Universidade Federal de São João Del Rei - UFSJ
Instituída pela Lei 10.425, de 19/04/2002 - D.O.U. de 22/04/2002
Pró-Reitoria de Ensino de Graduação - PROEN

Apostila de Matlab

Prof. Natã Goulart da Silva
Campus Alto Paraopeba

versão 0.91

Sumário

1 Introdução **3**

2 Tipo de Dados Fundamentais **4**

2.1 Strings 6

3 Comandos de leitura e Escrita **7**

4 Manipulação de Arquivos **8**

5 Operadores **9**

5.1 Operadores Aritméticos 9

5.2 Operadores Relacionais 9

5.3 Operadores Lógicos 10

5.4 Precedência de Operadores 11

6 Comandos Condicionais **12**

6.1 If, If elseif e else 12

6.2 Switch 14

7 Comandos for e while **15**

8 Scripts e Funções **17**

9 Estudos de Funções e Gráficos 2D **19**

9.1 Utilizando ezplot e variáveis simbólicas 21

9.2 Avaliando funções 22

9.3 Obtendo Raízes de Equações e Polinômios 22

10 Considerações sobre performance **23**

11 Alguns Comandos **25**

12 Erros mais comuns **28**

13 Referências **28**

1 Introdução

O software *Matlab*, inicialmente concebido para utilização em cálculos matriciais, possui também características que permitem o desenvolvimento de scripts e de algoritmos como em outras linguagens de programação. Um script é um arquivo que contém um ou mais comandos. Após salvar este arquivo, pode-se executá-lo digitando seu nome no interpretador de comandos do *Matlab*. Como propriedade fundamental, o *Matlab* faz uso das matrizes como estrutura de dados básica. Atualmente, o *Matlab* é um produto comercial extremamente completo utilizado por muitas indústrias de engenharia e por cientistas[1].

Após ser iniciado, o *Matlab* geralmente apresenta uma tela como a da Figura 1. Nesta tela vemos a Janela de Comandos (Command Window) na lateral direita, onde são digitados os comandos que o programa irá interpretar. No canto superior esquerdo, a janela Current Directory apresenta todos os arquivos disponíveis na pasta que o Matlab usa como diretório de trabalho. Esta pasta de trabalho pode ser alterada acionando a caixa que se encontra no centro superior da interface do Matlab, chamada também de Current Directory. Para que um script seja executado pelo *Matlab* é necessário que ele seja salvo na pasta de trabalho atual ou que o caminho onde o script estiver armazenado faça parte do path (caminho) do Matlab. Uma janela com o histórico dos comandos utilizados recentemente é apresentada no canto inferior esquerdo da interface. Pode-se alterar as janelas que são exibidas pelo Matlab, acionando o menu Desktop e marcando a janelas que devem aparecer. Na Figura 2 são apresentadas as janelas selecionadas.

Este documento apresenta informações básicas para a aplicação da ferramenta na disciplina de Cálculo Numérico. Outras informações estão disponíveis através do comando *help* do *Matlab* e no site disponível na segunda referência no final deste documento.

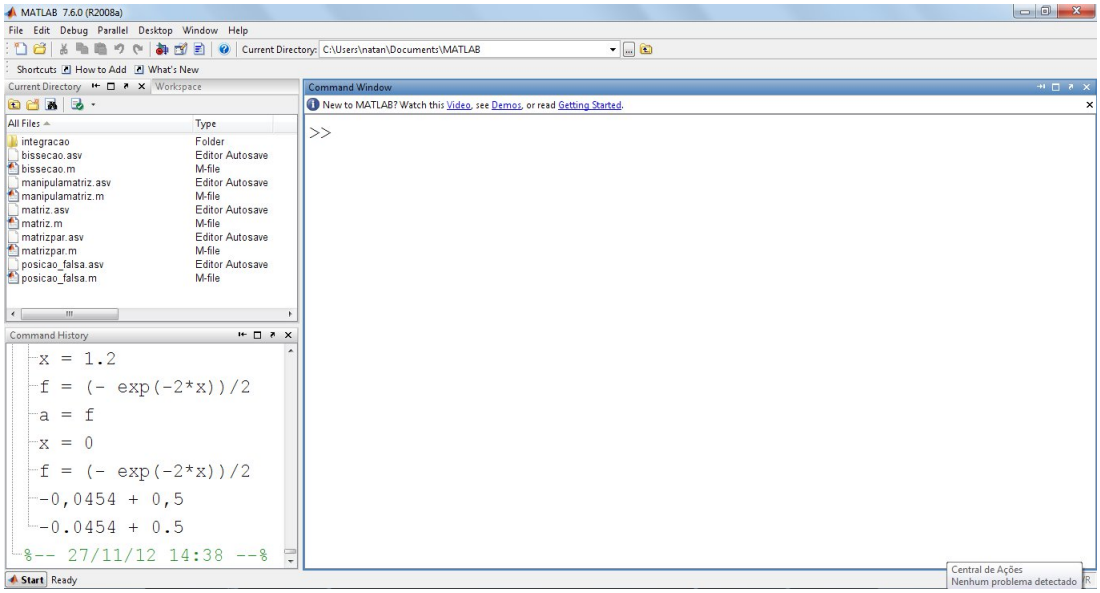

Figura 1: Janela Principal do Matlab.

