *input* para escrever na tela que você quer entrar com um valor para o raio. Ex: `r = input('Entre com o valor do raio: ')`
- Modifique o cálculo de `x` e `y`, multiplicando-os pelo valor do raio (`r`). Salve e execute o arquivo.

3.4. LIÇÃO 4: CRIANDO E EXECUTANDO UMA FUNÇÃO

O que você irá aprender:

- Como abrir e editar um M-file existente.
- Como definir e executar um arquivo função.

Método: Escreva um arquivo função para desenhar um círculo de um raio específico, com o raio sendo a entrada para a função. Você pode escrever um novo arquivo função ou modificar o procedimento da lição 3. Nós aconselhamos você a escolher a última opção.

1. Abra o arquivo **circle.m**:

Selecione a opção Abrir (Open) do menu Arquivo (File).
Dê um clique duplo no arquivo na caixa de diálogo.

2. Edite o arquivo `circle.m` da lição 3, conforme indicado abaixo:

```
function [x,y] = circlefn(r);  
% CIRCLE –Função para desenhar um círculo de raio r.  
% Arquivo escrito pelo Grupo Pet.  
% Input: r = raio especificado.  
% Output: [x,y] = coordenadas x e y dos pontos dados.  
% -----  
teta = linspace(0,2*pi,100);    % Cria o vetor teta.  
x = r*cos(teta);                % Gera coordenadas x.  
y = r*sin(teta);                % Gera coordenadas y.  
plot (x, y);                    % Plota o círculo.  
axis ('equal');                 % Iguala a escala dos eixos.  
title('Círculo de raio r = ', num2str(r)) % Põe um título com o valor de r.
```

3. Depois de modificado salve o arquivo com o nome `circlefn.m`, utilizando a opção Save as... do menu Arquivo.

4. Este é um exemplo de se executar uma função de 3 diferentes formas. Tente executá-los.

» R = 5;	Especifica a entrada e executa a função com as variáveis de saída especificadas.
» [x,y] = circlefn(R);	
» [cx,cy] = circlefn(2.5);	Você pode também especificar o valor de entrada diretamente.
» circlefn(1);	Se você não precisar da saída, não é necessário armazenar a saída numa variável.
» circlefn(R^2/(R+5*sin(R)));	É claro que a entrada pode ser uma expressão válida do MATLAB.

EXERCÍCIOS

1. Escreva uma função que retorne uma tabela de conversão de Celsius para Fahrenheit. A entrada da função deve ser dois números: T_i e T_f , especificando o menor e o maior variação da tabela em Celsius. A saída deve ser uma matriz de duas colunas: a primeira coluna mostrando a temperatura em Celsius de T_i para T_f no incremento de 1 °C e a segunda coluna mostrando a temperatura correspondente em Fahrenheit. (i) Crie um vetor coluna C de T_i para T_f com o comando $C = [T_i : T_f]'$, (ii) Calcule o número correspondente em Fahrenheit usando a fórmula $[F = 9 \cdot C / 5 + 32]$ e (iii) Faça o matriz final com o comando $\text{temp} = [C \ F];$. Note que a sua saída será nomeada temp.

2. Escreva uma função crossprod para computar o produto vetorial de dois vetores \mathbf{u} e \mathbf{v} , dado $\mathbf{u} = (u_1, u_2, u_3)$, e $\mathbf{v} = (v_1, v_2, v_3)$, e $\mathbf{u} \times \mathbf{v} = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$. Cheque sua função fazendo o produto vetorial de vetores unitários: (\mathbf{i}, \mathbf{j}) , (\mathbf{j}, \mathbf{k}) , etc. [$\mathbf{i} = (1,0,0)$, $\mathbf{j} = (0,1,0)$, $\mathbf{k} = (0,0,1)$].

3. Escreva a função para computar a soma (utilize a função sum) de uma série geométrica $1 + r + r^2 + r^3 + \dots + r^n$ para uma dado r e n . Portanto a entrada para a função deve ser r e n e a saída deve ser a soma da série. (Veja o exercício 4 do item 3.1. lição 1).

4. COMPUTAÇÃO INTERATIVA

Neste capítulo, nós introduziremos para vocês algumas das funções embutidas no MATLAB e suas capacidades, exemplos completos de computação interativa.

4.1. MATRIZES E VETORES

4.1.1. ENTRADA

Os valores de uma Matriz são colocados entre colchetes, sendo que, os elementos de uma linha são separados por espaço ou vírgula, e as colunas são separadas por ponto e vírgula.

Exemplos:

Matriz

$$A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 9 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 2x & \ln x + \sin y \\ 5i & 3 + 2i \end{bmatrix}$$

Entrada do comando no MATLAB

$$A = [1 \ 2 \ 5; 3 \ 9 \ 0]$$

$$B = [2*x \ \log(x) + \sin(y); 5i \ 3 + 2i]$$

Obs.: Usa-se a mesma notação para vetores com uma linha ou com uma coluna.

Continuação

Se a sua entrada é muito longa, você poderá usar reticências (...) e continuar a entrada na outra linha.

Exemplo:

$$A = [1/3 \ 5.55*\sin(x) \ 9.35 \ 0.097; \dots \\ 3/(x+2*\log(x)) \ 3 \ 0 \ 6.555; \dots \\ (5*x - 23)/55 \ x-3 \ x*\sin(x) \ \text{sqrt}(3)];$$

4.1.2. ÍNDICES

Dada uma matriz, os seus elementos podem ser acessados especificando os índices de suas linhas e colunas, sendo que $A(i, j)$ representa do elemento a_{ij} da matriz A .

Exemplos:

$$\gg A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 8]$$

$A =$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{bmatrix}$$

Entrada da Matriz.

Elementos de uma linha são separados por espaço ou vírgula, e as colunas são separadas por ponto e vírgula.

» A(2,3)

Acesso ao elemento da segunda linha e da terceira coluna da matriz A.

ans =

6

» A(3,3) = 9

Substitui o elemento da terceira linha e da terceira coluna da matriz A.

A =

1 2 3
4 5 6
7 8 9

» B = A(2:3,1:3)

É possível retirar uma parte da matriz A, usando os específicos limites para as linhas e as colunas.

B =

4 5 6
7 8 9

» B = A(2:3,:)

É possível selecionar toda uma linha ou coluna utilizando (:).

B =

4 5 6
7 8 9

» B(:,2) = []

Para deletar uma linha ou uma coluna, basta atribuir a matriz vazia.

B =

4 6
7 9

Dimensão

A dimensão da matriz é determinada automaticamente pelo MATLAB.

Exemplo:

B(2, 3) = 5; produz, $B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$

$$C(3,1:3) = [1 \ 2 \ 3]; \quad \text{produz. } B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 3 \end{bmatrix}$$

4.1.3. MANIPULAÇÃO DE MATRIZES

Você pode manipular facilmente a seleção de qualquer elemento de uma matriz, utilizando vetores como índices da matriz, para que seja criada uma submatriz a partir de uma matriz. Também, pode-se adicionar, remover ou substituir linhas e/ou colunas de uma matriz.

Como exemplo, suponha que A é uma matriz 10 x 10, B é uma matriz 5 x 10 e y é um vetor de 20 elementos, então:

$$A([1 \ 3 \ 6 \ 9], :) = [B(1:3, :); y(1:10)]$$

As linhas 1, 3 e 6 da matriz A serão substituídas pelas linhas 1, 2 e 3 da matriz B, e a linha 9 da matriz A é substituída pelo vetor y.

Exemplo:

$$\text{Se } Q = \begin{bmatrix} 2 & 3 & 6 & 0 & 5 \\ 0 & 0 & 20 & -4 & 3 \\ 1 & 2 & 3 & 9 & 8 \\ 2 & -5 & 5 & -5 & 6 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix} \quad \text{e } v = [1 \ 4 \ 5] \text{ então,}$$

$$Q(v, :) = \begin{bmatrix} 2 & 3 & 6 & 0 & 5 \\ 2 & -5 & 5 & -5 & 6 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix} \quad \text{e } Q(:, v) = \begin{bmatrix} 2 & 0 & 5 \\ 0 & -4 & 3 \\ 1 & 9 & 8 \\ 2 & -5 & 6 \\ 5 & 20 & 25 \end{bmatrix}$$

Reformando matrizes

Os elementos de uma matriz podem ser colocados dentro de um vetor ou também podem ser agrupados para formar uma matriz com a dimensão diferente da matriz de origem.

Suponha uma matriz $A_{4 \times 3}$; com o comando $b = A(:)$, os elementos da matriz A são armazenados em um vetor coluna b.

Já com o comando $\text{reshape}(A, 3, 4)$, transformamos a matriz $A_{3 \times 4}$. Note que o número de elementos da matriz nunca pode mudar.

Transposta

A transposta de uma matriz, é obtida colocando-se o nome da matriz seguida de um apóstrofo: matriz $A_{4 \times 3} \rightarrow A'_{3 \times 4}$.

Inicialização

A inicialização de uma matriz não é necessária em MATLAB. Contudo, ela pode ser útil em alguns casos:

$A = \text{zeros}(m,n)$ % Este comando cria uma matriz $m \times n$ onde todos os seus elementos são zeros.

$A = \text{ones}(m,n)$ % Este comando cria uma matriz $m \times n$ onde todos os seus elementos são 1's.

Inserindo uma linha ou uma coluna

Uma linha ou uma coluna pode ser facilmente inserida em uma matriz já existente. Deve-se atentar para o seguinte detalhe: que o tamanho da linha ou coluna a ser inserida deve ter o mesmo tamanho da linha ou coluna da matriz já existente.

Exemplo:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & & 0 \\ 0 & 0 & 1 \end{bmatrix}, u = [5 \ 6 \ 7] \quad \text{e} \quad v = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

Então

$$A = [A; u] \quad \text{produz} \quad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 5 & 6 & 7 \end{bmatrix}, \text{ uma matriz } 4 \times 3,$$

$$A = [A, v] \quad \text{produz} \quad A = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \end{bmatrix}, \text{ uma matriz } 3 \times 4,$$

$$A = [A, u'] \quad \text{produz} \quad A = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & & 0 & 6 \\ 0 & 0 & 1 & 7 \end{bmatrix}, \text{ uma matriz } 3 \times 4,$$

$A = [A \ u]$ produz um erro.

$B = []; B = [B; 1 \ 2 \ 3]$ produz $B = [1 \ 2 \ 3]$, e

$B = []; \text{for } k = 1:3, B = [B; k \ k+1 \ k+2]; \text{end}$ produz $B =$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Deletando uma linha ou coluna

Qualquer linha ou coluna de uma matriz pode ser deletada simplesmente atribuindo um vetor nulo à mesma.

Exemplo:

$A(2, :) = []$ deleta a 2ª. linha da matriz A.

O MATLAB possui alguns utilitários para a geração e manipulação de matrizes. Por exemplo:

eye (m,n)	retorna uma matriz mxn com 1 na diagonal principal.
zeros (m,n)	retorna uma matriz de zeros mxn.
ones (m,n)	retorna uma matriz de elementos 1 mxn.
rand (m,n)	retorna uma matriz mxn com número randômicos.
diag(v)	sendo v um vetor qualquer, gera uma matriz diagonal com o vetor v na diagonal.
diag(A)	extrai a diagonal principal de uma matriz para um vetor coluna.
diag(A,1)	extrai a primeira diagonal superior à diagonal principal para um vetor coluna.
rot90	rotaciona uma matriz em 90°.
fliplr	gira uma matriz da esquerda para a direita.
flipud	gira uma matriz de cima para baixo.
tril	extrai a parte triangular inferior de uma matriz.
triu	extrai a parte triangular superior de uma matriz.
reshape	muda o formato da matriz.

» eye(3)

ans =

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

» B = [ones(3) zeros(3,2); zeros(2,3) 4*eye(2)]

B =

```
1  1  1  0  0
1  1  1  0  0
1  1  1  0  0
0  0  0  4  0
0  0  0  0  4
```

Cria uma matriz B usando submatrizes elementares: ones, zeros, e a matriz identidade de tamanhos específicos.

» diag(B)'

ans =

1 1 1 4 4

Este comando extrai a diagonal principal da matriz B e a transforma num vetor. Sem a transposta ('), o resultado seria obviamente uma coluna.

» diag(B,1)'

ans =

1 1 0 0

O segundo argumento escolhe a diagonal que você deseja extrair.

» d = [2 4 6 8];

» d1 = [-3 -3 -3];

» d2 = [-1 -1];

» D = diag(d) + diag(d1,1) + diag(d2,-2)

D =

2 -3 0 0
0 4 -3 0
-1 0 6 -3
0 -1 0 8

Cria uma matriz D colocando d na diagonal principal, d1 na primeira diagonal acima da diagonal principal e d2 na segunda diagonal abaixo da diagonal principal.

4.1.4. CRIANDO VETORES

Suponha que você queira criar um vetor com um número grande de elementos, existe um comando que possibilita criar um vetor sem precisar digitar todos os elementos. A fórmula geral é:

v = valor inicial : incremento : valor final

Os três valores acima podem ser expressões válidas do MATLAB. Caso você não coloque o incremento, o MATLAB utilizará o valor padrão que é 1.

Exemplo:

a = 0:10:100

b = 0:pi/50:2*pi

c = 2:10

produz a = [0 10 20 ... 100],

produz b = [0 pi/50 2*pi/50 ... 2*pi]

produz c = [2 3 4 5 ... 10]

Como você pode perceber, não é necessário utilizar colchetes se um vetor é gerado deste modo, entretanto, um vetor atribuído como u = [1:10 33:-2:19] necessita a utilização de colchetes para fazer a concatenação dos dois vetores [1 2 3 ... 10] e [33 31 29 ... 19]. Finalmente, nós mencionamos o uso de duas funções embutidas freqüentemente usadas para gerar vetores:

`linspace(a, b, n)` gera um vetor linearmente espaçado de comprimento n de a até b .

Exemplo:

`u = linspace(0, 20, 5)` gera `u = [0 5 10 15 20]`. Então `u = linspace(a, b, n)` é o mesmo que `u = a: (b-a)/(n-1) : b`.

`logspace(a, b, n)` gera um vetor logaritmicamente espaçada e de comprimento n de 10^a até 10^b .

Exemplo:

`v = logspace(0, 3, 4)` gera `v = [1 10 100 1000]`. Portanto `logspace(a, b, n)` é o mesmo que `10.^(linspace(a, b, n))`.

Vetores especiais tais como vetores de 0's ou de 1's de um comprimento específico, podem ser criados com as funções `zeros`, `ones` etc.

4.2. OPERAÇÕES COM MATRIZES

4.2.1. OPERAÇÕES ARITMÉTICAS

As operações aritméticas com matrizes só serão válidas se elas forem matematicamente compatíveis, conforme as seguintes condições:

`A+B` ou `A-B` é válida se A e B são do mesmo tamanho

`A*B` é válida se o número de colunas de A é igual ao número de linhas de B .

`A/B` é válida e igual a $A \cdot B^{-1}$ para matrizes quadradas do mesmo tamanho.

`A^2` faz sentido somente se A é quadrada; o comando é igual a `A*A`.

Divisão à direita:

Adicionando à divisão à esquerda (`/`), existe também a divisão à direita (`\`). Em particular, o comando `x = A \ b` encontra o valor de b dividido por A . Portanto `A \ b` é quase o mesmo que `inv(A)*b`, mas é mais rápido e mais numericamente estável do que computar `inv(A)*b`.

Ordem de operações

É possível fazer a multiplicação, divisão ou exponenciação elemento por elemento entre matrizes ou vetores de mesmo tamanho colocando um ponto (`.`):

- `.*` Multiplicação de elemento por elemento,
- `./` Divisão à esquerda de elemento por elemento,
- `.\` Divisão à direita de elemento por elemento,
- `.^` Exponenciação de elemento por elemento,
- `.'` Transposta não conjugada.

Exemplos:

u.*v produz [u1v1 u2v2 u3v3 ...],
 u./v produz [u1/v1 u2/v2 u3/v3 ...], e
 u.^v produz [u1^{v1} u2^{v2} u3^{v3} ...].

O mesmo é verdadeiro para matrizes. Para duas matrizes de mesmo tamanho A e B, o comando C = A.*B produz uma matriz C com elementos $C_{ij} = A_{ij} \cdot B_{ij}$.

» A = [1 2 3;4 5 6;7 8 9];

» x = A(1,:)'

x =

1
2
3

Atribui ao vetor x, a transposta da primeira linha da matriz A.

» x'*x

ans =

14

Produto do vetor linha (x') e do vetor coluna (x); o resultado será um escalar, pela propriedade da multiplicação de matrizes.

» x*x'

ans =

1 2 3
2 4 6
3 6 9

A dimensão do resultado terá o número de linhas da matriz (x) e o número de colunas da matriz (x').

» A*x

Multiplicação de um vetor por uma matriz.

ans =

14
32
50

» A^2

Exponenciação da matriz A; é o mesmo que A*A.

ans =

30 36 42
66 81 96
102 126 150

» A.^2

Neste caso, a exponenciação é feita elemento por elemento.

ans =

1 4 9
16 25 36
49 64 81

4.2.2. OPERAÇÕES RELACIONAIS

Há seis operadores relacionais no MATLAB:

<	menor que
<=	menor ou igual que
>	maior que
>=	maior ou igual que
==	igual
~=	diferente

A relação é feita com vetores ou matrizes de mesmo tamanho, com 1 para verdadeiro ou 0 para falso.

Exemplos:

Se $x = [1 \ 5 \ 3 \ 7]$ e $y = [0 \ 2 \ 8 \ 7]$ então

$k = x < y$	resulta em $k = [0 \ 0 \ 1 \ 0]$	porque $x_i < y_i$ para $i = 3$.
$k = x <= y$	resulta em $k = [0 \ 0 \ 1 \ 1]$	porque $x_i \leq y_i$ para $i = 3$ e 4 .
$k = x > y$	resulta em $k = [1 \ 1 \ 0 \ 0]$	porque $x_i > y_i$ para $i = 1$ e 2 .
$k = x >= y$	resulta em $k = [1 \ 1 \ 0 \ 1]$	porque $x_i \geq y_i$ para $i = 1, 2$ e 4 .
$k = x == y$	resulta em $k = [0 \ 0 \ 0 \ 1]$	porque $x_i = y_i$ para $i = 4$.
$k = x ~= y$	resulta em $k = [1 \ 1 \ 1 \ 0]$	porque $x_i \neq y_i$ para $i = 1, 2$ e 3 .

4.2.3. OPERAÇÕES LÓGICAS

Existem quatro operações lógicas:

&	‘E’ lógico
	‘OU’ lógico
~	‘NÃO’ lógico
xor	‘OU’ exclusivo

Estes operadores trabalham de forma similar aos operadores relacionais.

Exemplos:

Para dois vetores $x = [0 \ 5 \ 3 \ 7]$ e $y = [0 \ 2 \ 8 \ 7]$,

$m = (x > y) \& (x > 4)$	resulta em $m = [0 \ 1 \ 0 \ 0]$,
$n = x y$	resulta em $n = [0 \ 1 \ 1 \ 1]$,
$m = \sim (x y)$	resulta em $m = [1 \ 0 \ 0 \ 0]$,
$p = \text{xor}(x, y)$	resulta em $p = [0 \ 0 \ 0 \ 0]$.

No MATLAB existem muitas funções lógicas embutidas, como:

all verdadeiro (=1) se todos os elementos de um vetor forem verdadeiros.

Exemplo: $\text{all}(x < 0)$ retorna 1 se todos os elementos de x forem negativos.

any	verdadeiro (=1) se algum elemento do vetor for verdadeiro. <i>Exemplo:</i> any(x) retorna 1 se algum elemento de x é diferente de zero.
exist	verdadeiro (=1) se o argumento (uma variável ou função) existe.
isempty	verdadeiro (=1) para uma matriz vazia.
isinf	verdadeiro para todos os elementos infinitos de uma matriz.
isfinite	verdadeiro para todos os elementos finitos de uma matriz.
isnan	verdadeiro para todos os elementos de uma matriz que não forem números (Not-a-Number).
find	encontra os índices de elementos não-nulos de uma matriz. <i>Exemplo:</i> find(x) retorna [2 3 4] para x=[0 2 5 7].

4.3. FUNÇÕES MATEMÁTICAS ELEMENTARES

Todas as funções a seguir são realizadas termo por termo, portanto elas produzem saídas com a mesma dimensão das entradas.

Funções Trigonômicas

sin	seno.	sinh	seno hiperbólico.
asin	arco seno.	asinh	arco seno hiperbólico.
cos	coseno.	cosh	cose hiperbólico.
acos	arco coseno.	acosh	arco coseno hiperbólico.
tan	tangente.	tanh	tangente hiperbólico.
atan, atan2	arco tangente.	atanh	arco tangente hiperbólico.
sec	secante.	sech	secante hiperbólico.
asec	arco secante.	asech	arco secante hiperbólico.
csc	cosecante	csch	cosecante hiperbólico.
acsc	arco cosecante	acsch	arco cosecante hiperbólico.
cot	cotangente	coth	cotangente hiperbólico.
acot	arco cotangente	acoth	arco cotangente hiperbólico.

Os ângulos devem ser dados em radianos. Todas essas funções, exceto atan2, pegam um escalar, vetor ou matriz como entrada de argumentos. A função atan2 pega 2 entradas: atan2(y, x) e retorna o arco tangente do ângulo y/x.

Funções Exponenciais

exp	Exponencial ($\exp(A) = e^{A_{ij}}$).
log	Logaritmo natural ($\log(A) = \ln(A_{ij})$).
log10	Logaritmo na base 10 ($\log_{10}(A) = \log_{10}(A_{ij})$).
sqrt	Raiz quadrada.

Funções Complexas

abs	Valor absoluto.
------------	-----------------

angle	Ângulo de fase.
conj	Complexo conjugado.
imag	Parte imaginária.
real	Parte real.

