1 - INTRODUÇÃO

1.1 - O que é o MATLAB?

MATLAB (MATrix LABoratory) é um "software" interativo de alta performance voltado para o cálculo numérico e científico. O MATLAB integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar onde problemas e soluções podem ser expressos como eles são escritos na matemática ou na forma de uma linguagem de programação.

No MATLAB o elemento básico de informação é uma matriz que não requer dimensionamento permitindo a resolução de muitos problemas numéricos em apenas uma fração do tempo que se gastaria para escrever um programa semelhante em linguagem Fortran, Basic ou C. A potencialidade do software está muito além do que será mostrado nesta apostila que vem para introduzir o MATLAB como uma linguagem de programação capaz de implementar um algoritmo qualquer.

1.2 - Inicializando o programa

No Gerenciador de Programas do Microsoft Windows deve-se abrir o grupo de programas do MATLAB for Windows, que contém o ícone do aplicativo MATLAB. Um duplo clique no ícone MATLAB carrega o aplicativo MATLAB.

Quando o MATLAB é carregado, duas janelas são exibidas: a Janela de Comando (Command Windows) e Janela Gráfica (Graphic Windows). A Janela de Comando é ativada quando se inicializa o MATLAB, e o "prompt" padrão (>>) é exibido na tela.

1.3 - Declarações, Constantes e Variáveis

As constantes e expressões são essenciais no uso do MATLAB. Uma constante numérica no MATLAB é formada por uma seqüência de dígitos que pode estar ou não precedida de um sinal positivo (+) ou um negativo (-) e pode conter um ponto decimal (.). Esta seqüência pode terminar ou não por uma das letras e, E, d ou D seguida de outra

seqüência de dígitos precedida ou não de um sinal positivo (+) ou um negativo (-). Esta segunda seqüência é a potência de 10 a qual a primeira seqüência deve ser multiplicada. Por exemplo, 1.23e-1 significa 0,123.

As declarações no MATLAB são freqüentemente da forma

```
>> variável = expressão
```

ou simplesmente

```
>> expressão
```

Uma variável pode também receber uma cadeia de caracteres ao invés de um número ou expressão. Estes caracteres devem ser manipulados como vetores linha, sendo que a cadeia deve estar delimitada por apóstrofes para ser atribuída à uma variável literal. Por exemplo

```
>>  x = ' Este ', y= ' teste. '
x =
      Este
y =
      teste.

>> z = [x ' e um ' y]
z =
     Este e um teste.
```

Já as expressões usadas no MATLAB são interpretadas e avaliadas pelo sistema sendo compostas de operadores e outros caracteres especiais, de funções e dos nomes das variáveis. A avaliação das expressões produz matrizes, que são então mostradas na tela e atribuídas às variáveis para uso futuro. Se o nome da variável e o sinal de igualdade “=” são omitidos, a variável com o nome **ans**, que representa a palavra “answer” (resposta), é automaticamente criada. Por exemplo, digite a expressão

```
>> 1900/81
```

que produz

```
ans=
      23.4568
```

Se o último caractere da declaração é um ponto e vírgula, “;”, a impressão na tela é suprimida, mas a tarefa é realizada. Esse procedimento é usado em arquivos com extensão ".m" e em situações onde o resultado é uma matriz de grandes dimensões e temos interesse em apenas alguns dos seus elementos.

Se a expressão é tão grande que não cabe em apenas uma linha, pode-se continuar a expressão na próxima linha usando um espaço em branco e três pontos, **”...”**, ao final das linhas incompletas. Por exemplo,

```
>> s = l - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
>>     - 1/8 + 1/9 - l/10 + 1/11 - 1/12 + 1/13;
```

calcula o resultado da série, atribuindo a somatória à variável **s**, mas não imprime o resultado na tela. Note que os espaços em branco entre os sinais “=”, “+” e “-“ são opcionais, mas o espaço em branco entre “1/7” e “...” é obrigatório.

As variáveis e funções podem ser formadas por um conjunto de letras, ou por um conjunto de letras e números, onde somente os primeiros 19 caracteres do conjunto são identificados. O MATLAB faz distinção entre letras maiúsculas e minúsculas, assim **a** e **A** não são as mesmas variáveis. Todas as funções devem ser escritas em letras minúsculas: **inv(A)** calcula a inversa de **A**, mas **INV(A)** é uma função indefinida.

2 – COMANDOS DE AUXÍLIO E FORMATAÇÃO

2.1 - Recuperando e Editando linhas de comando

As teclas com setas podem ser usadas para listar comandos dados anteriormente, para sua execução novamente ou sua reedição. Um exemplo é o caso de uma função com nome errado como a seguir:

```
>> sqt (169)
```

No caso para calcular a raiz quadrada o comando certo é **sqrt**, o MATLAB então responde com uma mensagem de erro:

```
??? Undefined funcion or variable sqt.
```

Ao invés de reescrever a linha inteira, simplesmente pressione a tecla -. O comando errado retorna, e você pode, então, mover o cursor para trás usando a tecla ¬ ou o ponto de inserção com o "mouse" ao lugar apropriado para inserir a letra "r". Então, o comando retorna a resposta correta

```
>> sqrt (169)
ans =
      13
```

Além dessas teclas com "setas", você pode utilizar as teclas abaixo para editar suas linhas de comando.

Ctrl ¬	Move uma palavra para a esquerda
Ctrl ®	Move uma palavra para a direita
Home	Move para o começo da linha
End	Move para o final da linha
Del	Apaga um caracter a direita
Backspace	Apaga um caracter a esquerda

2.2- Formatação

O formato numérico exibido na tela pode ser modificado utilizando-se o comando **format**, que afeta somente o modo como as

matrizes são mostradas, e não como elas são computadas ou salvas (o MATLAB efetua todas operações em dupla precisão).

Se todos os elementos das matrizes são inteiros exatos, a matrizes é mostrada em um formato sem qualquer ponto decimal. Por exemplo,

```
>> x = [-1 0 1]
```

sempre resulta em

```
x =
     -1 0 1
```

Se pelo menos um dos elementos da matriz não é inteiro exato, existem várias possibilidades de formatar a saída. O formato "default", chamado de formato **short**, mostra aproximadamente 5 dígitos significativos ou usam notação científica. Por exemplo, a expressão

```
>> x = [4/3 1.2345e-6]
```

é mostrada , para cada formato usado, da seguinte maneira:

Format short	1.3333 0.0000
Format short e	1.3333e+000 1.2345e-006
Format long	1.33333333333333 0.000000123450000
Format long e	1.333333333333333e+000 1.234500000000000e-006
Format hex	3ff5555555555555 3eb4b6231abfd271
Format rat	4/3 1/810045
Format bank	1.33 0.00
Format +	++

Com o formato **short** e **long**, se o maior elemento da matriz é maior que 1000 ou menor que 0.001, um fator de escala comum é aplicado para que a matriz completa seja mostrada. Por exemplo,

```
>> x = 1.e20*x
```

resultado da multiplicação será mostrado na tela.

```
x =
    l.0e+20 * 1.3333 0.0000
```

O formato + é uma maneira compacta de mostrar matrizes de grandes dimensões. Os símbolos "+", "-", e "espaço em branco" são mostrados, respectivamente para elementos positivos, elementos negativos e zeros.

2.3 - Utilizando o HELP

Como já foi dito no início do texto, esta apostila é apenas introdutória mas você poderá se aprofundar mais sobre qualquer tópico utilizando as vantagens do comando help. O help é um comando de ajuda que fornece informações sobre a maior parte dos tópicos. Digitando

```
>> help
```

obtêm-se uma lista desses tópicos disponíveis:

```
HELP topics:

c:\matlab          -Establish MATLAB session parameters.
Matlab\general       -General purpose commands.
Matlab\ops         -Operators and special characters.
Matlab\lang          -Language constructs and debugging.
Matlab\elmat         -Elementary matrices and matrix
manipulation.
Matlab\specmat                    -Specialized matrices.
Matlab\elfun         -Elementary math functions.
Matlab\specfun                    -Specialized math
functions.
Matlab\matfun                     -Matrix functions -
numerical linear algebra.
Matlab\datafun                     -Data analysis and
Fourier transform functions.
Matlab\polyfun                    -Polynomial and
```

interpolation functions.
Matlab\funfun -Function functions: nonlinear numerical methods.
Matlab\sparfun -Sparse matrix functions.
Matlab\plotxy -Two dimensional graphics.
Matlab\piotxyz -Three dimensional graphics.
Matlab\graphics -General purpose graphics functions.
Matlab\color -Color control and lighting model functions.
Matlab\sounds -Sound processing functions.
Matlab\strfun -Character string functions.
Matlab\iofun -Low-level file I/0 functions.
Matlab\demos -Demonstrations and samples.
simulink\simulink -SIMULINK model analysis.
simulink\blocks -SIMULINK block library.
simulink\simdemos -SIMULINK demonstrations and samples.
nnet\exampies - Neural Network Toolbox examples.
nnet\nnet - Neural Network Toolbox.