[1] http://www.mathworks.com/

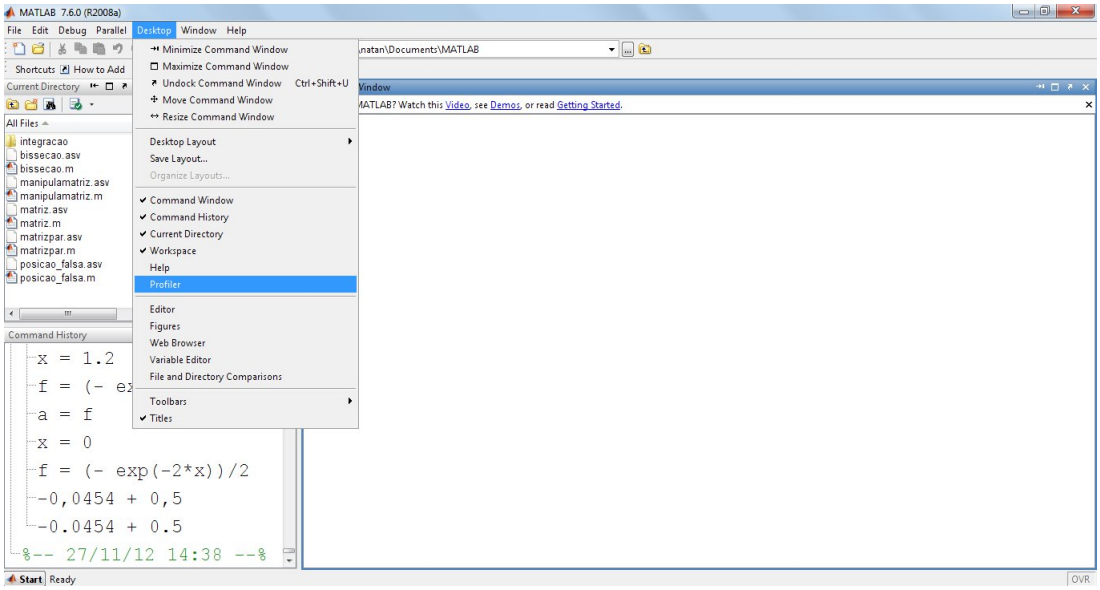

Figura 2: Alterando configurações de exibição de janelas.

Depois de se inicializar uma sessão em *Matlab* é visualizado o sinal de prompt (>>) que significa que o *Matlab* está pronto para receber os comandos.

Um aspecto importante na utilização dos comandos no *Matlab* é que há diferenciação de letras maiúsculas e minúsculas nos nomes de variáveis e comandos. Por exemplo, não existe o comando SUM e sim *sum* e as variáveis x e X são diferentes.

Em uma sessão do Matlab, para salvar todos os comandos digitados na janela de comandos e suas respectivas saídas, podemos usar o comando *diary*. Antes de iniciar o trabalho no Matlab, digite na janela de comandos: *diary arquivosaida.txt*. Assim, tudo que for digitado até que comando *diary off* seja executado, ou que o *Matlab* seja encerrado, estará salvo no arquivo *arquivosaida.txt*.

2 Tipo de Dados Fundamentais

O *Matlab* trabalha fundamentalmente com o tipo de dados matriz de números reais ou complexos. Vários tipos de dados podem ser representados na forma matricial. Um escalar x pode ser considerado uma matriz com uma linha e uma coluna: $x =$ matriz (1x1). Um ponto (x, y) como uma matriz de uma linha e duas colunas: $(x, y) =$ matriz (1x2).

Uma matriz pode ser inicializada de forma explícita como resultado de uma operação, como resultado da leitura de um arquivo de dados ou introduzida diretamente pelo utilizador. Para os exemplos que serão apresentados, consideram-se as seguintes matrizes:

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

Definindo uma matriz de forma explícita:

$>> A = [1, 2, 3; 4, 5, 6]$

, ou,

$>> A = [1, 2, 3$
$4, 5, 6]$

O nome de uma matriz deve ser iniciado por uma letra e pode conter até 19 caracteres. Quando os elementos de uma matriz são digitados, pode-se utilizar vírgulas ou espaços para separar as colunas e ponto e vírgula ou nova linha para separar as linhas.

Para se realizar a soma dos elementos de um vetor ou a soma das colunas de uma matriz, pode-se utilizar a função *sum*. Caso deseja-se obter a soma das linhas da matriz, usa-se o operador de matriz transposta $'$ e a função *sum*. Obtêm-se os valores da diagonal de uma matriz com a função *diag*. Outras funções que operam sobre matrizes são listadas no final da apostila. Antes de implementar alguma função para resolver um problema específico, pode ser interessante verificar se esta função já existe na ferramenta, fazendo consultas na documentação.

A função $fliplr$ faz com que a ordem das colunas da matriz seja invertida, ou seja, a primeira coluna passe a ser a última, a segunda a penúltima e assim sucessivamente.

Pode-se acessar aos elementos de uma matriz por um índice único ao invés de índices (i, j). A referência aos elementos da matriz acontece de cima para baixo da primeira para a ultima coluna. Assim, em uma matriz 3 x 3, o elemento a_{13} é referenciado pela índice 3, o elemento a_{12} pelo índice 4. Para a matriz A representada pela figura 3, o elemento $A(5)$ seria igual a 3.

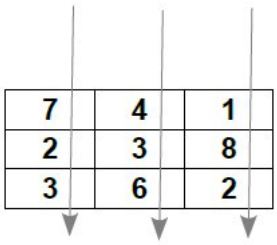

Figura 3: Acesso a elementos da Matriz

Caso tente-se acessar no *Matlab* elementos fora dos limites da matriz, irá ocorrer um erro com a mensagem: *indexoutofbounds*. Porém, caso seja atribuído um valor a uma posição fora dos limites iniciais da matriz, a matriz será expandida para receber este valor, e os outros elementos da linha ou coluna da matriz serão preenchidos com o valor zero.

Pode-se criar uma matriz ou vetor através de três parâmetros, valorinicial:passo:valorfinal. O valor que chamamos de passo é um valor de incremento ou decremento do valor inicial até atingir o valor final. No comando a seguir, será criado um vetor com primeiro valor igual a 70 e será subtraído o valor 5 aos próximos elementos até que o limite 0 seja atingido.

```
a = 70:-5:0
a =
```

```
    70    65    60    55    50    45    40    35    30    25    20    15    10     5     0
```

Pode-se fazer filtros ou alterar matrizes e vetores trabalhando com os intervalos dos índices das colunas e/ou linhas. Quando na posição do índice da linha ou coluna aparece o símbolo :, significa todas as linhas ou colunas. Dada uma matriz A 5 x 5, se quisermos exibir apenas as três primeiras linhas de A, podemos usar o seguinte comando: $A(1:3,:)$. Ou se quisermos exibir apenas as linhas ímpares de A, digitar A(1:2:end,:). Neste exemplo, temos os valores 1:2:end aplicando filtros às linhas da matriz. Estes valores indicam que será exibida a linha 1, com incremento de 2, será exibida a linha 3 e assim sucessivamente até a exibição da última linha da matriz (end). Assim, para esta matriz, seriam exibidas as linhas 1 , 3 e 5. Como na posição do índice das colunas aparece o símbolo :, todos os elementos das colunas destas linhas serão mostrados.

