Primeiras Lições de MATLAB*

Prof. Doherty Andrade
www.metodosnumericos.com.br

Resumo

Nessas notas apresentamos uma introdução básica ao MATLAB. Apresentamos seus comandos mais utilizados e algumas funções. Vamos aprender a trabalhar com M-files e dar uma pequena introdução a programação em MATLAB. Esperamos que essas notas introduzam o leitor menos experiente ao mundo do MATLAB.

Sumário

Lista de Figuras

Lista de Tabelas

1 O que é MATLAB

MATLAB é uma linguagem de programação e uma ferramenta de cálculo. MATLAB significa **Matrix Laboratory**. Foi originalmente escrito por Cleve Molder, da Math Works, Inc., para permitir acesso fácil ao software matricial desenvolvido nos projetos LINPACK e EISPACK. A versão inicial da década de 70 era destinada aos curos de Álgebra Linear Computacional. O MATLAB, como o nome diz, é uma ferramenta baseada em matrizes. As rotinas abertas do MATLAB possibilitou que fossem criadas vários aplicativos para o MATLAB, os chamados **toolboxes**. Visite o site http://www.mathworks.com

Quando entramos no Matlab, o símbolo $>>$ aparece na janaela de comandos, é o prompt do MATLAB. Este símbolo indica que você pode escrever um comando. Os comandos do MATLAB podem terminar com ponto e vírgula ou não. Quando um comando termina em ponto e vírgula, ele é executado mas o resultado não é mostrado para o usuário.

Explore o MATLAB iniciando no **Start**, ícone no canto inferior esquerdo. Use o help do MATLAB digitando **help** na janela de comandos.

O editor de textos do MATLAB oferece muitas facilidades. Com ele você pode salvar os seus arquivos MATLAB em vários formatos. Por exemplo, Tex, HTML, PowerPoint, Word e XLM. A partir do menu escolher do editor de textos escolher **Publish to**. Veremos mais adiante como criar um arquivo MATLAB, arquivos com terminação .m; depois disso, poderemos salvá-los em outros formatos.

Aprenda a usar o help. Digitando **doc name** obtemos informações sobre o objeto **name**. Experimente **doc dsolve** para aprender sobre esse comando.

Vejamos alguns comandos básicos do MATLAB.

Com o comando

```
>> t=1
```

obtemos o número

```
t=1
```

Com o comando

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

obtemos

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

O mesmo A pode ser obtido da seguinte forma

```
for n=1:1:10
A(n)=n-1;
end
```

O comando n:1:1:10 diz que n inicia com 1 e é acrescido de 1 até atingir 10.

Para plotar um gráfico simples, usamos o seguinte comando

```
>> x=-2:.1:2; % dominio da variavel, de -2 a 2 com incremento 0.1
              % observe o incremento de 0.1
>> y=x.^3-0.4*x+0.141; % funcao de variavel x
>> plot(x,y) % comando para plotar
```

Veja o gráfico gerado pelos comanados acima na figura 2.

No exemplo a seguir, plotamos com linhas mais grossa.

```
> x=-5:.01:5;
>> y=3*x.^2+3*x + 5;
>> plot(x,y,'LineWidth',3),grid
```

Com o comando **who** ou **whos** obtemos informações sobre o que está armazenado na memória. Outros comandos importantes são **clear** para limpar a memória, **quit e exit** para sair do MATLAB, **CRTL C** para interromper a execução do MATLAB.

Com o comando

```
A(6)
```

obtemos

```
ans =
     5
```

o elemento que está na sexta posição do vetor A.

Podemos calcular os valores que uma determinada função assume em todos os pontos do vetor A. Como exemplo, vamos calcular o seno de todos as entradas do vetor A, isto pode ser feito da seguinte forma

```
Y=sin(A)
```

O MATLAB retornará os valores da função seno calculado em cada ponto do vetor A, argumento em radianos:

```
Y =

0    0.8415    0.9093    0.1411   -0.7568   -0.9589   -0.2794
0.6570    0.9894    0.4121
```

Tente os comandos

```
>>a=2*A
>>b=A.^2
```

Note que como A é um vetor o ponto depois de A em

```
 >>b=A.^2
```

Operação	Símbolo	Exemplo
adição a+b	+	3+5
subtração a-b	-	4-2
multiplicação a.b	*	4.23 * 2.34
divisão $a \div b$	ou /	34/6 ou 45\9
potência a^b	^	5^3

Tabela 1: Símbolo das operações

é muito importante. Este tipo de operação .^2 chamada de pontual, indica que será realizada em cada componente de A.

Experimente utilizar as setas para cima, para baixo, esquerda e direita, aproveite esta facilidade.

O MatLaB opera com números complexos do mesmo modo que opera com os reais, os sinais para as operações são os mesmos. A seguir a tabela com os operações aritméticas básicas.

As expressões são executadas da esquerda para a direita com a seguinte ordem de precedência: operação de potência, seguida das operações de multiplicação e divisão, que por sua vez são seguidas pelas operações de adição e subtração. Parênteses podem ser usados para alterar esta ordem de precedências, onde as operações são executadas dos parênteses mais internos para os mais externos.

2 Operadores relacionais e conectivos lógicos

Veja a tabela 2 para os principais operadores.

Símbolo	Descrição
$<$	menor que
$>$	maior que
$<=$	menor ou igual a
$>=$	maior ou igual a
==	igual
$\sim=$	diferente
&	e
|	ou
$\sim$	não
xor	ou exclusivo

Tabela 2: Operadores Relacionais

3 Formato de saída dos números

Como default, se um resultado é inteiro, o MATLAB mostra o número como inteiro. Igualmente, quando o resultado é real, o MATLAB mostra o número com 4 dígitos a direita do ponto decimal. Se os dígitos do resultado estiverem fora desta faixa, o MATLAB mostra o resultado usando a notação científica como numa calculadora científica. Este default pode, entretanto, ser modificado usando-se o Numeric Format da pasta geral na linha **Preferences** do item **Files** na barra de menus. A seguir

Comando MATLAB	Saída	Comentários
format long	33.50000000000000	16 dígitos
format short e	33.500e+01	5 dígitos mais expoente
format long e	33.50000000000000 e +01	16 dígitos mais expoente
format hex	4040c00000000000	hexadecimal
format bank	33.50	2 dígitos decimais
format +	+	positivo, negativo ou zero
format rat	67/2	racional
format short	33.5000	4 dígitos decimais (formato default)

Tabela 3: Formatos de saída

algumas funções residentes do MATLAB

Com o comando **clear** limpamos a memória do MATLAB.

Vejamos um exemplo de solução de um sistema de equações lineares $Ax = b$:

```
>> A=[3 -1 1; 1 4 1; 2 -1 -5]
A =
     3    -1     1
     1     4     1
     2    -1    -5
>> b=[-3 -4 -3]'
b =
    -3
    -4
    -3
>> x=A\b
x =
   -1.3151
   -0.7260
    0.2192
```

O comando bf inv(A) determina a inversa de A

```
>>B=inv(A)

B =
```

Função	Descrição
sin	seno
cos	cosseno
tan	tangente
asin	arcsin
acos	arccos
atan	arctan
exp	exponencial
log	logaritmo natural
sqrt	raiz quadrada
rem	resto
sign	sinal
abs	valor absoluto
max	maior componente de um vetor
min	menor componente de um vetor
length	comprimento de um vetor
sum	soma das componentes de um vetor
prod	produto das componentes de um vetor
means	média das componentes de um vetor
std	desvio padrão das componentes de um vetor
norm	norma euclidiana de vetor
sort	ordena as componentes de um vetor em ordem crescente

Tabela 4: Algumas funções

```
    0.2603    0.0822    0.0685
   -0.0959    0.2329    0.0274
    0.1233   -0.0137   -0.1781
```

Podemos usar a inversa de A para determinar a solução do sistema $Ax = b$ por meio de $x = B * b$:

```
>> x=B*b
x =
   -1.3151
   -0.7260
    0.2192
```

Com o comando **rank(A)** obtemos o posto de A.

Com o comando A' obtemos a transposta de A.

O MATLAB tem rotinas para determinar a decomposição LU de uma matriz A dada:

```
[L U]= lu(A)
L =
    1.0000         0         0
    0.3333    1.0000         0
    0.6667   -0.0769    1.0000

U =
    3.0000   -1.0000    1.0000
         0    4.3333    0.6667
         0         0   -5.6154
```

Com o comando **[V D] = eig(A)** obtemos uma matriz diagonal D de autovalores e uma matriz V cujas colunas são os correspondentes autovetores tal que $A^*V = V^*D$:

```
>> [V D] = eig(A)
V =
   0.1321            -0.1902 + 0.6396i  -0.1902 - 0.6396i
   0.0932             0.7188             0.7188
  -0.9868            -0.1069 + 0.1629i  -0.1069 - 0.1629i


D =
  -5.1734                  0                  0
        0            3.5867 + 1.1164i         0
        0                  0            3.5867 - 1.1164i
```

4 Construção de Vetores e Matrizes

Nas construções das funções implementadas até agora, como pode ser notado, utilizou-se da construção de vetores. Mostraremos algumas outras formas de manipular vetores no MATLAB. Na tabela, tem-se um resumo das diversas formas de se construir um vetor no MATLAB. Veja a tabela 5.