For more help on directory/topic, type 'help topic".

Para obter informações sobre um tópico específico, digite **help tópico.** Por exemplo,

```
>> help plotxy
```

que fornece uma lista de todos os comandos relacionados com gráficos bidimensionais:

Two dimensional graphics.

Elementary X-Y graphs.

plot	- Linear plot.
loglog	- Log-log scafe plot.
semilogx	- Semi-log scale plot.
semilogy	- Semi-log scale plot.
fill	- Draw filled 2-D polygons.

Specialized X-Y graphs.

polar	- Polar coordinate plot.
bar	- Bar graph.
stem	- Discrete sequence or & "stemm" plot.
stairs	- Stairstep plot.
errorbar	- Error bar plot.
hist	- Histogram plot.
rose	- Angle histogram plot.
compass	- Compass plot.
feather	- Feather plot.
fplot	- Plot function
comet	- Comet-like trajectory.

Graph annotation.

title	- Graph title.
xlabel	- X-axis label.
ylabel	- Y-axis label.
text	- Text annotation.
gtext	- Mouse placement of text.
grid	- Grid lines.

See also PLOTXYZ, GRAPHICS

Finalmente, para obter informações sobre um comando específico, por exemplo, **title**, digite:

```
>> help title
```

e informações mais detalhadas sobre este comando serão exibidas:

```
TITLE Titles for 2-D and 3-D plots.
      TITLE ('text') adds text at the top of the current axis.

      See also XLABEL, YLABEL, ZLABEL, TEXT.
```

Note que no exemplo mostrado para adicionar o título em um gráfico, **TITLE ('TEXT')** está escrito em letras maiúsculas somente para destacar. Deve-se lembrar que todos os comandos do MATLAB devem ser escritos em letras minúsculas, portanto, para adicionar o texto "*Título do Gráfico*" em um gráfico, digite:

```
>> title ('Título do Gráfico')
```

3 – MATRIZES

3.1 - Matrizes Simples

O MATLAB trabalha essencialmente com um tipo de objeto, uma matriz numérica retangular podendo conter elementos complexos (deve-se lembrar que um escalar é uma matriz de dimensão l x l e que um vetor é uma matriz que possui somente uma linha ou uma coluna).

As matrizes podem ser introduzidas no MATLAB por diferentes caminhos:

- digitadas na Janela de Comando (lista explícita de elementos),
- geradas por comandos e funções,
- criadas em arquivos ".m",
- carregadas a partir de um arquivo de dados externo.

O método mais fácil de entrar com pequenas matrizes no MATLAB é usando uma lista explícita. Os elementos de cada linha da matriz são separados por espaços em branco ou vírgulas e as colunas separadas por ponto e vírgula, colocando-se colchetes em volta do grupo de elementos que formam a matriz. Por exemplo, entre com a expressão

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

Pressionando <enter> o MATLAB mostra o resultado

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

A matriz **A** é salva na memória RAM do computador, ficando armazenada para uso posterior.

As matrizes podem, também, ser introduzidas linha a linha, o que é indicado para matrizes de grande dimensão. Por exemplo:

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

Outra maneira para entrar com matrizes no MATLAB é através de um arquivo no formato texto com extensão ".m". Por exemplo, se um arquivo chamado "gera.m" contém estas três linhas de texto,

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

então, a expressão "gera" lê o arquivo e introduz a matriz A.

```
>>gera
```

O comando **load** pode ler matrizes geradas pelo MATLAB e armazenadas em arquivos binários ou matrizes geradas por outros programas armazenadas em arquivos ASCII.

3.2 - Elementos das Matrizes

Os elementos das matrizes podem ser qualquer expressão do MATLAB, por exemplo.

```
>> x = [-1.3 sqrt(2) ((1+2+3)*4/5)^2]
```

resulta em

```
x =
    -l.3000    1.4142    23.0400
```

Um elemento individual da matriz pode ser reverenciado com índice entre parênteses. Continuando o exemplo,

```
>> x(6) = abs(x(l))
```

produz:

```
x =
   -1.3000   1.4142   23.0400   0   0   1.3000
```

Note que a dimensão do vetor x é aumentada automaticamente para acomodar o novo elemento e que os elementos do intervalo indefinido são estabelecidos como zero.

Grandes matrizes podem ser construídas a partir de pequenas matrizes. Por exemplo, pode-se anexar outra linha na matriz **A** usando

```
>> r= [ l0 11 12];
>> A= [A;r]
```

que resulta em

```
A=
    1   2    3
```

```
 4  5  6
 7  8  9
10 11 12
```

Note que o vetor **r** não foi listado porque ao seu final foi acrescentado “;”.
Pequenas, matrizes podem ser extraídas de grandes matrizes usando “:”.
Por exemplo:

```
>> A = A(1:3,:);
```

seleciona as três primeiras linhas e todas as colunas da matriz **A** atual, modificando-a para sua forma original.

3.3 – Operações com Matrizes

As operações com matrizes no MATLAB são as seguintes:

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

A seguir cada uma dessas operações é mostrada com mais detalhe.

3.3.1 - Transposta

O caractere apóstrofo, " ’ " , indica a transposta de uma matriz. A declaração

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

que resulta em

```
A =
    1     2     3
```

```
     4     5     6
     7     8     0

B =
     l     4     7
     2     5     8
     3     6     0
```

e

```
>> x = [-1 O 2]'
```

produz

```
x =
    -1
     0
     2
```

Se **Z** é uma matriz complexa, **Z'** será o conjugado complexo composto. Para obter simplesmente a transposta de **Z** deve-se usar **Z'**, como mostra o exemplo

```
>> Z = [1 2; 3 4] + [5 6; 7 8]*i
>> Z1 = Z'
>> Z2 = Z'
```

que resulta em

```
Z =
   1.0000 + 5.0000i   2.0000 + 6.0000i
   6.0000 + 7.0000i   4.0000 + 8.0000i
Z1 =
   1.0000 - 5.0000i   3.0000 - 7.0000i
   2.0000 - 6.0000i   4.0000 - 8.0000i
Z2 =
   1.0000 + 5.0000i   3.0000 + 7.0000i
   2.0000 + 6.0000i   4.0000 + 8.0000i
```

3.3.2 - Adição e Subtração

A adição e subtração de matrizes são indicadas, respectivamente, por "+" e "-". As operações são definidas somente se as matrizes as

mesmas dimensões. Por exemplo, a soma com as matrizes mostradas acima, **A + x**, não é correta porque **A** é 3x3 e **x** é 3x1. Porém,

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

é aceitável, e o resultado da soma é

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

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

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

resulta em

```
y =
 -2
 -1
```

3.3.3- Multiplicação

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

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

é aceitável, e resulta em

```
ans =
     4
```

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

```
>> x*y'
ans =
```

```
     2     l     -l
     0     0      0
    -4    -2      2

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

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

>> b = **A'x**

que resulta em

```
b =
     5
     8
    -7
```

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

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

3.3.4 - Divisão

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

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

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

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

resulta em

```
z =
     -1
      0
      2
```

3.3.5 - Exponenciação

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

```
>> A^3
     ans =

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

3.4 -Manipulação de Vetores e Matrizes

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

3.4.1 - Gerando Vetores

Os dois pontos, “ : ”, é um caractere importante no MATLAB. A declaração

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

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

```
x =
    l  2  3  4  5
```

Outros incrementos, diferentes de um, podem ser usados.

```
>> y = 0 : pi/4 : pi
```

que resulta em

```
y =
        0.0000  0.7854  1.5708  2.3562  3.1416
```

Incrementos negativos também são possíveis.

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

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

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

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

3.4.2 - Elementos das Matrizes

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

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

a declaração

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

resulta em

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

Um subscrito pode ser um vetor. Se X e V são vetores, então **X(V)** é **[X(V(1)), X(V(2)), ..., X(V(n))]**. Para as matrizes, os subscritos vetores permitem o acesso à submatrizes contínuas e descontínuas. Por exemplo, suponha que A é uma matriz 10x10.

```
      A =
92  99  11  18  15  67  74  51  58  40
98  80  17  14  16  73  55  57  64  41
14  81  88  20  22  54  56  63  70  47
85  87  19  21  13  60  62  69  71  28
86  93  25  12  19  61  68  75  52  34
17  24  76  83  90  42  49  26  33  65
23  15  82  89  91  48  30  32  39  66
79  16  13  95  97  29  31  38  45  72
10  12  94  96  78  35  37  44  46  53
11  18  10  77  84  36  43  50  27  59
          0
```

então

```
>> A(1:5,3)
ans =
      11
      17
      88
      19
      25
```

especifica uma submatriz 5x1, ou vetor coluna, que consiste dos cinco primeiros elementos da terceira coluna da matriz **A**. Analogamente,

```
>> A(1:5,7:10)
ans =
      74    51    58    40
      55    57    64    41
      56    63    70    47
      62    69    71    28
      68    75    52    34
```

é uma submatriz 5x4, consiste das primeiras cinco linhas e as últimas quatro colunas.