Utilizando a lógica dos filtros anteriores, o comando $A(:,2) = []$ pode ser utilizado para apagar a segunda coluna de uma matriz. Esta operação pode ser realizada para qualquer coluna ou linha.

Uma matriz ou vetor pode ser criado através de operações realizadas sobre dados existentes, conforme o exemplo:

```
>>n =1:3
n = 1     2     3

>> pot = [n n.^2 n.^3]
pot =   1     2     3     1     4     9     1     8    27
```

2.1 Strings

Uma string é uma sequência ordenada de caracteres representada no Matlab na forma de um vector linha de caracteres. Uma das formas de se manipular strings é atribuí-la a uma variável. O conteúdo precisa estar delimitado por aspas simples. Após atribuir a string, podemos acessar cada letra com um índice do vetor. O primeiro elemento do vetor corresponde a primeira letra da string e assim sucessivamente. O exemplo a seguir apresenta esta atribuição:

```
str = 'Isto é uma string';
str(1)
ans = I
str(3)
ans=t
```

Se quisermos armazenar uma lista de *strings*, como por exemplo, os meses do ano, de maneira que sejam acessados pelos índices podemos usar a atribuição informando os itens escritos com aspas simples, delimitados por ponto e vírgula e utilizando chaves com delimitador do vetor. O exemplo a seguir apresenta a leitura e acesso de um vetor nessa sintaxe.

```
semana = {'Domingo';'Segunda';'Terça';'Quarta';'Quinta';'Sexta';'Sábado'};
```

```
semana(1)

ans = 'Domingo'

semana{1}

ans = Domingo
```

Para realizar a comparação entre duas *strings*, podemos usar a função *strcmp*.

```
string = 'a';
strcmp(string, 'a')
 ans = 1
strcmp(string1, 'A')
 ans = 0
```

3 Comandos de leitura e Escrita

Uma das formas mais simples de imprimir mensagens no *Matlab* é através do comando *disp*. Este comando recebe como parâmetro, entre parênteses e aspas simples, o texto que deve ser impresso na tela. Para a leitura de informações, pode-se armazenar em variáveis, valores digitados no teclado através da utilização da função *input*. Os comandos a seguir apresentam exemplos de uso das funções *disp* e *input*.

```
disp('Imprimi uma mensagem simples na tela')
%Armazena em n1 e n2 os valores digitados pelo teclado
n1 = input('Digite um número:' );
n2 = input('Digite outro número:' );
```

O comando *disp* pode imprimir mensagens que contenham *strings* delimitadas por aspas e valores de variáveis. O exemplo a seguir apresenta uma mensagem onde ocorre a transformação de um valor em cm para polegadas. A função *num2str* transforma um valor numérico em *string*.

```
entrada = 5;
unidade = 'cm';
disp([num2str(entrada) ' ' unidade ' é igual a ' num2str(entrada/2.54) ' polegadas.']);
%O resultado será:
5 cm é igual a 1.9685 polegadas.
```

O *Matlab* possui funções mais completas para leitura e escrita de valores que apresentam sintaxe parecidas com as utilizadas na linguagem C. Dentre estas funções, é extremamente útil o uso da função *fprintf*. Através desta função podemos imprimir na tela mensagens com várias formatações, strings, valores inteiros e em ponto flutuante. Para formatar as informações o comando *fprintf* utiliza-se os seguintes caracteres:

%s - utilizado para imprimir uma string

%c - utilizado para imprimir uma character

%d - utilizado para imprimir um número inteiro

%f - utilizado para imprimir um número ponto flutuante

\n - Gera uma quebra de linha

\t - Gera tabulação

\\- utilizado para imprimir uma barra

\% - utilizado para imprimir por cento

Exemplos de utilização do comando fprintf:

```
fprintf('Olá Mundo!!!\n');
fprintf('Meu nome é %s\n', 'José');
x = 0:.1:1;
A = [x; exp(x)];
fprintf('%6s %12s\n','x','exp(x)');
fprintf('%6.2f %12.8f\n',A);
%Resultado
     x        exp(x)
  0.00    1.00000000
  0.10    1.10517092
  0.20    1.22140276
  0.30    1.34985881
  0.40    1.49182470
  0.50    1.64872127
  0.60    1.82211880
  0.70    2.01375271
  0.80    2.22554093
  0.90    2.45960311
  1.00    2.71828183

a = [1.02, 3.04, 5.06];
fprintf('%d\n',round(a));
%Resultado:
1
3
5
```

4 Manipulação de Arquivos

O *Matlab* possui comandos equivalentes aos da linguagem C para manipular arquivos. Para abrir um arquivo texto para leitura como entrada de dados, utilize o seguinte comando:

```
fid = fopen('arquivo.dat','r')
```

Onde: *fid* é uma variável que contém o endereço lógico do arquivo de leitura 'arquivo.dat'. Para ler os dados, utiliza-se o comando fscanf. Supondo que o arquivo de dados contenha uma coluna de valores no formato real, eles serão lidos de forma sequencial pelo comando:

```
x = fscanf(fid, '%f');
```

Terminada a leitura dos dados do arquivo fecha-se o arquivo com o comando: *fclose(fid)*. Para gravar um arquivo de dados, emprega-se a seguinte sequência de comandos:

```
fid = fopen('arquivo_gravacao.dat','w');
fprintf(fid, '%f', x);
```

Observe que o status do *arquivo_gravacao.dat* é de escrita ('w' - write). Ao terminar a gravação dos dados, fecha-se o arquivo com o mesmo comando visto anteriormente, *fclose(fid)*.

5 Operadores

5.1 Operadores Aritméticos

Os operadores aritméticos são utilizados para realizar operações matemáticas sobre números ou matrizes no Matlab. A seguir são listados alguns operadores aritméticos:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão à Direita
\	Divisão à Esquerda
^	Elevado à potência
'	Transposta

5.2 Operadores Relacionais

Estes operadores realizam comparações quantitativas entre dois valores, como por exemplo, maior que, menor que, etc. A Tabela apresenta os operadores utilizados pelo *Matlab*.