Funções de aproximação

fix	Aproxima na direção de zero. <i>Exemplo:</i> $\text{fix}([-2.33 \ 2.66]) = [-2 \ 2]$.
floor	Aproxima na direção de $-\infty$. <i>Exemplo:</i> $\text{floor}([-2.33 \ 2.66]) = [-3 \ 2]$.
ceil	Aproxima na direção de $+\infty$. <i>Exemplo:</i> $\text{ceil}([-2.33 \ 2.66]) = [-2 \ 3]$.
round	Aproxima para o inteiro mais próximo. <i>Exemplo:</i> $\text{round}([-2.33 \ 2.66]) = [-2 \ 3]$.
rem	Resto da divisão. $\text{rem}(a, b)$ é o mesmo que $a - b \cdot \text{fix}(a./b)$. <i>Exemplo:</i> Se $a = [-1.5 \ 7]$, $b = [2 \ 3]$, então $\text{rem}(a, b) = [-1.5 \ 1]$.
sign	Retorna o sinal. <i>Exemplo:</i> $\text{sign}([-2.33 \ 2.66]) = [-1 \ 1]$.

4.4. FUNÇÕES DE MATRIZES

As funções de matrizes são:

$\text{expm}(A)$	encontra o exponencial da matriz A (e^A).
$\text{logm}(A)$	encontra $\log(A)$, tais que $A = e^{\log(A)}$.
$\text{sqrtn}(A)$	encontra a raiz da matriz A .

Obs: As funções normais realizam as operações termo por termo enquanto as funções de matrizes realizam uma operação com a matriz. Veja os exemplos a seguir.

» $A = [1 \ 2; 3 \ 4]$;
» $\text{asqrt} = \text{sqrt}(A)$

$\text{asqrt} =$

1.0000 1.4142
1.7321 2.0000

Raiz quadrada de cada elemento da matriz A .

» $\text{Asqrt} = \text{sqrtn}(A)$

$\text{Asqrt} =$

0.5537 + 0.4644i 0.8070 - 0.2124i
1.2104 - 0.3186i 1.7641 + 0.1458i

Raiz quadrada da matriz A . Portanto, $\text{Asqrt} \cdot \text{Asqrt} = A$.


```
» exp_aij = exp(A)
```

```
exp_aij =
```

```
    2.7183    7.3891  
    20.0855   54.5982
```

Analogamente, **exp** retorna a exponencial de cada elemento da matriz, enquanto **expm** retorna a exponencial da matriz A.

```
» exp_A = expm(A)
```

```
exp_A =
```

```
    51.9690   74.7366  
   112.1048  164.0738
```

4.5. CARACTERES DE “STRINGS”

Todos os caracteres de “strings” são colocados entre apóstrofo. O MATLAB considera toda “string” como um vetor linha com 1 elemento para cada caracter. Por exemplo,

```
mensagem = ‘Deixe-me sozinho’
```

cria um vetor, nomeado mensagem, de tamanho 1 x 16 (os espaços também contam como caracteres). Portanto, para criar um vetor coluna com “strings” em cada linha, cada texto “string” deve ter exatamente o mesmo número de caracteres. Por exemplo o comando:

```
nomes = [‘John’; ‘Ravi’; ‘Mary’; ‘Xiao’]
```

cria um vetor coluna com um nome por linha. Contudo, para o MATLAB a variável nomes é uma matriz 4 x 4. Claramente, o comando howdy = [‘Hi’; ‘Hello’; ‘Namaste’] resultará em um erro porque cada uma das linhas tem tamanhos diferentes. Textos de diferentes tamanhos podem ser feitos preenchendo-os com espaços vazios. Assim a entrada correta para o howdy será:

```
howdy = [‘Hi_____’; ‘Hello__’; ‘Namaste’]
```

Um modo mais fácil de fazer a mesma coisa é utilizar o comando **char**, que converte “strings” para uma matriz. Assim ele pode criar o mesmo howdy acima preenchendo os espaços em branco em cada linha automaticamente.

```
howdy = char(‘Hi’, ‘Hello’, ‘Namaste’)
```

Manipulando caracteres “strings”

Os caracteres “strings” podem ser manipulados apenas como matrizes. Assim

```
c = [howdy(2, :) nomes(3, :)]
```

produz Hello Mary como uma saída na variável *c*. Esta característica pode ser usada durante a execução de funções de conversão número-para-texto, como **num2str** e **int2str**. Por exemplo, se você quiser apresentar o resultado de um programa que calcula a hipotenusa *h* de um triângulo retângulo, que varia dependendo da entrada do programa. Assim para apresentar o resultado você deverá digitar:

```
disp( ['A hipotenusa é ', num2str(h)] )
```

Há várias funções embutidas para a manipulação de “strings”.

char	converte “strings” para uma matriz.
abs	converte caracteres para os correspondentes numéricos de acordo com a tabela ASCII.
blanks(n)	cria <i>n</i> espaços em branco.
deblank	remove os espaços em branco de uma “string”.
eval	executa a “string” como um comando.
findstr	encontra uma especificada “substring” em uma “string”.
int2str	converte número inteiros para textos.
ischar	verdadeiro (=1) para uma sequência de caracteres.
isletter	verdadeiro (=1) para um caracter alfabético.
isstring	verdadeiro (=1) se o argumento é um texto.
lower	converte letras maiúsculas em letras minúsculas.
mat2str	converte uma matriz para uma “string”.
num2str	converte números para textos (similar ao int2str).
strcmp	compara duas matrizes e retorna 1 se forem iguais.
strncmp	compara os primeiros <i>n</i> caracteres de uma dada “string”.
strcat	concatena “strings” horizontalmente ignorando os espaços em branco.
strvcat	concatena “strings” verticalmente ignorando os espaços em branco.
upper	converte letras minúsculas em letras maiúsculas.

A função eval

MATLAB possui uma poderosa função chamada **eval**, que executa o que tem no interior de uma “string”.

Se a “string” possui um comando que você deseja executar esta função é ideal. Por exemplo,

```
eval('x = 5*sin(pi/3)')
```

este comando atribui o valor $5 \cdot \sin(\pi/3)$ à variável *x*.

Exercícios

1. Entre com as matrizes a seguir:

$$A = \begin{bmatrix} 2 & 6 \\ 3 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad C = \begin{bmatrix} -5 & 5 \\ 5 & 3 \end{bmatrix}$$

2. Criando uma matriz nula, de 1's e identidade. Crie as seguintes matrizes com a ajuda das matrizes genéricas citadas acima.

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} E = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix} F = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$$

3. A seguinte matriz G é criada colocando as matrizes A, B e C citadas acima, na sua diagonal. De quantas maneiras você pode criar a matriz G, utilizando as submatrizes A, B e C?

$$G = \begin{bmatrix} 2 & 6 & 0 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 3 \end{bmatrix}$$

4. Crie uma matriz 20 x 20 com o comando `H = ones(20)`. Agora substitua a submatriz 10 x 10 entre as linhas 6 e 15 e as colunas 6 e 15 com zeros. Veja a estrutura da matriz (em termos de entradas diferentes de zero) com o comando `spy(H)`. Escolha as submatrizes 5 x 5 no canto superior direito e no canto inferior esquerdo para zerar e ver a estrutura de novo.
5. Crie uma matriz 10 x 10 qualquer com o comando `J = rand(10)`. Agora realize as seguintes operações:
- Multiplique todos os elementos por 100 e aproxime todos os elementos para inteiros com o comando `J = fix(J)`.
 - Substitua todos os elementos de `J < 10` por zeros.
 - Substitua todos os elementos de `J < 90` por infinito (`inf`).
 - Extraia todos os elementos entre 30 e 50 e coloque-os em um vetor `b`.

5. GRÁFICOS

5.1. GRÁFICOS BÁSICOS EM 2-D

O mais básico e talvez o mais útil comando para produzir um gráfico simples em 2-D é:

plot (x, y, ‘Opções de estilos’)

onde x e y são vetores contendo as coordenadas correspondentes de pontos no gráfico e *opções de estilos* é um argumento opcional para especificar a cor, estilo de linha (contínuo, tracejado, pontilhado, etc.), e o estilo da marcação dos pontos (o, -, *, +, etc) todas as três opções podem ser especificadas juntas na seguinte sequência: cor, estilo de linha e estilo de marcador.

Os dois vetores x e y DEVEM ter o mesmo número de elementos. Tamanhos diferentes entre os dois vetores é o caso mais comum de erro com o uso do comando **plot**.

Se a função **plot** for utilizada sem o componente x, ela considera que o valor das abscissas (x) são os valores dos índices i de y (i). Se por exemplo

y = [2 3 5 9], o comando plot(y) irá plotar os pontos: (1, 2),(2, 3),(3, 5),(4, 9).

5.2. OPÇÕES DE ESTILO

Cores		Estilo de linha		Estilo de marcação	
y	yellow	-	contínuo	+	sinal de mais
m	magenta	--	tracejado	o	círculo
c	cyan	..	pontilhado	*	asterisco
r	red	-.	traço ponto	x	marcação de x
g	green	none	sem linha	.	marcação de ponto
B	blue			^	circunflexo
W	white			square	quadrado
K	black			diamond	losango

5.3. RÓTULOS, TÍTULO, LEGENDA E OUTROS OBJETOS DE TEXTO

Gráficos podem ser anotados com os comandos **xlabel**, **ylabel**, **title** e **text**. O primeiro dos três comandos utilizam apenas 1 argumento de texto, enquanto o último utiliza 3.

Exemplo:

xlabel(“Comprimento do cano”)

Rótulo da abcissa.

ylabel(“Pressão do fluido”)

Rótulo da ordenada.

title(“Variação de Pressão”)

Título do gráfico.

text(2, 6, “Note este ponto”)

Insere um comentário no ponto (2, 6)

Os argumentos do comando text(x,y,’texto’) devem ser vetores, sendo que x e y devem ter o mesmo número de elementos e o texto deve ser uma “string” ou um

conjunto de “string”. Se ‘texto’ é um vetor então ele deve ter o mesmo número de elementos de x e y. Uma variável útil do comando **text** é o **gtext**, que pega os valores de x e y com um simples “clik” do mouse.

Legenda:

O comando **legend** produz uma caixa de legenda no gráfico.

<code>legend(string1, string2, ...)</code>	Produz uma legenda usando os textos das correspondentes “strings”.
<code>legend(LineStyle1, string1,...)</code>	Especifica o estilo de linha de cada rótulo.
<code>legend(...,pos)</code>	Escreve a legenda fora do “quadro de plotagem” se <i>pos</i> = -1 e dentro se <i>pos</i> = 0. (há outras opções para <i>pos</i> .)
<code>legend off</code>	Deleta a legenda do gráfico.”.

Quando o MATLAB é solicitado para produzir uma legenda, ele tenta encontrar um lugar no gráfico onde caiba o que está escrito nela sem atrapalhar os traçados do gráfico, o grid, e outros objetos. Ao argumento opcional *pos* especifica o local da caixa de legenda. *Pos* = 1 – superior direito (default); *pos* = 2 – superior esquerdo; *pos* = 3 – inferior esquerdo; *pos* = 4 – inferior direito. Mas é melhor mover a caixa de legenda com o mouse.

5.1 CONTROLE DOS EIXOS E DE ZOOM

Uma vez que um gráfico foi gerado você pode mudar os limites dos eixos com o comando **axis**. Escrevendo:

`axis([xmin xmax ymin ymax])`

onde os valores *xmin*, *xmax*, *ymin* e *ymax* são os respectivos valores dos limites dos eixos a serem colocados no comando.

O comando **axis** pode assim ser usado para aumentar uma seção particular do gráfico, ou diminuir (“zoom-in e zoom-out”). Há também alguns argumentos úteis ao comando.

<code>axis('equal')</code>	Escolhe uma escala igual para todos os eixos.
<code>axis('square')</code>	Transforma a tela padrão, que é retangular, para quadrada.
<code>axis('normal')</code>	Retorna os eixos para os valores padrão.
<code>axis('axis')</code>	Congela os limites do eixo atual.
<code>axis('off')</code>	Remove o quadro de plotagem e os valores do eixo.

Controle parcial dos eixos

É possível controlar somente uma parte dos limites dos eixos e deixar que o MATLAB complete os outros limites automaticamente. Isto é feito especificando os limites desejados com o comando `axis` e preenchendo com **inf** os limites que vocês gostaria que fossem escolhidos automaticamente. Por exemplo:

`axis([-5 5 -inf inf])` Somente os limites do eixo x são especificados, enquanto os do y são escolhidos pelo MATLAB.

5.4. GRÁFICOS SOBREPOSTOS

Há três diferentes modos de gerar gráficos sobrepostos no MATLAB: o `plot`, o `hold` e linhas de comandos.

Método 1: Usando o comando `plot`.

Se toda a seleção de dados está disponível, o comando `plot` com múltiplos argumentos pode ser usado para gerar gráficos sobrepostos. Por exemplo, se nós tivermos três seleções de dados (x_1, y_1) , (x_2, y_2) e (x_3, y_3) , o comando `plot(x1, y1, x2, y2, ':', x3, y3, 'o')`, desenha (x_1, y_1) com uma linha sólida, (x_2, y_2) com uma linha pontilhada e (x_3, y_3) como vários pontos separados em forma de círculos ('o'). Note que todos os vetores devem ter o mesmo tamanho.

Se todos os componente (x_1, y_1) , (x_2, y_2) e (x_3, y_3) , tiverem o mesmo tamanho, então é conveniente fazer uma matriz de X vetores e uma matriz de Y vetores de modo que $X = [x_1 \ x_2 \ x_3]$ e $Y = [y_1 \ y_2 \ y_3]$, assim o comando `plot(X,Y)` irá desenhar os três gráficos na mesma figura com cores diferentes.

Exemplo:

A seqüência de comandos abaixo irá gerar o gráfico a direita.

» `x = [1 2 3; 4 5 6; 7 8 9; 10 11 12]`

`x =`

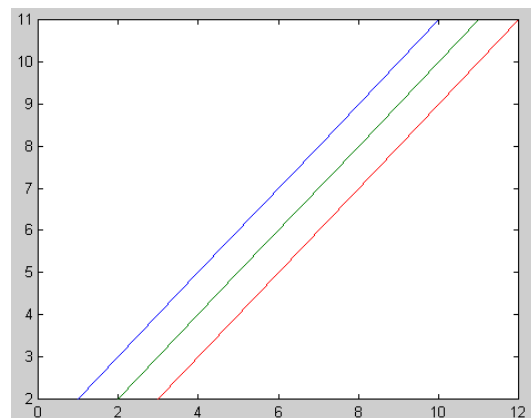
```
1   2   3
4   5   6
7   8   9
10  11  12
```

» `y = [2 2 2; 5 5 5; 8 8 8; 11 11 11]`

`y =`

```
2   2   2
5   5   5
8   8   8
11  11  11
```

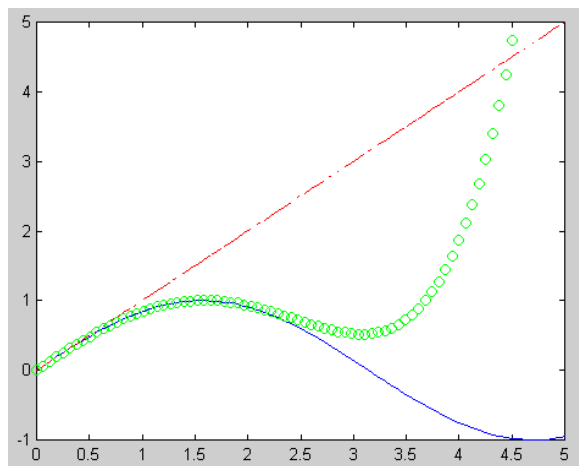
» `plot(x,y)`



Método 2: Usando o comando hold para gerar gráficos sobrepostos.

Um outro modo de fazer gráficos sobrepostos é usando o comando hold. Digitando 'hold on' em qualquer ponto durante a sessão congela o gráfico corrente na janela de gráficos. Todos os subseqüentes gráficos gerados pelo comando plot são simplesmente adicionados (e não substituídos) ao gráfico existente. O seguinte arquivo procedimento (*Script file*) mostra como gerar o mesmo gráfico como na figura abaixo, usando o comando hold.

```
% - Procedimento que gerar gráficos sobrepostos com o comando hold -
x = linspace(0,2*pi,100);      % Cria o vetor x.
y1 = sin(x);                  % Calcula y1.
plot(x,y1)                    % Desenha (x,y1) com uma linha sólida.
hold on                       % Congela a figura.
y2 = x; plot(x,y2,'r--')      % Desenha (x,y2) com uma linha tracejada.
y3 = x-(x.^3)/6+(x.^5)/120;   % Calcula y3.
plot(x,y3,'go')               % Desenha (x,y3) com círculo.
axis([0 5 -1 5])              % ajusta os eixos do gráfico.
hold off                      % Descongela a figura.
```



Método 3: Usando o comando line para gerar gráficos sobrepostos

O line é um comando gráfico de baixo nível o qual é usado pelo comando plot para gerar linhas. Uma vez que exista um gráfico na janela de gráfico, você pode adicionar linhas utilizando o comando line diretamente. O comando line utiliza um par de vetores (ou um trio em 3D) seguidos de *ParameterName* / *ParameterValue* como argumentos:

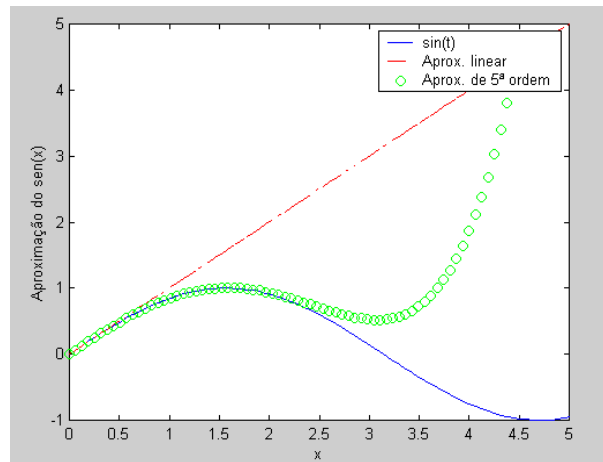
`line(xdata, ydata, ParameterName, ParameterValue)`

O mesmo gráfico acima pode ser feito com o comando line ao invés de ser feito com o comando hold, conforme a seguir:

```
% - Procedimento que gerar gráficos sobrepostos com o comando line -
x = linspace(0,2*pi,100); % Cria o vetor x.
y1 = sin(x); % Calcula y1.
y2 = x; % Calcula y2.
y3 = x-(x.^3)/6+(x.^5)/120; % Calcula y3.

plot(x,y1) % Desenha (x,y1) com uma linha sólida.
line(x,y2,'linestyle','r--') % Desenha (x,y2) como uma linha tracejada.
line(x,y3,'linestyle','go') % Desenha (x,y3) com círculos.

axis([0 5 -1 5]) % ajusta os eixos do gráfico.
xlabel('x')
ylabel('Aproximação do sen(x)')
legend('sin(t)', 'Aprox. linear', 'Aprox. de 5ª ordem')
```



A legenda da figura acima pode ser arrastada para uma outra posição clicando e segurando o com o mouse em cima da legenda.

5.5. GRÁFICOS ESPECÍFICOS EM 2-D

Existem muitas funções específicas de gráficos em 2-D. Elas são usadas como alternativas para o comando plot. Aqui está uma lista dos possíveis modos de desenho em 2D:

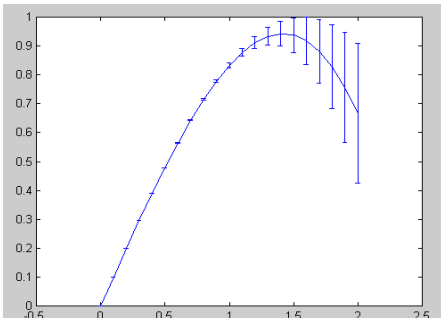
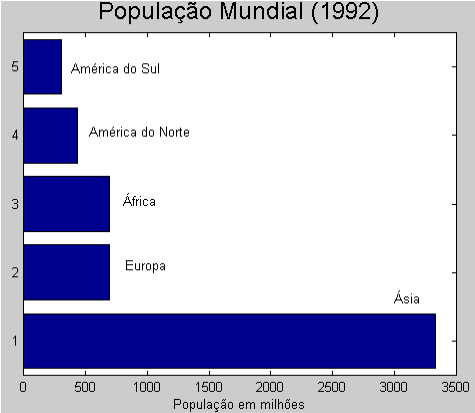
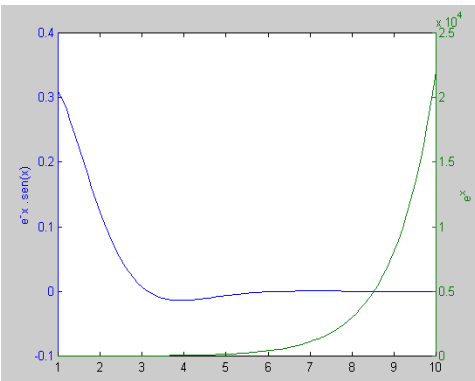
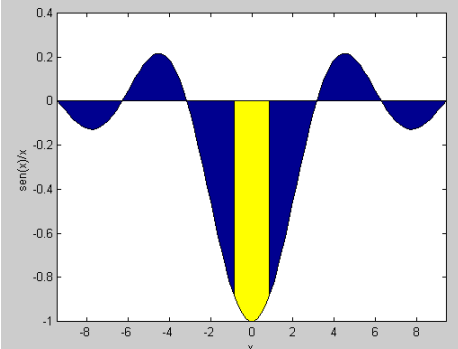
area	cria uma área preenchida.
bar	cria um gráfico de barras.
barh	cria um gráfico de barras horizontais.
comet	faz uma animação 2D.
compass	cria um gráfico de vetores para números complexos.
contour	faz um gráfico de curvas de nível.
contourf	faz um gráfico de curvas de nível preenchidas.
errorbar	desenha um gráfico com barras de erro.
feather	faz um gráfico do vetor velocidade.
fill	desenha um polígono preenchido com uma cor especificada.
fplot	desenha uma função de uma única variável.
hist	faz um histograma.