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

```
>> A(1:2:5,:)
ans =
      92    99    11    18    15    67    74    51    58    40
      14    81    88    20    22    54    56    63    70    47
      86    93    25    12    19    61    68    75    52    34
```

é uma submatriz 3x10 que consiste da primeira, terceira e quinta linhas e todas colunas da matriz **A**.

Muitos efeitos sofisticados são obtidos usando submatrizes em ambos os lados das declarações. Por exemplo, sendo **B** uma matriz 10x10 unitária,

```
>> B = ones (10)
B =
      1  1  1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1  1  1  1
      1  1  1  1  1  1  1  1  1  1
```

```
1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1
```

a declaração,

```
>> B(1:2:7,6:l0) = A(S:-1:2,1:5)
```

produz

```
1  1  1 1  1  86  93  25  12  19
1  1  1 1  1   1
                   1   1   1   1
1  1  1 1  1  85  87  19  21  13
1  1  1 1  1   1
                   1   1   1   1
1  1  1 1  1  14  81  88  20  22
1  1  1 1  1   1
                   1   1   1   1
1  1  1 1  1  98  80  17  14  16
1  1  1 1  1   1
                   1   1   1   1
1  1  1 1  1   1
                   1   1   1   1
1  1  1 1  1   1
                   1   1   1   1
```

4– CONJUNTOS

4.1 – Operações com conjuntos

O termo *operações com conjuntos* é usado quando as operações aritméticas são realizadas entre os elementos que ocupam as mesmas posições em cada matriz (elemento por elemento). As operações com conjuntos são feitas como as operações usuais, utilizando-se dos

mesmos caracteres (“*****”, ”**/**”, “****”, “^” e “ **‘** “) precedidos por um ponto “**.**” (“**.***”, ”**./**”, “**.**”, “.**^**” e “ **.‘** “).

4.1.1 - Adição e Subtração

Para a adição e a subtração, a operação com conjuntos e as operações com matrizes são as mesmas. Deste modo os caracteres "+" e "-" podem ser utilizados tanto para operações com matrizes como para operações com conjuntos.

4.1.2 - Multiplicação e Divisão

A multiplicação de conjuntos é indicada por “**.***”. Se **A** e **B** são matrizes com as mesmas dimensões, então **A.*B** indica um conjunto cujos elementos são simplesmente o produto dos elementos individuais de **A** e **B**. Por exemplo, se

```
>> x = [1 2 3];    y = [4 5 6];
```

então,

```
>> z = x .* y
```

resulta em

```
z=
     4 10 18
```

As expressões **A./B** e **A.\B** formam um conjunto cujos elementos são simplesmente os quocientes dos elementos individuais de **A** e **B.** Assim,

```
>> z = x .\ y
```

resulta em

```
z =
   4.0000    2.5000    2.0000
```

4.1.3 - Exponenciação

A exponenciação de conjuntos é indicada por "**.**^". A seguir são mostrados alguns exemplos usando os vetores **x** e **y.** A expressão

```
>> z = x .^ y
```

resulta em

```
z =
    l          32          729
```

A exponenciação pode usar um escalar.

```
>> z = x..^2
z =
    l      4      9
```

Ou, a base pode ser um escalar.

```
>> z = 2.^[x y]
z =
    2   4   8   16   32   64
```

4.1.4 - Operações Comparativas

Estes são os seis operadores usados para comparação de duas matrizes com as mesmas dimensões:

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

A comparação é feita entre os pares de elementos correspondentes e o resultado é uma matriz composta dos números

um e zero, com um representando **VERDADEIRO** e zero, **FALSO**. Por exemplo**,**

```
>> 2 + 2 ~= 4
ans =
        0
```

Pode-se usar, também os operadores lógicos **&** (e) e **|** (ou). Por exemplo,

```
>> 1= = 1 & 4 = = 3
ans =
        0

>> 1 = = 1 | 4 = = 3
ans =
        1
```

5 - POLINÔMIOS

5.1 – Operações com polinômios

Um polinômio no MATLAB é definido como

$$P(x) = c1\ x^{n} + c2\ x^{n-1} + \ldots + cn\ x + cn+1$$

onde os coeficientes do polinômio são definidos como elementos de um vetor qualquer, no caso representado pelo vetor c.

Estudaremos agora, uma série de funções que facilitam as operações com polinômios.

5.1.1 - Adição e Subtração

No MATLAB não existe uma função que faça especialmente a soma de polinômios, mas isso pode ser feito através da soma vetorial. Portanto somando dois
polinômios $p1(x) = 3x^2 + 2x + 2$ e $p2(x) = 5x^2 - x - 1$, resulta em

```
>> p1 = [3 2 2];
>> p2 = [5 -1 -1];
>> soma = p1 +p2
soma =
          8     1     1
```

subtraindo,

```
>> sub = p1 -p2
sub =
        -2     3     3
```

Obs: Caso os vetores possuam graus diferentes ou não possuam uma ou mais constante c (c=0) então o vetor correspondente ao de menor grau deve ser preenchido com zeros de modo que os vetores fiquem com o mesmo tamanho.

Exemplo:

```
>> a = [5 4 3 2 1];

>> b = [0 –4  2 1 1];
soma = a+b

soma =
    5     0     5     3     2
```

5.1.2 - Multiplicação

Entrando com os vetores p1 e p2 que contém os coeficientes dos polinômios p1(x) e p2(x), respectivamente, então os coeficientes do polinômio resultante da multiplicação de p1(x) por p2(x) pode ser obtido pela função conv(p1,p2). Para o caso de $p1(x) = 3x^2 - 5x + 4$ e $p2(x) = 2x - 1$ temos

```
>> p1 = [3 -5 4];
>> p2 = [0 2 -1];
>> c = conv(p1,p2)
c =
    0     6   -13    13    -4
```

que é o polinômio $c(x) = 6x^3 - 13x^2 + 13x - 4$.

5.1.3 - Divisão

O comando que faz a divisão entre polinômios cujos os elementos são os coeficientes dos vetores p1 e p2, é o comando [q , r] = deconv (p1,p2), onde q é o vetor dos coeficientes do polinômio quociente e o vetor r contém os coeficientes do polinômio obtido pelo resto da divisão, ou seja, p1 = conv (q,b) + r. Veejamos o exemplo:

```
>> p1 = [2 –3 4 –5 6];
>> p2 = [1 –3 1];
>> [q, r] = deconv (p1,p2)
q =
     2   3  11

r =
     0    0    0   25  -5
```

lembrando que q é o quociente e r é o resto.

5.1.4 - Derivação

A função polyder é responsável pela derivação polinomial. Assim para obter a primeira e segunda derivada do polinômio $P(x) = x^3 + 3x^2 + 1$, fazemos

```
>> p = [1 3 0 1];
>> p1= polyder(p)
p1 =
    3     6     0                          % Derivada primeira

>> p2= polyder(p1)
p2 =
    6     6                                % Derivada segunda
```

ou seja $P'(x) = 3x^2 + 6x$ e $P''(x) = 6x + 6$.