Comandos	Descrição-Cria um vetor x contendo
x=[2 3 sqrt(2) 2-3i]	os elementos especificados
x=primero : último	começando com o valor primeiro, incrementando-se de 1(um) em 1(um) até atingir o valor último ou o valor mais próximo possível de último
x=linspace(primeiro, último, n)	com o valor primeiro e terminado-se no valor último, contendo n elementos
x=linspace(primeiro, último)	com o valor primeiro e terminado-se no valor último, contendo 100 elementos
x=logspace(primeiro, último, n)	os elementos espaçado logaritmicamente, começando-se com o valor $10^{primeiro}$ e terminando-se no valor 10^{ultimo}, contendo n elementos

Tabela 5: Criando vetores

É muito fácil criar uma matriz no MATLAB, por exemplo, como o comando

```
>>A=[1 2 3; 3 4 5; 6 -2 3]

A =
     1     2     3
     3     4     5
     6    -2     3
```

criamos uma matriz 3×3 as linhas são separdas por ponto e vírgula.

O MATLAB tem uma lógica interessante para tratar com matrizes, digitando

```
>>A >3
```

ele retorna uma matriz 3×3 indicando com 1 as posições em que os elementos de A são maiores do que 3. Tente os comandos

```
>>A >=3
>>A <=3
>>A~=3
```

Este comando é importante pois, com ele fica fácil saber se temos algum elemento de uma matriz 50 mil por 50 mil maior do 5 que 5, por exemplo.

Digite na janela de comandos o comando abaixo para mais operadores. Veja a tabela 6

```
>>help  ops
```

Comando	Função
eig	autovalores e outovetores
chol	decomposição Cholesky
svd	decomposição a valores singulares
inv	inversa
lu	decompoisção LU
qr	decomposição QR
hess	forma hessenberg
schur	decomposição schur
rref	forma escalonada reduzidapor linhas
expm	exponencial de matrix
sqrtm	raiz quadrada de matriz
poly	poliômio característico
det	determinante
size	dimensões de matriz
norm	1-norma, 2-norma, F-norm, ∞-norm*
cond	número de condição na 2-norma
rank	posto

Tabela 6: Comandos para matrizes

5 Criando variáveis

Os nomes das variáveis devem consistir de uma única palavra, de acordo com as três regras abaixo:

Regras de Criação de Variáveis	Exemplos
Letras minúsculas e maísculas são diferentes	XY e xy são diferentes
Podem ter até 19 letras	abcdefghij
Devem iniciar com letra, pode ser seguido por quaisquer letras, dígitos ou subescrito. Caracteres de pontuação não podem ser utilizados	 how_about, x512 ,$a\_b\_c\_d$

Em adição às regras acima para formação das variáveis, as seguintes variáveis são especiais no MATLAB : ans, pi, eps, flops, inf, NaN, i, j, nargin, nargout, realmin e realmax. Veja a tabela 7 para detalhes.

As variáveis **realmin e realmax** denotam o menor e o maior real positivo no MATLAB. Por exemplo,

Variáveis especiais	Significado
ans	Nome ”default”da variável usado para resultados
pi	Constante π
eps	O menor número que somado a outro resulta em número diferente
flops	Conta o número de operações em ponto- flutuante
inf	Indica um número infinito
NaN	Indica que não é um número
i (e) j	$i = j = \sqrt{-1}$
nargin	Número de argumentos de entrada usados em uma função
nargout	Número de argumentos de saída usados em uma função
realmin	O menor número real positivo utilizável
realmax	O maior número real positivo utilizável

Tabela 7: Variáveis protegidas

```
>> realmin

ans =   2.2251e-308
>> realmax

ans = 1.7977e+308
```

MATLAB tem três variáveis para representar importantes: **-Inf , Inf , NaN**. O –Inf e Inf são as representações do IEEE para o $-\infty$ e o $+\infty$. O variável NaN afirma que não é um número.

6 Gerenciamento de arquivos

O MATLAB possui uma série de comandos para gerenciamento de arquivos, tais como listar os nomes de arquivos, vizualizar, deletar, etc. Na tabela 8, abaixo tem-se um resumo dos principais comandos:

7 O comando diary

O comando **diary** <**nome do arquivo**> permite que guardemos tudo o que aparece na tela, exceto gráficos, no arquivo < **nome do arquivo** > até que digitemos **diary off**.

8 Controle de Fluxo

8.1 Operadores Lógicos

Os operadores

Comando	Descrição
cd	mostra o diretório de trabalho
p=cd	retorna para a variável **p** o diretório de trablaho corrente
cd temp	muda para o diretório temp
cd..	muda para o diretório um nível acima
chdir	o mesmo que cd
chdir path	o mesmo que cd temp
delete test	deleta o arquivo test.m
dir	lista todos os arquivos do diretório presente
ls	o mesmo que dir
matlabroot	retorna o caminho do diretório onde se encontra o programa MATLAB executável
path	visualiza todos os caminhos de diretoório do MATLAB
pwd	o mesmo que cd
type test	visualiza o arquivo M-file teste.m na janela de comandos
what	retorna uma lista de todos os M-files do diretório corrente
which test	visualiza o caminho diretório do arquivo test.m

Tabela 8: Gerenciando arquivos

```
&, | , ~
```

correspondem aos operadores lógicos ”e”, ”ou”e ”não”.

Podemos utilizar esses operadores lógicos, como nos exemplos,

```
C = A & B
```

é uma matriz cujos elementos são 1s onde ambas as matrizes A e B são elementos não-nulos, e 0s onde uma das matrizes ou ambas são elementos nulos.

```
C = A | B
```

é uma matriz cujos elementos são 1s onde tanto A ou B possuem elementos não-nulos, e 0s onde ambas posssuem elementos nulos.

```
B = ~A
```

é uma matriz cujos elementos são 1s onde a matriz A é um elemento nulo, e 0s quando A é um elemento não-nulo. Todas as operações acima são válidas apenas para quando A e B possuem mesma dimensão, ou quando uma das duas matrizes é um escalar.

8.2 O Comando for

A sintaxe do comando **for** é a seguinte Vejamos um exemplo: Quando o incremento é omitido, o valor assumido é 1, isto ocorre no exemplo acima.

Mais um exemplo usando **for**, os seguintes comandos determinam um ponto fixo da função $f(x) = \cos(x)$ com aproximação inicial $x = 1.$ e 100 iterações. A resposta encontrada é $x = 0.7391$ radianos.

for i=valorinicial:incremento:valorfinal instruções end

Exemplo 1	Exemplo 2
for i=0:1:10 end	for i=0:10 end

```
x=1; for i=1:100, x=feval('cos',x), end
```

Vejamos um exemplo menos simples. Queremos determinar uma raiz de $f(x) = x^2 + x - 6$. Isto é, desejamos resolver a equação $f(x) = 0$ ou equivalentemente, $x = \sqrt{6 - x}$. Veja o gráfico 1, que mostra que há um ponto fixo em $x = 2$.

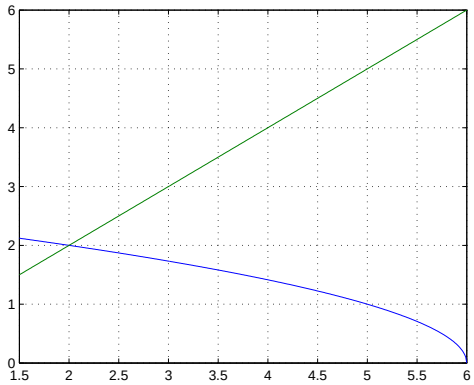


Figura 1: Existência de ponto fixo

Para determinarmos este ponto fixo, precisamos escrever um programinha definindo a função $\phi(x) = \sqrt{6 - x}$. Crie esta função de modo anônimo como indicado

```
>>func = @(x)(6-x)^.5;
```

Agora o programinha que determina o seu ponto fixo

```
>>x=1.; %chute incial
for i=1:100, x=func(x); end
x %chamando o ponto fixo

x =
    2
```

Note que este programinha chama a função *func* que foi definida anteriormente.