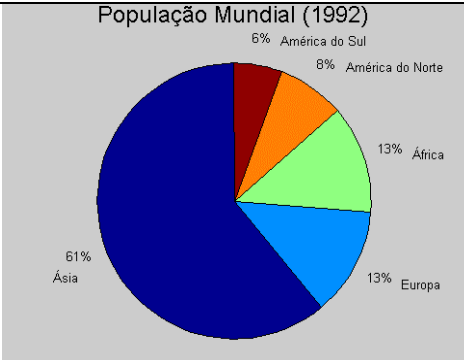
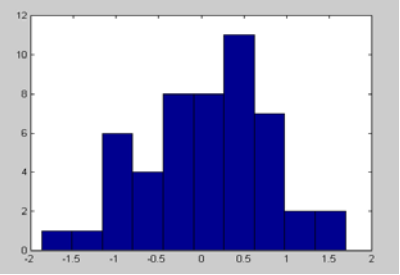
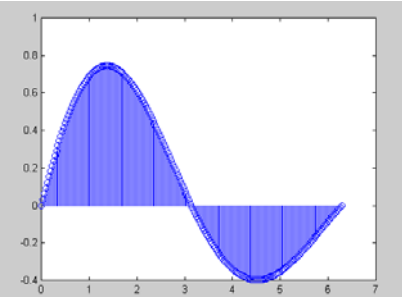
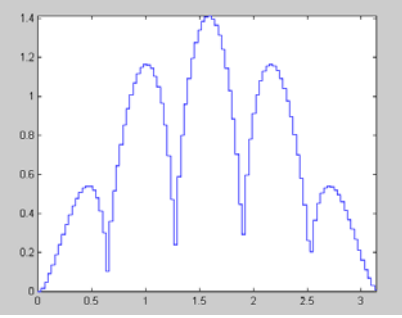
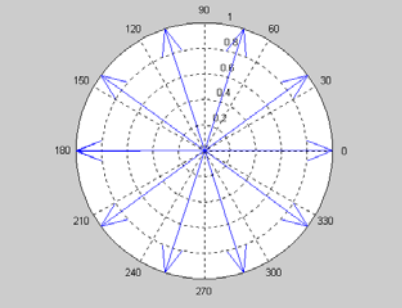
loglog	cria gráficos com escala logarítmica em ambos os eixos.
pcolor	faz um gráfico de uma matriz, onde cada célula possui uma cor de acordo com o seu valor.
pie	faz um gráfico em forma de pizza.
plotyy	faz um gráfico com duplo eixo y.
plotmatrix	faz uma matriz espalhada.
polar	desenha curvas em coordenadas polares.
quiver	desenha campos de vetores.
rose	faz histogramas em coordenadas polares.
scatter	cria um gráfico espalhado.
semilogx	faz um gráfico semilog com escala logarítmica no eixo x.
semilogy	faz um gráfico semilog com escala logarítmica no eixo y.
stairs	desenha um gráfico em degraus.
stem	desenha um gráfico em fatias.

Mostraremos exemplos destas funções a seguir.

Função	Comando	Resultado
Fplot	$f(t) = t \sin(t), 0 \leq t \leq 10\pi$ <code>fplot('x.*sin(x)', [0 10*pi])</code> Note que a função para ser desenhada deve ser escrita como uma função de x.	
Semilogx	$x = e^{-t}, y = t, 0 \leq t \leq 2\pi$ <code>t = linspace(0, 2*pi, 200);</code> <code>x = exp(-t); y = t;</code> <code>semilogx(x, y), grid;</code>	
Semilog y	$x = t, y = e^t, 0 \leq t \leq 2\pi$ <code>t = linspace(0, 2*pi, 200);</code> <code>semilogy(t, exp(t))</code> <code>grid</code>	

Função	Comando	Resultado
Loglog	$x = e^t, y = 100 + e^{2t}, 0 \leq t \leq 2\pi$ <code>t = linspace (0, 2*pi, 200);</code> <code>x = exp(t); y = 100 + exp(2*t);</code> <code>loglog (x, y);</code> <code>grid</code>	
Polar	$r^2 = 2 \sin(5t), 0 \leq t \leq 2\pi$ <code>t = linspace (0, 2*pi, 200);</code> <code>r = sqrt (abs(2*sin(5*t)));</code> <code>polar (t, r)</code>	
fill	$r^2 = 2 \sin(5t), 0 \leq t \leq 2\pi$ $x = r \cos(t), y = r \sin(t);$ <code>t = linspace (0, 2*pi, 200);</code> <code>r = sqrt (abs(2*sin(5*t)));</code> <code>x = r.*cos(t); y = r.*sin(t);</code> <code>fill (x, y, 'k');</code> <code>axis('square');</code>	
bar	$r^2 = 2 \sin(5t), 0 \leq t \leq 2\pi$ $y = r \sin(t);$ <code>t = linspace (0, 2*pi, 200);</code> <code>r = sqrt (abs(2*sin(5*t)));</code> <code>y = r.*sin(t);</code> <code>bar (t, y);</code> <code>axis([0 pi 0 inf]);</code>	

Função	Comando	Resultado
errorbar	$f_{\text{aprox}} = x - x^3/3!, 0 \leq t \leq 2\pi$ $\text{error} = f_{\text{aprox}} - \text{sen}(x).$ $x = 0: .1: 2;$ $\text{aprx2} = x - x.^3/6;$ $\text{er} = \text{aprx2} - \text{sin}(x);$ $\text{errorbar}(x, \text{aprx2}, \text{er});$	
hist	<p>População mundial por continentes</p> $\text{cont} = \text{char}(\text{'Ásia'}, \text{'Europa'}, \text{'África'}, \dots \text{'América do Norte'}, \text{'América do Sul'});$ $\text{pop} = [3332; 696; 694; 437; 307];$ $\text{barh}(\text{pop})$ $\text{for } i = 1:5,$ $\quad \text{gtext}(\text{cont}(i, :));$ end $\text{xlabel}(\text{'População em milhões'});$ $\text{title}(\text{'População Mundial (1992)'}, \dots \text{'FontSize', } 18);$	
Plotyy	$y_1 = e^{-x} \text{sen}(x), 0 \leq t \leq 10$ $y_2 = e^x$ $x = 1: .1: 10;$ $y_1 = \exp(-x) \cdot \text{sin}(x);$ $y_2 = \exp(x);$ $\text{Ax} = \text{plotyy}(x, y_1, x, y_2);$ $\text{hy1} = \text{get}(\text{Ax}(1), \text{'ylabel'});$ $\text{hy2} = \text{get}(\text{Ax}(2), \text{'ylabel'});$ $\text{set}(\text{hy1}, \text{'string'}, \text{'e}^{-x} \cdot \text{sen}(x));$ $\text{set}(\text{hy2}, \text{'string'}, \text{'e}^x);$	
Area	$y = \text{sen}(x)/x, -3\pi \leq x \leq 3\pi$ $x = \text{linspace}(-3 \cdot \pi, 3 \cdot \pi, 100);$ $y = -\text{sin}(x)/x;$ $\text{area}(x, y)$ $\text{xlabel}(\text{'x'}); \text{ylabel}(\text{'sen}(x)/x'});$ hold on $x_1 = x(46:55); y_1 = y(46:55);$ $\text{area}(x_1, y_1, \text{'facecolor'}, \text{'y'});$	

Função	Comando	Resultado																		
Pie	<p>População mundial por continentes</p> <pre> cont = char('Ásia', 'Europa', 'África', ... 'América do Norte', 'América do Sul'); pop = [3332; 696; 694; 437; 307]; pie (pop) for i = 1:5, gtext (cont (i, :)); end title ('População Mundial (1992)', ... 'FontSize', 18); </pre>	 <p>População Mundial (1992)</p> <table border="1"> <thead> <tr> <th>Continent</th> <th>Population (millions)</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Ásia</td> <td>3332</td> <td>61%</td> </tr> <tr> <td>Europa</td> <td>696</td> <td>13%</td> </tr> <tr> <td>África</td> <td>694</td> <td>13%</td> </tr> <tr> <td>América do Norte</td> <td>437</td> <td>8%</td> </tr> <tr> <td>América do Sul</td> <td>307</td> <td>6%</td> </tr> </tbody> </table>	Continent	Population (millions)	Percentage	Ásia	3332	61%	Europa	696	13%	África	694	13%	América do Norte	437	8%	América do Sul	307	6%
Continent	Population (millions)	Percentage																		
Ásia	3332	61%																		
Europa	696	13%																		
África	694	13%																		
América do Norte	437	8%																		
América do Sul	307	6%																		
Hist	<p>Histograma de 50 de números distribuídos aleatoriamente entre 0 e 1.</p> <pre> y = randn (50,1); hist (y) </pre>																			
stem	<p>$f = e^{-t/5} \sin t, 0 \leq t \leq 2\pi$</p> <pre> t = linspace(0, 2*pi, 200); f = exp(-.2*t).* sin(t); stem(t, f) </pre>																			
stairs	<p>$r^2 = 2 \sin 5t, 0 \leq t \leq 2\pi$ $y = r \sin t$</p> <pre> t = linspace(0, 2*pi, 200); r = sqrt(abs(2*sin(5*t))); y = r.*sin(t); stairs (t, y) axis([0 pi 0 inf]) </pre>																			
compass	<p>$z = \cos \theta + i \sin \theta, 0 \leq \theta \leq 2\pi$</p> <pre> th = -pi: pi/5:pi; zx = cos(th); zy = sin(th); z = zx + i*zy; compass (z); </pre>																			

Função	Comando	Resultado
comet	$y = t \sin t, 0 \leq t \leq 10\pi$ <code>q = linspace(0, 10*pi, 200);</code> <code>y = q.*sin(q);</code> <code>comet (q, y);</code> Obs.: É melhor visualizar no MATLAB, pois ele é desenhado gradualmente.	
contour	$z = -(x^2/2) + xy + y^2, -5 \leq x \leq 5$ $-5 \leq y \leq 5$ <code>r = -5: .2:5;</code> <code>[X, Y] = meshgrid (r, r);</code> <code>Z = -.5*X.^2 + X.*Y + Y.^2;</code> <code>cs = contour (X, Y, Z);</code> <code>clabel (cs);</code>	
quiver	$z = x^2 + y^2 - 5\sin(xy),$ $-2 \leq x \leq 2, -2 \leq y \leq 2$ <code>r = -2: .2: 2;</code> <code>[X, Y] = meshgrid (r, r);</code> <code>Z = X.^2 + Y.^2 - 5.*sin(X.*Y);</code> <code>[dx, dy] = gradient (Z, .2, .2);</code> <code>quiver (X, Y, dx, dy, 2);</code>	
pcolor	$z = x^2 + y^2 - 5\sin(xy),$ $-2 \leq x \leq 2, -2 \leq y \leq 2$ <code>r = -2: .2: 2;</code> <code>[X, Y] = meshgrid (r, r);</code> <code>Z = X.^2 + Y.^2 - 5.*sin(X.*Y);</code> <code>pcolor (Z), axis('off');</code> <code>shading interp</code>	

5.6. USANDO O COMANDO SUBPLOT

Se você quer fazer alguns gráficos lado a lado numa mesma figura (não sobrepostos), use o comando subplot para desenhar os seus gráficos.

O comando subplot requer 3 argumentos:

subplot (m, n, p)

O subplot divide a janela da figura em m x n sub-janelas e coloca o próximo gráfico gerado na p-ésima sub-janela, sendo p contado da esquerda para a direita e de cima para baixo. Então, o comando subplot (2, 2, 3), plot (x, y) divide a janela de gráficos

em 4 partes e insere o gráfico de (x, y) na terceira sub-janela, que é a primeira sub-janela da segunda linha.

5.7. GRÁFICOS 3D

O MATLAB dispõe de diversas funções que facilitam a visualização do gráfico 3D.

De fato os colormap embutidos podem ser usados para representar a quarta dimensão. As facilidades dispostas incluem funções embutidas para plotar curvas espaciais, objetos em forma de grade, superfícies, superfícies com textura, gerando contornos automaticamente, especificando os pontos de luz, interpolando cores e textura e mesmo imagens digitando help graph3d no comando você obtém uma lista de funções viáveis para gráficos 3D gerais. Aqui está uma lista dos mais comuns:

plot3	plot curvas no espaço
stem3	cria dados discretos na plotagem fatiados.
bar3	desenha gráficos 3D em barras.
bar3h	idem acima para barras horizontais.
pie3	desenha o gráfico de pizza 3D.
comet3	faz animação gráfica para gráficos 3D.
fill3	desenha figuras 3D preenchidas.
contour3	faz os contornos de um gráfico 3D.
guiver3	desenha vetores de campo em 3D.
scatter3	faz gráficos 3D espalhados.
mesh	desenha superfície.
meshc	desenha superfície ao longo de contornos.
meshz	desenha superfície com cortinas.
surf	desenha superfície em 3D (igual ao mesh, porém preenchido).
surfc	idem meshc, porém preenchido.
surfl	cria superfície com as fontes de luz especificadas.
trimesh	mesh com triângulos.
trisurf	surf com triângulos.
slice	desenha uma superfície volumétrica em pedaços.
waterfall	cria um gráfico em forma de cachoeira.
cylinder	cria um cilindro.
sphere	cria uma esfera.

Entre estas funções plot3 e comet3 são análogos ao plot e ao comet do 2D.

5.8. COMANDO VIEW

O ângulo de vista do observador é especificado com o comando
view (azimuth, elevação)

O azimuth e a elevação são em graus. O azimuth é medido tendo o eixo z como referência, contado no sentido anti-horário. A elevação é positiva acima do plano xy.

O script file abaixo gera os dados, plota as curvas e obtém diferentes pontos de vista.

Existem dois tipos específicos do comando view, especificando o padrão 2D e 3D.

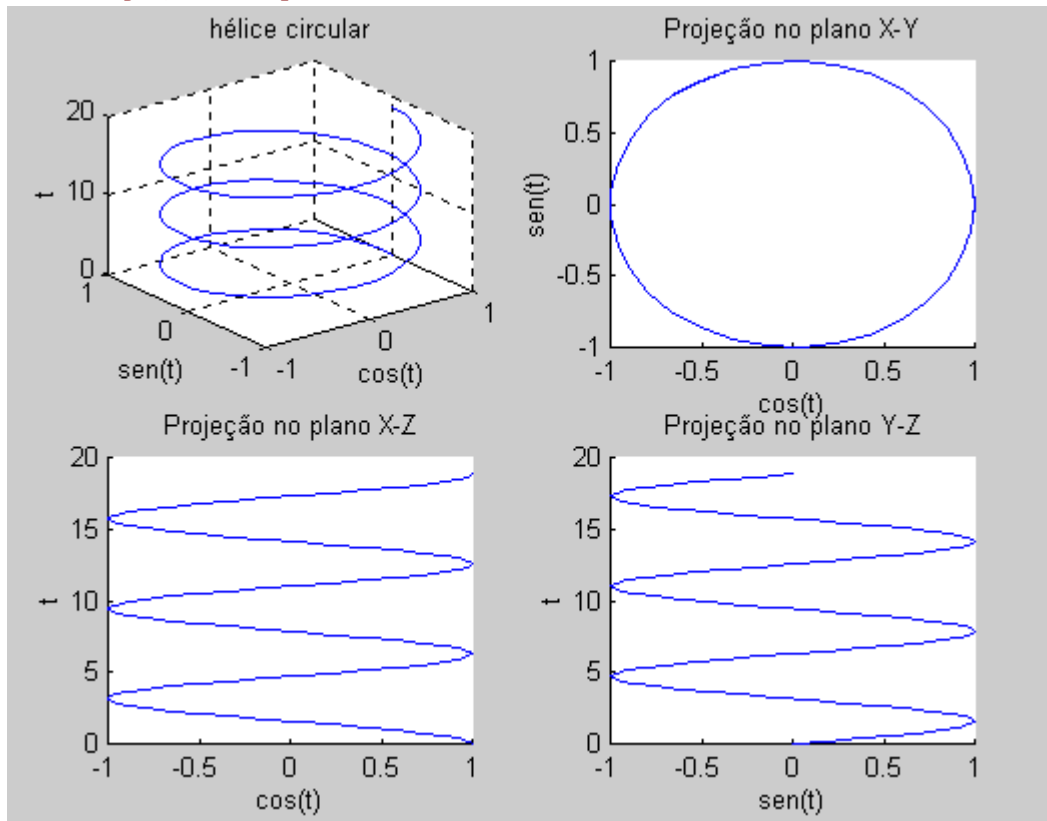
```
view(2)  é o mesmo que view(0, 90). (padrão 2D)
view(3)  é o mesmo que view(37.5, 30). (padrão 3D)
% --- Script file com exemplos de subplot e view ---
clf                                %limpar a figura anterior.
t = linspace(0, 6*pi, 100);       %Gera o vetor t.

x = cos(t); y = sin(t); z = t;    %Calcula x, y, z.
subplot(2,2,1)                    %Divide a janela de gráfico em 4
                                  %e plota na primeira janela.
plot3(x,y,z); grid                %Plota a curva 3D com grades.
xlabel('cos(t)'), ylabel('sen(t)'), zlabel('t')
title('hélice circular');

subplot(2,2,2)                    %Plota na segunda sub-janela.
plot3(x,y,z), view(0,90),         %Vista de cima pelo eixo z.
xlabel('cos(t)'), ylabel('sen(t)'), zlabel('t')
title('Projeção no plano X-Y')

subplot(2,2,3)                    %Plota na terceira sub-janela.
plot3(x,y,z), view(0,0),          %Vista ao longo do eixo y.
xlabel('cos(t)'), ylabel('sen(t)'), zlabel('t')
title('Projeção no plano X-Z')

subplot(2,2,4)                    %Plota na quarta sub-janela.
plot3(x,y,z), view(90,0),         %Vista ao longo do eixo x.
xlabel('cos(t)'), ylabel('sen(t)'), zlabel('t')
title('Projeção no plano Y-Z')
```



5.9. ROTACIONAR A VISTA

Existe um comando chamado `rotate3d`, que simplesmente possibilita girar o gráfico 3D com o mouse. Existem também funções como `zoom`, `roll`, `pan`, etc. Veja a ajuda on-line no `graph3D`.

5.10. GRÁFICOS DE MALHA E SUPERFÍCIE

As funções para plotagem de malhas e superfícies (`mesh` e `surf`) e suas variáveis `meshz`, `meshc`, `surfc` e `surfz` possuem múltiplos argumentos opcionais, a maioria formando `mesh(Z)` e `surf(Z)` onde z representa uma matriz. Superfícies geralmente são representadas pelos valores de coordenada z mostrados em uma grade de valores (x,y) . Portanto para criar uma superfície primeiramente precisamos gerar uma grade (x,y) de coordenadas e definir a “altura” (a coordenada z) das superfícies de cada um dos pontos da grade. Note que você precisa fazer o mesmo para plotar uma função de duas variáveis. O MATLAB dispõe da função `meshgrid` para criar uma grade de pontos abaixo de um alcance especificado.

Exemplo: Plotar a função $z = x^2 - y^2$,
entre o domínio de $-4 \leq x \leq 4$ e $-4 \leq y \leq 4$.