5.2 - Calculando raízes com a função roots

Utilizando um vetor linha c como coeficientes de um polinômio podemos calcular as raízes deste com a função roots(c), que fornece um vetor coluna as n raízes de $P(x) = 0$. Utilizando como exemplo o polinômio $P(x) = x^2 + 2x + 1$ temos

```
>> c = [1 2 1];
>> r =roots(c)
r =
   -1
   -1
```

6 – GRÁFICOS

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

6.1 - Gráficos Bidimensionais

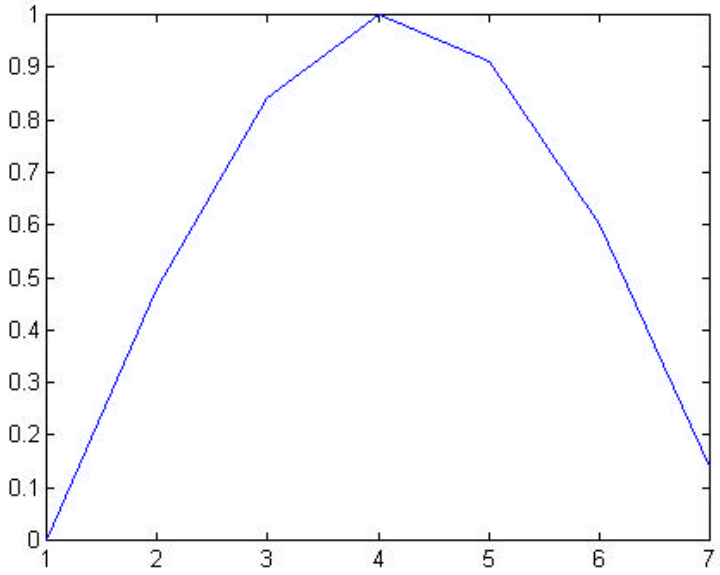
Estes são os comandos para plotar gráficos bidimensionais:

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

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

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

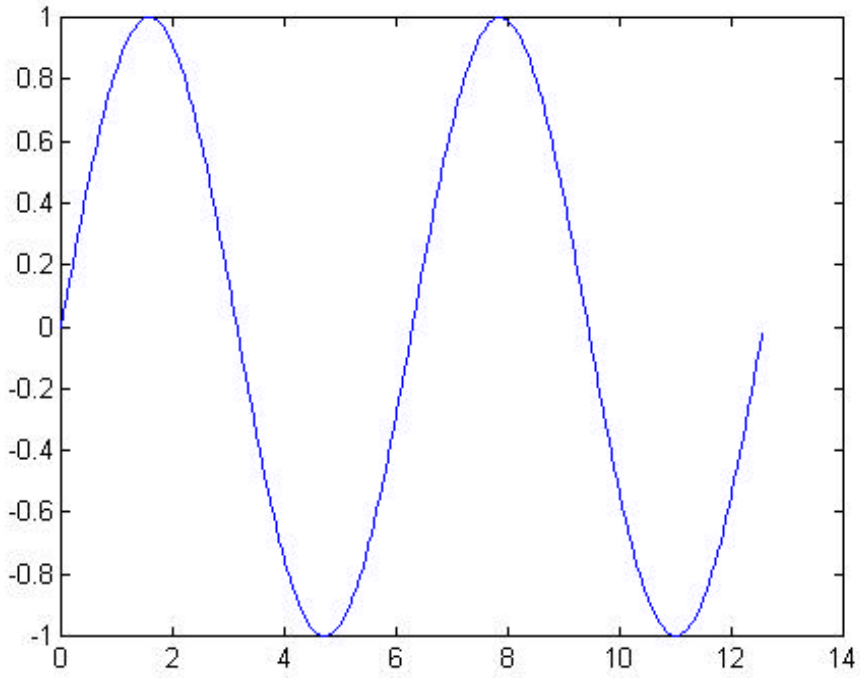
e o resultado é mostrado na Janela Gráfica:



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

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

resulta em



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

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

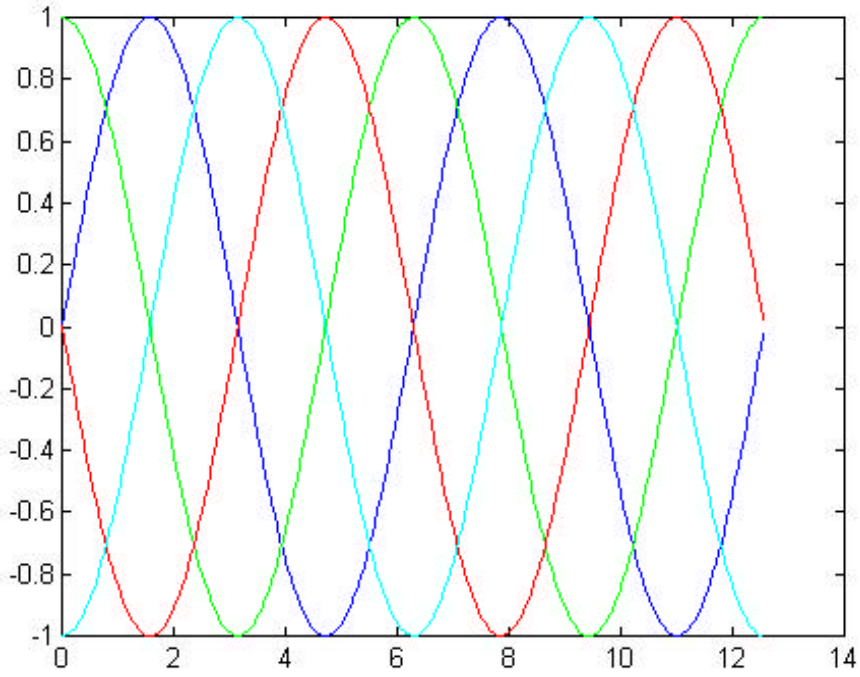
- Se **X** é uma matriz e **Y** é um vetor, **plot(X,Y)** plota sucessivamente as linhas ou colunas de **X** versus o vetor **Y**.

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

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

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

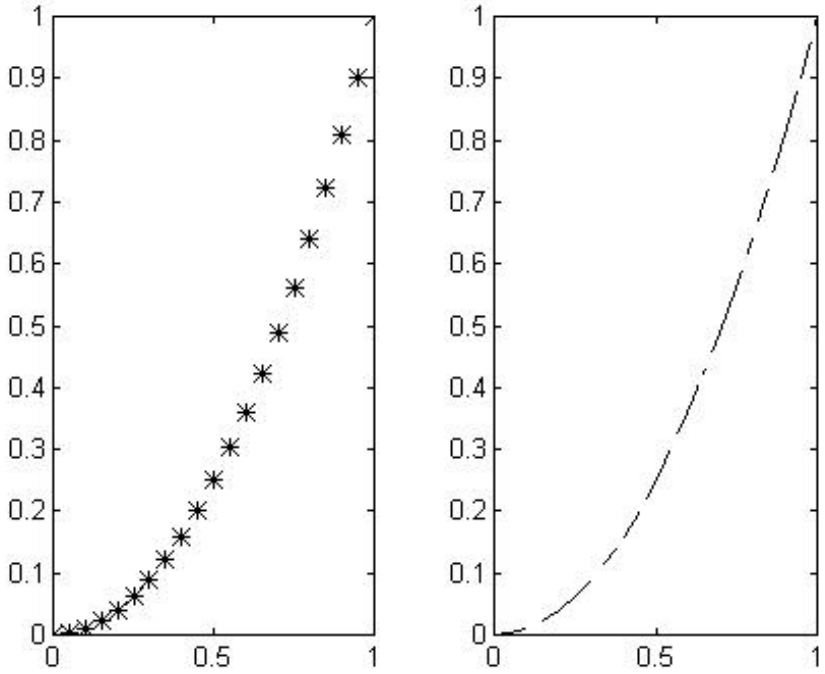
```
>> plot(t, sin(t), t, cos(t), t, sin(t + pi), t, cos(t + pi))
```



6.2 - Estilos de Linha e Símbolo

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

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



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

Símbolo	Cor	Símbolo	Marcador	Símbolo	Tipo de Linha

y	amarelo	.	Ponto	-	Linha contínua
m	Lilás	*	Estrela	- -	Linha tracejada
c	azul claro	°	Círculo	-.	Traços e pontos
r	vermelho	+	+	:	Linha pontilhada
g	verde	x	x		
b	azul escuro	s	Quadrado		
w	branco	d	Losango		
k	preto	∨	Triângulo p/ baixo		
		∧	Triângulo p/ cima		
		<	Triângulo p/ esquerda		
		>	Triângulo p/ direita		
		p	Pentagrama		
		h	hexagrama		

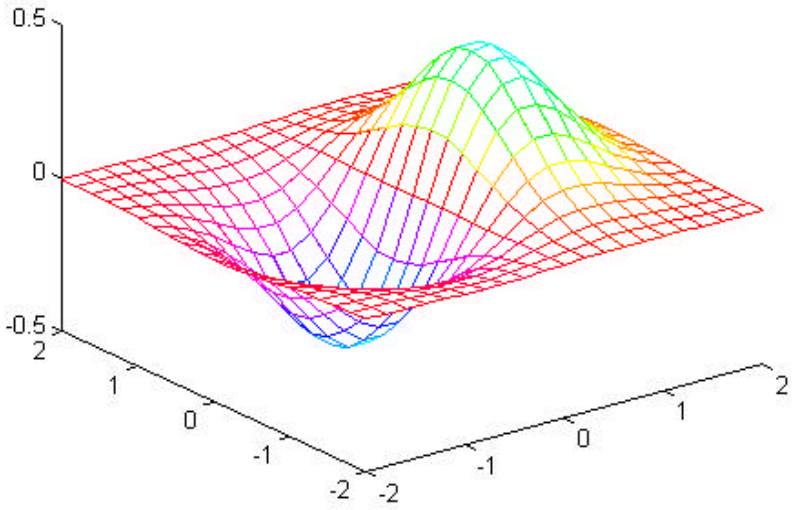
6.3 - Plotando Gráficos Tridimensionais e Contornos

Estes são alguns comandos para plotar gráficos tridimensionais e contornos.