Operador	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior que ou igual
==	Igual
~=	Diferente
'	Transposta

Caso os operadores relacionais sejam aplicados sobre matrizes de iguais dimensões, as comparações serão realizadas elemento a elemento e será gerada uma matriz resultado com as mesmas dimensões das matrizes avaliadas. Veja o exemplo da comparação entre duas matrizes A e B.

$$A = \begin{bmatrix} 2 & 7 & 6 \\ 9 & 0 & 5 \\ 3 & 0,5 & 6 \end{bmatrix} B = \begin{bmatrix} 8 & 7 & 0 \\ 3 & 2 & 5 \\ 4 & -1 & 7 \end{bmatrix}$$

A == B

$$ans = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Pode ser realizada a comparação entre um valor escalar e uma matriz.

5.3 Operadores Lógicos

Os operadores lógicos são utilizados para agrupar dois testes, no caso de operadores binários, ou modificar um operador, no caso de operadores unários. Os principais operadores binários são: *and* que retorna o valor verdadeiro se os dois valores agrupados pelo operador são verdadeiros e o operador *or* que retorna verdadeiro se qualquer um dos dois operadores agrupados forem verdadeiros. No *Matlab* temos várias maneiras de utilizar operadores lógicos. Os primeiros operadores lógicos citados são chamados de operadores *short-circuiting*. O funcionamento destes operadores ocorre da seguinte maneira:

Operador	Descrição
&&	Retorna um valor lógico verdadeiro (1) se ambos os valores são verdadeiros e o valor lógico falso (0) caso contrário.
||	Retorna um valor lógico verdadeiro (1) se quaisquer valores, ou ambos, são verdadeiros e o valor lógico falso (0) caso contrário.

Estes operadores avaliam a segunda expressão apenas quando necessário. Se for avaliado por exemplo, $A > 1$ && $B < 3$, o valor de B será avaliado somente se o teste para a primeira expressão for verdadeiro. De outra forma, no teste $A > 1$ $||B < 3$, a segunda expressão não será avaliada se a primeira expressão for verdadeira.

No exemplo a seguir, a função será executada apenas se esta estiver no *path* do Matlab, evitando assim erros como: *myfun.m, cannot be found.*

$comp = (exist('myfun.m') == 2)\&\&(myfun(x) >= y)$

Podemos realizar a avaliação de dois vetores através dos operadores lógicos. Sejam dois vetores A e B descritos a seguir:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

O operador & irá retornar 1 (true) se ambos os elementos de uma mesma posição forem diferentes de zero em ambos os vetores e zero caso contrário, no caso do operador | será

retornado 1 (true) se qualquer elemento de uma posição dos vetores for diferente de zero, e retornará zero, caso contrário. O operador $\sim$ retorna o valor simétrico do elemento.

Exemplo:

A & B = 01001

A |B = 11101

$\sim$A = 10010

Podemos usar funções equivalentes aos operadores lógicos

$A\&B \rightarrow and(A, B)$

$A\ |B \rightarrow or(A, B)$

$\sim$A $\rightarrow not(A)$

A função *find* determina o índice de um vetor que coincide com uma dada condição lógica. Esta função pode ser usada para alterar, por exemplo, elementos de uma matriz ou vetor que satisfaçam algum critério de teste.

Seja a matriz A criada pela função *magic* que gera uma matriz de tamanho n com valores de seus elementos variando em 1 e n^2, e que a soma de cada linha ou coluna tem sempre o mesmo valor.

A = magic(4)

$$A = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

i = find(A > 8); A(i) = 100

$$A = \begin{bmatrix} 100 & 2 & 3 & 100 \\ 5 & 100 & 100 & 8 \\ 100 & 7 & 6 & 100 \\ 4 & 100 & 100 & 1 \end{bmatrix}$$

O mesmo resultado pode ser obtido pelo seguinte comando:

A(A > 8) = 100;

5.4 Precedência de Operadores

Em uma expressão, deve-se avaliar a precedência de cada um operadores: aritméticos, relacionais e lógicos. A precedência determina qual operação ou operador será avaliado primeiro. Operadores com mesma procedência são avaliados da esquerda para a direita na ordem em que aparecem.

A lista a seguir informa o operadores com sua ordem de precedência.

1. Parenteses ();
2. transposta de uma matriz e potenciação;

3. negação e operadores unários (+,-);
4. Multiplicação e divisão
5. soma e subtração
6. operador separação :
7. menor que (<), menor igual(<=), maior que (>), maior igual (>=), igual (==), diferente (~=)
8. Operador *and* de elementos (&)
9. Operador *or* de elementos (|)
10. Operador *and* (&&)
11. Operador *or* (||)

A ordem de avaliação dos operadores pode ser alterada fazendo uso de parenteses para aumentar a precedência de determinadas parcelas de uma expressão.

O *Matlab* sempre dá ao operador & precedência sobre o operador |, mesmo que o operador |esteja posicionado mais a esquerda. A expressão a |b & c é avaliada como se fosse escrita a |(b & c). Então, deve-se usar parenteses para explicitar a precedência entre estes operadores. O mesmo vale para os operadores && e ||.

6 Comandos Condicionais

6.1 If, If elseif e else

No Matlab o comando if pode apresentar as seguintes sintaxes:

```
if expressão
   Instruções...
end

if expressão
   Instruções...
else
   Instruções...
end

if expressão1
   Instruções...
elseif expressão2
   Instruções...
else
   Instruções...
end
```

O comando *if* avalia uma ou um conjunto de expressões e caso o resultado seja verdadeiro, as instruções contidas dentro do *if* são executadas. Exemplo:

```
n=5;
if n > 0
  disp('O número lido é maior que zero');
end
```

No trecho de código descrito anteriormente, verifica-se que a variável n tem armazenado um valor maior que zero. A mensagem contida no comando *disp* apenas será impressa se a condição for verdadeira, o que irá ocorrer neste exemplo. Todos os comandos que estiverem entre o *if* e o respectivo *end* serão executados caso o teste seja verdadeiro. Um comando *if* termina com um comando *end*. Neste exemplo, utiliza-se um teste que verifica apenas uma condição.