8.3 O comando while

A sintaxe do comando **while** é a seguinte:

```
while condição
instruções
end
```

As instruções são executadas enquanto a condição satisfeita. Vejamos um exemplo simples.

```
>> n=0;
>> while n<=10;
n=n+2
 end
```

8.4 O comando if

A sintaxe básica desse comando é

```
if Condição
Instruções
end
```

Caso a condição seja satisfeita, as instruções são executadas. Outra forma deste comando é a seguinte: Nesta forma, se a condição for satisfeita as Instruções I serão

```
if Condição
Instruções I
else  Instruções II
end
```

executadas; caso contrário as Instruções II serão executadas.

Há ainda outra forma para este comando: Neste caso, se a Condição 1 não for

```
if  Condição 1
Instruções I
elseif  Condição 2
Instruções II
else  Instruções III
end
```

satisfeita, uma nova condição é verificada, se for verdadeira as Instruções II serão executadas e se for falsa, as Instruções III serão executadas.

Resumindo, o MATLAB suporta as seguintes variantes do **if**

if ... end
if ... else ... end
if ... elseif ... else ... end

Vejamos um exemplo

```
%%%% testar se a*x^2+b*x+c=0 tem raiz complexa
clear
a=input('Entre com o valor de a=');
b=input('Entre com o valor de b=');
c=input('Entre com o valor de c=');
x=-5:.01:b;
y=a*x.^2+b*x + c;
plot(x,y), grid
d = b^2 - 4*a*c;
if d<0
disp('Cuidado: o discriminante é  negativo, raízes complexas');
elseif d==0
disp('o discriminante é zero, raízes repetidas');
else
disp('OK: raízes reais e distintas');
end
```

9 Strings

String é um array de caracteres. Cada caracter é representado internamente pelo seu valor em ASCII.

Vejamos um exemplo:

```
>> srt='Estou aprendendo MATLAB.'

srt = Estou aprendendo MATLAB.
```

10 M-files: Scripts e Functions

A maneira mais simples de utilizar o MATLAB é utilizá-lo como se fosse uma calculadora, entrando com os comandos diretamente no prompt. Entretanto, a medida que o número de comandos aumenta, ou quando se deseja mudar o valor de uma ou mais variáveis e executar novamente os comandos, o uso do MATLAB desta forma é simplesmente ineficiente, tedioso e pouco inteligente. É nesta hora que entram em ação os M-files. No MATLAB existem três tipos de M-files: scripts,functions e live script .

Devemos utilizar o MATLAB como uma linguagem de programação de alto nível, isto é, escrever um arquivo M-file: "script"ou "function´´ ou "live script". Os M-files são os programas fontes do MATLAB e levam a extensão .m , por exemplo, exemplo1.m.

Para escrever um programa no MATLAB, você pode fazer:

• Escolha **File** na barra de menu. Dentro do menu **File** escolha **New** e selecione **M-file**. Abre-se, então, um editor de textos, onde pode-se escrever os comandos do MATLAB. Escreva o programa. Uma vez escrito o programa, entre no menu **File** da janela do editor de textos e escollha a opção **Save as**. Nesta opção do menu, salve o programa como file1.m (por exemplo) no seu diretório de trabalho. Em seguida, feche a janela do editor de textos, o que faz com que volte à janela de comandos do MATLAB.

• Outra alternativa, é simplesmente digitar na janela de comandos o comando "edit´´ que se abrirá o editor.

Na janela de comandos, use o comando cd para ir ao diretório onde o programa file1.m foi salvo. Como default o MATLAB salva em um subdiretório seu chamado de **work**.

Como exemplo digite o seguinte script file1.m

```
>> x=-2:.1:2; % dominio da variavel, de -2 a 2 com incremento 0.1
              % observe o incremento de 0.1
>> y=x.^3-0.4*x+0.141; % funcao de variavel x
>> plot(x,y) % comando para plotar
```

Para executar os arquivos com extensão .m, basta digitar o nome do arquivo na janela de comandos. No nosso exemplo,

```
>> file1
```

e o MATLAB executará as instruções contidas no file1.m gerando a seguinte figura 2.

• Outro tipo de arquivo M-file é o **function** (Não vamos traduzir, para não confundir com função de matemática). Na verdade um **function** é um programa em MATLAB como estamos acostumados. Criamos um **function** do mesmo modo que o script. Embora a criação de um function se faça do mesmo modo que um script, no function a primeira linha deve conter o nome **function**.

A diferença entre o script e o function é a seguinte: no script o MATLAB apenas executa os comandos e no function é preciso entrar com dados e o MATLAB retorna o resultado dos cálculos. Mas a principal diferença está no aproveitamento de memória do MATLAB. No function as variáveis envolvvidas na execução do programa não são guardadas na memória do MATLAB. No script as variáveis são globais.

Vejamos um exemplo de **function** chamado de func1.m:

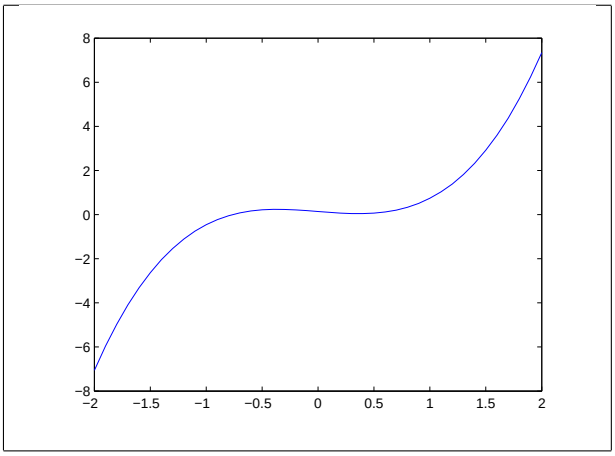


Figura 2: Resultado do script file1.m

```
%%%exemplo de function
function y=func1(x)
 y=5*x*sin(x);
```

Digitando $>>$**func1(3)** na tela de comandos do MATLAB, o programa retornará o valor desta função em $x = 3$.

```
>> func1(3)

ans =
    2.1168
```

Podemos sofisticar mais, MATLAB é uma excelente linguagem de programação. Muitas vezes precisamos fazer um programa interativo de modo que as entradas possam ser dadas via teclado. Para isso, usamos os comandos **input** e **disp**. Por exemplo, crie o script func22.m:

```
clear
t=input('Entre com o valor de t=');
a=exp(t);
b=cos(t);
x=a*b
```

Chame o programa func22 com o comando

```
>> func22
```

O MATLAB retornará

```
Entre com o valor de t=
```

Entrando com $t = 2$, no MATLAB temos

```
>> func22
Entre com o valor de t=2
x =
   -3.0749
```

Como outro exemplo, crie o seguinte M-file

```
clear
a=input('Entre com o valor de a=');

b=input('Entre com o valor de b=');
x=a:.01:b; y= x.*exp(-x.^2);
plot(x,y),grid
```

Por causa da grande utilidade dos M-files, o MATLAB possui diversas funções que tornam os M-files ainda mais interessantes. Algumas dessas funções estão listadas a seguir. Outros exemplos de de M-file:

Função	Função
echo	ecoa cada um dos comandos do M-file na janela de comandos
input	permite a entrada de dados durante a execução
pause	faz uma pausa na execução do programa até que uma tecla seja pressionada
pause(n)	faz uma pausa de n segundos
disp(ans)	visualiza os resultados sem mostrar os nomes das variáveis

```
%% vários desenhos com quantidade de pontos diferentes
%%Exemplo1
for n=1:.5:5;
n10 = 10*n;
x = linspace(-2,2,n10);
y = x./(1+x.^2);
plot(x,y,'r'),
title(sprintf('Gráfico %g. Desenho com  n = %g pontos.' ...
, (n+1)/2, n10)),
 axis([-2,2,-.8,.8]),
 xlabel('x'),
 ylabel('y'),
grid ,
pause(1),
 end

%%%Exemplo2
k = 0; for n=1:3:10 n10 = 10*n;
```

```
x = linspace(-2,2,n10);

y = x./(1+x.^2);
k = k+1;
subplot(2,2,k) plot(x,y,'r')
title(sprintf('Gráfico %g. Desenho com  n = %g pontos.' ...
, k, n10))
xlabel('x')
ylabel('y')
axis([-2,2,-.8,.8]) grid
pause(3); end

%%Exemplo3
x = -1:.05:1;
y = x;
[xi,yi] = meshgrid(x,y); zi = yi.^2 - xi.^2;
surfc(xi,yi,zi)

%%Exemplo 4
x = -1:.05:1;
y = x;
[xi,yi] = meshgrid(x,y); zi = yi.^2 - xi.^2;
surfc(xi,yi,zi)
pause(5)
contourf(zi),
hold on


%%Exemplo 5 - animação
m = moviein(5);
x = 0:pi/100:pi;
for i=1:5,
    h1_line =plot(x,sin(i*x));
set(h1_line,'LineWidth',1.5,'Color','m'),
grid,
 title('Funções sin(kx), k = 1, 2, 3, 4, 5'),
h = get(gca,'Title'); set(h,'FontSize',12), xlabel('x'), k =
num2str(i); if i > 1, s = strcat('sin(',k,'x)'); else s =
'sin(x)'; end ylabel(s), h = get(gca,'ylabel');
set(h,'FontSize',12), m(:,i) = getframe; pause(2), end,
 movie(m)
```