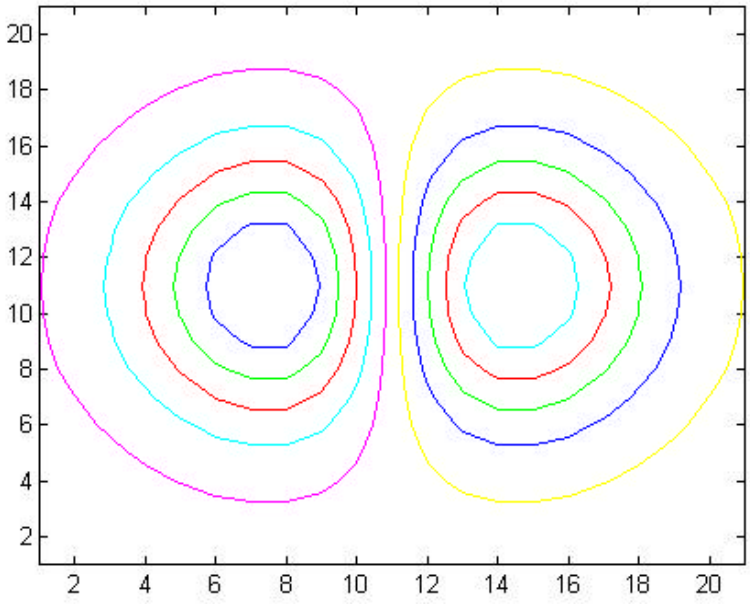
Plot3	Plotar em espaço 3D.
fill3	Desenhar polígono 3D.
Comet3	Plotar em 3D com trajetória de cometa.
Contour	Plotar contorno 2D.
contour 3	Plotar contorno 3D.
Clabel	Plotar contorno com valores.
Quiver	Plotar gradiente.
Mesh	Plotar malha 3D.
Bar3	Plotar em forma de barra 3D.
Meshc	Combinação mesh/contour.
Surf	Plotar superfície 3D.
Surfc	Combinação surf/contour.
Surfil	Plotar superfície 3D com iluminação.
Slice	Plot visualização volumétrica.
Cylinder	Gerar cilindro.
Sphere	Gerar esfera.

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

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



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



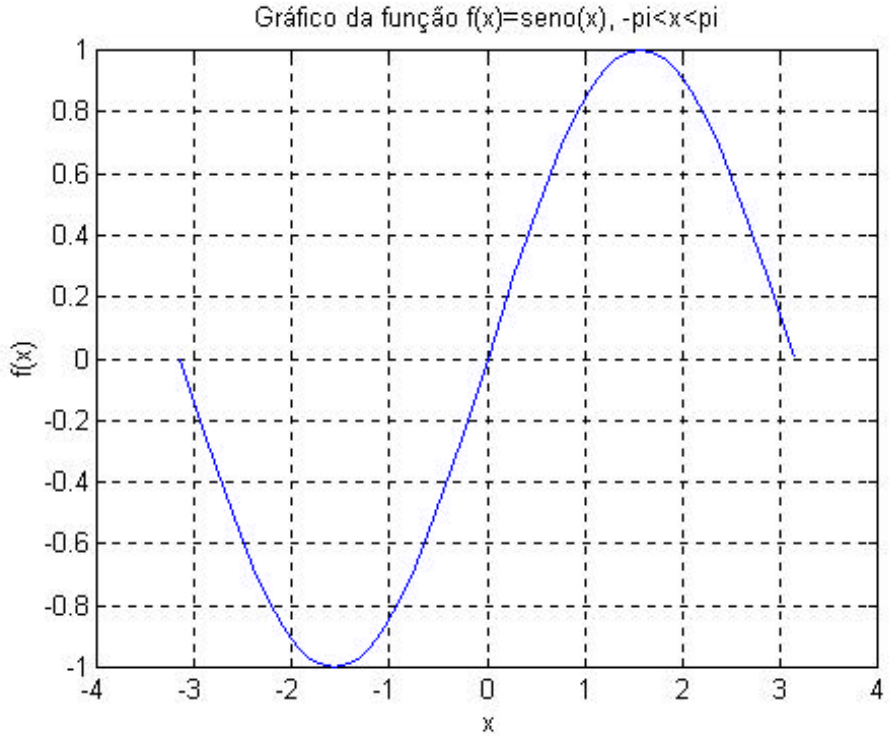
6.4 - Anotações no Gráfico

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

title	Título do gráfico.
xlabel	Título do eixo-X.
ylabel	Título do eixo-Y.
zlabel	Título do eixo-Z.
text	Inserir anotação no gráfico.
gtext	Inserir anotação com o mouse.
ginput	Selecionar pontos no gráfico com o mouse.
grid	Linhas de grade.
axis	Define os valores mínimos e máximos dos eixos

Por exemplo:

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



6.5 - Manipulação de Gráficos

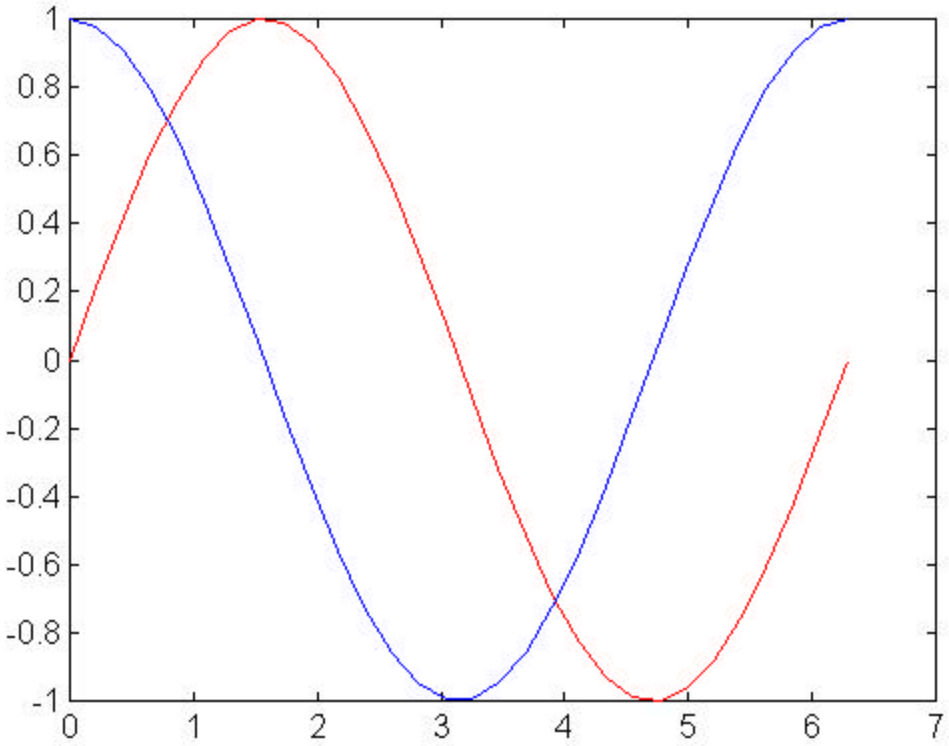
Para adicionar curvas a um gráfico já existente utiliza-se o comando *hold*. Depois de executado o comando *hold on* as curvas já existente no gráfico não são removidas a cada comando plot, mas sim acrescenta as novas curvas ao gráfico. O comando *hold off* libera a janela de figuras atual para novos gráficos.

Por exemplo:

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

Para incluir a curva do cos(x):

```
>> hold on
>> plot(x, z, 'b');
>> hold off
```



7 - FUNÇÕES

.

· Solução de equações diferenciais.