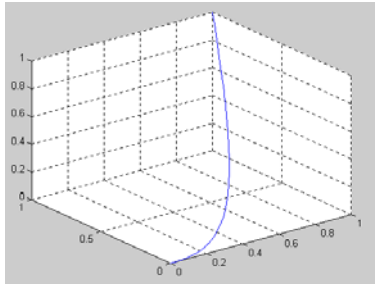
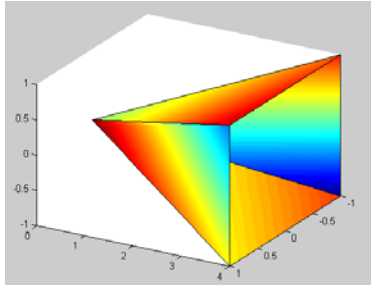
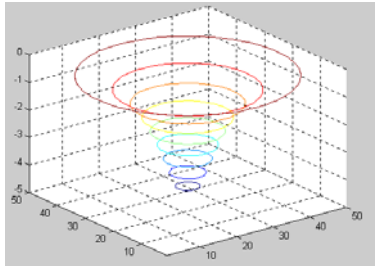
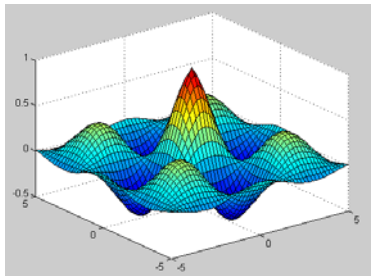
Para fazer isso, nós primeiro formamos 25 pontos na grade. Nós podemos criar duas matrizes X e Y sendo cada uma de tamanho 5×5 e escrever as coordenadas xy de cada ponto dessas matrizes. Nós podemos então avaliar z com o comando $z = X.^2 - Y.^2$. Criar as duas matrizes X e Y é muito mais fácil com o comando `meshgrid`:

```
rx = 0:4;           % cria um vetor rx = [0 1 2 3 4]
ry = -4:2:4;        % cria um vetor ry = [-4 -2 0 2 4]
[X,Y] = meshgrid(rx,ry); % cria um grid de 25 pontos e armazena
                        % essas % coordenadas em X e Y.
```

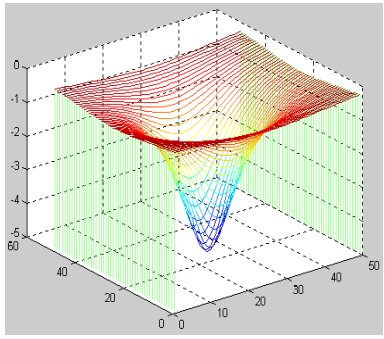
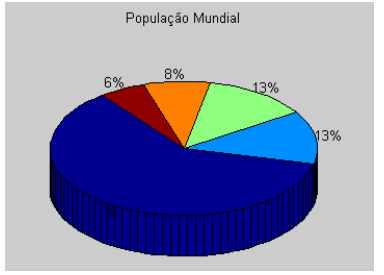
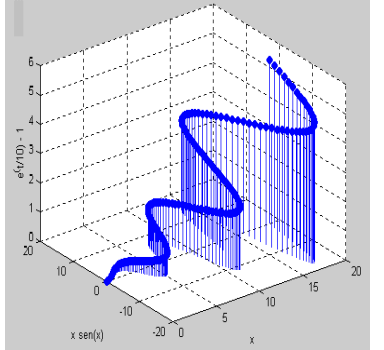
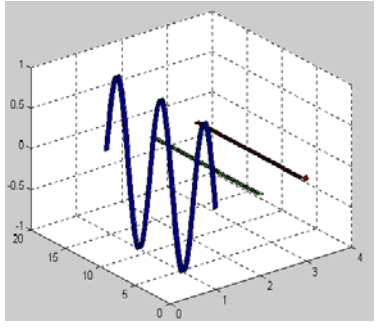
```
X =
    0    1    2    3    4
    0    1    2    3    4
    0    1    2    3    4
    0    1    2    3    4
    0    1    2    3    4
```

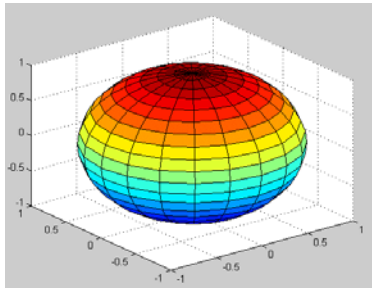
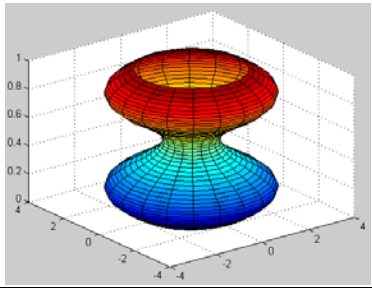
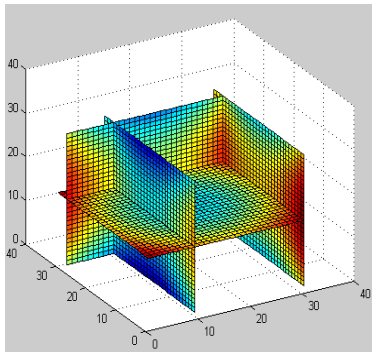
```
Y =
   -4   -4   -4   -4   -4
   -2   -2   -2   -2   -2
    0    0    0    0    0
    2    2    2    2    2
    4    4    4    4    4
```


Os comandos mostrados acima geraram os 25 pontos necessários. A grade pode ser criada com o comando meshgrid: [X,Y] = meshgrid(rx,ry); quando rx e ry são vetores especificando o local das linhas de grade ao longo dos eixos x e y. Tudo o que nós precisamos é gerar dois vetores, rx e ry, para definir a região de interesse de distribuição dos pontos. Também os dois vetores não precisam ser do mesmo tamanho ou linearmente espaçados (contudo, na maioria das vezes nós pegamos a região quadrada e criamos os pontos de grade igualmente espaçados em ambas as direções). Para utilizar bem gráficos 3D é preciso entender o uso do meshgrid

Função	Comando	Resultado
plot3	<p>Desenho de uma curva paramétrica espacial:</p> $x(t) = t, y(t) = t^2, z(t) = t^3. 0 \leq t \leq 1$ <pre>t = linspace (0, 1, 100); x = t; y = t.^2; z = t.^3; plot3 (x, y, z), grid</pre>	
fill3	<p>Desenho de 4 polígonos preenchidos com 3 vértices cada:</p> <pre>X = [0 0 0 0; 1 1 -1 1; 1 -1 -1 -1]; Y = [0 0 0 0; 4 4 4 4; 4 4 4 4]; Z = [0 0 0 0; 1 1 -1 -1; -1 1 1 -1]; fill3 (X, Y, Z, rand (3, 4)); view (120, 30)</pre>	
contour3	<p>Desenho de linhas de contorno 3D de:</p> $Z = -5 / (1 + x^2 + y^2), -3 \leq x \leq 3, -3 \leq y \leq 3.$ <pre>r = linspace (-3, 3, 50); [x, y] = meshgrid (r, r); z = -5./(1 + x.^2 + y.^2); contour3 (z).</pre>	
surf	$z = \cos x \cdot \cos y \cdot e^{-\frac{\sqrt{x^2+y^2}}{4}}, -5 \leq x \leq 5, -5 \leq y \leq 5$ <pre>u = -5: .2: 5; [X, Y] = meshgrid (u, u); Z = cos(X).* cos(Y).* ... exp(-sqrt(X.^2 +Y.^2)/4); surf(X, Y, Z)</pre>	

Função	Comando	Resultado
surfc	$z = \cos x \cdot \cos y \cdot e^{-\frac{\sqrt{x^2+y^2}}{4}}, -5 \leq x \leq 5, -5 \leq y \leq 5$ <pre> u = -5: .2: 5; [X, Y] = meshgrid (u, u); Z = cos(X).* cos(Y).* ... exp(-sqrt(X.^2 +Y.^2)/4); surfc (Z) view (-37.5, 20) axis('off ') </pre>	
surfl	$z = \cos x \cdot \cos y \cdot e^{-\frac{\sqrt{x^2+y^2}}{4}}, -5 \leq x \leq 5, -5 \leq y \leq 5$ <pre> u = -5: .2: 5; [X, Y] = meshgrid (u, u); Z = cos(X).* cos(Y).* ... exp(-sqrt(X.^2 +Y.^2)/4); surfl (Z) shading interp colormap hot </pre>	
mesh	$z = -5/(1+x^2+y^2), -3 \leq x \leq 3, -3 \leq y \leq 3.$ <pre> x = linspace (-3, 3, 50); y = x; [x, y] = meshgrid (x, y); z = -5./(1+x.^2+y.^2); mesh (z) </pre>	
meshz	$z = -5/(1+x^2+y^2), -3 \leq x \leq 3, -3 \leq y \leq 3.$ <pre> x = linspace (-3, 3, 50); y = x; [x, y] = meshgrid (x, y); z = -5./(1+x.^2+y.^2); meshz (z) view (-37.5, 50) </pre>	

Função	Comando	Resultado
waterfall	$z = -5/(1+x^2+y^2)$, $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. <code>x = linspace (-3, 3, 50);</code> <code>y = x;</code> <code>[x, y] = meshgrid (x, y);</code> <code>z = -5./(1+x.^2+y.^2);</code> <code>waterfall (z)</code> <code>hidden off</code>	
pie3	<p>População mundial por continente.</p> <p>% Popdata: As, Eu, Af, NA, AS</p> <p>pop = [3332; 696; 694; 437; 307];</p> <p>p ie3 (pop)</p> <p>title ('População Mundial')</p>	
stem3	<p>Desenho de dados discretos com fatias.</p> <p>$x = t$, $y = t \sin(t)$, $z = e^{t/10} - 1$. $0 \leq t \leq 6\pi$.</p> <p><code>t = linspace (0, 6*pi, 200);</code> <code>x = t; y= t.*sin(t);</code> <code>z = exp(t/10) - 1;</code> <code>stem3 (x, y, z, 'filled')</code> <code>xlabel ('x')</code>, <code>ylabel ('x sen(x)')</code>, <code>zlabel ('e^(t/10) - 1')</code>;</p>	
ribbon	<p>Curvas 2D desenhadas como tiras 3D</p> <p>$y_1 = \sin t$, $y_2 = e^{-15t} \sin t$, $y_3 = e^{-8t} \sin t$ $0 \leq t \leq 5\pi$.</p> <p><code>t = linspace (0, 5*pi, 100);</code> <code>y1 = sin(t);</code> <code>y2 = exp(-15*t).*sin(t);</code> <code>y3 = exp(-8*t).*sin(t);</code> <code>y = [y1; y2; y3];</code> <code>ribbon (t', y', .1);</code></p>	

Função	Comando	Resultado
Sphere	<p>Uma unidade de esfera centrada na origem e gerada por 3 matrizes x, y e z de tamanho 21x21 cada.</p> <pre>sphere (20) ou [x, y, z] = sphere (20); surf (x, y, z);</pre>	
cylinder	<p>Um cilindro gerado por</p> $r = \sin(3\pi z) + 2,$ $0 \leq z \leq 1, 0 \leq \theta \leq 2\pi.$ <pre>z = [0: .02: 1]'; r = sin(3*pi*z) + 2; cylinder (r)</pre>	
slice	<p>Pedaços de uma função volumétrica.</p> $f(x, y, z) = x^2 + y^2 - z^2;$ $-3 \leq x \leq 3, -3 \leq y \leq 3, -3 \leq z \leq 3.$ <pre>v = [-3: .2: 3]; [x, y, z] = meshgrid (v, v, v); f = (x.^2 + y.^2 - z.^2); xrows = [10, 31]; yrows = 28; zrows = 16; slice (f, xrows, yrows, zrow); view ([-30 30])</pre> <p>O valor da função é indicada por uma intensidade de cor.</p>	

5.10.1. GRÁFICOS DE SUPERFÍCIE INTERPOLADOS

Muitas vezes, nós obtemos dados (geralmente a partir de experimentos) na forma da tripla (x, y, z) , e queremos ajustar uma superfície a partir dos dados. Assim, nós temos um vetor Z que contém os z -valores correspondendo aos valores x e y irregularmente espaçados. Aqui nós não temos uma grade regular, como a criada pelo comando `meshgrid`, e não temos uma matriz Z que contém os z -valores da superfície naqueles pontos da grade. Portanto, nós temos que ajustar a superfície a partir das variáveis dadas (x_i, y_i, z_i) . A incumbência é muito mais simples do que parece. O MATLAB dispõe de uma função, **griddata**, que faz esta interpolação para nós. A sintaxe geral desta função é

$[X_i, Y_i, Z_i] = \text{griddata}(x, y, z, x_i, y_i, \text{método})$

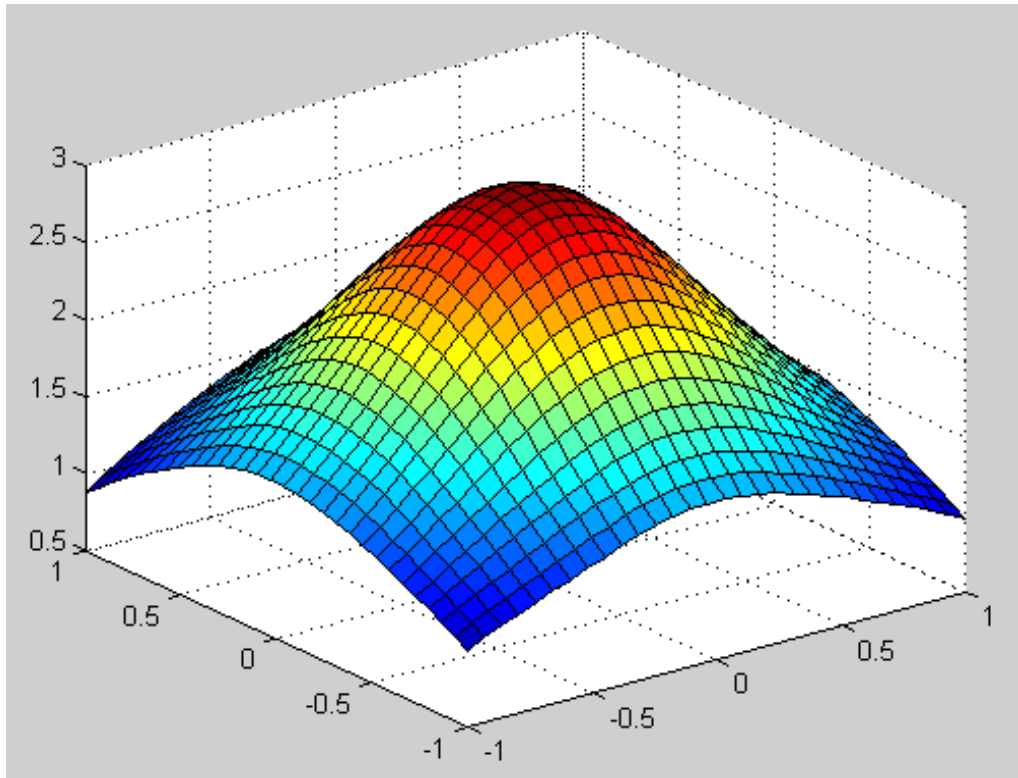
onde x, y, z são os vetores dados (espaçados não-uniformemente), x_i e y_i são os pontos prescritos pelo usuário (com certeza, uniformemente espaçados) no qual z_i são calculados pela interpolação, e método é a escolha do algoritmo de interpolação. Os algoritmos disponíveis são *nearest*, *linear*, *cubic* e *v4*. Veja a documentação on-line para a descrição destes métodos.

Como exemplo vamos considerar 50 pontos randomicamente distribuídos no plano xy , no limite $-1 < x < 1$, e $-1 < y < 1$. Os valores de z nestes pontos são determinados por $z = 3/(1+x^2+y^2)$. Assim, nós temos 3 vetores de comprimento 50 cada um. Os dados dos pontos são mostrados na figura pelo *stem plot*.

```
% SURFINTERP: Script file que gera uma superfície interpolada
% Dado os vetores x, y, e z, gera a matriz de dados Zi da interpolação para ajustar uma
superfície
%-----

xv=2*rand(1,100)-1;           %este é o vetor x
yv=2*rand(1,100)-1;           %este é o vetor y
zv=3./(1+xv.^2+yv.^2)         %este é o vetor z
stem3(xv, yv, zv)              %mostra os dados como um gráfico stem
xi=linspace(-1, 1, 30);        %cria xi com os dados uniformemente
espaçados
yi=xi';                        %cria yi com os dados uniformemente
espaçados                      %observe que yi é uma coluna

[Xi, Yi, Zi]=griddata(xv, yv, zv, xi, yi, 'v4');
%superfície interpolada usando o método v4 (MATLAB 4
% griddata) de interpolação
surf(Xi,Yi,Zi)                 %plota a superfície
```



5.10.2. MANEJAR COM GRÁFICOS

A linha é um objeto gráfico. Ela tem várias propriedades – tipo de linha, cor, intensidade, visibilidade, etc. Uma vez que uma linha é desenhada numa tela de um gráfico, é possível mudar qualquer uma das propriedades depois. Suponha que você queira mudar umas das linhas, você primeiro deve escolher a linha que você quer mudar e então mudar o que você não gosta nela. Num gráfico, uma linha pode estar entre vários objetos gráficos (eixos, textos, legendas, etc). Então como você pode identificar esta linha? Você consegue identificar uma linha pela sua *handle*.

O que é handle? O MATLAB atribui um floating-point number para todo objeto na figura (incluindo objetos invisíveis), e usa este número como um endereço ou nome para o objeto na figura. Este número é o handle do objeto.

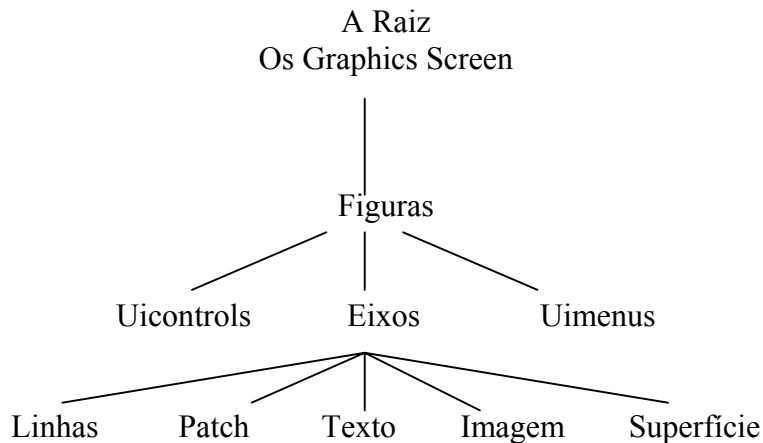
Uma vez que você obtenha o handle, você pode acessar todas as propriedades do objeto. Em suma, o handle identifica o objeto e o objeto carrega com ele a lista de suas propriedades. Em programação, este approach de definir objetos e suas propriedades é chamado programação orientada a objeto. A vantagem que isso oferece é que você pode acessar objetos individuais e suas propriedades e mudar qualquer propriedade de um objeto sem afetar outras propriedades ou objetos. Assim, você obtém um controle completo sobre os objetos gráficos. Todo o sistema do MATLAB de gráficos orientados a objeto e sua controlabilidade é referido ao HANDLE GRAPHICS. Aqui nós discutiremos resumidamente este sistema e seu uso, mas só aconselhamos aos leitores mais interessados a consultar o User Guide para maiores detalhes.

A chave para o entendimento e uso do sistema HANDLE GRAPHICS é saber como obter os handles dos objetos gráficos e como usar os handles para obter e mudar as

propriedades dos objetos. Uma vez que nem todos os objetos gráficos são independentes (por exemplo, o aparecimento de uma linha está ligado à um eixo em uso), uma certa propriedade de um deles pode afetar as propriedades de outros, então é importante saber como os objetos se relacionam.

5.10.3. A HIERARQUIA DOS OBJETOS

Os objetos gráficos seguem uma hierarquia de parentesco pai-filho.



É importante conhecer esta estrutura por dois motivos:

Mostra para você quais objetos são afetados se você mudar um valor default de uma propriedade em um nível em particular, e

Diz a você em qual nível você pode perguntar por handles de quaisquer objetos.

5.10.4. OBJETOS HANDLES

Objetos Handles são identificadores únicos associados a cada objeto gráfico. Estes handles têm uma representação em ponto flutuante. Handles são criados no momento da criação do objeto por funções gráficas tais como **plot(x , y)**, **contour(z)**, **line(z1 , z2)**, **text(xc , yc , ‘Olhe aqui seu mané’)**, etc.

Obtendo Objetos Handles

Existem duas formas de obtenção de handles:

1. Criação de handles explicitamente por meio dos comandos object-creation-level (isto é, você pode plotar e obter seu handle na mesma hora):

`h1 = plot (x , y , ‘r-‘)` retorna o handle da linha para h1

`hx1 = xlabel (‘ângulo’)` retorna o handle do eixo x para hx1

2. Pelo uso explícito de funções de retorno de handles:

gfc obtém o handle da figura atual

Exemplo: `hfig = gfc`, retorna o handle da figura em hfig.

gca obtém o handle do eixo atual
Exemplo: haxes = gca, retorna o handle do eixo atual em haxes.
gco obtém o handle do objeto atual

Handles de outros objetos pode ser obtido através do comando **get**. Por exemplo, **hlines = get (gca , ‘children’)** retorna os handles de todos os “filhos” do eixo atual em um vetor coluna hlines. A função **get** é usada para obter o valor da propriedade de um objeto, especificado por seu handle, na forma do comando
get (handle, ‘Nome da Propriedade’)

Para um objeto com handle h, digite **get(h)** para obter uma lista de todos os nomes das propriedades e seus valores atuais.

Exemplos:

h1 = plot (x , y) plota uma linha e retorna o handle h1 da linha plotada.
get (h1) lista todas as propriedades da linha e seus valores.
get (h1 , ‘type’) mostra a classe do objeto (neste caso é uma linha).
get (h1 , ‘linestyle’) retorna o estilo de linha atual da linha.
 Para maiores informações sobre **get**, veja o on-line help.

5.10.5. PROPRIEDADES DOS OBJETOS

Todos os objetos gráficos sobre a tela têm certas propriedades associadas a eles. Por exemplo, as propriedades de uma linha incluem type, parent, visible, color, linestyle, linewidth, xdata, ydata, etc. igualmente, as propriedades de um objeto de texto, tais quais xlabel ou title, incluem type, parent, visible, color, fontname, fontsize, fontweight, string, etc. Uma vez conhecido o handle de um objeto, você pode ver uma lista de suas propriedades e seus valores atuais com o comando **get (handle)**. Por exemplo, veja o algoritmo as propriedades de uma linha e seus valores atuais.