10.1 Funçao Anônima

Além dos M-files, uma segunda maneira de representar ou especificar uma função matemática é criando uma função anônima de uma expressão. Fazemos isto usando

o comando @. Por exemplo, a expressão $x^2 - 2x + 1$ pode ser transformada em função anônima por

```
>> f=@(x) x^2-2*x+1;
```

Podemos usar f para avaliar a expressão

```
>> f(1)

ans =
     0
```

Mais um exemplo simples,

```
X = fminbnd(@cos,3,4) % minimo de cos no intervalo [3,4]
```

Podemos realizar a composta de funções. Vejamos um exemplo simples.

```
 >>syms x, f=@(x) x.^2; g=@(x) x-1-cos(x);
h=@(x) g(f(x))

h =

    @(x) g(f(x))

>> h(x)

ans =

x^2-1-cos(x^2)
```

Muitas vezes temos uma função com mais de uma variável, mas desejamos estudá-la olhando apenas para uma delas e mantendo as outras fixas. Por exemplo,

```
function y = poly(x, y, z) % escreve o polinomio
y = x^3 + y*x + z;
```

Se desejamos tomar $y = 2$ e $z = 3$ e determinar as raízes deste novo polinômio usamos o comando @. Quando fazemos isto, estamos tratando com função anônima.

```
 y= 2;
 z = 3;
 x = fzero(@(x) poly(x, y, z), 0)

x =
   -1.0000
```

11 Funções inline

Muitas vezes é prático definir uma função que será usada apenas durante uma sessão no MATLAB. MATLAB tem um comando usado para definir a chamada **função inline**.

Exemplos:

```
%%%exemplo1
>> f = inline('sqrt(1 -x.^2-y.^2)','x','y')
f =
     Inline function:
     f(x,y) = sqrt(1-x.^2-y.^2)
>> f(.3,.4)
ans =
    0.8660

%%% exemplo2
>> f = inline('1 +x.^2','x')
x=-2:.1:2;
 y=f(x);
 plot(x,y), grid
```

12 Live Script

É mais ou menos recente no MatLab a existência de arquivos tipo *live script*. São arquivos com extensão mlx e nestes arquivos os comandos são executados e os resultados são apresentados imediatamente abaixo ou ao lado, tudo no mesmo ambiente.

Não é possível definir *function* em live script. Se necessário, function deve ser criada em um arquivo com extensão m. Procure criar e usar live script porque representam um avanço nas versões mais recentes.

13 Criando Gráficos

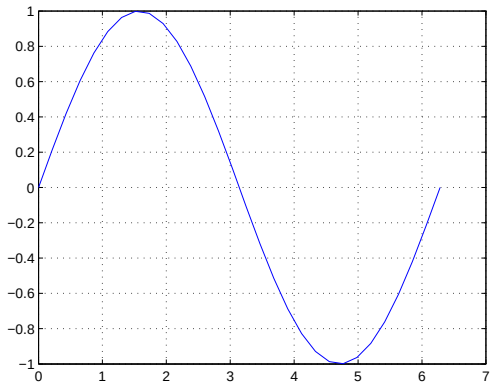
Vejamos alguns exemplos

• Exemplo simples, veja a figura na tabela 9.

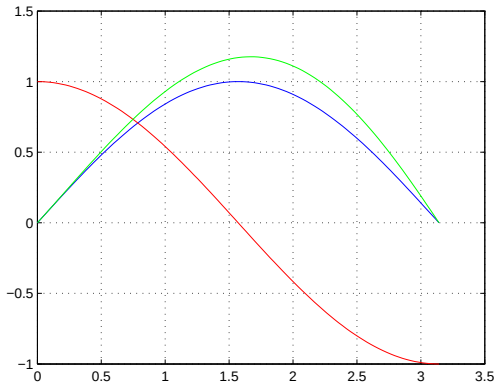
```
>> x=linspace(0,2*pi, 30);
y=sin(x);
plot(x,y); grid on
title('Exemplo')
xlabel('x em rads')
ylabel('y=sin(x)')
```

O MatLab pode plotar vários gráficos juntos num mesmo sistema de eixos. Veja o exemplo a seguir, onde **b**, **r** e **g** representam cores óbvias. Veja a figura na tabela 9.

```
t=0:pi/100:pi;
x=sin(t);
y=cos(t);
z=x.*exp(.1*t);
plot(t,x,'b',t,y,'r',t,z,'g'),grid
```



Plotando um gráfico



Vários gráficos juntos

Tabela 9: Gráficos

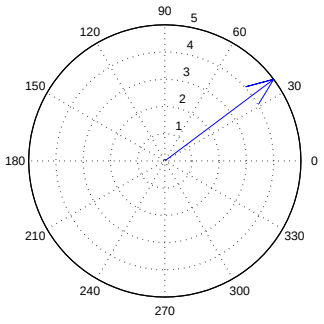
A função **fplot** fornece uma representação gráfica melhor, pois concentra a sua avaliação sobre regiões onde a taxa de variação da função é maior. Para avaliar uma função, deve-se criar um arquivo function e passar o nome do arquivo para **fplot**. Vejamos um xemplo:

```
fplot(@(x)(x.^2+1), [-2, 2]),grid
```

geramos um gráfico melhor, **fplot** utiliza menos pontos para avaliar a função, mas mostra a função em intervalos menores em uma região onde a taxa de variação é maior, gerando uma figura mais precisa. A opção *grid* ou *grid minor* imprimem junto ao gráfico uma malha para melhor estudar o comportamento da função.

O MatLab pode plotar números complexos escritos na forma polar: a função **compass** realiza essa tarefa. Vejamos um exemplo.

```
z=4+i*3, compass(z)
```



O MATLAB plota numa mesma tela vários gráficos. O comando **subplot(m,n,k)** divide a tela em m linhas e n colunas que serão ocupadas pelos $m \times n$ gráficos, onde k

é a posição referência do gráfico. Vejamos um exemplo simples, os comandos abaixo geram a figura 3.

```
>> clear
 t=0:.01:10;
 x=3*t.*sin(2*t);
 y=t.*cos(t)+t.^2;
 z=exp(-.2*t).*t;
 w=x.*y;
 subplot(2,2,1)
 plot(t,x)
 title('Grafico 1-x')
 xlabel('t')
 ylabel('x')
 subplot(2,2,2)
 plot(t,y)
 title('Grafico 2-y')
 xlabel('t')
 ylabel('y')
 subplot(2,2,3)
 plot(t,z)
 title('Grafico 3-z')
 xlabel('t')
 ylabel('z')
 subplot(2,2,4)
 plot(t,w)
 title('Grafico 4-w')
 xlabel('t')
 ylabel('w')
```

Para plotar gráficos em 3 dimensões, existem vários comandos possíveis. Um exemplo simples é dado pela figura 4.

```
>> t=0:.01:10;
y=sin(5*t); z=cos(5*t); plot3(x,y,z); xlabel('eixo X');
ylabel('eixo y'); zlabel('eixo z');
```

Os comandos **mesh, surf** são usados para gráficos de superficies e **contour** para gráficos de contornos. Veja exemplo e o resultado na figura 5.

```
>>clear
x=-2:.1:2; y=-5:.1:5; n=length(x); m=length(y); for j=1:n
    for i=1:m
        Z(i,j)=sin(x(j))*cos(y(i));
    end
end
```