Caso existam apenas dois estados possíveis para um teste, pode-se utilizar o comando condicional *if* juntamente com o *else*. A seguir é apresentado um trecho de código que verifica se um número é par ou ímpar. Para verificar se um número é par, avaliamos se o resto da divisão inteira do número por dois é igual a zero. Se isso ocorrer o número é par, caso contrário, o número é impar e não precisamos fazer um segundo teste para chegar a essa conclusão, por isso usamos o *else*. Para calcular o resto da divisão inteira, podemos usar a função *mod* ou a função *rem*:

```
n = 5;
if mod(n,2) ==  0
  disp('O número lido é par');
else
  disp('O número lido é impar');
end
```

mod(x,y) é obtido pela expressão: $x - floor(x./y).*y$ para y diferente de zero. Onde a função *floor* arredonda o resultado para o menor número inteiro, se aproximando de zero.

rem(x,y) é obtido pela expressão; $x - fix(x./y).*y$, se y é diferente de zero. Onde a função *fix* arredonda o resultado para o menor número inteiro, se aproximando de menos infinito.

Por convenção:

mod(x,0) == x.

mod(x,x) == 0.

mod(x,y), para x =y e y =0, tem o mesmo sinal de y.

rem(x,0) == NaN.

rem(x,x), para x =0 == 0.

rem(x,y), para x =y e y =0, tem o mesmo sinal de x.

mod(x,y) e rem(x,y) são iguais se x e y têm o mesmo sinal.

Se houver vários possíveis estados para um teste, podemos utilizar vários comandos *if* independentes ou utilizar comandos *elseif* juntamente com comandos *if*. Os comandos *elseif*

e *else* são opcionais e são executados apenas quando o teste realizado anteriormente apresenta resultado igual a falso. Vários comandos *elseif* podem estar incluídos após um comando *if*. Em uma estrutura de testes, o comando *else* é único e suas instruções serão executadas se todos os testes anteriores falharem. Exemplo:

```
n=5;
if n > 0
  disp('O número lido é maior que zero');
elseif n < 0
  disp('O número lido é menor que zero');
else
  disp('O número é igual zero');
end
```

O resultado da avaliação de uma expressão é verdade ou *true* se o resultado é diferente de zero ou de vazio, falso caso contrário. Pode-se utilizar operadores relacionais e lógicos nos testes realizados.

Podemos inserir mais de um comando dentro de um comando if inclusive outros comandos if. Um comando if pode avaliar várias testes separados por operadores relacionais. O exemplo a seguir verifica e imprimi o resultado se for digitada uma nota válida.

```
n = input('Entre com uma nota');
if n > 100 || n < 0
  disp('Nota inválida');
else
  if n >= 60
     disp('Aprovado');
  else
     disp('Aprovado');
  end
end
```

6.2 Switch

A sintaxe do comando switch deve seguir as seguintes regras:

```
variavel_teste = 1;
 switch variavel_teste
   case expressao1
   % Se a expressão1 for igual ao valor contido na variável varivel_teste, a Instrução1 ↩
       será executada
      Instruções1
   case expressao2
   % Se a expressão2 for igual ao valor contido na variável varivel_teste, a Instrução1 ↩
       será executada
      Instruções2
    :
   otherwise
   % A instruçãon será executada se nenhuma outra anterior for.
      Instruçõesn
end
```

O comando *Switch* avalia várias opções e executa as instruções incluídas dentro da opção na qual o teste é verdadeiro. Cada teste é avaliado pela palavra reservada *case*. Exemplo:

```
numero = input('Entre com um número:');

switch numero
     case -1
         disp('Número Negativo.');
     case 0
         disp('Zero');
     case 1
         disp('Número Positivo');
     otherwise
         disp('Outro Valor');
end
```

Após executar as instruções compreendidas dentro de um case, o *Matlab* continua processando os comandos após o bloco *switch*. As instruções contidas dentro da seção otherwise são executas apenas se nenhum teste anterior for verdadeiro.

No Matlab, apenas o primeiro case que apresenta resultado true é executado. Os valores testados pelo comando case podem ser uma lista de valores contidos em um vetor conforme exemplo({52,78}).

```
n= 52;
switch (n)
    case 52
       disp('Número 52')
    case {52, 78}
       disp('Número 52 ou 78')
 end
```

Resultado: Número 52

7 Comandos for e while

Os comandos *for* e *while* são utilizados para executar uma ou mais ações por um número repetido de vezes até que uma determinada condição seja satisfeita. Ambos os comandos utilizam uma variável, chamada variável de controle, que é responsável pela interrupção da execução. Estes comandos, for e while, são geralmente chamados de laços. A variável de controle é iniciada, a cada execução do laço é verificada se a condição de teste que envolve a variável de controle é verdadeiro e então os comandos dentro do laço são executados. A cada vez que os comandos do laço são executados, geralmente, a variável de controle é alterada, tendo seu valor se aproximando gradativamente da condição de parada.

O comando for é geralmente executado por um número determinado de vezes, mas é possível interromper sua execução através do comando break ou alterando internamente o valor da variável de controle. A sintaxe do comando for é mostrada no exemplo a seguir:

```
for i=1:1:10
    i
 end
```

No exemplo anterior, a variável i é a variável de controle. O laço será executado com o i tendo como valor inicial igual a 1 e sendo incrementado a cada execução em uma unidade. O laço será interrompido quando i for igual a 11, pois 10 é o limite da variável de controle. O comando for i=1:10 tem o mesmo efeito. O valor utilizado como incremento pode ser qualquer inteiro positivo ou negativo, caso seja igual a 1, não precisa ser apresentado. É importante verificar que o laço deve ser executado um número finito de vezes. Assim, especial atenção deve ser dada aos possíveis valores da variável de controle. A seguir são apresentados exemplos do uso do for:

```
% Imprimir os valor 1.0 até 0.0 em ordem decrescente, com decremento igual a -0.1
for s = 1.0:-0.1:0.0
  disp(s)
end

% Imprimi a cada execução do laço, um elemento do vetor
for s = [1,5,8,17]
  disp(s)
end

%Imprimi a cada execução do laço um vetor coluna da matriz identidade 5 x 5. Serão ↩
    impressos 5 vetores coluna.
for e = eye(5)
  disp(e)
 end

%Outra forma de alterar valores de um vetor. Somando 4 a cada elemento do vetor.
a = [ 3 4 6 2 5]
for i = 1:length(a)
  a(i) = a(i) + 4;
end
%Somando 5 aos elementos pares da matriz A
A = [ 3 4 6; 2 5 4; 6 3 2]
for i = 1:length(A)
  for j=1:length(A)
    if rem(A(i,j),2)==0
       A(i,j) = A(i,j) + 5;
    end
  end
end
```