Existem algumas propriedades comuns para todos os objetos gráficos. Estas propriedades são: children, clipping, parent, type, userdata, e visible.

```
>> t=linspace(0, pi, 50);
>> hL=line(t, sin(t) );
>> get (hL);
```

```
Color = [0 0 1]
EraseMode = normal
Linestyle = -
Linewidth = [0.5]
Marker = none
MarkerSize = [6]
MarkeredgeColor = auto
MarkerFaceColor = none
Xdata = [(1 by 50) double array]
Ydata = [(1 by 50) double array]
Zdata = [ ]
```



```
ButtonDownFcn =  
Children = [ ]  
Clipping = on  
CreateFcn =  
DeleteFcn =  
BusyAction = queue  
HandleVisibility = on  
Interruptible = on  
Parent = [3.00037]  
Selected = off  
SelectionHighlight = on  
Tag =  
Type = line  
UserData = [ ]  
Visible = on
```

Estabelecendo os valores das propriedades

Você pode ver a lista de propriedades e seus valores com o comando **set (handle)**. Qualquer propriedade pode ser mudada através do comando:

set (handle , ‘Nome da Propriedade’ , ‘Valor da Propriedade’)

onde ‘*Valor da Propriedade*’ pode ser um caracter string ou um número. Se o ‘*Valor da Propriedade*’ é uma string então ele deve estar entre apóstrofes.

```
>>t=linspace(0, pi, 50);  
>>x=.*sin(t);  
>>hL=line(t, x);  
>>set(hL)
```

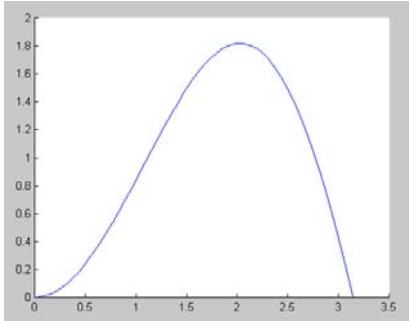
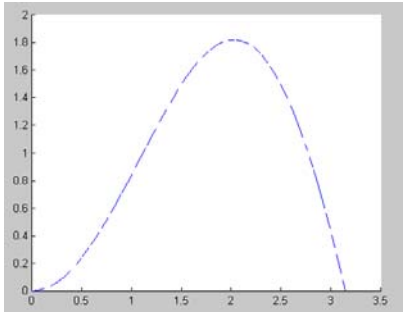
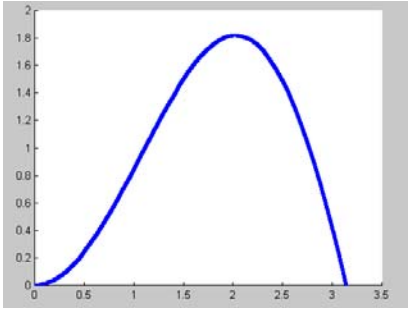
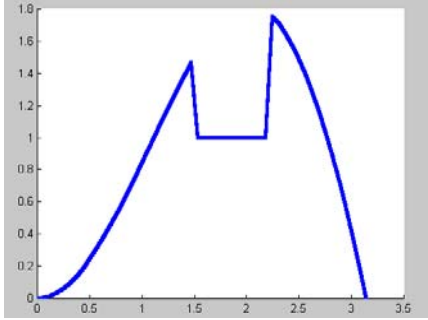
```
Color  
EraseMode: [ {normal} | background | xor | none ]  
LineStyle: [ {-} | -- | : | -. | none ]  
LineWidth  
Marker: [ + | o | * | . | x | square | diamond ..  
MarkerSize  
MarkerEdgeColor: [ none | {auto} ] -ou- um ColorSpec.  
MarkerFaceColor: [ {none} | auto ] -ou- um ColorSpec.  
XData  
YData  
Zdata
```

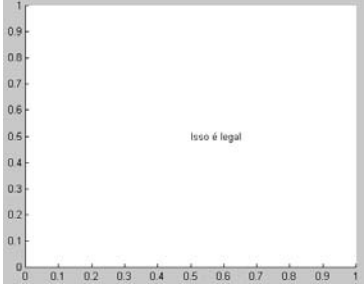

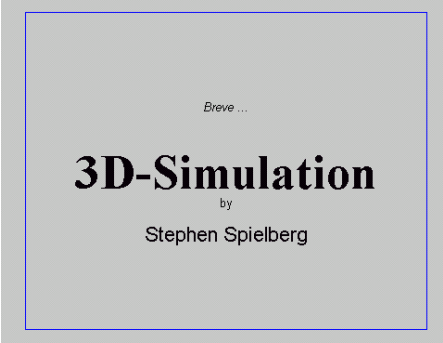

```
ButtonDownFcn  
Children  
Clipping: [ {on} | off]  
CreatFcn  
DeleteFcn  
BusyAction: [ {queue} | cancel]
```

Agora vejamos dois exemplos:

Exemplo-1: Criamos uma linha com um handle explícito e então usamos o comando set para mudar o estilo da linha, sua espessura, e alguns dos dados.

Exemplo-2: Escrevemos algum texto em uma posição específica (na janela figure), criamos seu handle, e então usamos o comando set para mudar o tamanho da fonte, a fonte, e a string do texto.

Exemplos de Script	Saída
<p>Cria uma linha simples e retorna o seu handle hL.</p> <pre>t = linspace(0, pi, 50); x = t.*sin(t); hL = line(t,x);</pre>	
<p>Muda o estilo de linha para tracejado.</p> <pre>set(hL,'linestyle','--');</pre>	
<p>Muda a espessura da linha.</p> <pre>set(hL,'linewidth',3);</pre>	
<p>Muda os valores de algumas coordenadas y.</p> <pre>yvec = get(hL, 'ydata'); yvec(25:35) = ones(size(yvec(25:35))); set(hL,'ydata',yvec)</pre>	

<p>Escreve algum texto no local (0.5, 0.5) e cria um handle para isso.</p> <pre>x = 0.5; y = 0.5; hT = text(x,y,'Isso é legal', ... 'erasemode','xor');</pre>	
<p>Fazer o texto centrado em (0.5, 0.5) e muda a fonte e o tamanho da fonte.</p> <pre>set(hT, 'horizontal','center','fontsize', 36, ... 'fontname','symbol'); set(gca,'visible','off')</pre>	
<p>Agora cria uma apresentação.</p> <pre>clf line([0 0 1 1 0], [0 1 1 0 0]); h1 = text(.5, .7, 'Breve ...', ... 'fontangle','italic', ... 'horizontal','center'); set(gca,'visible','off') h2 = text(.5, .5, '3D-Simulation', ... 'horizontal','center', ... 'fontsize', 40, 'fontname', 'times', ... 'fontweight','bold', 'erase','xor'); h3 = text(.5, .4, 'by', 'horizontal','center'); h4 = text(.5, .3, 'Stephen Spielberg', ... 'fontsize', 16, 'horizontal','center', ... 'erase','xor');</pre>	
<p>Próximo slide por favor ...</p> <pre>set(h1, 'string', ' '); set(h2, 'string', 'O modelo'); set(h3, 'string', ' '); set(h4, 'string', 'Previsões & Idealizações');</pre>	

5.10.6. MODIFICANDO UM GRÁFICO EXISTENTE

Mesmo que você crie um plot sem explicitamente criar objetos handles, o MATLAB cria handles para cada objeto plotado. Se você quer modificar algum objeto você primeiro deve obter seu handle. Aqui é onde você precisa conhecer o parentesco pai-filho entre os vários objetos gráficos.

Modificando plots com PropEdit

Agora, que você tem algum entendimento sobre Handle Graphics, objetos handles, e propriedades de objetos, você pode querer usar o editor gráfico “point-and-click”, PropEdit, existente no MATLAB. Simplesmente digite propedit para ativar o editor. Todos os objetos gráficos ativos na janela figure são mostrados com suas propriedades na janela do PropEdit. Você pode selecionar uma propriedade a partir da lista clicando nela e mudá-la no retângulo estreito no centro. Um objeto gráfico com um sinal de adição (+) à sua esquerda indica que você pode dar um duplo click nele para ver seus objetos “filhos”.

5.10.7. CONTROLE COMPLETO SOBRE O PLANO GRÁFICO (LAYOUT)

Nós encerramos esta seção com um exemplo de colocação arbitrária de eixos e figuras na janela gráfica. Com ferramentas Handle Graphics como estas, você ainda tem controle completo sobre o plano gráfico. Aqui estão dois exemplos:

Exemplo-1: “Placing Insets”

O seguinte script file mostra como criar múltiplos eixos, “size them and place them so that they look like insets. A saída é mostrada na figura a seguir.

```
%-----
%      Exemplo de posicionamento gráfico com Handle Graphics
%-----

clf
t=linspace(0, 2*pi); t(1)=eps;
y=sin(t);
%-----
h1=axes('position', [0.1 0.1 0.8 0.8] );

plot(t,y), xlabel('t'), ylabel('sint')
set(h1, 'Box', 'Off' );
xh1=get(gca, 'ylabel' );

set(xh1, 'fontsize', 16, 'fontweight', 'bold' )

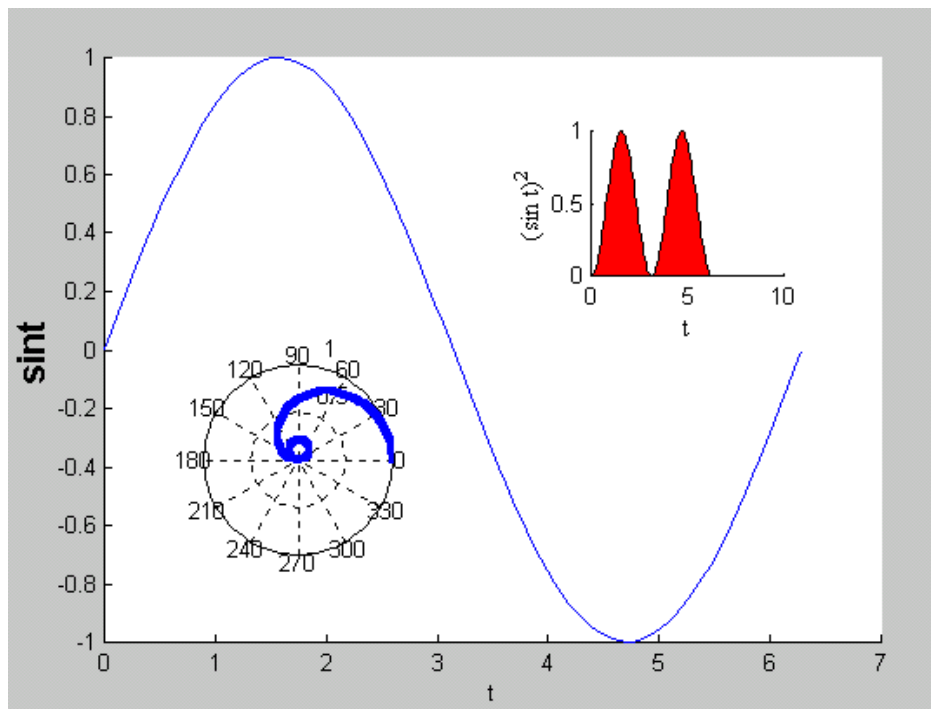
yh1=get(gca, 'ylabel' );
set(yh1, 'fontsize', 16, 'fontweight', 'bold' )
%-----
h2=axes('position', [0.6 0.6 0.2 0.2 ] );
fill(t, y.^2, 'r' )
set(h2, 'Box', 'Off' );
```

```

xlabel('t'), ylabel('(sin t)^2')
set(get(h2, 'xlabel'), 'FontName', 'Times')
set(get(h2, 'ylabel'), 'FontName', 'Times')
%-----
h3=axes('position', [0.15 0.2 0.3 0.3]);
polar(t, y./t);
polarch=get(gca, 'children');
set(polarch(1), 'linewidth', 3)

for i=1: length(polarch)
    if strcmp(get(polarch(i), 'type'))
        delete(polarch(i))
    end
end
end
%-----

```



5.10.8. SALVANDO E IMPRIMINDO GRÁFICOS

O jeito mais simples de “to get a hardcopy” de um gráfico é digitar print na janela de comando depois que o gráfico aparecer na janela figure. O comando print envia o gráfico atual da janela figure para a área de impressão na forma apropriada. Em PC's (rodando o Windows) e MAC's você poderia, alternativamente, ativar a janela figure (trazer à frente dando um click na janela) e então selecionar print a partir do menu.

A figura pode também ser salva em um arquivo específico no formato Post Script ou Encapsulated Post Script (EPS). Estes formatos estão disponíveis para impressoras preto e branco bem como para impressoras coloridas. O Post Script inclui

ambos LEVEL 1 e LEVEL 2 Post Script. O comando que salva gráficos para um arquivo tem a forma:

`Print -ddevicetype -options filename`

onde devicetype “for postscript printers” pode ser um dos seguintes:

<i>devicetype</i>	Descrição	<i>devicetype</i>	Descrição
ps	PostScript preto e branco	eps	EPSF Preto e Branco
psc	PostScript colorido	epsc	EPSF colorido
ps2	PostScript preto e branco nível 2	eps2	EPSF Preto e Branco nível 2
psc2	PostScript colorido nível 2	epsc	EPSF colorido nível 2

Por exemplo, o comando:

`print -deps sineplot`

salva a figura atual no arquivo Encapsulated Post Script sineplot.eps. A extensão ‘.eps’ é automaticamente gerada pelo MATLAB.

Os padrões do argumento opcional –options são append, epsi, Pprinter, e fhandle. Existem várias outras plataformas de opções dependentes. Veja o on-line help em print para maiores informações. Em adição ao dispositivo Post Script, MATLAB suporta um número de outros dispositivos de impressoras nos sistemas UNIX e PC. Existem ações de dispositivos disponíveis para HP Laser Jet, Desk jet, e impressoras Paint jet, DEC LN03, impressoras Epson e outros tipos de impressoras. Veja no on-line help em print para checar a disponibilidade de dispositivos e opções.

“Other than printer devices”, o MATLAB também pode gerar um arquivo gráfico nos seguintes formatos:

-dill	salva o arquivo no formato Adobe Illustrator
-djpeg	salva o arquivo como uma imagem JPEG
-dtiff	salva o arquivo como uma imagem comprimida TIFF
-dmfile	salva o arquivo como um M-file com gráficos handles

Agora, é possível salvar um arquivo gráfico como uma lista de comandos (um M-file), restaurar o gráfico depois, e modificá-lo. Para salvar os gráficos na janela ativa atual, digite:

`Print filename -dmfile`

Depois, para abrir o gráfico novamente, simplesmente execute o arquivo que você criou.

O formato Adobe Illustrator é completamente útil se você quer dar um retoque ou modificar a figura de forma que fica muito difícil fazê-lo no MATLAB. É claro, você deve ter acesso ao Adobe Illustrator para estar apto a abrir e editar o gráfico salvo. O aspecto mais desagradável das versões anteriores às do MATLAB 5 era a falta de facilidade para escrever subscritos e sobrescritos e misturar fontes nas legendas. A partir do MATLAB 5 este problema foi resolvido incorporando um subconjunto de comandos LATEX, os quais podem ser usados em rótulos e textos. Se você não conhece

LATEX, vale a pena consultar o MATLAB helpdesk para ver a lista de comandos e suas saídas.

5.10.9. ANIMAÇÃO

Todos nós sabemos o impacto visual da animação. Se você tem uma grande quantidade de dados representando uma função ou um sistema em várias seqüências de tempo, você pode desejar aproveitar a capacidade do MATLAB “animar seus dados”.

Existem três tipos de ferramentas de animação no MATLAB.

1. **Comet pot:** Esta é a mais simples e mais restrita ferramenta para exibir um gráfico de linha 2-D ou 3-D como uma animação. O comando comet (x , y) plota os dados em vetores x e y com um cometa se movendo. Então, preferivelmente do que ver o conjunto todo do plot aparecer na tela de uma vez, você pode ver o gráfico sendo plotado. Esta ferramenta pode ser utilizada na visualização de trajetórias em um “phase planes”. Como um exemplo veja o demo embutido “on the Lorenz attractor”.
2. **Movies:** Se você tem uma seqüência de gráficos que você gostaria de animar, use a ferramenta embutida movie. A idéia básica é armazenar cada figura como um pedaço de filme, frame, com cada frame armazenado como um vetor coluna de uma grande matriz chamada M, e então executar os frames na tela com o comando movie (M). Um frame é armazenado em um vetor coluna usando-se o comando getframe. Para um eficiente armazenamento você deveria primeiro inicializar a matriz M. O comando embutido moviein é fornecido precisamente para esta inicialização, embora você mesmo possa fazê-la. Um exemplo de script file que faz um movie pode ser visto a seguir:

```
% ---- Esqueleto de um script file para gera e mostra uma animação.----
%
nframes = 36; % número de quadros na animação.
Frames = moviein(nframes); % inicializa a matriz 'Frames'.
for i = 1:nframes,
    . % Você pode ter cálculos aqui para
    . % gerar dados.
    .
    x = ....;
    y = ....;
    plot(x,y) % Você pode usar alguma função plot
    Frames(:,i) = getframe; % Armazena a figura atual em um quadro.
end
movie(Frames,5) % passa a animação armazenada
                  % em quadros 5 vezes.
```

3. **Handle Graphics:** Uma outra forma, e talvez a forma mais versátil de criar animações é usar a ferramenta Handle Graphics. A idéia básica aqui é plotar um objeto na tela, pegar seu handle para mudar as propriedades desejadas do objeto (mais comumente os valores do seu xdata e ydata), e plotar novamente o objeto numa seqüência de vezes selecionada. Existem duas coisas importantes a saber para estar apto a criar animações usando Handle Graphics:

- O comando `drawnow`, o qual transporta os gráficos para a tela de saída sem aguardar o controle retornar para o MATLAB. O on-line help explica como funciona o `drawnow`.
- A propriedade do objeto `'erasemode'` o qual pode ser ajustado para `normal`, `background`, `none`, ou `xor` para controlar o aparecimento do objeto quando a tela gráfica é redesenhada. Por exemplo, se um script file contendo as seguintes linhas de comando é executado:

```
h1=plot(x1,y1,'erasemode','none');
h2=plot(x2,y2,'erasemode','xor');
.
.
newx1=...; newy1=...; newx2=...; newy2=...;
.
.
set(h1,'xdata',newx1,'ydata',newy1);
set(h2,'xdata',newx2,'ydata',newy2);
.
.
```

então o primeiro comando `set` desenha o primeiro objeto com o novo `xdata` e `ydata`, mas o mesmo objeto desenhado antes permanece na tela, enquanto que o segundo comando `set` redesenha o segundo objeto com o novo dado e também apaga o objeto desenhado com os dados anteriores `x2` e `y2`. Assim é possível manter alguns objetos fixados na tela enquanto outros objetos mudam a cada passagem de controle. Agora, vejamos alguns exemplos que ilustram o uso do Handle Graphics em animações.

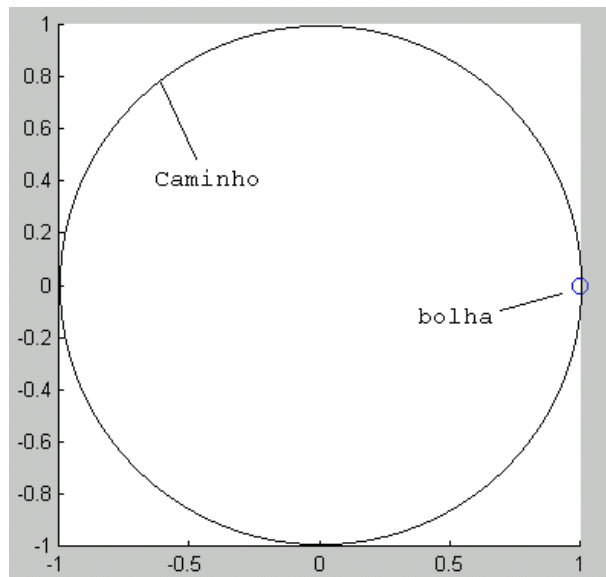
Exemplo-1: Uma bolha se move através de um caminho circular:

A idéia básica é primeiro calcular as diferentes posições da bolha ao longo do caminho circular, desenhar a bolha como um ponto na posição inicial e criar seu handle para determinar a coordenada `x` e `y` da bolha compreendendo todos os valores pertencentes à circunferência. A propriedade `erasemode` da bolha é ajustada para `xor` (Exclusive or) assim a bolha anterior é apagada da tela quando a nova bolha é desenhada. Execute o seguinte script file:

```
% Script file para animar um movimento circular de uma bolha.
% -----
clf
theta = linspace(0, 2*pi, 100);
x = cos(theta); y = sin(theta);
hbolha = line(x(1),y(1),'marker','o', ...
    'markersize',8, ...
    'erase','xor'); % Desenha a bolha nas suas condições iniciais.
axis([-1 1 -1 1]);
axis('square');
for k = 2:length(theta),
    set(hbolha,'xdata',x(k), ...
        'ydata',y(k)); % Desenha a bolha em uma nova posição.
```



```
drawnow
end
```



Exemplo-2: A bolha se move ao redor de um caminho circular e deixa seu rasto:

No exemplo-1 acima, a bolha se move no caminho circular, mas não deixa vestígios de que ela passou por ali. Para ficar evidente, nós podemos fazer a bolha deixar um rasto quando se move. Para tanto, nós basicamente desenhamos a bolha duas vezes, uma vez como uma bolha (com uma marca de maior tamanho) e uma vez como um ponto em cada posição. Mas nós ajustamos a propriedade `erasemode` do ponto para `none`, assim este ponto (a posição anterior da bolha) permanece na tela enquanto a bolha se move e assim cria o rasto da bolha.

```
% Script file para animar um movimento circular de uma bolha.
% Quando ela se move, ela deixa um rasto.
% -----
clf
theta = linspace(0, 2*pi, 100);
x = cos(theta); y = sin(theta);
hbolha = line(x(1),y(1), 'marker', 'o', 'markersize', 8, 'erase', 'xor');
hrasto = line(x(1),y(1), 'marker', '.', 'color', 'r', 'erase', 'none');
axis([-1 1 -1 1]);
axis('square');
for k = 2:length(theta),
    set(hbolha, 'xdata', x(k), 'ydata', y(k));
    set(hrasto, 'xdata', x(k), 'ydata', y(k));
    drawnow
end
```

Exemplo-3: Um pêndulo de barra oscila em 2-D:

Aqui está um slightly mais complicado exemplo. Ele envolve animação do movimento de um pêndulo em barra determinado pela EDO $\theta'' + \sin\theta = 0$. Agora que você está familiarizado com definição de objetos gráficos e usar seus handles para mudar sua posição, a adição da complicação da resolução de uma equação diferencial não seria tão difícil.

```
% ---- script file para animar um pendulo simples -----
```

```

clf                                % limpa a figura
data = [0 0; -1.5 0];              % coordenadas dos pontos finais da barra.
phi = 0;                           % orientação inicial.
R = [cos(phi) -sin(phi); sin(phi) cos(phi)];
% rotação da matriz
data = R*data;
axis([-2 2 -2 2])                 % seleciona os limites dos eixos.
axis('equal')

%---- define oo objetos chamados bar, hinge e path.
Bar = line('xdata',data(1,:), ...
           'ydata',data(2,:), 'linewidth',3, 'erase','xor');
hinge = line('xdata',0, 'ydata',0, 'marker','o', 'markersize',[10]);
path = line('xdata',[], 'ydata',[], 'marker','.', 'erasemode','none');

theta = pi-pi/1000;               % ângulo inicial.
thetadot = 0;                     % velocidade angular inicial.
dt = .1; tfinal=50; t=0;          % tempo de passo, tempo final e inicial.

%-----Método de Euler para integração numérica.
while(t<tfinal);
    t = t + dt;
    theta = theta + thetadot*dt;
    thetadot = thetadot -sin(theta)*dt;
    R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
    datanew = R*data;

    %----- muda os valores das propriedades dos objetos: path e bar.
    set(path, 'xdata',datanew(1,1), 'ydata',datanew(2,1));
    set(bar, 'xdata',datanew(1,:), 'ydata',datanew(2,:));
    drawnow;
end

```

Exemplo-4: O pêndulo de barra oscila, e outros dados são exibidos:

Agora aqui está o desafio. Se você puder entender o seguinte script file, você está em boa forma! Você está pronto para fazer praticamente qualquer animação. O exemplo seguinte divide a tela gráfica em quatro partes, mostra o movimento do pêndulo em uma parte, mostra a posição da extremidade na segunda parte, plota o deslocamento angular θ na terceira parte e a velocidade angular θ' na quarta parte (veja fig.6.18). Existem quatro animações ocorrendo simultaneamente. Tente! Há um erro intencional em uma das quatro animações. Se você iniciasse o pêndulo da posição vertical perpendicular com uma velocidade angular inicial (você precisa mudar a declaração `thetadot = 0` dentro do programa), então você veria o erro. Vá em frente, encontre o erro e elimine-o.

```

% ----- script file para aminorar um pêndulo e os dados -----

% Conseguir os dados básicos para a animação.
% Perguntar para o usuário sobre as posições iniciais.
disp('Por favor especifique o ângulo inicial a partir da');
disp('posição vertical superior direita. ');
disp(' ')
offset = input('Entre com o ângulo inicial agora: ');
% Pede para o usuário o tempo de simulação.