7.2 - Equações Diferenciais

Os comandos do MATLAB para resolver equações diferenciais ordinárias são:

ode23	Resolver equação diferencial. método

	baixa ordem.
ode23p	Resolver e plotar soluções.
ode45	Resolver equação diferencial. Método para alta ordem

Considere a equação diferencial de segunda ordem chamada de *Equação de Van der Pol*

$$x + (x^2 - 1) \cdot x + x = 0$$

Pode-se reescrever esta equação como um sistema acoplado de equações diferenciais de primeira ordem

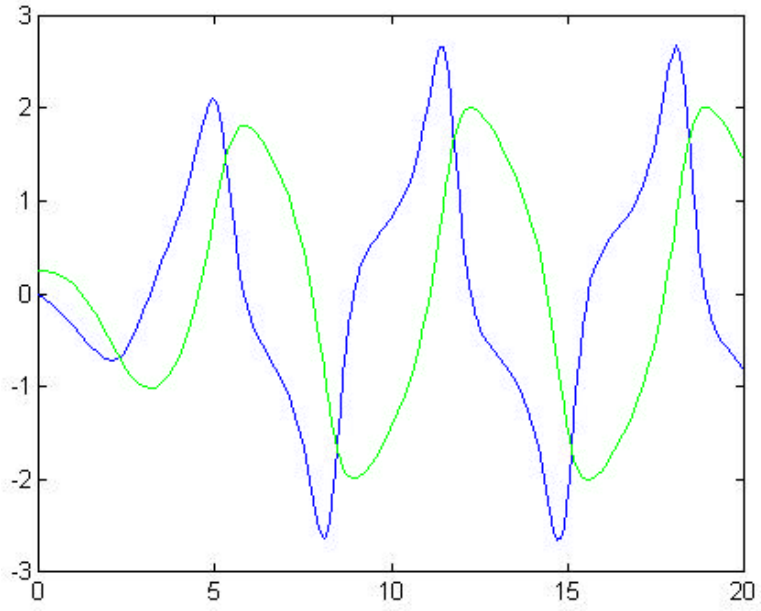
$$x_1 = x_1 \cdot (1-x_2^2) - x_2$$
$$x_2 = x_1$$

O primeiro passo para simular esse sistema é criar um arquivo ".m" contendo essas equações diferenciais. Por exemplo, o arquivo **volpol.m**:

```
function xdot=volpol(t,x)

xdot=[0 0]

xdot(l)=x(l).*(1- x(2).^2) - x(2);

xdot(2)=x(l);
```

Para simular a equação diferencial no intervalo $0 \leq t \leq 20$, utiliza-se o comando **ode23**

```
>> t0 = 0; tf  = 20;
>> x0 = [0 0.25];
>>  [t,x] = ode23('volpol', t0, tf, x0);
>>  plot(t,x)
```



9 – INTERPOLAÇÃO E AJUSTE DE CURVAS

9.1 - Interpolação unidimensional

Os valores contidos em um vetor qualquer podem ser interpolados através da função **interp1** , cuja sintaxe é

<resultado> = interp1 (x,y,z,'<método>')

onde <método> especifica o método a ser utilizado na interpolação, sendo

'linear'	Interpolação linear
'cubic'	Interpolação por polinômios cúbicos
'spline'	Interpolação por splines cúbicos

Caso não seja especificado o método a ser utilizado o MATLAB assumirá como uma interpolação linear. Os valores do vetor x devem

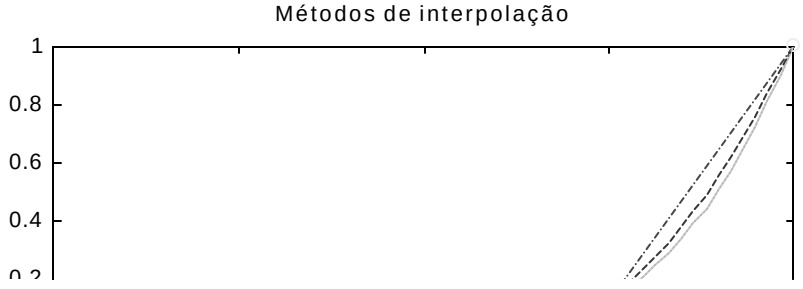
crescer ou decrescer monotonicamente e, além disso, no caso de interpolação cúbica os valores de x devem ser igualmente espaçados. Utilizando como exemplo a função $y = x^3$ para x maior igual a –1 ou menor igual a 1.

```
>> x = linspace(-1,1,5);          % abcissas dos pontos
>> y = x.^3;                      % ordenadas
>> interp1(x,y,0.4,'linear')
ans =
    0.1000
>> interp1(x,y,0.4,'cubic')
ans =
    0.0520
>> interp1(x,y,0.4,'spline')
ans =
    0.0640
```

Considerando que o valor exato é 0.4^3, ou seja, 0.064, o método de interpolação com splines cúbicos produziu o melhor resultado para este caso.

Para visualizar os métodos fazemos:

```
>> x1 = linspace (-1,1,60);                    % abcissas para a interpolação
>> y1 = x1.^3;                                 % ordenadas da curva real
>> cubica = interp1(x,y,x1,'cubic');           % interpolação cúbica
>> spline = interp1(x,y,x1,'spline');          % interpolação por splines
>> plot(x,y,'o',x,y,'-.',x1,y1,'-',x1,cubica,'--',x1,spline,' : ')
>> xlabel('x'); ylabel('y'); title('Métodos de interpolação')
```



Com :

o	pontos a ser interpolados
.-.-	Interpolação linear
- -	Interpolação por polinômios cúbicos
....	Interpolação por splines cúbicos

E tendo como a função $y = x^3$ a linha contínua.

9.2 - Ajuste Polinomial (Mínimos quadrados)

O comando c = polyfit(x,y,n) atribui ao vetor c os valores dos coeficientes do polinômio de mínimos quadrados de grau n que melhor ajusta aos pontos (xi,yi) definidos nos vetores x e y . O polinômio é na forma

$$Pn(x) = c1x^n + c2x^{n-1} + c3x^{n-2} + \ldots \ldots + cnx + cn+1$$

Utilizaremos agora um exemplo do uso da função polyfit, com dados relacionados a temperatura em graus Celsius e Farenheit obtidos a partir de um termômetro com pouca exatidão

```
>> c = [15 17 18 20 23 25 26 28 30 32];
>> f = [40 41 42 43 45 46 46 48 49 50];
```

A reta de mínimos quadrados é obtida por $f^{o} = P1(c) = b1c + b2$

então fazemos

```
>> b = polyfit(c,f,1)
b =
    0.5925   31.1345
```

Mesmo sendo os dados obtidos através de um termômetro com pouca exatidão o resultado foi muito próximo ao valor exato obtido pela equação F =5/9C + 32. Os pontos P(ci) do polinômio de regressão de grau 1 em relação a c são obtidos pelo comando

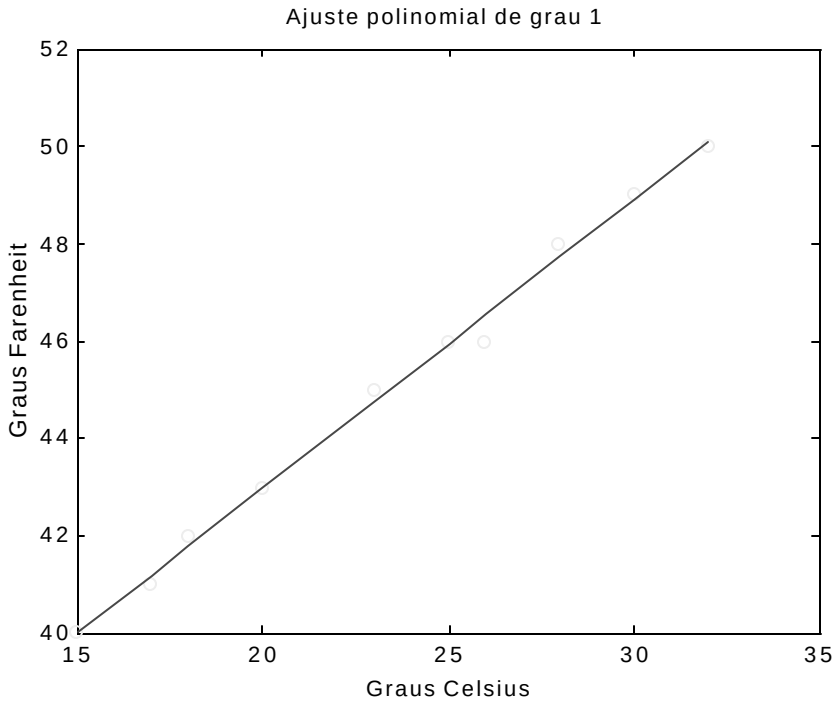
```
>> p = polyval(b,c);
```

e os resíduos do ajuste ri = fi – P(ci)

```
>> [c; f ; p ; f-p]'
ans =
   15.0000   40.0000   40.0226   -0.0226
   17.0000   41.0000   41.2077   -0.2077
   18.0000   42.0000   41.8003    0.1997
   20.0000   43.0000   42.9854    0.0146
   23.0000   45.0000   44.7630    0.2370
   25.0000   46.0000   45.9481    0.0519
   26.0000   46.0000   46.5406   -0.5406
   28.0000   48.0000   47.7257    0.2743
   30.0000   49.0000   48.9108    0.0892
   32.0000   50.0000   50.0959   -0.0959
```

Visualizando o ajuste:

```
» plot(c,f,'o',c,p)
» xlabel('Graus Celsius'); ylabel('Graus Farenheit');
» title('Ajuste polinomial de grau 1');
```



10 – ESTRUTURAS CONDICIONAIS

Os comandos que controlam o fluxo especificam a ordem em que a computação é feita. No MATLAB estes comandos são semelhantes aos usados na linguagem C, mas com uma estrutura diferente.