O comando *while* tem funcionamento semelhante ao comando for. No comando for, a variável de controle é normalmente declarada, iniciada e alterada no princípio do laço. No caso do comando *while*, a variável de controle é iniciada antes da execução do laço. A cada execução do laço, a condição de teste é verificada e caso o resultado seja verdadeiro, as instruções internas do laço são executadas. Para que a condição de parada seja alcançada, a variável de controle deve ser alterada dentro das instruções do laço. Exemplos de uso do *while*:

```
%Imprimir os valores de n = 0, 2, 4, 6, 8, 10
 n = 0
 while n <= 10
   disp(n);
   n = n + 2;
 end
```

```
%Este exemplo verifica o número de linhas do arquivo magic.m, despresando linhas em branco ↩
    e comentários.
% Abre o arquivo magic.m para leitura
fid = fopen('magic.m','r');
%inicia uma variável como contador de linhas
count = 0;
%Enquanto não chegar ao final do arquivo aberto e cujo identificador está armazenado na ↩
    variável fid
while ~feof(fid)
% A variável line recebe o conteúdo de uma linha do arquivo
  line = fgetl(fid);
  % O comando if verifica se a linha é vazia, se tem comentários ou se ela é diferente de ↩
      string
  if isempty(line) || strncmp(line,'%',1) || ~ischar(line)
     %Se o teste do if for verdadeiro, a linha não é uma linha válida na contabilidade do ↩
         arquivo e o comando continue
     % faz com que a execução do laço seja enterrompida neste ponto e volte ao inicio, na ↩
         linha do comando while.
     continue
     end
     % Se a linha não é vazia, o contador de linhas é incrementado
     count = count + 1;
end
%Imprime o total de linhas
fprintf('%d Linhas\n',count);
%Fecha o arquivo aberto
fclose(fid);
```

No exemplo a seguir, a função *myfunction* apenas será executada se existir o arquivo no *path* do *Matlab* que a implementa, verificação feita pelo comando *exist*. Se a função existir e recebendo um parâmetro x retornar um valor maior ou igual a *pi*, a mensagem *Funcionou!!!* é apresentada e o laço é interrompido pelo comando *break*.

```
while exist('myfunction.m') && (myfunction(x) >= pi)
   disp('Funcionou!!!')
   break
end
```

8 Scripts e Funções

Um *script* é um arquivo com comandos interpretáveis pelo *Matlab* cujo nome termina com a extensão *.m*. Digitando-se o nome deste arquivo sem a extensão na janela de comandos do *Matlab*, os comandos contidos no *script* são executados sequencialmente. Para isso, é necessário que o arquivo m esteja em um diretório listado no caminho *path* do *Matlab*. Um *script* é criado através da opção *criar novo arquivo m* no menu arquivo ou na barra de tarefas

do *Matlab*.

Por exemplo, suponha que o arquivo *plotseno.m* contenha as seguintes linhas:

```
x = 0:2*pi/N:2*pi;
y = sin(h*x);
plot(x,y)
```

Ao ser executado, a sequência de comandos contida no arquivo *plotseno.m* irá gerar um gráfico com a função definida em *y*.

Os comandos no *script* podem se referir a variáveis já definidas no Matlab, neste exemplo, variáveis N e h utilizadas em plotseno.m.

Além dos scripts, as funções são um tipo de arquivo m que permitem ao usuário criar novos comandos no Matlab. Uma função é definida como um arquivo m que começa com um cabeçalho na seguinte forma:

$function[saida1, saida2, ...] = nome\_funcao(var1, var2, ...)$

Após a criação do cabeçalho da função, o restante do arquivo m consiste de comandos do *Matlab* que utilizam os valores das variáveis var1, var2, ..., para calcular o valor das variáveis saida1, saida2, etc. É importante observar que quando uma função é chamada, o *Matlab* cria um espaço de trabalho separado. Dessa forma, os comandos na função não podem se referir às variáveis da área de trabalho, a menos que elas sejam passadas como argumentos da função (no lugar das variáveis var1, var2, etc.). Inversamente, as variáveis criadas dentro da função são apagadas quando a sua execução termina, exceto se elas forem passadas como argumentos de saída da função (variáveis saida1, saida2, etc.).

A seguir é apresentado um exemplo de uma função, chamada fcn, que calcula f(x) = $sin(x^2)$.

```
function y = fcn(x)
y = sin(x.^2);
% Utiliza-se o operador escalar .^ para que a função fcn retorne um vetor de mesma dimensão↩
    da variável x.
```

Com esta função definida, podemos usar *fcn* como se fosse uma função padrão do *Matlab*:

```
x = (-pi:2*pi/100:pi)';
y = sin(x);
z = fcn(x);
plot(x,y,x,z)
grid
```

Que gera um gráfico do cálculo da função sin(x) e outro com a função personalizada fcn(x).

A seguir outro exemplo de uma função que recebe três números (x,y,z) e retorna os valores da soma e do produto destes números.

```
function [ soma, produto] = soma_produto( x, y, z)
soma = x + y + z ;
produto = x * y * z ;
```

```
end
```

Para usar esta função em *Matlab* executa-se: $[s, p] = somaprod(2, 2, 3)$ que tem como resultado s= 7 e p= 12.

O *Matlab* identifica a função pelo nome do arquivo, neste caso somaprod, e não pelo nome que lhe atribui dentro do arquivo.

É possível codificar várias funções em um arquivo e definir funções dentro de outras funções. Deve-se salvar os arquivos com o nome da primeira e principal função codificada. Quando existe mais de uma função em um arquivo é obrigatório o uso do comando *end* para encerrar cada função.