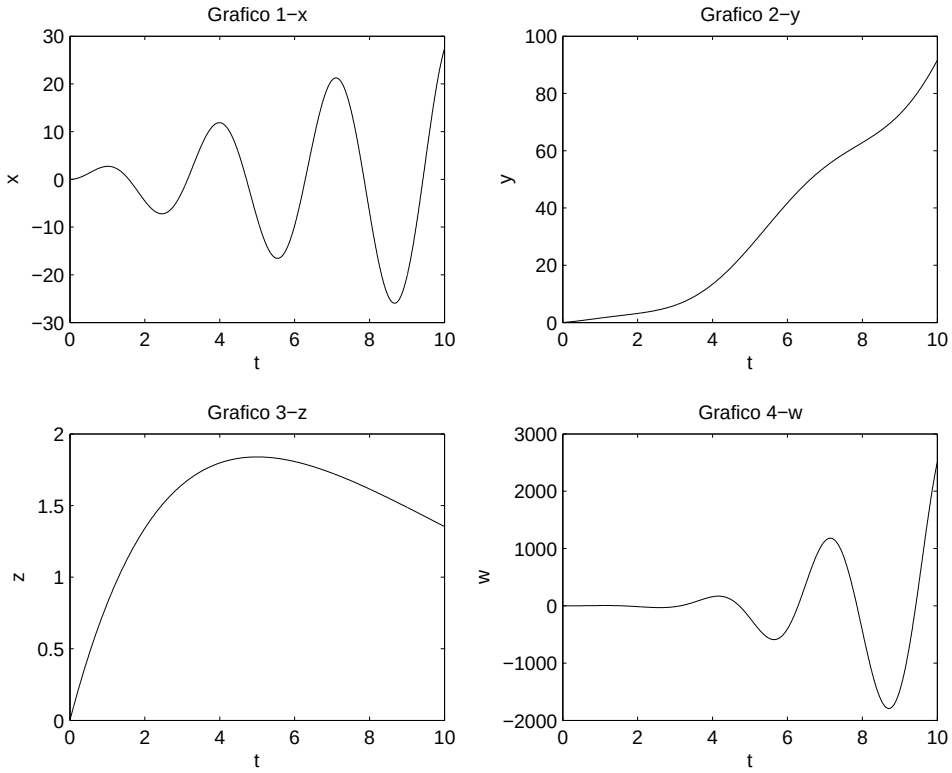


Figura 3: Várias figuras numa mesma tela

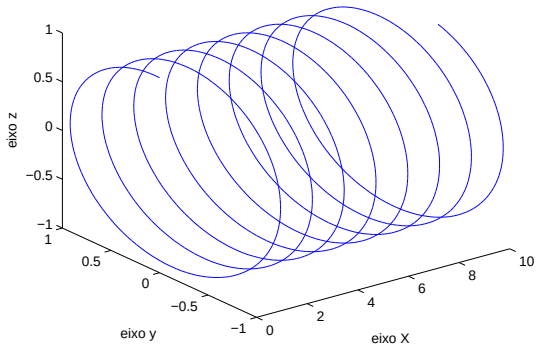


Figura 4: gráfico em 3D

```
 subplot(2, 2, 1);
 mesh(x,y,Z)
 title('Usando mesh')
 xlabel('X')
 ylabel('Y')
 zlabel('Z')
 subplot(2, 2, 2);
 surf(x,y,Z)
 title('Usando surf')
 xlabel('X')
 ylabel('Y')
 zlabel('Z')
 subplot(2, 2, 3);
 contour(x,y,Z,10)
 title('Usando contour com 10 linhas')
 xlabel('X')
 ylabel('Y')
 zlabel('Z')
 subplot(2, 2, 4);
 contour(x,y,Z,30)
 title('Usando contour com 30 linhas')
 xlabel('eixo X')
 ylabel('eixo Y')
 zlabel('eixo Z')
```

Experimente os seguintes exemplos:

```
%%%exemplo1
>>[x, y] = meshgrid([1:.5:10],[1:.5:10]);
>>z = x.^2 - y.^2 ;
>>surf(x,y,z)
>>mesh(x,y,z)
>>contour(x,y,z)
%%%exemplo 2
>>c = x.^2 + y.^2;
>>mesh(x,y,z,c)
```

Exemplo de um script com plot

```
%Plota uma função y=ax^2 + bx + c no intervalo -5<x<5
clear
a=input('Entre com o valor de a=');
b=input('Entre com o valor de b=');
c=input('Entre com o valor de c=');
x=-5:.01:5;
y= a*x.^2+b*x+c;
```

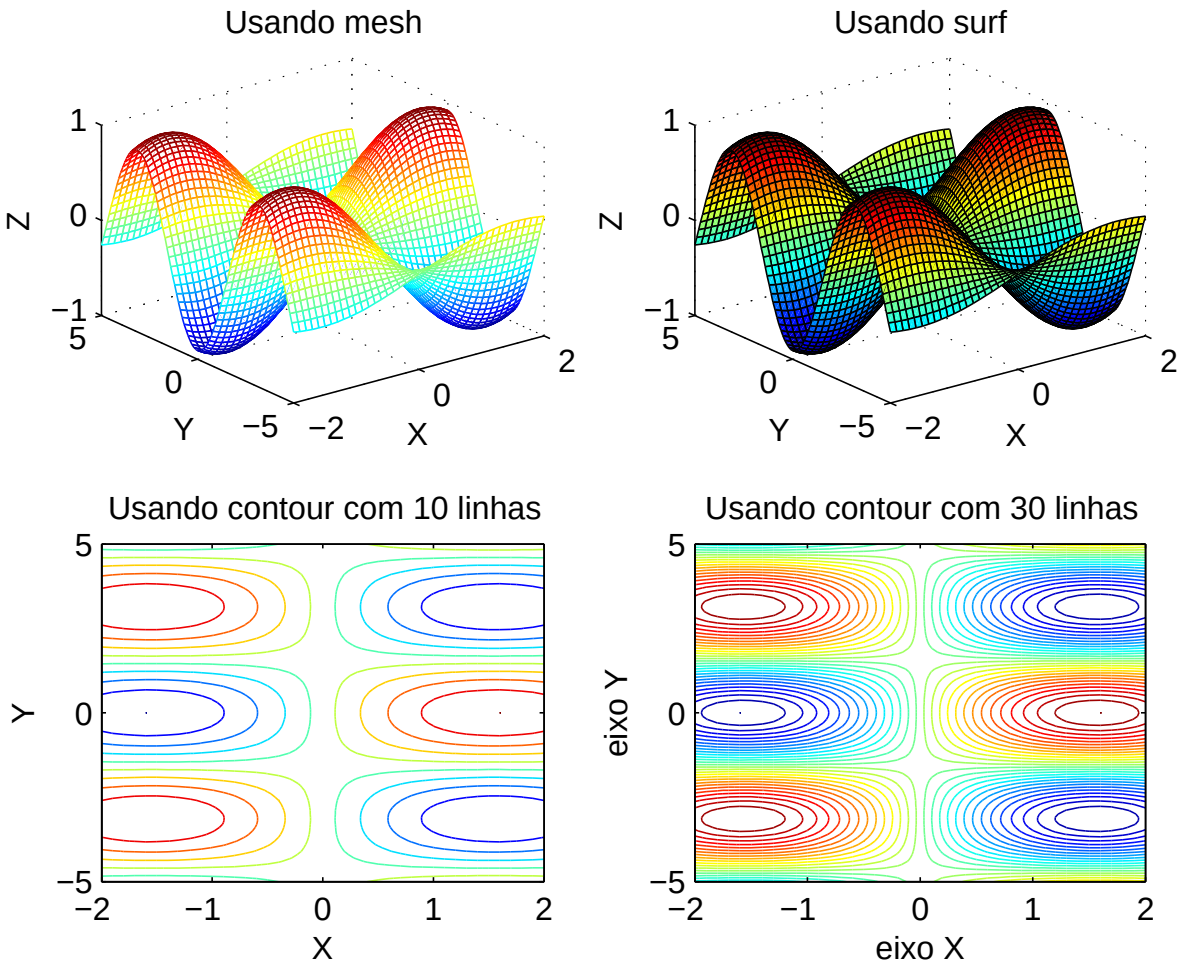
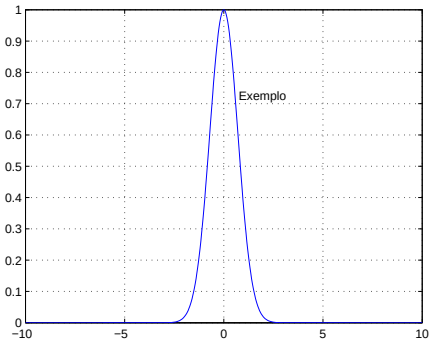


Figura 5: Usando mesh, surf e contour

```
plot(x,y),grid
figura(1)
```

Podemos incluir um texto junto com o gráfico. O seguinte exemplo ilustra o uso do comando **gtext**.Veja

```
>> syms x,fplot(@(x)exp(-x^2), [-10 10]
grid,gtext('Exemplo')
%>> syms x,fplot('exp(-x^2)', [-10 10])
%grid,gtext('Exemplo')
```



13.1 Ângulo de Visão

MATLAB permite que visualize-se um gráfico 3-D de um determinado ângulo. A função **view** define o ângulo de visão em coordenadas esféricas através da especificação do azimute (rotação horizontal) e da elevação vertical do ponto de vista, com relação a origem dos eixos. O azimute é um ângulo polar no plano x-y, sendo positivo quando a rotação for no sentido horário com relação ao ponto de vista. A elevação vertical é o ângulo acima (ângulo positivo) ou abaixo (ângulo negativo) do plano x-y.