```

```

tfinal = input('Por favor entre com a duração da simulação: ');
disp('Eu estou trabalhando ...')

theta = pi-offset; % ângulo inicial.
thetadot = 0; % Velocidade angular inicial.
dt = .2; t = 0; tf = tfinal; % Tempo de passo, tempo inicial e final.

disp('Veja os gráficos na tela')
clf % Limpar a figura
h1 = axes('position', [.6 .1 .4 .3]);
axis([0 tf -4 4]); % Seleciona os limites do eixo.
xlabel('tempo'); ylabel('Deslocamento');
Displ = line('xdata', [], 'ydata', [], 'marker', '.', 'erasemode', 'none');

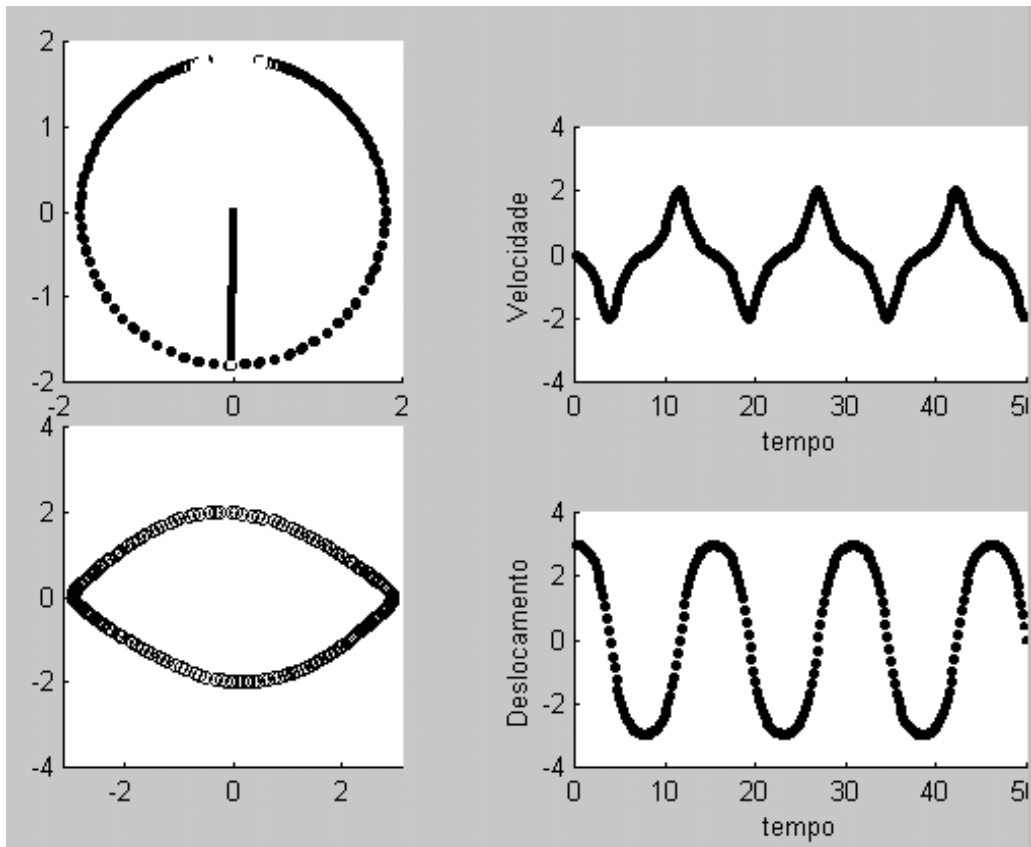
h2 = axes('position', [.6 .55 .4 .3]);
axis([0 tf -4 4]); % Seleciona os limites do eixo.
xlabel('tempo'); ylabel('Velocidade');
Vel = line('xdata', [], 'ydata', [], 'marker', '.', 'erasemode', 'none');

h3 = axes('position', [.1 .1 .4 .4]);
axis([-pi pi -4 4]); % Seleciona os limites do eixo.
axis('square');
phase = line('xdata', [], 'ydata', [], ...
            'marker', 'o', 'markersize', 5, 'erasemode', 'none');

h4 = axes('position', [.1 .55 .4 .4]);
axis([-2 2 -2 2]); % Seleciona os limites do eixo.
axis('square');

data = [0 0; -1.8 0]; % Coordenadas de pontos finais da barra.
phi = 0; % Orientação inicial.
R = [cos(phi) -sin(phi); +sin(phi) cos(phi)]; % Rotaciona a matriz.
data = R*data;
% ---- Define os objetos chamados bar, hinge e path
bar = line('xdata', data(1,:), ...
          'ydata', data(2,:), 'linewidth', 3, 'erase', 'xor');
hinge = line('xdata', [], 'ydata', [], 'marker', '.', 'erasemode', 'none');
path = line('xdata', [], 'ydata', [], 'marker', '.', 'erasemode', 'none');
% ---- Método de Euler para a integração numérica. ----
while (t < tfinal);
    t = t + dt;
    theta = theta + thetadot*dt;
    thetadot = thetadot - sin(theta)*dt;
    R = [cos(theta) (-sin(theta)); sin(theta) cos(theta)];
    datanew = R*data;
    % ---- Muda os valores das propriedades dos objetos: path e bar.
    set(path, 'xdata', datanew(1,1), 'ydata', datanew(2,1));
    set(bar, 'xdata', datanew(1,:), 'ydata', datanew(2,:));
    set(phase, 'xdata', theta, 'ydata', thetadot);
    set(Displ, 'xdata', t, 'ydata', theta);
    set(Vel, 'xdata', t, 'ydata', thetadot);
    drawnow;
end

```



6. PROGRAMANDO NO MATLAB: FUNÇÕES E PROCEDIMENTOS

Para o estudo deste capítulo, o essencial é saber a diferença entre uma função e um procedimento. Ambos são subrotinas, que são uma seqüência de comandos que são dados ao computador pelo programador, mas a função possui entradas e saídas e o procedimento não.

Algumas vezes chamaremos os procedimentos de *script files*.

6.1. PROCEDIMENTOS

Sempre que você quiser escrever um procedimento basta escrever os mesmos comandos que você digitaria na área de trabalho (parte interativa do MATLAB) dentro de um arquivo *.m.

É interessante fazer um procedimento sempre que você tiver um grupo de comandos muito grande e é preciso executá-lo várias vezes. Armazenando este grupo de comandos num arquivo script, basta escrever o nome do arquivo na área de trabalho e o mesmo grupo de comandos será executado.

Cuidado:

- Nunca nomeie um arquivo script com o mesmo nome de uma variável que ele computa. Quando o MATLAB procura por um nome, ele busca primeiro na lista de variáveis existentes na área de trabalho, ou seja, se existir uma variável com o mesmo nome de um arquivo script ele jamais será aberto.
- O nome do arquivo deve começar com uma letra e os restantes devem ser dígitos, exceto o ponto (.). Você pode escrever nomes longos, mas o MATLAB irá pegar somente os primeiros 19 caracteres.
- Fique atento com as variáveis que você trabalha nos procedimentos, porque todas as variáveis que você declarar ficarão na área de trabalho a menos que você as limpe. Evite nomes que coincidam com as funções embutidas. É bom checar se os nomes propostos já existem. Você pode fazer isso com o comando `exist('nome')`, que retornará zero se não houver nada com este *nome*.

6.2. FUNÇÕES

Para se fazer uma função, deve-se fazer quase a mesma coisa do que com o procedimento. A única diferença é que a função necessita de uma linha de definição com a seguinte sintaxe:

```
function [variáveis_de_saída] = nome_da_função (variáveis_de_entrada);
```

Exemplos:

Linha de Definição da função	Nome da função
<code>function [rho, H, F] = motion(x, y, t);</code>	<code>motion.m</code>
<code>function [theta] = anguloTH(x, y, t);</code>	<code>anguloTH.m</code>
<code>function theta = THETA(x, y, t);</code>	<code>THETA.m</code>
<code>function [] = circulo(r);</code>	<code>circulo.m</code>
<code>function circulo(r);</code>	<code>circulo.m</code>

Cuidado:

A palavra `function` deve ser escrita em letras minúsculas. Um erro muito comum é iniciar a declaração com `Function`.

6.2.1. EXECUTANDO UMA FUNÇÃO

Existem dois modos em que uma função pode ser executada, ela pode ser embutida ou escrita pelo usuário:

1. Com saída explícita: Esta é a sintaxe integral de chamar uma função. Ambas as listas de saída e entrada são especificadas na declaração. Por exemplo, se a linha de definição de uma função é:

```
function [rho,H,F] = motion(x,y,t);
```

então todos os seguintes comandos representam a sua execução;

- `[r,angmom,force] = motion(xt,yt,time);` As variáveis de entrada `xt`, `yt` e `time` devem ser definidas antes de executar este comando.
- `[r,h,f] = motion(rx,ry,[0:100]);` As variáveis de entrada `rx` e `ry` devem ser definidas anteriormente, a terceira entrada, a variável `t` é especificada na execução.
- `[r,h,f] = motion(2,3.5,0.001);` Todos os valores de entrada são especificados na execução.
- `[radius,h] = motion(rx,ry);` Chamada com lista parcial de entrada e saída. Na terceira variável de entrada deve ser nomeado um valor default (padrão) dentro da função se ela for requerida nos cálculos. A saída corresponde aos primeiros dois elementos da lista de saída da função `motion`.

2. Sem saída: A lista de saída pode ser omitida completamente se as quantidades computadas não são de algum interesse. Este pode ser o caso quando a função exibe o resultado desejado graficamente. Para executar a função deste modo, apenas escreva o nome da função com a lista de entrada. Por exemplo, `motion(xt,yt,time);` executará a função `motion` sem gerar alguma saída explícita, contando que `xt`, `yt`, e `time` são definidos. Se o ponto e vírgula no fim da declaração é omitido, a primeira variável de saída na lista de saída das funções é mostrada na variável default (padrão) chamada `ans`.

A função pode ser escrita para aceitar uma lista parcial de entradas se alguns valores default (padrão) das outras entradas não especificadas estão definidas dentro da função. Este tipo de lista de entrada pode ser feita com a função embutida `nargin`, o qual se estende por um número de argumentos de entrada. Da mesma forma, a lista de saída pode ser manipulada com a função embutida `nargout`. Veja a ajuda on-line em `nargin` e `nargout`. Para um exemplo de como usa-los, olhe na função `fplot` digitando `type fplot.m`.

Exemplo de um Arquivo Função:

Vamos escrever uma função para resolver o mesmo sistema de equações lineares que nós resolvemos usando o procedimento. Desta vez, nós faremos `r` uma

entrada para a função e det_A e x será a saída. Vamos chamar esta função de solvexf. Como uma regra, ela deve ser salva em um arquivo chamado sovexf.m.

```
function [det_A,x] = solvexf(r);
% SOLVEXF resolve uma equação de matriz 3x3 com parâmetro r
% Esta é a função 'solvexf.m'
% Para chamar esta função, digite:
% [det_A,x] = solvexf(r) ;
% r é a entrada e det_A e x são as saídas.
%-----

A = [5 2*r r; 3 6 2*r-1; 2 r-1 3*r] ;      % cria a matrix A
b = [2;3;5] ;                             % cria o vetor b
det_A = det(A) ;                          % resolve o determinante
x = A\b ;                                  % encontra x.
```

Agora r, x e det_A são todas variáveis locais. Portanto, alguns outros nomes de variáveis podem ser usados no lugar delas no momento da execução da função (repare que no exemplo abaixo usamos detA ao invés de det_A e y no lugar de x). Vamos executar esta função no MATLAB.

```
>> [detA, y] = solvexf(1) ;                % Pegua r=1 e executa solvexf.m

>> detA                                    % Mostra o valor de detA

ans =
    64

>> y                                       % Mostra o valor de y

ans =
   -0.0312
    0.2344
    1.6875
```

Os valores de detA e y serão automaticamente mostrados se o ponto e vírgula no final da função de comando executada for omitido.

Após a execução de uma função, as únicas variáveis deixadas na área de trabalho pela função serão as variáveis de saída da lista. Isto nos dá um maior controle da entrada e saída do que nós podemos conseguir com arquivos script. Nós podemos também incluir checagem de erros e mensagem dentro das funções. Por exemplo, nós poderíamos modificar a função acima para checar se a matriz A é vazia ou não e mostrar uma mensagem apropriada antes de resolver o sistema, mudando a última linha para:

```
if isempty(A)                             % se a matrix A for vazia
    disp('A matriz A é vazia');
else                                       % se A não for vazia
```

```
x = A\b ;           % encontra x
end                 % fim da declaração if
```

6.2.2. MAIS SOBRE FUNÇÕES

Até agora a diferença entre scripts e funções devia estar clara para você. As variáveis dentro de uma função são locais e são apagadas após a execução da função. Mas as variáveis dentro de um arquivo script são deixadas na área de trabalho do MATLAB após a execução do script. Funções podem ter argumentos, já os arquivos script não. E quanto as funções dentro de uma outra função? Elas são locais? Como elas são executadas? Uma função pode ser passada como uma variável de entrada para uma outra função? Agora nós vamos esclarecer essas questões.

Executando uma função dentro de uma outra função

Na função `solvexf` acima, nós usamos uma função embutida, `det`, para calcular o determinante de A . Nós usamos a função justamente do modo que nós a usaríamos no prompt do MATLAB ou no arquivo script. Isto é verdade para todas as funções embutidas ou escritas pelo usuário. A história é diferente somente quando você quer que o nome da função seja dinâmico, isto é, se a função a ser executada dentro puder ser diferente com diferentes execuções da função de chamada. Em tais casos, o atual nome da função é passado para a função denominada através de uma lista de entrada e de um nome fictício que é usado dentro da função de chamada. O mecanismo de passar o nome da função e avaliá-la dentro de uma função de chamada é bem diferente daquele para uma variável. Nós explicaremos isto logo abaixo.

Uma função na lista de entrada

Quando uma função precisa ser passada na lista de entrada de uma outra função, o nome da função a ser passada precisa aparecer como um caracter string, na lista de entrada. Por exemplo, a função embutida `fzero` encontra um zero de uma função fornecida pelo usuário de uma variável simples. A chamada sintaxe da função é `fzero(f,x)` onde f é o nome da função e x é uma incógnita inicial. Se nós escrevermos a função `fr3` (salvada como `fr3.m`) para resolver-la, dizemos, $f(r) = r^3 - 32 \cdot 5r^2 + (r - 22)r + 100$, nós podemos denominar `fzero` com a declaração `fzero('fr3', 5)`. Note as cotas singulares ao redor do nome da função `fr3` de entrada.

Avaliando uma função com `Feval`

A função `feval` avalia uma função cujo o nome é especificado como uma string na dada lista de variáveis de entrada. Por exemplo `[y, z] = feval('Hfunction', x, t)`; avalia a função `Hfunction` nas variáveis de entrada x e t e retorna a saída em y e z . Isto é equivalente a escrever `[y, z] = Hfunction(x,t)`. Então porque você alguma vez avaliaria uma função usando `feval` quando você pode avaliar a função diretamente? O uso mais comum é quando você quer avaliar funções com diferentes nomes mas na mesma lista de entrada. Considere a seguinte situação. Você quer avaliar alguma dada função de x e y na origem $x = 0, y = 0$. Você pode escrever um arquivo script com um comando do seguinte modo:


```
value = feval ('funxy', 0, 0);
```

Este comando é o mesmo que escrever, $\text{value} = \text{funxy}(0, 0)$. Agora suponha que Harry tem uma função $z(x,y) = \sin xy + xy^2$, programada como:

```
function z = harrysf(x,y)
% função para avaliar z(x,y)
z = sin(x*y) + x*y^2;
```

e Kelly tem uma função $h(x,y) = 20xy - 3y^3 - 2x^3 + 10$, programada como:

```
function z = kellysf(x,y)
% função para avaliar h(x,y)
h = 20*x*y - 3*y^3 - 2*x^3 + 10;
```

Ambas as funções podem ser avaliadas com seu arquivo script mudando o nome `funxy` para `harrysf` e `kellysf`, respectivamente. A questão aqui é que o comando no seu arquivo script leva nomes de arquivos dinâmicos.

O uso de `feval` torna-se essencial quando a função é passada como uma variável de entrada para uma outra função. Em tais casos, a função passada como a variável de entrada deve ser avaliada usando `feval` dentro da função de chamada. Por exemplo, os solucionadores de ODE (ordinary differential equation) `ode23` e `ode45` levam funções definidas pelo usuário como entradas nas quais o usuário especifica a equação diferencial. Dentro de `ode23` e `ode45`, a função de definida pelo usuário é avaliada no tempo corrente t e no valor corrente de x para computar a derivada x' usando `feval`, `ode23` e `ode45` são arquivos M, os quais você pode copiar e editar. Faça o printout de um deles e veja como ele usa `feval`.

Escrevendo boas funções

Escrever boas funções no MATLAB é mais fácil do que escrever funções na maioria dos programas de linguagem padrão, ou, para devido interesse, na maioria dos pacotes de software que suportam seus próprios programas de desenvolvimento. Entretanto, escrever eficientes e elegantes funções é uma arte que se consegue apenas através da experiência. Para iniciantes, manter os seguintes pontos em mente ajuda:

- **Algoritmo:** Antes de você começar a escrever uma função, escreva um algoritmo. Ele é essencialmente uma função inteira no português simples. Pense sobre a estrutura lógica e a sequência de computações, defina a entrada e a saída de variáveis, escreva a função em simples palavras e então comece a tradução na linguagem MATLAB.
- **Clareza:** Selecione um nome sensato para a função e as variáveis dentro dela. Escreva comentários no corpo da função. Esboce e escreva um comentário útil para uma ajuda on-line (as linhas de comentário devem ser escritas no começo de uma função). Verifique se você incluiu a sintaxe de como usar a função.

- **Modularidade:** Mantenha sua função modular, isto é, divida grandes cálculos em pedaços menores (sub-rotinas) e escreva funções separadas para eles. Mantenha suas funções pequenas em comprimento.
- **Robustez:** Providencie checagem para erros e saída com úteis mensagens de erros.
- **Expansibilidade:** O programador deve prever uma expansão do programa. Por exemplo, se você está escrevendo uma função com variáveis escalares, mas você acha que pode usar variáveis vetoriais depois, escreva a função com isto em mente. Evite números de difícil código.

6.2.3. SUB-FUNÇÕES

O MATLAB 5 admite que várias funções sejam escritas em um único arquivo. Enquanto esta facilidade é útil de um ponto de vista de organização de arquivo, vem com severas limitações. Todas as funções escritas após a primeira função no arquivo são tratadas como sub-funções e não estão disponíveis para o mundo lá fora. Isto significa que enquanto a primeira função pode acessar todas suas sub-funções e as sub-funções escritas no mesmo arquivo podem também acessar uma a outra, funções de fora do arquivo não podem acessar essas sub-funções.

6.2.4. FUNÇÕES COMPILADAS (ANALISADAS): P-CODE

Quando uma função é executada no MATLAB, cada comando da função é primeiramente interpretado pelo MATLAB e então traduzido para um nível de linguagem mais baixo. Este processo é chamado ‘parsing’. Não é exatamente como compilar uma função em C ou Fortran mas, superficialmente, é um processo similar. O MATLAB 5 permite que você salve uma função analisada para uso posterior. Para analisar uma função dita *projectile*, que existe em um arquivo M chamado *projectile.m*, digite o comando.

pcode projectile

Este comando gera uma função analisada que é salva em um arquivo *projectile.p*. Na próxima vez em que você chamar a função *projectile*, o MATLAB diretamente executa a função pré-analisada.

Para todas as funções de tamanho moderado, salva-las como funções analisadas não gastam tanto tempo durante a execução como você poderia pensar. O analisador do MATLAB é rápido o suficiente para reduzir o tempo gasto na análise a uma fração de tempo desprezível. O melhor uso de p-codes, talvez, é proteger suas propriedades corretas quando você envia suas funções para outras pessoas. Enviando p-codes, os receptores podem executá-los mas não podem modificá-los.