10.1 - Laço for

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

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

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

- A primeira parte (**i=1**) é realizada uma vez, antes do laço ser inicializado.

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

- A terceira parte acontece quando a condição se torna falsa e o laço termina.

O comando **end** é usado como limite inferior do corpo do laço.

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

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

10.2 - Laço while

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

```
a = l; b = 15;
while a<b,
```

```
    clc
    a = a+l
    b = b-l
    pause(l)
  end
  disp('fim do loop')
```

A condição **a<b** é testada. Se ela for verdadeira o corpo do laço, será executado.

Então a condição é retestada, e se verdadeira o corpo será executado novamente. Quando o teste se tornar falso o laço terminará, e a execução continuará no comando que segue o laço após o **end**.

10.3 - Declarações if e break

A seguir, é apresentado um exemplo do uso da declaração **if** no MATLAB.

```
for i = l:5,
    for j = l:5,
        if i = = j
            A(i,j) = 2;
        elseif abs(i-j) = = 1
            A(i,j) = -1;
        else
            A(i,j) = 0;
        end
    end
end
A
```

Os valores de **i e j** variam de 1 a 5, varrendo toda a matriz **A.** Se **(if) i** for igual a **j**, **A(i,j)=2**, ou se **(elseif)** o valor absoluto de **i-j** for igual a 1, **A(i,j)=-1**, ou **(else) A(i,j)=0**, se nenhuma das condições anteriores forem satisfeitas.

É conveniente, às vezes, controlarmos a saída deu m laço de outro modo além do teste, no início ou no fim do mesmo. O comando **break** permite uma saída antecipada de um **for** ou **while.** Um comando **break** faz com que o laço mais interno seja terminado imediatamente. Por exemplo,

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

11 - "m – files" (ARQUIVOS ".m")

Os comandos do MATLAB são normalmente digitados na Janela de Comando, onde uma única linha de comando é introduzida e

processada imediatamente. O MATLAB é também capaz de executar seqüências de comandos armazenadas em arquivos.

Os arquivos que contêm as declarações do MATLAB são chamadas arquivos ".m", e consistem de uma seqüência de comandos normais do MATLAB, possibilitando incluir outros arquivos ".m" escritos no formato texto (ASCII).

Para editar um arquivo texto na Janela de Comando do MATLAB selecione **New M-File** para criar um novo arquivo ou **Open M-File** para editar um arquivo já existente, a partir do menu **File**. Os arquivos podem, também, ser editados fora do MATLAB utilizando qualquer editor de texto.

Existem alguns comandos e declarações especiais para serem usados nos arquivos, por exemplo:

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

O caractere % é usado para inserir um comentário no texto, o comando **clear** apaga todos os dados da memória, o comando **input** é usado quando se deseja entrar com um dado a partir da Janela de Comando, **pause** provoca uma pausa na execução do arquivo até que

qualquer tecla seja digitada, **clc** limpa a Janela de Comando, **figure(1)** mostra a Janela Gráfica número 1 e **close** fecha todas as Janelas Gráficas.

12 - OPERAÇÕES COM O DISCO

Os comandos **load** e **save** são usados, respectivamente, para importar dados do disco (rígido ou flexível) para a área de trabalho do MATLAB e exportar dados da área de trabalho para o disco. Outras operações com o disco podem ser efetuadas, como executar programas externos, trocar o diretório de trabalho, listagem do diretório, e serão detalhadas a seguir.

12.1 - Manipulação do Disco

Os comandos **cd**, **dir**, **delete**, **type** e **what** do MATLAB são usados da mesma maneira que os comandos similares do sistema operacional.

Cd	troca o diretório de trabalho atual
dir	lista o conteúdo do diretório atual
delete	exclui arquivo
type	mostra o conteúdo do arquivo texto
what	lista arquivos ".m", ".mat" e ".mex".

Para maiores detalhes sobre estes comandos utilize o **help**.

12.2 - Executando Programas Externos

O caracter ponto de exclamação, **!**, é um desvio e indica que o restante da linha será um comando a ser executado pelo sistema operacional. Este procedimento vem sendo historicamente utilizado em todos as versões do MATLAB como "prompt" para indicar a execução de um colando do DOS, sendo muito útil nas versões que

usavam somente o DOS. No ambiente Windows, entretanto, este comando é desnecessário, mas foi mantido nas versões do MATLAB para Windows.

Para entrar com o caracter de desvio no "prompt" do MATLAB, deve-se coloca-lo no Início do comando do DOS ou Windows que se deseja executar. Por exemplo, para carregar um aplicativo como o programa **Notepad** do Windows (Bloco de Notas), sem sair do MATLAB, entre com

```
>> ! Notepad
```

Uma nova janela é aberta, o Notepad é carregado, podendo ser utilizado da maneira usual.

Pode-se usar, também, qualquer comando implícito do DOS, por exemplo:

copy, fomat, ren, mkdjr, rmdir, ...

12.3 - Importando e Exportando Dados

Os dados contidos na Área de Trabalho do MATLAB podem ser armazenados em arquivos, no formato texto ou binário, utilizando o comando **save**. Existem diversas maneiras de utilizar este comando. Por exemplo. para armazenar as variáveis X, Y e Z pode-se fazer:

save	salva os dados no arquivos binário "matlab.mat".
save X	salva a matriz X no arquivo o binário "x.mat".
save arql X Y Z	salva as matrizes X, Y e Z no arquivo binário "arq1.mat".
save arq2.sai X Y Z -ascii	salva as matrizes X., Y e Z no arquivo texto "arq2.sai" com 8 dígitos.
Save arq3.sai X Y Z -ascii -double	salva as matrizes X., Y e Z no arquivo texto "arq3.sai" com 16 dígitos.

Os dados obtidos por outros programas podem ser importados pelo MATLAB, desde que estes dados sejam gravados em disco no formato apropriado. Se os dados são armazenados no formato ASCII, e no caso de matrizes, com colunas separadas por espaços e cada linha da matriz em uma linha do texto, o comando **load** pode ser usado.

Por exemplo suponha que um programa em linguagem C, depois de executado, monta o arquivo "teste.sai" (mostrado abaixo) que contém uma matriz.

```
1.0000  2.0000  3.0000

4.0000  5.0000  6.0000

7.0000  8.0000  9.0000
```

Executando o comando:

```
>> load teste.sai
```

o MATLAB importa a matriz, que passa a se chamar **teste**:

```
>> teste
teste =
     1  2  3
     4  5  6
     7  8  9
```

Obviamente, o MATLAB pode também importar (através do comando **load**) os dados que foram anteriormente exportados por ele. Por exemplo, para importar as variáveis X, Y e Z, anteriormente exportadas usando o comando **save**, pode-se fazer:

save	load
save X	load x
save arq1 X Y Z	load arq1
save arq2.sai X Y Z -ascii	load arq2.sai
save arq3.sai X Y Z -ascii -	load

double	arq3.sai

Deve-se ressaltar que o comando **save**, quando usado para exportar os dados do MATLAB em formato texto, exporta apenas um bloco contendo todas as variáveis. E quando importamos estes comandos através do comando **load**, apenas uma variável com nome do arquivo é importada. Por exemplo:

```
>> X=rand(3,3)
X =
      0.2190    0.6793    0.5194
      0.0470    0.9347    0.8310
      0.6789    0.3835    0.0346

>> Y = rand(3,3)
Y =
      0.0535   0.0077   0.4175
      0.5297   0.3835   0.6868
      0.6711   0.0668   0.5890

>> save arq2.sai X Y -ascii
>> clear
>> load arq2.sai
>> arq2
arq2 =
   0.2190  0.6793  0.5194
   0.0470   0.9347  0.8310
   0.6789            0.3835       0.0346
   0.0535  0.0077  0.4175
   0.5297  0.3834  0.6868
   0.6711   0.0668  0.5890
```

13 - EXEMPLOS DE EXERCÍCIOS RESOLVIDOS UTILIZANDO "m - files"

Exemplo 1 - Dinâmica de Populações

Os arquivos-m a seguir contém os comandos necessários para solucionar o problema de dinâmica de populações. Para fácil utilização o código do arquivo deve ser gravado no diretório do MATLAB com o nome runpop.m

Primeiramente vamos criar o que chamaremos de “programa principal” (runpop.m),utilizaremos este nome pois este arquivo “chamará” o arquivo subsequente (pop.m) que funciona neste caso como “função”.