A função **peaks** é uma função residente no MATLAB, vamos experimentar ângulos de visão com ela. Por exemplo, as quatro linhas de comando abaixo proporcionam quatro maneiras diferentes de se visualizar a função peaks.

```
%escolha uma superficie e um ângulode visão
%habilite um e desabilite outro angulo de visão
surf(z)
z = peaks;
mesh(z),
view(-37.5,30)
%view(-7,80),
%view(-90,0),
% view(-7,-10),
%view(-37.5,30),
%view(-7,80),
%view(-90,0),
%view(-7,-10)
```

13.2 Imprimindo Gráficos

Tendo a figura na tela, é simples ver como fica a sua impressão a partir do Menu no **Print Preview** e imprimi-la. Outra forma é enviar o gráfico diretamente para a impressora para ser impresso usando o comando **print**. Por exemplo,

```
x = 0:0.01:1;
plot(x, x.^2),
print % para imprimir
```

Tendo na tela um gráfico podemos salvá-lo ou imprimi-lo (como um arquivo) salvando-o como um arquivo com extensão eps, BMP, fig e outras extensões. Digite na linha de comando, um dos seguintes comandos:

depsc Level 1 color Encapsulated PostScript
deps2 Level 2 black and white Encapsulated PostScript
depsc2 Level 2 color Encapsulated PostScript

13.3 Plotando Dados

Suponha que temos os dados dados.dat, representam as temperatura e a precipitação pluviométrica média mensal em cada um dos 12 meses do ano de uma determinada cidade.

```
%% dados. dat
30 4.0

31 3.7

38 4.1

49 3.7

59 3.5

68 2.9

74 2.7

72 3.7

65 3.4

55 3.4

45 4.2

34 4.9
```

Para plotar a precipitação ao longo do ano e a temperatura ao longo do ano, procedemos como indicam os comandos

```
>> load dados.dat
precip = dados(:,2);
temp = dados(:,1);
subplot(2,1,1),
plot(temp),
subplot(2,1,2),
plot(precip),
```

14 Polinômios

MATLAB pode tratar um polinômio como um vetor. Isto é, um vetor contendo os coeficientes do polinômio (inclusive os nulos), escritos da mais alta ordem para menor ordem. Por exemplo, o polinômio $p(x) = x^5 - 5x^3 + 2x^2 + 4x + 1$ é representado pelo vetor

```
>>p = [1 0 -5 2 4 1],
p =
     1     0    -5     2     4     1
```

Vejamos algumas funções do MATLAB para tratar com polinômios e raíz.

```
>>roots(p) %determina todas as raizes de p

ans =
  -2.2786
   1.5488 + 0.3981i
   1.5488 - 0.3981i
  -0.4095 + 0.0626i
  -0.4095 - 0.0626i
```

Para calcular o valor do polinômio p em um ponto $x = 3$ basta escrever

```
 >>polyval(p,3)
```

Do mesmo modo, com o comando

```
>>polyval(p,[1:10])
```

o MATLAB calcula o valor do polinômio em todos os pontos do vetor $[1 : 10]$, retornando um outro vetor de mesmo tamanho.

O MATLAB pode ajustar facilmente um polinômio a um conjunto de dados $[x, y]$ chamado de **dados**

```
x=[1,2 3, 4, 5,6, 7,8 9, 10, 11, 12];
y=[5.35 ,3.68,3.54,2.39,2.06,1.48,0.63,1.09,1.75,2.66,5.34,6.13];
>>%p = polyfit(x,y,n) % n é o grau do pol. interpolador
>>p = polyfit(x,y,2)  % se n=2
p =

    0.1567   -2.0010    7.5259
```

O polinômio de grau 2 que melhor se ajusta aos dados é $p(x) = 0.1567x^2 - 2.0010x + 7.5259$

Com o comando **polyval** e o polinômio p podemos predizer o valor y para um outro valor de x com o comando

```
>>ypred = polyval(p,x)
```

15 Zero de Funções e Otimização

Com o comando **fzero** do MATLAB é possível determinar raízes de uma função. Vejamos um exemplo, como antes, teremos que definir a função e chamá-lo num script. Consideremos achar os zeros de $f(x) = x^2 - \exp(-5x) + \sin(x)$.

A rotina *fzero* necessita e exige uma aproximação inicial para o zero. Neste exemplo, escolhermos $x_0 = 0.5$ como aproximação inicial

```
>> fzero('x^2-exp(-5*x)+sin(x)',0.5)
```

Como exemplo, vamos apresentar um script para usar o método de Newton-Raphson para determinar uma aproximação para a raiz de $f(x) = x^2 - (\exp(-5 * x) + \sin(x))$

```
>> format long %%mais dígitos
>>fun = @(x)( x-(x^2-(exp(-5*x)+sin(x))/(2*x+5*exp(-5*x)+cos(x))))
>>x=1;%aprox inicial
>>for i=0:100,x=feval(fun,x);
 end;
x

x =
    0.49373535126018 % resposta
```

16 Interpolação

O comando **interp1(x,y,xi,metodo)**, é usado para interpolação unidimensional, onde os vetores x e y são os pontos a serem interpolados, e xi são os pontos de avaliação, $f(yi) = xi$, o método é opcional.

O problema de interpolação bidimensional pode ser formulado como segue: dado uma grade de pontos (x_k, y_l) e um conjunto de números z_{kl} associados, determinar uma função $z = f(x, y)$ que interpola os dados, isto é, $f(x_k, y_l) = z_{kl}$, para todos os valores de k, l. A grade de pontos deve ser armazenada monotonicamente, i.e., $x_1 < x_2 < \ldots < x_m$ e analogamente na ordenada y. A função MATLAB **zi = interp2(x, y, z, xi, yi, 'metodo')** gera uma interpolante bidimensional.

Existem seis tipos do parâmetro 'metodo' e é opcional:

'nearest' - interpolação "nearest neighbor´´, localmente constante.

'linear' - interpolação bilinear,

'cubic' - interpolação bicúbica,
'spline' - interpolação por spline

```
Tente os exemplos:
%%%bilinear
[x, y] = meshgrid(-1:.25:1);
z = sin(x.^2 + y.^2);
[xi, yi] =
meshgrid(-1:.05:1);
zi = interp2(x, y, z, xi, yi, 'linear');

surf(xi, yi, zi), title('Interpolação bilinear de  sin(x^2 +...
y^2)')

%%bicubica
[x, y] = meshgrid(-1:.25:1);
z = sin(x.^2 + y.^2);
[xi, yi] =
meshgrid(-1:.05:1);

zi = interp2(x, y, z, xi, yi, 'cubic');

surf(xi, yi, zi), title('Interpolação bicubica de sin(x^2 + y^2)')
```

Vejamos um exemplo eo gráfico gerado por ele em 6

```
[x, y] = meshgrid(-1:.25:1);
z = exp(-x.^2 - y.^2);

[xi, yi] = meshgrid(-1:.05:1);

zi = interp2(x, y, z, xi, yi, 'cubic');
surf(xi, yi, zi),
title('Interpolação bicubica')
```

Otimização está relacionado a zeros de funções, por isso vale a pena lembrar dois comandos para equações não-lineares e otimização:

Comando	Função do Comando
fmin	Minimiza função de uma variável
fzero	Encontra zeros de função de uma variável
fminbnd	Minimiza uma função de uma variável com restrição de fronteira
fminsearch	Minimiza uma função de várias variáveis

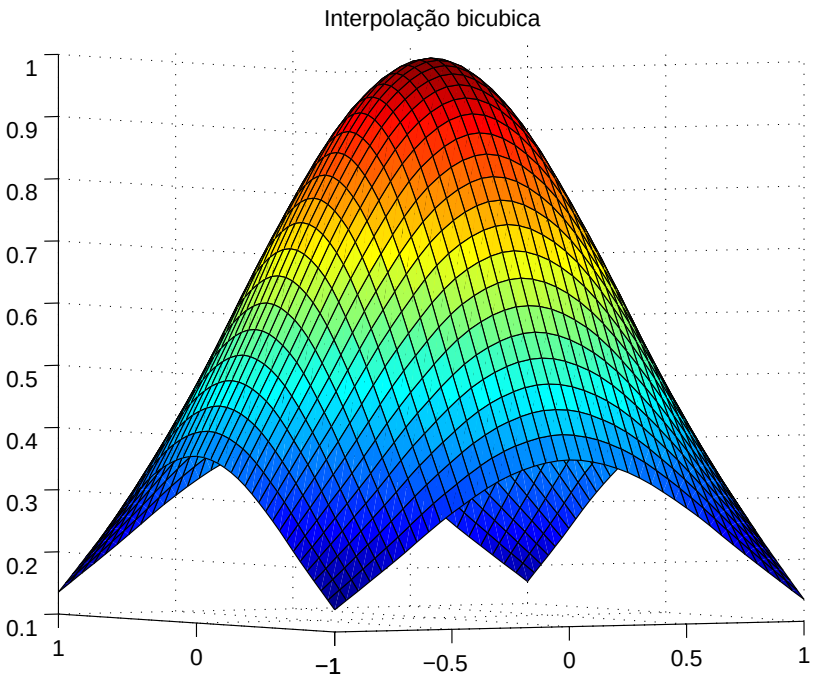


Figura 6: Interpolação bicúbica

A determinação do mínimo da função $fun(x)$ no intervalo de $[a,b]$ é obtido da seguinte maneira,

```
>> xm = fmin('fun',a,b)
>> x = a:0.01:b
>> plot(x, fun(x), xm, ym, 'o')
```