6.2.5. O PROFILER

Para avaliar a performance das funções, o MATLAB 5 possui uma ferramenta chamada *profiler*. O *profiler* mantém o rastro do tempo gasto em cada linha da função (em unidades de 0.01 segundos) assim que a função é executada. O *profiler* mostra

quanto tempo quanto tempo foi gasto em cada linha. A saída do profiler podem também ser vista graficamente.

O profiler é surpreendentemente fácil de usar. Vejamos um falso exemplo. Para desenhar uma função chamada projectile, digite os seguintes comandos:

profile projectile	% ativa o profiler no projectile.m
[a,b] = projectile (v, theta)	% executa a função projectile
profile report	% mostra as informações do profiler
profile plot	% mostra a informação graficamente
profile done	% obrigado, tchau

Observação: O profiler pode ser usado somente em funções (não scripts) e nessas funções que existem como arquivos M. Então, você não pode esboçar funções embutidas que não são providas como arquivos M.

6.3. CARACTERÍSTICAS ESPECÍFICAS DA LINGUAGEM

Foram discutidas várias características da linguagem do MATLAB através de muitos exemplos nas seções anteriores. Você é advertido a prestar uma atenção especial ao uso formal de sinais de pontuação e diferentes delimitadores, operadores, especialmente a ordem dos operadores (um ponto (.) precedendo os operadores aritméticos, e os operadores relacionados para controle de fluxo, o MATLAB é provido de laços for e **while**, e **if-elseif-else** e um **switch-case-otherwise**. Todo o controle de fluxo das declarações devem terminar com a correspondente declaração **end**. Nós agora discutiremos o controle de fluxo e algumas outras características da linguagem. Veja a ajuda on-line para mais detalhes.

6.3.1. O USO DE COMENTÁRIO PARA CRIAR AJUDA ON-LINE

Como já foi mostrado na discussão sobre funções, as linhas de comentários no princípio (antes de qualquer declaração executável) de um script ou uma função são usadas pelo MATLAB como a ajuda on-line sobre o arquivo. Este automaticamente cria a ajuda on-line para funções escrita pelo usuário. Isto é uma boa idéia para reproduzir a linha de definição de função sem a palavra function entre essas primeiras poucas linhas de comentário de forma que a sintaxe de execução da função é exibida pela ajuda on-line. O comando lookfor procura o argumento string na primeira linha comentada dos arquivos M. Portanto, de acordo com a convenção um pouco confusa das funções embutidas do MATLAB, você deveria escrever o nome do script ou função no caso acima com letras seguidas por uma curta descrição com palavras chaves, como a primeira linha comentada. Veja o exemplo abaixo.

```
% SOLVEX resolve uma matriz de equação 3x3 com parâmetro r.
```

6.3.2. CONTINUAÇÃO

Três pontos consecutivos (...) ao término de uma linha denota continuação. Logo, se você digita um comando que não cabe em uma única linha, você pode dividi-lo em duas ou mais linhas usando os três pontos no final de cada linha exceto na última linha. Exemplos:

```
A = [1 3 3 3; 5 10 -2 -20; 3 5 ...
      10 2; 1 0 0 9];
x = sin(linspace(1,6*pi,100)) .* cos(linspace(1,6*pi,100)) + ...
    0.5*ones(1,100);
plot(comprimento do tubo, pressão do fluido, ': ', comprimento do tubo, ...
      pressão teórica, '-')
```

Você não pode, entretanto, usar o recurso continuação dentro de um caracter string. Por exemplo, digitando

```
logo = ' Eu não sou somente o Presidente da empresa, ...
        mas também um cliente';
```

produz um erro. Para criar strings longas, quebre a string em pequenos segmentos de string e use concatenação.

6.3.3. VARIÁVEIS GLOBAIS

É possível declarar um jogo de variáveis para ser globalmente acessível para todas ou algumas funções sem declarar as variáveis na lista de entrada das funções. Isto é feito com o comando **global**. Por exemplo, a declaração **global x y z** declara as variáveis x, y e z para serem globais. Esta declaração deve estar antes de qualquer declaração executável nas funções e scripts que precisam acessar valores de variáveis globais. Tenha cuidado com os nomes das variáveis globais. É aconselhável utilizar longos nomes para tais variáveis com o fim de evitar semelhança com outras variáveis locais.

Exemplo: Considere a resolução da seguinte EDO de 1ª ordem.

$$\dot{x} = kx + c \sin t, \quad x(0) = 1.0$$

onde você está interessado em soluções para vários valores de k e c. Seu arquivo script pode parecer:

```
% scriptfile para resolver a EDO de 1ª-ordem .
ts = [0 20]; % especifica o intervalo de tempo = [t_0 t_final]
x0 = 1.0; % especifica a condição inicial
[t, x] = ode23 ('ode1', ts, x0); % executa ode23 para resolver EDO.
```

e a função 'ode1' pode parecer:

```
função xdot = ode1(t,x);
% EDO1: função para computar a derivada xdot
```

```
% dados t e x.
% sintaxe chamada: xdot = ode1 (t,x);
% -----
xdot = k*x + c*sin(t);
```

Isto, entretanto, não funcionará. De fato para ode1 computar xdot. os valores de k e c devem estar prescritos. Esses valores podiam ser prescritos dentro da função ode1 mas você teria de editar essa função a cada vez que você mudar os valores de k e c. Uma alternativa é prescrever os valores na arquivo script e fazê-los disponíveis para a função ode1 através de uma declaração global.

```
% scriptfile para resolver EDO de 1ª ordem.
global k_value c_value                                % declara variáveis globais.
k_value c_value = 2 ;                                % especifica o valor das variáveis globais
t0 = 0 ; tf = 20 ;                                    % especifica o tempo inicial e final
x0 = 1.0 ;                                             % especifica a condição inicial
[t, x] = ode23 ('ode1', t0,x0) ;                     % executa ode23 para resolver EDO.
```

Agora você pode modificar a função ode1 e desse modo pode acessar as variáveis globais.

```
function xdot = ode1(t,x) ;
% EDO1: função para computar a derivada xdot
% dados t e x.
% sintaxe chamada: xdot = ode1 (t,x);
% -----
global k_value c_value
xdot = k_value + c_value*sin(t)
```

Dessa forma, se os valores de k_value e c_value for mudado no arquivo script, os novos valores tornam-se disponíveis para ode1 também. Observe que a declaração global é somente no script e na função do usuário ode1, portanto k_value e c_value estarão disponíveis somente para esses arquivos.

6.3.4. LAÇOS, RAMIFICAÇÕES E CONTROLE DE FLUXO

O MATLAB tem sua própria sintaxe para declarações de controle de fluxo como o laço for, laço while e claro, if-elseif-else. Além do mais, ele prove três comandos – break, error, e return para controlar a execução de scripts e funções. Abaixo descreveremos cada uma dessas funções.

Laço For:

Um laço é usado para repetir uma declaração ou um grupo de declarações por um número fixo de vezes. Temos aqui dois exemplos:

Exemplo – 1: for m = 1:100

```
        Num = 1/(m+1)
    end
```

O contador no laço for pode também ser especificado: for = m:k:n para avançar o contador i em relação a k a cada vez (no exemplo abaixo n vai de 100 a 0 da seguinte forma 100, 98, 96, ..., etc.). Você pode ter laços for aninhados, isto é, colocar laços for dentro de laços for. Todo for, entretanto, deve ser terminado com um end.

Exemplo – 2: for n = 100:-2:0, k = 1/(exp(m)), end

Laço While:

Um laço while é usado para executar uma declaração ou um grupo de declarações para um número indefinido de vezes até que a condição especificada por while não seja satisfeita. Por exemplo:

```
% Vamos encontrar todas as potências de 2 abaixo de 10000
v = 1 ; i = 1 ;
while num < 10000
    num = 2^i ;
    v = [v ; num] ;
    i = i + 1 ;
end
```

Novamente, um laço while deve ter um end fechando o laço.

Declarações If-elseif-else :

Esta construção provê uma ramificação lógica para computação. Por exemplo:

```
if i > 5 ;
    k = i ;
elseif (i > 1) & (j == 20)
    k = 5*i + j ;
else
    k = 1;
end
```

Você pode colocar a declaração if, contanto que você feche o laço com a declaração end. Você pode colocar todos os três tipos de laços, em uma combinação.

Switch-case-otherwise:

Essa construção (é uma característica do MATLAB 5.x) que prove uma outra ramificação lógica para computação. Uma variável é usada como uma chave e os valores da variável fazem com que os diferentes casos sejam executados. A sintaxe geral é:

```
switch variável
```

```
case valor1
    1º bloco de comandos
case valor2
    2º bloco de comandos
otherwise
    último bloco de comandos
end
```

Neste exemplo, o primeiro bloco de comandos é executado se a variável for igual ao valor 1; o segundo bloco de comandos é executado se a variável for igual ao valor 2. Caso a variável não combine com nenhum dos casos o último bloco de comandos é executado.

O switch pode ser usado como uma variável numérica ou uma variável string. Vamos ver um exemplo mais concreto usando uma variável string com o comando switch.

```
switch cor
case 'vermelho'
    c = [1 0 0];
case 'verde'
    c = [0 1 0];
case 'azul'
    c = [0 0 1];
otherwise
    error('escolha de cor inválida')
end
```

Break

O comando break dentro de um laço for ou while termina a execução do laço, até mesmo se a condição do laço for verdadeira.

Exemplos:

```
1.
for i = 1: length(v)
    if u(i) < 0                % verifica para u negativo
        break                % termina a execução do laço
    end
    a = v(i) + .....        % faz alguma coisa
end

2.
while 1
    n = input('Entre com o nº máximo de interações ');
    if n <= 0
        break                % termina execução do laço
    end
    for i=1:n
```

```
..... % faz alguma coisa
end
end
```

Caso haja um laço dentro de outro, o comando **break** finaliza somente o laço mais interno.

Error

O comando **error** (*'mensagem'*) dentro de uma função ou um script aborta a execução, mostra uma mensagem de erro e retorna o controle para o teclado.

Exemplo:

```
function c = crossprod (a,b) ;
% crossprod (a,b) calcula o produto vetorial axb.
if nargin~=2 % se não há 2 argumento de entrada
    error ('Desculpe, precisa da entrada de 2 vetores')
end
if length(a)==2 % inicia os cálculos
    .....
end
```

Return

O comando **return** simplesmente retorna o controle da função chamada.

Exemplo:

```
Function animatebar(t0, tf, x0);
% animatebar anima um pêndulo
:
disp('Você deseja ver o retrato da fase?')
ans = input ('Digite 1 para SIM, e 0 para NÃO'); % olhe abaixo para descrição
if ans= 0 % se o input é 0
    return % sair da função
else
    plot(x,...) % mostra a plotagem da fase
end
```

6.3.5. ENTRADA INTERATIVA

Os comandos **input**, **keyboard**, **menu** e **pause** podem ser usados dentro de um script ou uma função para entradas interativas feitas pelo usuário. Suas descrições estão logo abaixo:

Input

O comando **input**(*'Texto'*), usado no exemplo anterior, mostra na tela o texto que está no lugar de *'Texto'* e espera o usuário a dar entrada pelo teclado.

Exemplo:

- `n = input('Maior tamanho da matriz quadrada');` alerta o usuário a entrar com o tamanho da maior matriz quadrada e salva a entrada em `n`.

- `more = input('Mais simulações? (S/N)', 's');` alerta o usuário a digitar S para SIM e N para NÃO e armazena a entrada como uma string em `'more'`. Note que o segundo comando, `'s'`, do comando, direciona o MATLAB a salvar a entrada como uma string.

Este comando pode ser usado para escrever programas interativos amigáveis no MATLAB.

Keyboard

O comando **keyboard** dentro de um script ou uma função retorna o controle para o teclado no exato ponto onde o comando ocorre. A execução da função ou o script não estão terminados. O prompt da janela de comandos `'>>'`, muda para `'k>>'` para mostrar o status especial. Neste ponto, você pode checar variáveis já computadas, mudar seus valores e editar qualquer comando MATLAB válido. O controle é retornado à função digitando a palavra **return** no prompt especial `'k>>'` e pressionando enter.

Este comando é útil para funções que eliminam erros (debugging functions). Às vezes, em grandes algoritmos, você pode querer checar alguns resultados intermediários, plotá-los e ver se a colocação dos dados no algoritmo está seguindo uma ordem logicamente correta, e então deixar que a execução continue.

Exemplo:

```
% EXKEYBRD: um arquivo de script como exemplo de comando no teclado
% -----

A = ones(10)                                % monta uma matriz 10x10 de 1's

for i=1:10
    disp(i)                                % mostra o valor de I
    A(:,i) = i*A(:,i);                    % remaneja a i-ésima coluna de A
    if i==5                                % quando i = 5
        keyboard                          % retorna o controle do teclado
    end
end
```

Durante a execução do script acima (exekeybrd.m) o controle é retornado ao teclado quando o valor do contador `i` alcança 5. A execução do procedimento recomeça após o controle ser retornado ao arquivo script digitando **return** no prompt especial `'k>>'`.

Menu

O comando `menu('Nome do menu', 'opção 1', 'opção 2'...)` cria um menu na tela com o `'Nome do menu'` e lista as opções no menu. O usuário pode selecionar qualquer opção pelo mouse ou pelo teclado, dependendo do computador. A implementação desse comando em MACs e PCs cria menus de janelas interessantes com botões.

Exemplo:

```
% Plotando um círculo
r = input('Entre com o raio desejado: ');
theta = linspace(0, 2*pi, 100);
r = r*ones(size(theta));           %faz r do mesmo tamanho de theta.
coord = menu('Plotagem de Círculo', 'Cartesian', 'Polar');
if coord == 1                       % Se a primeira opção do menu for selecionada
    plot(r.*cos(theta), r.*sin(theta))
    axis('square')
else                                % Se a segunda opção do menu for selecionada
    polar(theta, r);
end
```

No exemplo acima, o comando **menu** cria um menu com o nome Plotagem de Círculo e duas opções: Cartesiano e Polar. As opções são internamente numeradas. Quando o usuário seleciona uma das opções, o número correspondente é passado para a variável coord. O laço if-else no script mostra (através do comando menu) o que fazer com cada opção.

Pause

Este comando temporariamente pára o processo em decorrência. Pode ser usado com ou sem um argumento opcional:

- **pause** pára o processo em andamento e espera o usuário dar o sinal de “vá em frente”. Pressionando qualquer tecla reinicia-se o processo.
- **pause(n)** Pára o processo em andamento, por n segundos, e então recomeça o processo.

Exemplo: for i = 1:n, plot(X(:,i),Y(:,i)), pause(5), end pára por 5 segundos antes de plotar o próximo gráfico.

6.3.6. RECURSION

A linguagem do MATLAB suporta *recursion*, isto é, uma função pode chamar a si própria durante sua execução. Assim, algoritmos recursivos podem ser diretamente implementados no MATLAB.

6.3.7. ENTRADA/SAÍDA

O MATLAB suporta várias funções I/O de arquivos de linguagem C padrão para leitura e escrita de binários formatados e arquivos de texto. As funções suportadas incluem:

fopen	Abre um arquivo existente ou cria um novo
fclose	Fecha um arquivo aberto
fread	Lê dados binários de um arquivo
fwrite	Escreve dados binários para um arquivo
fscanf	Lê strings em formato especificado
fprintf	Escreve dados em um string formatado
fgets	Lê uma linha de um arquivo excluindo NEW-LINE CHARACTER
fgetl	Lê uma linha de um arquivo incluindo NEW-LINE CHARACTER
frewind	“rebobina” um arquivo
fseek	Escolhe o indicador de posição do arquivo
ftell	Dá a indicação correta da posição do arquivo
ferror	Investiga o status de erro do arquivo I/O

Já é satisfatório usar os seis primeiros comandos listados acima. Para a maioria dos propósitos **fopen**, **fprintf** e **fclose** são suficientes.

Este é um exemplo simples que usa **fopen**, **fprint** e **fclose** para criar e escrever dados formatados para um arquivo:

```
% TEMTABLE – gera e escreve a tabela de temperatura
% Arquivo script para gerar uma tabela de temperatura Fahrenheit-Celsius.
% A tabela está escrita em um arquivo chamado ‘Temperature.table’.
%-----
F = -40:5:100;
C = (F - 32)*5/9;
t = [F; C];
fid = fopen('Temperature.table', 'w');
fprintf(fid, 'Temperature.table\n ');
fprintf(fid, '~~~~~\n');
fprintf(fid, 'Fahrenheit Celsius\n');
fprintf(fid, '%4i    %8.2f\n', t);
fclose(fid);
```

No arquivo de script acima, o primeiro comando I/O, **fopen**, abre um arquivo `temperature.table` no modo de escrita (write) – especificado por um ‘w’ no comando – e designa o *identificador de arquivo* para `fid`. Os seguintes **fprintf** usam `fid` para escrever as strings e os dados no arquivo em questão. Os dados são formatados de acordo com as especificações no argumento-string de **fprintf**. No comando acima, `\n` insere uma nova linha, `%4i` insere um campo inteiro de largura 4, e `%8.2f` insere um ponto fixo de largura 8 e 2 casas decimais após a vírgula. O arquivo de saída, `Temperature.table`, é mostrado abaixo. Note que a matriz de dados `t` tem duas filas, no qual o arquivo de saída escreve a matriz em 2 colunas. Isto é porque `t` é lida columnwise (coluna sábia) e então escrito no formato especificado (2 linhas em cada coluna).

```
Temperature Table
~~~~~
Fahrenheit    Celsius
-40           -40.00
```

-35	-37.22
-30	-34.44
-25	-31.67
-20	-28.89
-15	-26.11
-10	-23.33
-5	-20.56
0	-17.78
5	-15.00
10	-12.22
15	-9.44
20	-6.67
25	-3.89
30	-1.11
35	1.67
40	4.44
45	7.22
50	10.00
55	12.78
60	15.56
65	18.33
70	21.11
75	23.89
80	26.67
85	29.44
90	32.22
95	35.00
100	37.78

6.4. OBJETOS DE DADOS AVANÇADOS

No MATLAB 5 alguns e novos objetos de dados foram introduzidos, nomeados *estruturas* e *células*. Além disso, o mais familiar objeto de dados, a matriz, foi transformada – agora ela pode ser multidimensional. Embora uma detalhada discussão desses objetos e suas aplicações além do espaço imaginável não tenham ainda sido abrangidos nessa apostila, nesta seção dar-se-á uma introdução suficiente para que o usuário se sinta inicialmente preparado. A princípio, esses objetos parecem ser bem diferentes do principal elemento do MATLAB, a matriz. O fato, entretanto, é que esses objetos, bem como a matriz, são somente casos especiais do tipo fundamental de dados, o vetor. Eles se “encaixam” em seus lugares corretos e começa-se a trabalhar com eles do mesmo modo que com vetores e matrizes.

6.4.1. MATRIZES MULTIDIMENSIONAIS

O MATLAB 5 suporta matrizes multidimensionais. Você pode criar matrizes de dimensão n especificando n índices. As funções de criação das matrizes usuais, zeros, ones, rand e randn aceitam n índices para criar tais matrizes. Por exemplo:

A = zeros (4, 4, 3) inicializa uma matriz A 4x4x3 com todos os elementos nulos
B = rand (2,4,5,6) cria uma matriz aleatória B 2x4x5x6 (na 4ª dimensão!)

É fácil visualizar uma matriz 2-D. Há como escreve-la em uma folha plana (uma página). Já matrizes 3-D seriam um pouco mais complexas. Imaginemos (l, c, p). O primeiro termo indicaria o número de linhas, o segundo o número de colunas, e o terceiro indicaria o número de páginas em que seriam escritas p matrizes $l \times c$. Assim, A = zeros (4, 4, 3) poderia ser representado por 3 páginas, cada uma delas contendo uma matriz 4 x 4 cheia de zeros. Agora, suponhamos que se queira mudar o termo da quarta linha e quarta coluna da matriz na página 2. Então, escrevemos, no MATLAB, A(4, 4, 2) = 6. então, as regras usuais de ordenação se aplicam: Deve-se unicamente pensar qual matriz – no total de páginas – está sendo acessada. Podemos associar então a terceira dimensão a um caderno cheio de páginas que representam o 2-D. Assim fica fácil expandir o conceito de “dimensões” para matrizes. Pensemos em uma prateleira com n cadernos (4-D), uma estante com n_1 prateleiras (5-D), uma sala com n_2 estantes (6-D), um bloco com n_3 salas (7-D), um prédio com n_4 blocos (8-D), e assim por diante. Assim você pode acessar qualquer matriz 2-D nesse conjunto. Por exemplo, para acessar o elemento a_{11} de cada página de cada caderno de cada prateleira de cada estante de cada sala de cada bloco de cada prédio, escreve-se C(1, 1, :, :, :, :, :, :). As regras de ordenação se aplicam em cada dimensão. Portanto, você pode acessar submatrizes com índices variáveis do mesmo modo como você faria para acessar uma matriz 2-D.

Quando se opera em matrizes multidimensionais deve-se ser cuidadoso. Todas as funções de álgebra linear são aplicadas a matrizes 2-D. Não se pode multiplicar matrizes de dimensões diferentes, e nem mesmo que sejam da mesma dimensão, mas

diferente de 2-D. Porém, todas as operações elemento por elemento são válidas para qualquer dimensão. (Ex: $5*A$, $\sin(A)$, $\log(A)$). Também operações $A+B$ e $A-B$ são aceitas, desde que as dimensões sejam iguais entre si.

6.4.2. ESTRUTURAS OU REGISTROS

Uma estrutura é uma construção de dados cujo agrupamento é denominado *campo*. Diferentes campos podem conter diferentes tipos de dados, mas um único campo deve conter dados do mesmo tipo. Uma estrutura é um registro. Um registro (estrutura) pode conter uma informação (dados) sobre vários assuntos de diferentes títulos (campos). Por exemplo, você poderia manter um livro de registros com uma página dedicada a cada um de seus parentes. Você poderia listar informações para cada um deles sob os títulos: grau de parentesco, nome, telefone, filhos (quantos), filhos (nomes), etc. Embora os títulos sejam os mesmos em cada registro, a informação contida nos títulos é diferente para cada registro. Implementar esse registro no MATLAB é montar uma estrutura. O melhor é que não há limites para a quantidade de registros dentro do “livro de registros”.

Vejamos um exemplo de estrutura. Está sendo feita uma estrutura chamada *FallSem* com os campos *course*, *prof* e *score*. Deseja-se registrar os nomes dos cursos, de seus professores correspondentes e sua performance no curso:

```
FallSem.course = 'cs101';  
FallSem.prof = 'Turing';  
FallSem.score = [80 75 95];
```

Assim, campos são indicados adicionando seus nomes após o nome da estrutura, separados por um ponto (.). Os campos são valores designados simplesmente como uma outra variável no MATLAB. No exemplo acima, os campos *course* e *score* contêm um vetor de números. Agora, como pode ser gerado um registro para o próximo curso? Estruturas são bem como seus campos: multidimensionais. Portanto, pode-se gerar o próximo registro de diferentes maneiras:

Múltiplos registros em um vetor de estrutura

Pode-se fazer com que a estrutura *FallSem* seja um vetor (neste exemplo, um vetor será o bastante), então armazene um registro completo como um elemento do vetor:

```
» FallSem.course = 'cs101';  
» FallSem.prof = 'Turing';  
» FallSem.score = [80 75 95];  
» FallSem
```

Cria uma estrutura *FallSem* com 3 campos:
course, *prof*, *score*;

```
FallSem =  
  course: 'cs101'  
    prof: 'Turing'  
   score: [80 75 95]
```

Quando requisitado, o MATLAB mostra a estrutura.