9 Estudos de Funções e Gráficos 2D

Uma das formas de se construir gráficos em duas dimensões consiste em utilizar um conjunto de pares ordenados de pontos. Para esta abordagem, inicializamos dois vetores, com o mesmo número de elementos. Após armazenar valores nestes vetores, podemos criar o gráfico com o comando plot. Sejam os vetores x e y inseridos no *Matlab* pelo seguintes comandos:

x =[0: 2: 22]

y =[9 8 6 5 8 10 14 17 15 13 11 10]

Pode-se criar um gráfico x,y com o comando plot que na sua forma mais básica é assim utilizado:

plot(x,y)

Este gráfico está representado na Figura 4

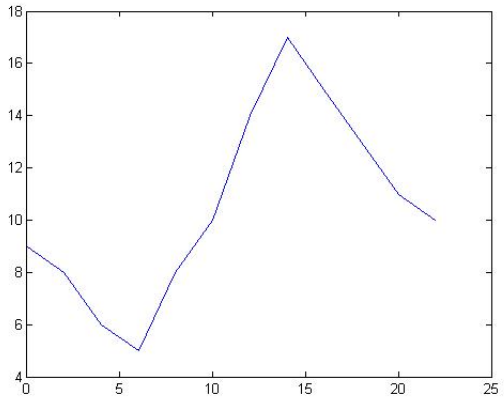

Figura 4: Gráfico do comando plot

Para o comando plot, é ainda possível introduzir facilmente títulos, identificação nos eixos e linhas de grade com os seguintes comandos:

xlabel('Horas do dia')

ylabel('Valores da Temperatura')

title('Valores da Temperatura do Ar')

grid

Através do comando *plot*, pode-se criar também gráficos de funções $f(x)$ utilizando um vetor x com pontos extremos definidos pelo intervalo que a função será avaliada. O vetor x é criado com valores intermediários variando por um passo pequeno, algo como $0,1$. Vejamos os comandos que criam o gráfico da função $f(x) = x^2 + x - 9$ no intervalo $[-4, 4]$, apresentada na Figura 5.

$x = -4 : 0.1 : 4;$

$y = x.^2 + x - 6;$

plot(x,y);

ylabel('f(x)')

ylabel('Valores de x')

title('Geração de Gráficos')

grid

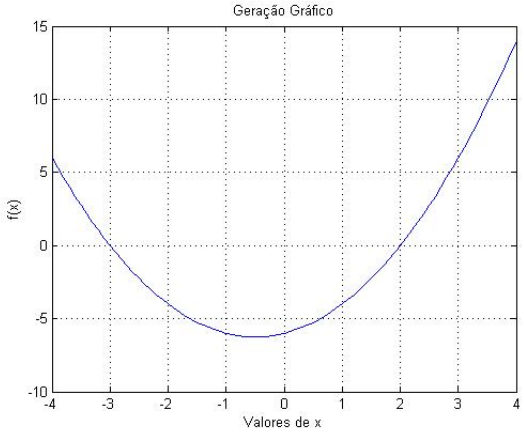

Figura 5: Gráfico $f(x) = x^2 + x - 9$

Para a geração de duas curvas de funções no mesmo gráfico, podemos usar o comando plot, passando como parâmetro vários vetores. Exemplo: O comando a seguir gera dois gráficos na mesma janela, y em função de x e z em função de x.

plot(x,y,x,z);

Outra forma é utilizar o comando hold após a geração do primeiro gráfico.

- hold → congela a tela do gráfico atual de forma que gráficos subsequentes são sobrepostos sobre o atual
- hold → novamente desabilita-se a opção.
- hold on → habilita superposição de gráficos
- hold off → desabilita superposição de gráficos

Outro comando que pode ser utilizado para plotar gráficos é o comando *fplot*. O comando *fplot* recebe como parâmetros a função e o intervalo a ser avaliado e tem sintaxe semelhante a fplot($'f'$, $[a, b]$).

Exemplo: Gerar o gráfico de $f(x) = 3 * x^2 - e^x$. Utiliza-se a função *fplot* como a seguir:

fplot('$3 * x^2 - exp(x)$',[0 4 0 8])
Este gráfico está representado pela Figura 6

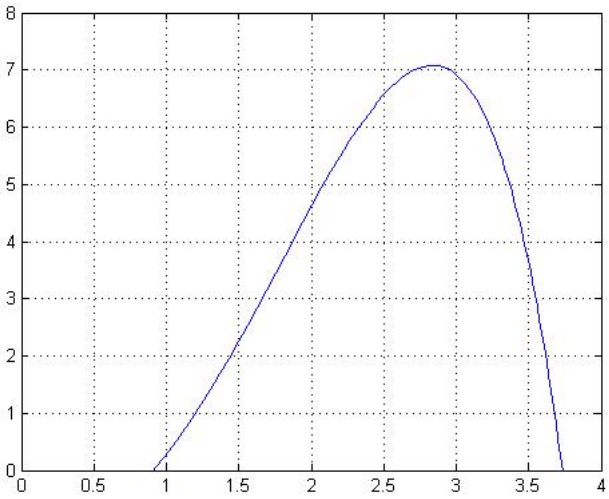

Figura 6: Gráfico com o comando fplot

9.1 Utilizando ezplot e variáveis simbólicas

O *Matlab* possui um pacote adicional que permite operações e geração de gráficos com variáveis simbólicas. Variáveis simbólicas são variáveis que não possuem valores numéricos. Com este tipo de variável é possível realizar cálculos como derivação e integração de funções. O emprego de variáveis simbólicas na geração de gráficos evita que se tenha que definir um vetor x com os seus limites e intervalos previamente a geração do gráfico. Assim, é possível gerar um gráfico sem determinar o intervalo de estudo.