Vejamos alguns exemplos. Primeiro crie as funções myfun2.m e myfun3.m, por exemplo dadas por

```
%% função myfun2
 >>myfun2 = @(x) (x - a)^2;
```

Agora vamos usar o comando **fminbnd**

```
>> a = 1.5; % define o primeiro parâmetro
      x = fminbnd(@(x) myfun2(x,a),0,1)
      %invoca o a função de forma anônima

x =
     1
```

Agora vamos maximizar a função $myfun3(x,y)$. O comando exige um candidato inicial, vamos tomar $x = -1.2$ e $y = 1$, assim temos

```
>> myfun3 = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
>> [x,fval] = fminsearch(myfun3,[-1.2, 1])

x =
    1.0000    1.0000


fval =
  8.1777e-010
```

17 Calculando integrais numericamente

• Para calcular uma aproximação da integral usando a regra dos trapézios:

```
>> x=0:.1:1;
>> y=exp(-x.^2);
>> trapz(x,y)
ans =
    0.7462
```

• O comando **quad('fc', a, b, tol)** retorna uma aproximação integral da função **fc** no intervalo [a,b] usando a regra de Simpson. A fc é o nome de uma função pré-definida ou um arquivo .m correspondente à função. O parâmetro tol refere-se a tolerânica desejada.

Com o comando **dblquad('integr2', a,b,c,d)** calculamos a integral dupla da função **integr2** no retãngulo $[a,b] \times [c,d]$

```
dblquad('integr2', a,b,c,d)  %integral dupla de integr2 no retangulo
```

18 EDOs

Embora MatLab seja primeiramente um pacote numérico ele pode também resolver simbolicamente EDO's para isto usamos o comando dsolve( ). Consideremos como exemplo, a EDO dada por $y'(x) = -xy$, a solução explícita é obtida por

```
>>dsolve('Dy=-y*x','x')
```

Para um problema de valor inicial

$$y'(x) = -xy, y(1) = 1.$$

```
>>eq1 = 'Dy = -y*x'
>>inic = 'y(1)=1'
>>dsolve(eq1,inic,'x')
```

O MATLAB tem os métodos de Runge-Kutta-Felberg de ordem 2(3) e ordem 4(5), respectivamente, as funções **ode23** e **ode45**, para resolver numericamente equações diferenciais ordinárias.

Como exemplo, seja o PVI

$$y' = -2xy + 1, y(0) = 1$$

use os comandos abaixo para determinar a solução numérica

```
>> xspan = [0 5];
>>y0 = 1;
>> [x,y] = ode45(@(x,y)-2*x*y+1,xspan, y0)
>>plot(x,y,'o ') % para plorar
```

Podemos resolver a EDO simbolicamente, veja mais um exemplo. Obter a solução do PVI

$$\begin{cases} y'(t) = 4t - 2ty(t) \\ y(0) = 1, t \in [0,1]. \end{cases}$$

```
>>dsolve('Dy=4*t-2*t*y','y(0)=1')

ans =

2-exp(-t^2)
```

Como outro exemplo, vamos resolver a EDO

$$\begin{cases} y'(x) = -y^3 + x, \ x \in [0, 0.5] \\ y(0) = 1 \end{cases}$$

usando RK2 e RK4. do modo antigo. Primeiramente, vamos escrever uma function, chamada edo1.m correspondente à função. isto pode ser feito como

```
function f=edo1(x,y)
    f=-y^3+x;
end
```

Agora vamos chamar o MATLAB para resolver, usando RK23 com o comando

```
xx=[0,.5]; [x,y]=ode23('edo1',x,1); plot(x,y,'*')
```

e usando RK45 com o comando

```
xx=[0,.5]; [x,y]=ode45('edo1',x,1); plot(x,y,'*')
```

Para resolver um sistema de EDOs de primeira ordem, como por exemplo,

$$\begin{cases} y_1'(x) = y_2(x) \\ y_2' = 2(1 - y_1^2)y_2 + y_1, \\ y_2(0) = 2, \ y_1(0) = 0 \end{cases}$$

no intervalo $[0, 10]$. Como antes vamos escrever uma function, chamada edo2.m correspondente à função:

```
function f=edo2(x,y)
    f=[y(2); 2*(1-y(1)^2)*y(2)-y(1)];
end
```

Vamos chamar o MATLAB para resolver o sistema com o comando

```
>> xx=[0,10]; %intervalo
y0=[2,0]; %condição inicial
[x,y]=ode45('edo2',xx,y0);
plot(x,y(:,1),x,y(:,2),'.') %plotando a solução e sua derivada
xlabel{'Tempo,x')
 title('Solução do sistema')
```

Veja a tabela 10 com o alguns comandos para EDOs.

ode23	Resolve equação diferencial
ode45	Resolve equação diferencial, método RK45

Tabela 10: Tabela com comandos para EDOS

19 Usando Toolboxes

Toolboxes são ferramentas do MATLAB desenvolvidas para resolver um problemas específicos. Existem cerca de 75 toolboxes no MATLAB, cada uma delas especialmente criada para resolver um problema. Consulte o help para aprender sobre os toolboxes. a presença dos toolboxes pode ser verificada clicando no **start** do MATLAB.

19.1 Symbolic Math Toolbox

Vamos aprender um pouco sobre o toolbox de Matemática simbólica. O toolbox de matemática simbólica do MATLAB é o núcleo de matemática simbólica do Maple. Após a instalação do MATLAB, o Symbolic Math Toolbox fica disponível de forma trasnparente, como qualquer função do núcleo do MATLAB. Digite o comando abaixo para a versão

```
>> ver symbolic
```

Digite **help sym** para aprender mais no help mais sobre o toolbox de matemática simbólica. Para calcularmos uma integral simbolicamente, podemos fazer de duas formas, nos dois casos o retorno é um objeto simbólico.

```
>> int('x')
%%%% ou
>> syms x
>> int(x)
```

Faça o seguinte exemplo.

```
>> syms x y
>>A=[sin(x) y^3; cos(y) x^2]
>> det(A) %% det é um comando do Maple
>>B=inv(A) %% inv é um comando do Maple
>> C=A*B
>> simplify(C) %% simplify é um comando do Maple
```

A função **sym** também define expressões,

```
>> eq1=sym('a*x+b');
>> eq2=sym('a*x^2+b*x+c');
>> eq3=sym('a*x^3+b*x^2+c*x+d');
>> x=solve(eq1) %% solve é um comando do Maple
>> y=solve(eq2)
>> z=solve(eq3)
```

O comando do Maple **subs** permite calcular o valor numérico de uma expressão simbólica

```
>> syms t
>> v=[sin(t) t, t^2 t*cos(t)];
>> w=subs(v,t,2)
w=
    0.9093    2.0000    4.0000   -0.8323
```

Uma vez que definimos que a variável x é uma variável simbólica, podemos definir expressões que envolvam esta variável. Faça o seguinte exemplo de **plot** com objetos simbólicos e veja o gráfico na figura 7

```
%%exemplo1
>> syms x
f=-x^2*log(x); subplot(1,2,1) ezplot(f) subplot(1,2,2)
ezplot(int(f))
%% exemplo 2
>> syms A B C x
>> solve(A*x^2+B*x+C)
```

Tente os comandos .

```
>> diff(x^2*cos(x))
>> diff(diff(x^2*cos(x)))
>> diff(x^2-y^2+x*y,x)
>> diff(x^2-y^2+x*y,y)
>> diff(diff(x^2-y^2+x*y,x),y)
```