Programa principal

```
% leitura dos dados iniciais para um sistema de edo 2x2

%initial = [0 0.25];
clg

%axis;

initial(1) = input('Primeiro dado inicial da presa  ');
initial(2) = input('Segundo dado inicial do predador  ');
% Intervalo de tempo usado [ti,tf]
ti = input('Tempo inicial ');
tf = input('Tempo final ');
% Constantes referentes a presa e ao predador
%k1 = input('Taxa de nascimento da presa  ');
%k2 = input('Taxa de mortandade da  presa  ');
%k3 = input('Taxa de nascimento do predador  ');
%k4 = input('Taxa de mortandade do predador  ');
```

```
k1 = 3.
k2 = 0.002
k3 = 0.0006
k4 = 0.5

[x, num_y] = ode23('pop',ti,tf,initial);

subplot(211), plot(x,num_y(:,1),'- g'),...
title('Populacao da Presa'),xlabel('t'),grid,...
subplot(212),plot(x,num_y(:,2),'- g'),...
title('Populacao do Predador'),xlabel('t'),grid
```

Função pop (Salvar como pop.m)

```
function xf = pop (t,x)

global k1 k2 k3 k4
k1=3.;
k2=0.002;
k3=0.0006;
k4=0.5;

xf(1)= k1*x(1)-k2*x(1)*x(2);
xf(2)= k3*x(1)*x(2)-k4*x(2);
```

Visualizando a solução

Agora que já criamos os arquivos necessários para a solução do problema, é só digitar no prompt do MATLAB

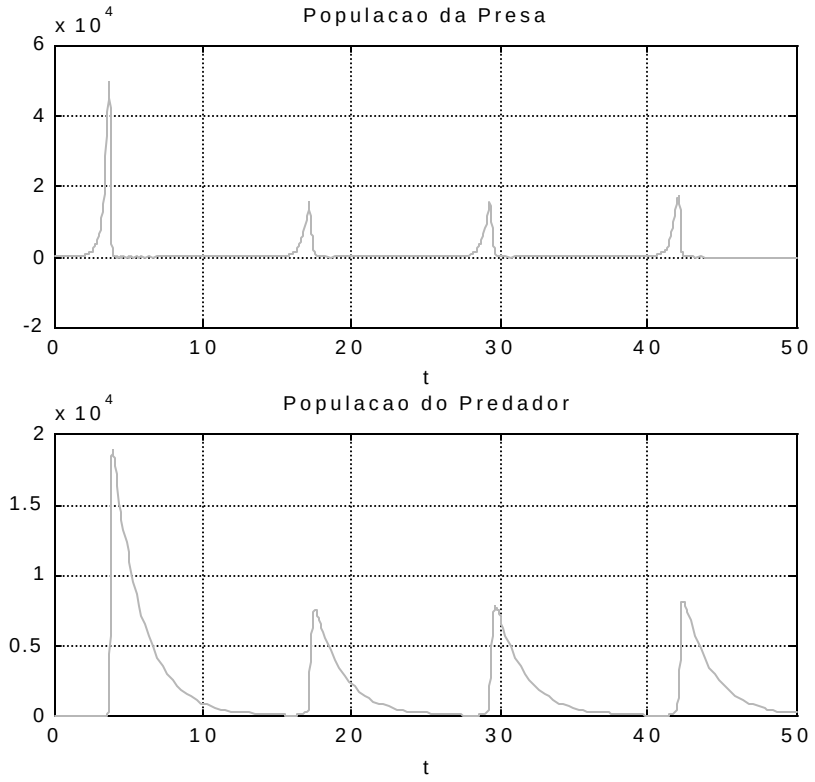
```
>> runpop
```

serão pedidos alguns dados como população inicial de presa , predadores, tempo inicial e final. Para os dados

População inicial da presa = 0,8
População inicial do predador = 0,2
Tempo Inicial = 0

Tempo Final = 50

Obtemos o gráfico ,



Observe a relação entre crescimento da presa em relação ao predator e vice-versa.

Exemplo 2 - Trajetória de uma bola de tênis com " top –spin "

Os arquivos-m a seguir contém os comandos necessários para solucionar e disponibilizar visualmente o problema da trajetória de tênis em diferentes casos.

Veja o código do programa principal que será salvo com o nome "tenis.m" (Preste atenção no diretório aonde está sendo gravado os arquivos , se não quiser ter problemas salve todos os arquivos no diretório do MATLAB).

Programa Principal

```
% Problema para a trajetória de uma bola de tênis viajando no vácuo,
% e no ar na presença de um spin (rotação)
%                          tenisV = caso no vácuo
% As 4 funções          tenisA = caso no ar sem spin
% utilizadas são :      tenisAsp = caso no ar com rotação positiva
%                          tenisAfs = caso no ar com rotação negativa

% Variáveis que serão usadas pelas funções com os campos vetoriais

global g alpha w etha

% Inicializando as variáveis

% Constantes básicas em unidades MKS
clg
g = 9.81; d = 0.063; m = 0.05; rho = 1.29;
alpha = pi*d^2/(8*m)*rho;
etha = 1;
w = 20;

 % Condições iniciais
h = 1; v0 = 25; theta  = pi/180*15;
xin = [0, h, v0*cos(theta), v0*sin(theta)];

 % Tempo de vôo no vácuo
tmaxid = (xin(4) + sqrt(xin(4)^2 + 2*g*xin(2)))/g;

 % solução no vácuo
[tV, xV] = ode23('tenisV',0,tmaxid,xin);

% solução no ar sem spin
```

```
[tA, xA] = ode23('tenisA',0,tmaxid,xin);

% solução com spin
[tAsp, xAsp] = ode23('tenisAsp',0,tmaxid,xin);

% Preparando a saída gráfica do problema
N = max(xV(:,1)); x =0:N/100:N;
axis([0, max(xV(:,1)), 0 , max(xV(:,2))])
hold    % comando que permite sobrepor os três gráficos (plots) seguintes

%1 plot(x, spline(xV(:,1), xV(:,2), x), ':g');
%2 plot(x, spline(xA(:,1), xA(:,2), x), '-- g');
%3 plot(x, spline(xAsp(:,1), xAsp(:,2), x), '-w');

plot(xV(:,1), xV(:,2), ':g');
plot(xA(:,1), xA(:,2), '-- g');
plot(xAsp(:,1), xAsp(:,2), '-w');
```

Função tenisV.m

Função que calcula o campo vetorial para a bola no vácuo.

```
function xdot = tenisV(t,x)
global g
g=9.81;
xdot(1) = x(3);
xdot(2) = x(4);
xdot(3) = 0;
xdot(4) = -g;
```

Função tenisA.m

Função que calcula o campo vetorial para a bola no ar sem rotação.

```
function xdot = tenisA(t,x)
%sem spin a constante de Magnus CM e ZERO.
global g alpha
v = sqrt(x(3)^2+x(4)^2);
xdot(1) = x(3);
xdot(2) = x(4);
xdot(3) = -alpha*0.508*x(3)*v;
xdot(4) = -g -alpha*0.508*x(4)*v;
```

Função tenisAsp.m

Função que calcula o campo vetorial para a bola no ar com rotação.

```
function xdot = tenisAsp(t,x)
global g alpha w etha
v = sqrt(x(3)^2+x(4)^2);

%  cálculo do campo  levando em conta as constantes de arraste CD
%  e a de Magnus CM
aux1 = (0.508 +1/(22.503 + 4.196*(v/w)^0.4))*alpha*v;
aux2 = etha*w/(2.022*w + 0.981*v)*alpha*v;
xdot(1) = x(3);
xdot(2) = x(4);
xdot(3) = -aux1*x(3) + aux2*x(4);
xdot(4) = -g -aux1*x(4) - aux2*x(3);
```

Visualizando a solução

Agora que já criamos os arquivos necessários para a solução do problema, é só digitar no prompt do MATLAB

```
>> tenis
```

Então obteremos o seguinte gráfico que ilustra a trajetória da bola de tênis nos 3 casos que foram estudados:

1- A linha pontilhada indica a trajetória da bola no vácuo.

2- A linha tracejada indica a trajetória da bola no ar com rotação.

3- A linha contínua indica a trajetória da bola no ar com rotação.