```
» FallSem(2).course = 'phy200'; FallSem(3).course = 'math211';  
» FallSem(2).prof = 'Fiegenbaum'; FallSem(3).prof = 'Ramanujan';  
» FallSem(2).score = [72 75 78]; FallSem(3).score = [85 35 66];  
» FallSem
```

```
FallSem =  
1x3 struct array with fields:  
    course  
    prof  
    score
```

Após adicionar mais 2 registros, FallSem torna-se um vetor, e, quando requisitado, o MATLAB mostra informações sobre a estrutura;

```
» FallSem(2).course  
  
ans =  
phy200
```

Usa o índice do vetor na estrutura para acessar os seus elementos.

```
» FallSem(3).score(1)  
  
ans =  
85
```

Pode-se usar a notação de índices para a estrutura assim como para os seus campos.

```
» FallSem.score  
  
ans =  
80 75 95  
  
ans =  
72 75 78  
  
ans =  
85 35 66
```

Quando nenhum índice é especificado para a estrutura, o MATLAB mostra na tela os valores do campo de todos os registros até agora executados.

```
» for k = 1:3,  
    all_scores(k,:) = FallSem(k).score;  
end
```

Para acessar os valores a partir de um campo de muitas gravações use um laço.

Assim, foi criado o arranjo de estrutura FallSem 1 x 3. Cada elemento de FallSem pode ser acessado assim como se acessa um elemento de um vetor comum – faz FallSem(2) ou FallSem(1), etc. Digitando FallSem(1), obtêm-se 2 valores de todos os campos junto com os nomes do campo. Pode-se também acessar campos individuais (FallSem(1).prof ou FallSem(1).score(3)).

Obs.: Em um vetor de estrutura, cada elemento deve ter o mesmo número de campos. Cada campo, entretanto, poder conter dados de diferentes tamanhos. Portanto, FallSem(1).score pode um vetor linha de 3 espaços enquanto que FallSem(2).score um vetor coluna de 5 espaços.

Múltiplas gravações em campos de vetores.

Para o exemplo escolhido, pode-se armazenar múltiplos registros em uma única estrutura (ou seja, guarda FallSem 1 x 1) fazendo com que os campos sejam de tamanhos apropriados para acomodar os registros:

```
FallSem.course = char('cs101', 'phy200', 'math211');  
FallSem.prof = char('Turing', 'Fiegenbaum', 'Ramanujan');  
FallSem.score = [80 75 95; 72 75 78; 85 35 66];
```

Neste exemplo, a função char é usada para criar de caracteres string separadamente das variáveis de entrada. Aqui, FallSem é uma estrutura 1 x 1, mas o campo course é um arranjo de caracteres 3 x 7, prof é de tamanho 3 x 10 e score é um vetor de números 3 x 3.

Bem, este exemplo funciona muito bem porque se poderia criar um vetor coluna de nomes de curso, outra de nomes de professores, e uma matriz de scores onde cada linha corresponde a um curso diferente. O que aconteceria se o terceiro registro tivesse uma matriz para cada curso? Poderia-se ainda armazenar o registro por ambos os caminhos mencionados anteriormente. Enquanto o primeiro método de criação de um vetor de estrutura parece ser o caminho mais fácil, pode-se também usar o segundo método e obter o terceiro campo score sendo uma matriz 3-D.

Criando Estruturas

Nos exemplos anteriores, já se tinha visto como criar estruturas por designação direta. Assim como pode-se criar uma matriz digitando seu nome e designando valores para ela – $A = [1 \ 2 \ 3 \ 4]$ – pode-se criar uma estrutura digitando seu nome junto a um campo e designando valores ao campo, assim como feito nos exemplos anteriores. O outro caminho para criar uma estrutura é com a função struct. A sintaxe geral de struct é:

```
str_name = struct('nome do campo1','campo1',' nome do campo2','  
campo2',...)
```

Assim, a estrutura criada FallSem poderia ser criada como se segue:

Como uma simples estrutura:

```
FallSem = struct('course',char('cs101','phy200','math121'),...  
    'prof',char('Turing','Fiegenbaum','Ramanujan'),...  
    'score',char([80 75 95; 72 75 78; 85 35 66]));
```

Como um vetor de estrutura:

```
Fall_Sem = [struct('course','cs101','prof','Turing',...  
    'score',[80 75 95]);
```



```
struct('course','phy200','prof','Fiegenbaum',...  
      'score',[72 75 78]);  
struct('course','math121','prof','Ramanujam',...  
      'score',[85 35 66]);
```

Manipulando estruturas

Manipulação de estruturas é similar à manipulação de arranjos em geral – acessar elementos de estrutura pelo próprio índice e manipular seus valores. Há, porém, uma grande diferença: não se pode designar todos os valores de um campo através de um vetor de uma estrutura a uma variável com 2 pontos variáveis e especificadores.

Assim, se Fall_Sem é um arranjo de estrutura 3 x1 então:

Fall_Sem(1).score(2)	é válido e mostra o elemento do score oriundo do primeiro registro de Fall_Sem
Fall_Sem(1).score(:)	mostra todos os elementos de score vindos de Fall_Sem(1).score
Fall_Sem(:).score	É inválido, não designa score vindos de todos os registros para, embora o comando Fall_Sem(:).score ou (Fall_Sem.score) mostra escores de todos os registros.

Então, embora se possa ver na tela os valores de campo através de múltiplos registros com Fall_Sem(:).score, deve-se usar um loop para designar os valores à variável:

```
For k=1:3, all_scores(k,:)=Fall_Sem(k).score; end
```

A designação não pode ser feita diretamente com o operador de coluna porque os valores dos campos de muitos registros são tratados como entidades diferentes. Os conteúdos de campo são também permitidos serem de diferentes tamanhos. Portanto, embora se possa usar um laço for para designação, deve tomar cuidado extra para assegurar que a designação faz sentido. Por exemplo, no laço for citado, se Fall_Sem(2).score tem somente 2 scores de teste, então a designação produzirá um erro.

Está claro, dos exemplos mostrados, que se pode usar índices para a estrutura tanto como campos para acessar informação. Até agora, têm-se usado somente arranjos de caracter e de números nos campos. Pode-se, entretanto, haver também estruturas dentro de estruturas. Porém, o nível de ordenação torna-se completamente envolvido e requer cuidado extra quando se têm estruturas “entrelaçadas”.

Há também várias funções que ajudam na manipulação das estruturas – fieldnames, setfield, getfield, rmfield, isfield, etc. Os nomes da maioria dessas funções já sugere seu papel.

6.4.3. CÉLULAS

Uma célula é o mais versátil objetode dados no MATLAB. Pode conter qualquer tipo de dados – um arranjo de números, strings, estruturas, ou outras células. Um arranjo de células é chamado de *cell array*. Pode-se pensar que uma célula é como um arranjo de recipientes de dados. Imagine que se tenha um arranjo 3 x3, `my_riches`. Agora colocam-se roupas em uma caixa, sapatos em outra, um computador em outra, etc. Cada caixa é um recipiente que simplesmente parece com cada uma das outras. Entretanto, o conteúdo de cada uma é diferente. Então, tem-se um arranjo que se pode acomodar qualquer coisa e ainda parecer sumariamente similar. Analogamente, pense nisto como no Universo do MATLAB, trocando-se as caixas, por células, roupas por uma matriz, sapatos por uma estrutura, etc., então teremos um objeto de dados no MATLAB chamado **célula**.

Vamos primeiro criar uma célula, colocando somente dados nela, e então discutir vários aspectos das células usando esta como exemplo:

```
C = cell(2,2);           % cria uma célula 2 x 2. Cell(2) teria o mesmo efeito
C{1,1} = rand(3);        % insere uma matriz randômica 3 x 3 na primeira “caixa”
C{1,2} = char('john', 'raj') % insere um arranjo de string na segunda caixa
C{2,1} = Fall_Sem;        % insere a estrutura Fall_Sem na terceira caixa
C{2,2} = (3, 3);          % insere um célula 3 x 3 na quarta caixa
```

Neste exemplo, criar uma célula parece superficialmente criar um arranjo ordenado, porém há algumas diferenças evidentes. Primeiramente, os conteúdos são tão variados quanto se desejar. Além disso, para designar a dimensão da célula há chaves `{}` no lugar dos parênteses. Mas, por que não se usam os parênteses no lugar das chaves?

Uma célula é diferente de um vetor numérico ordenado que ele distingue entre os recipientes de dados e os conteúdos, e isso permite acesso a ambos separadamente. Quando se trata uma célula como um vetor de recipientes, as células se comportam somente como um arranjo e você pode acessar um recipiente com a sintaxe familiar `C(i,j)`. O que você consegue é o recipiente da *i*-ésima linha e da *j*-ésima coluna. O recipiente carregará um rótulo que indicará se os conteúdos são `doublé`, `char`, `struct` ou `cell`, e de que dimensão. Se você deseja acessar os conteúdos de um recipiente, então deve-se usar a especial *célula-conteúdo-ordenação* – índices dentro de chaves. Assim, para visualizar a matriz randômica em `C(1,1)`, deve-se digitar `C{1,1}`.

Criando Células

Já foi discutido como criar células com a função **cell**. Pode-se também criar células diretamente.

```
C = {rand(3)      char('John', 'raj');   Fall_Sem      cell(3,3)};
```

Este exemplo ilustra que as chaves estão para as células assim como os colchetes estão para um vetor comum quando usado do lado direito da declaração.

Manipulando Células

Manipular células é tão fácil quanto manipular vetores comuns. Entretanto, fica bem mais fácil com a indexação de célula, que não foi visto antes. Crie a célula do último exemplo mostrado, com a estrutura `Fall_Sem` feita como exemplo nesta apostila. Depois, tente os seguintes comandos e veja o que acontece.

```
C(1,1) = {[1 2; 3 4]};  
C{1,2}(1,:)   
C{2,1}(1).score(3) = 100;  
C{2,1}(2).prof  
C{2,2}{1,1} = eye(2);  
C{2,2}{1,1}(1,1) = 5;
```

Há várias funções viáveis para manipulação de células. Algumas delas: **dellstr**, **iscellstr**, **cell2struct**, **struct2cell**, **iscell**, **num2cell**, dentre outros.

Duas funções merecem atenção especial:

<code>celldisp</code>	mostra na tela os conteúdos de uma célula.
<code>cellplot</code>	plota o vetor de células esquematicamente.

7. ERROS

Erros fazem parte de nossas vidas interagindo com computadores ou não. A diferença é que se você interage com computadores os seus erros são indicados imediatamente (geralmente sem rodeios e sem muitos conselhos). A interação com o MATLAB não é diferente. Errar é humano, mas ao programar não podemos permanecer no erro se quisermos que o programa funcione. O MATLAB fornece muitas indicações de erros, mas não esclarece muito bem o que está indicando e o que é necessário fazer para corrigir o erro, portanto iremos dar algumas breves dicas para auxiliar o programador a entender os seus erros no MATLAB.

Aqui estão algumas das mais comuns mensagens de erro, em ordem alfabética, seguidos de algum comando.

- `>> D(2:3, :) = sin(d)`

??? In an assignment A(matrix, :) = B, the number of columns in A and B must be the same.

Esta é um típico problema de transferência de matrizes onde as dimensões das matrizes dos dois lados não são iguais. Use os comandos `size` e `length` para verificar as dimensões em ambos os lados. Por exemplo, para o caso acima execute `size(D(2:3, :))` e `size(sin(d))` ou `size(d)`.

Um erro semelhante ocorre ao tentar transferir uma matriz para um vetor.

- `>> D(:, 2) = d1`

??? In an assignment A(matrix) = B, a vector A can't be resized to a matrix.

Neste exemplo, `D` e `d1` são matrizes, mas `D(:, 2)` é um vetor (a segunda coluna de `D`), então `d1` não pode ser colocado dentro de `D(:, 2)`.

- `>> (x, y) = circlefn(5);`

??? (x,

|

A closing right parenthesis is missing.

Check for a missing “)” or a missing operator.

O problema aqui é que o MATLAB utiliza parênteses como índice de matrizes. Para se representar uma lista de vetores para a saída de uma função deve-se utilizar colchetes, portanto o comando correto é: `[x, y] = circlefn(5);`

Quando os colchetes são misturados com os parênteses a mesma mensagem aparece.

- `>> (x, y] = circlefn(5);`

??? (x,

|

A closing right parenthesis is missing.

Check for a missing “)” or a missing operator.

- `x = 1:10;`
`v = [0 3 6];`
`x(v)`
??? Index into matrix is negative or zero.

O primeiro elemento do vetor `v` é zero. Assim nós estamos tentando pegar o elemento zero de `x`. Mas zero não é um índice válido para qualquer matriz ou vetor no MATLAB. O mesmo problema surge quando um número negativo é usado como índice. É claro que um erro também aparece se o índice excede a dimensão da variável.

- `>> x = 1:10; y = 10:-2:-8;`
`x*y`
??? Error using ==> *
Inner matrix dimensions must agree.

Na multiplicação de `x*y` o número de colunas de `x` deve ser igual ao número de linhas de `y` e vice e versa. Aqui `x` e `y` são ambos vetores linha de 10 elementos cada, portanto, não podem multiplicar, mas `x'*y` ou `x*y'` irá executar sem erro.

Muitas outras operações envolvendo dimensões impróprias de matrizes produzem erros parecidos. Por exemplo, `A^2` só tem lógica para uma matriz `A`, se `A` for uma matriz quadrada.

Um outro erro comum é usar o operador de matriz quando você quer usar o operador seqüencial. Por exemplo, para os mesmos vetores `x` e `y` acima, `y.^x` retorna a exponenciação elemento por elemento, já `y^x` produz um erro:

```
>> y^x
??? Error using ==> ^
Matrix dimensions must agree.
```

- `[x, y] = Circlefn;`
??? Input argument r is undefined.
Error in ==> Macintosh HD:MATLAB 4.1:Circlefn.m
On line 4 ==> `x = r*cos(theta); y = ...`

Uma função foi executada sem uma entrada apropriada. Esta é uma das poucas funções que fornecem informações suficientes (nome da função, o diretório onde está a função e a linha em que o erro ocorreu).

- `[t, x] = Circle (5);`
??? Attempt to execute SCRIPT Circle as a function.

Aqui temos um caso comum de erro em que `Circle`, que é um procedimento (script file), está sendo usado como função (function file). Nas funções podemos especificar as entradas e as saídas, mas nos procedimentos não.

Outro caso interessante que produz o mesmo erro é quando tentamos executar a função abaixo.

```
Function [x, y] = circlefn (r);  
% CIRCLEFN – função que desenha um círculo de raio r.  
  
theta = linspace (0, 2*pi, 100);  
x = r*cos (theta); y = r*cos (theta);  
plot (x, y);
```

O erro será:

```
>> [x, y] = circlefn (5);  
??? Attempt to execute SCRIPT circlefn as a function.
```

A princípio você se assusta: “Opa!? Esse arquivo não é um procedimento!”. Tudo bem, mas também não é função, porque um arquivo só é classificado como função se no início estiver escrito “function” com f minúsculo, e não “Function”.

- [x, y] = circlefn ();
??? [x, y] = circlefn ()

Missing variable or function.

Este erro é claro. Você não colocou a variável de entrada.

- circlefn [5];
??? circlefn [

Missing operator, comma, or semi-colon.

A entrada das funções deve ser especificada entre parênteses.

- >> x = b+2.33
??? Undefined function or variable b.

Neste caso a mensagem já indica o erro exato: a variável b não foi criada. Mas se o mesmo erro ocorrer com uma função ou com um procedimento que você já criou, qual seria o erro? Provavelmente, o seu arquivo está em outro diretório. Para verificar utilize o comando **dir**.

- >> global a, b
??? Undefined function or variable b.

Neste caso você pretende declarar as variáveis a e b como globais, mas você não pode utilizar aquela vírgula depois do a, pois o MATLAB enxerga vírgulas como separadores, assim ele executa global a, e depois executa b. Quando digitamos apenas o

nome de uma variável no MATLAB ele retorna o valor dessa variável, e como o b ainda não existe ocorreu este erro. Portanto, não use vírgulas nos comandos **global**, **save** ou **load**.

Um outro erro parecido ocorre no laço for quando se digita: for i = 1, n. Contudo o MATLAB não dará uma mensagem de erro, ele irá executar o laço apenas uma vez (para i = 1) e retornará o valor de n.

- plot (d, d1)
??? Error using ==> plot
Vectors must be the same lengths.

Os vetores de entrada no comando plot devem ser compatíveis. Para mais detalhes, veja os comentários sobre o comando plot no capítulo de gráficos.

Apêndice (Respostas dos exercícios do tutorial)

Os comandos abaixo resolvem os problemas do capítulo de tutorial.

Lição 1: Criando e Trabalhando com Vetores

1. `x = [0 1.5 3 4 5 7 9 10];`
`y = 0.5*x-2`
Resposta: `y = [-2.0000 -1.2500 -0.5000 0 0.5000 1.5000 2.5000 3.0000].`
2. `t = 1:10;`
`x = t.*sin(t)`
`y = (t-1)./(t+1)`
`z = sin(t.^2)./(t.^2)`
3. `theta = [0; pi/4; pi/2; 3*pi/4; pi; 5*pi/4]`
`r = 2;`
`x = r*cos(theta); y = r*sin(theta);`
`x.^2 + y.^2`
4. `n = 0:10;`
`r = 0.5; x = r.^n;`
`s1 = sum(x)`
`n = 0:50; x = r.^n; s2 = sum(x)`
`n = 0:100; x = r.^n; s3 = sum(x)`
Resposta: `s1 = 1.9990, s2 = 2.0000 e s3 = 2.`

Lição 2: Plotando Gráficos Simples

1. `x = linspace(0, 4*pi, 10);` % Com 10 pontos
`y = exp(-.4*x).*sin(x);`
`plot(x, y)`
`x = linspace(0, 4*pi, 50);` % Com 50 pontos
`y = exp(-.4*x).*sin(x);`
`plot(x, y)`
`x = linspace(0, 4*pi, 100);` % Com 100 pontos
`y = exp(-.4*x).*sin(x);`
`plot(x, y)`
2. `t = linspace(0, 20, 100);`
`plot3(sin(t), cos(t), t)`
3. `x = 0:10:1000;`
`y = x.^3;`
`semilogx(x, y)`
`semilogy(x, y)`
`loglog(x, y)`

Lição 3: Criando, Salvando e Executando Procedimentos

1. Substitua o comando `plot(x, y)` pelo comando `plot(x, y, 0, 0, '+')` no arquivo `circle.m`.
2. O seu script file modificado deve se parecer com isso:


```
% CIRCLE – procedimento que desenha um círculo unitário
% -----
r = input('Entre com o raio do círculo: ');
theta = linspace(0, 2*pi, 100); % cria o vetor theta.
x = r*cos(theta); % gera a coordenada x
y = r*sin(theta); % gera a coordenada y
plot(x, y); % Desenha o círculo
axis('equal'); % iguala a escala dos eixos
title('Círculo com o raio fornecido') % põe um título
```

Lição 4: Criando e Executando uma Função

1.

```
function temptable = ctof(tinitial, tfinal);
% CTOF : Função que converte a temperatura de C para F.
% sintaxe de chamada:
% temptable = ctof(tinitial, tfinal);
% -----
C = [tinitial:tfinal]'; % cria um vetor coluna C
F = (9/5)*C + 32; % calcula o correspondente em F.
temptable = [C F]; % cria uma matriz de 2 colunas com C e F.
```
2.

```
function w = prodvet(u, v);
% PRODVET: função que calcula w = u x v para vetores u e v.
% sintaxe de chamada:
% w = prodvet(u, v);
% -----
if length(u) > 3 | length(v) > 3, % se u OU v têm mais de 3 elementos
    error('Pergunte ao Euler. Este produto vetorial vai além de mim.')
end
w = [u(2)*v(3)-u(3)*v(2); u(3)*v(1)-u(1)*v(3); u(1)*v(2)-u(2)*v(1)];
```
3.

```
function s = somaserieg(r,n);
% SOMASERIEG: função que calcula a soma de uma série geométrica.
% A série é 1 + r + r^2 + r^3 + ... + r^n.
% sintaxe de chamada:
% s = somaserieg(r,n);
% -----
nvetor = 0:n; % cria um vetor de 0 a n.
serie = r.^nvetor; % cria um vetor com os termos da série.
s = sum(serie); % soma todos os elementos do vetor 'serie'.
```