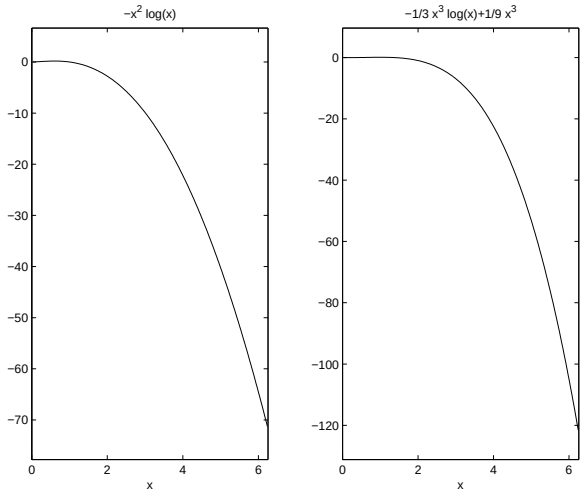


Figura 7: Gráfico com Symbolic

```
>> f=x^2*cos(y)
>> J=jacobian([f f],[x y])
>>dsolve('Dy=cos(x)+x')
```

A função **ezplot('funcao simbolica')** toma uma equação e plota como uma função de x. A sintaxe é **ezplot('funcao simbolica',[xmin xmax]')**. Veja o exemplo

```
ezplot('x^2-x',[-5 5]')
```

Mais exemplos. Defina a function

```
  function z=fun(x)
z=x^2+x+sin(x)}
```

Vamos derivar e integral simbolicamente. Veja os comandos

```
>> syms x
>> fun(x)

z =

x^2+x+sin(x)


>> diff(fun(x),x)

z =

x^2+x+sin(x)
```

```
ans =

2*x+1+cos(x)


>> int(fun(x),x)

z =

x^2+x+sin(x)



ans =

1/3*x^3+1/2*x^2-cos(x)
```

O comando **int(f,t,a,b)** retorna a integral definida f com relação a variável t variando de a a até b: $\int_a^b f(t)dt)$.

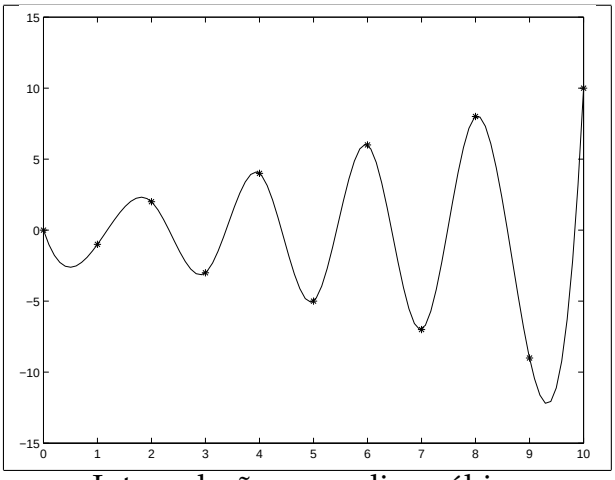
```
int(-2*x/(1+x^2)^2,x,0,1)                %retorna

ans =

-1/2
```

19.2 Splines

A interpolação por meio de splines cúbicas consiste na colagem de pedaços de polinômios de grau 3 com a condição de continuidade da segunda derivada nos nós que definem a malha usada. O comando **spline** do MATLAB realiza esta interpolação.

O Toolbox Spline contém programas básicos de splines. Para mais detalhes digite **help splines**. Vejamos um exemplo e o gráfico produzido na figura.

```
>>x=0:10;
>>y= x.*cos(pi*x);
>>xi=linspace(0,10);
>>yi=spline(x,y,xi); %% equivalentemente ou yi=interp(x,y,xi,'spline')
>>plot(x,y,'o',xi,yi)
```



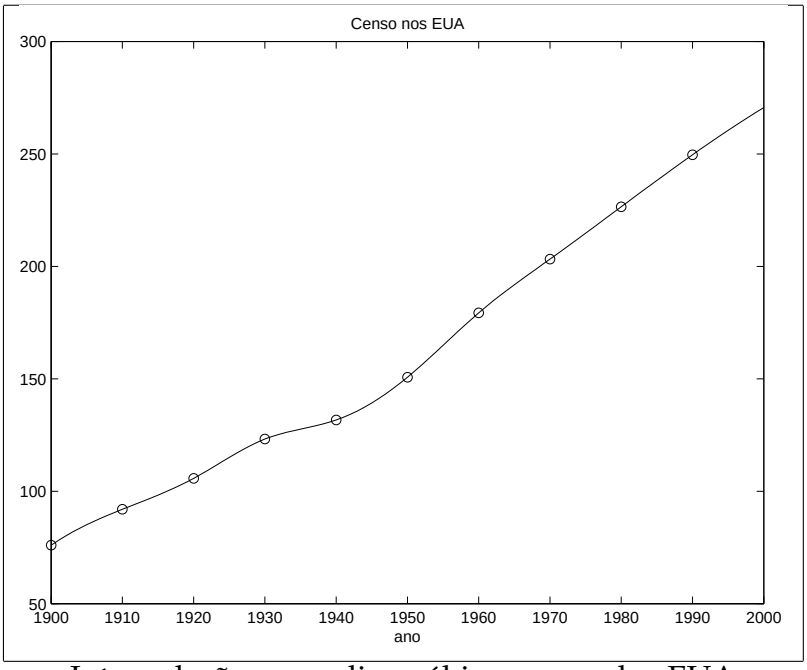
Interpolação por spline cúbica

Outro exemplo é dado por t os anos de 1900 a 1990 e p a população dos EUA nesses anos. Deseja-se estimar a população do EUA no ano de 2000. Veja os comandos,

```
>>t = 1900:10:1990;
>>p = [ 75.995  91.972  105.711  123.203  131.669
... 150.697 179.323  203.212  226.505  249.633 ];
>>spline(t,p,2000)

ans =   270.6060
```

Os seguinte comandos interpolam os dados com uma spline cúbica, avaliada em cada ano de 1900 a 2000 e plota o resultado.

```
 >>x = 1900:1:2000;
 >>y = spline(t,p,x);
 >>plot(t,p,'o',x,y)
 >>title('Censo nos EUA')
>> xlabel('ano')
```



Interpolação por spline cúbica: censo dos EUA

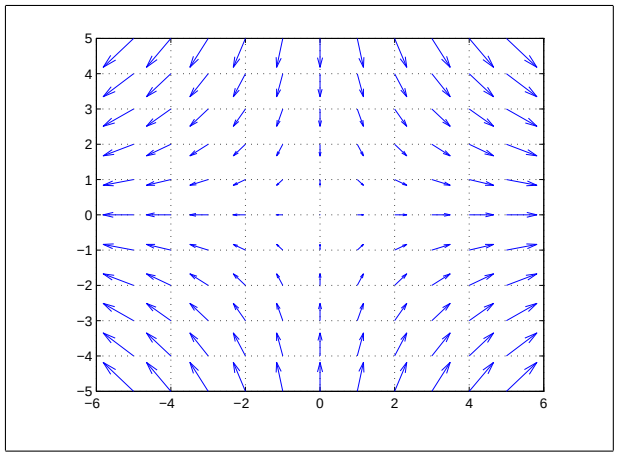
20 Campos Vetoriais

Com o MATLAB você pode plotar um campo de vetores, por exemplo

```
>>x=-5:5
>>y=-5:5
>>[X,Y]=meshgrid(x,y)
```

O comando **quiver(X,Y,U,V)** plota o campo de velocidades (U, V) nos pontos (X, Y). Como exemplo, tomemos o campo vetorial $(3x, -3y)$.

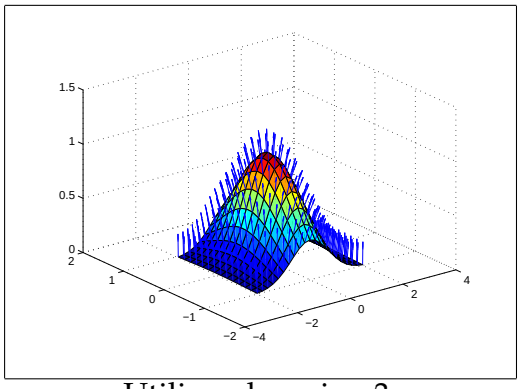
```
>>U=3*x
>>V=-3*y
>>quiver(X,Y,U,V)[X,Y]
>>grid
```



Campo vetorial $(3x, -3y)$.

Digite **help quiver** para mais informações o quiver e descubra o **quiver3**. Veja o exemplo de utilização,

```
>> z =  exp(-x.^2 - y.^2);
>> [u,v,w] = surfnorm(x,y,z);
>> quiver3(x,y,z,u,v,w);
hold on, surf(x,y,z), hold off
```



Utilizando quiver3.

No exemplo a seguir, plotamos curvas de nível de uma superfície e em seguida o campo gradiente.

Referências

[1] Using MATLAB, Version 7, The MathWorks, Inc., 2015.

Índice Remissivo