Após a geração do gráfico e uma análise inicial do comportamento da função, podemos passar o intervalo que se deseja analisar como parâmetro para as funções do *Matlab* que geram os gráficos. Para a geração de gráficos em duas dimensões que utilizam variáveis simbólicas, utilizaremos o comando *ezplot*. O comando *ezplot* recebe como parâmetro obrigatório a função que se deseja plotar entre aspas simples. Outros parâmetros podem ser passados para a função como o intervalo da variável independente. Assim como no comando *plot*, podem ser definidas características adicionais do gráfico como rótulos de eixos. Para declarar uma variável simbólica utilizamos o comando do *Matlab syms*. Assim, o comando *syms x y* cria duas variáveis simbólicas. O exemplo a seguir cria uma variável simbólica x e gera o gráfico de função f(x) = $exp(-x^2) - cos(x)$ no intervalo [-6,6] representado pela Figura 7.

```
syms x
ezplot('exp(-x^2)-cos(x)',[-6,6])
```

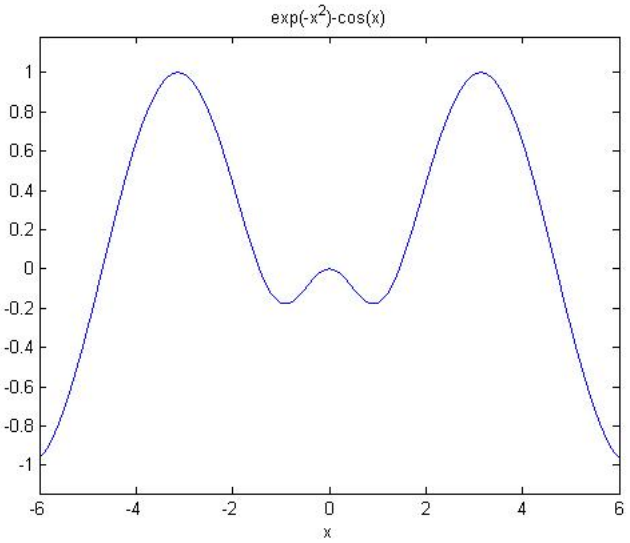

Figura 7: Gráfico com o comando ezplot

9.2 Avaliando funções

Dado qualquer x, podemos encontrar o valor da função $f(x)$ executando o comando de atribuição da função em uma variável qualquer após o valor de x ser definido. Exemplo: Se quisermos calcular o valor da função $f(x) = 3 * x^2 - exp(x)$ para $x = 1,5$, devemos executar os seguintes comandos no *Matlab*:

$x = 1.5;$

$f = 3 * x^2 - exp(x)$

Será armazenado em f o valor da função no ponto x = 1,5 que, para o exemplo, será $f = 2,2683$.

Utilizando esta forma de calcular o valor da função em um ponto, a cada alteração do valor de x, o comando de avaliação da função deve ser executado novamente.

Uma outra forma de avaliar uma função em um determinado ponto é utilizar o comando *feval*. A função *feval* recebe como parâmetros o nome da variável que corresponde a função e o ponto onde a função será avaliada. Uma das formas de passar a função como parâmetro para a *feval* é mostrado a seguir. Veja que é necessário colocar antes da função os símbolos $@(x)$. O resultado dos comandos digitados será -7.1250

$x = 1.5;$

$f = @(x)x.^3 - 9 * x + 3$

$feval(f, x)$

Com o uso dos dois comandos citados anteriormente, podemos calcular o valor de uma função qualquer em um ponto fornecido.

9.3 Obtendo Raízes de Equações e Polinômios

Uma das formas de se armazenar um polinômio de grau n é colocar os coeficientes do polinômio em um vetor com tamanho $n+1$. Como exemplo, seja o polinômio $x^4+4*x^2-5*x+6$. Podemos armazenar este polinômio em um vetor v com a seguinte representação v = [1 0

4 -5 6]. Vemos que os coeficientes que não aparecem na forma tradicional do polinômio são armazenados no vetor como zero. Assim, dado um polinômio armazenado na forma de um vetor, podemos encontrar suas raízes no *Matlab* com o comando *roots*. Vejamos o exemplo:

v = [1 0 4 -5 6];

$roots(v)$

$ans =$

$-0.7119 + 2.0646i$

$-0.7119 - 2.0646i$

$0.7119 + 0.8667i$

$0.7119 - 0.8667i$

Vemos que para o exemplo foram encontradas quatro raízes imaginárias.

Outra forma de encontrar uma raiz para uma dada função, visto que é conhecido um valor aproximado para esta raiz, é utilizar o comando *fzero*. A partir do ponto informado como aproximação, o comando *fzero* encontra um intervalo onde a função tem variação de sinal e então procura pela raiz. Se o ponto informado como aproximação não pertencer a vizinhança de uma raiz, a função não convergirá para a solução. Sabendo que a função $f = 3*x^2 - exp(x)$ tem um zero próximo a x =1, podemos executar o comando:

fzero('$3 * x^2 - exp(x)$',1)

ans = 0.9100

10 Considerações sobre performance

Uma das formas de se verificar o tempo gasto para execução de um *script* ou função no *Matlab* é utilizar os comandos *tic* e *toc*. Use o comando *tic* no início do código que deve ser contabilizado e o comando *toc* logo após o encerramento do código. Será informado o tempo gasto na execução. Se o trecho de código for pequeno ou se desejar verificar a média de n execuções, pode-se usar um laço para o cálculo.

```
tic
  for k = 1:100
      executar o código
  end
toc
```

Conforme citado no início deste material, o *Matlab* é otimizado para trabalhar com matrizes e vetores. Assim, em algumas implementações, é mais eficiente utilizar estas estruturas do que por exemplo, usar laços. Vejamos o exemplo de dois códigos que calculam o valor de uma função em vários pontos. O primeiro utiliza um laço e o segundo vetores. Estes códigos calculam o valor de seno de 1001 valores variando entre 0 e 10.

```
i = 0;
for t = 0:.01:10
   i = i + 1;
   y(i) = sin(t);
end
```

Utilizando apenas vetores o código fica mais simples e eficiente.

```
t = 0:.01:10;
y = sin(t);
```

É mais eficiente pré-alocar variáveis matrizes ou vetores, principalmente com dimensões significativamente grandes para evitar que a cada incremento do tamanho da variável, seja necessário que o *Matlab* procure novas posições contínuas de memória para armazenar a estrutura de dados. A seguir são apresentados dois trechos de código que incrementam o tamanho do vetor x com os respectivos tempos de execução que exemplificam este problema. Verifica-se que a segunda codificação é mais eficiente com relação ao tempo computacional necessário para a execução.

```
tic
x = 0;
for k = 2:1000000
  x(k) = x(k-1) + 5;
end
toc
```

Tempo de execução = 0,301528 segundos.

```
tic
x = zeros(1, 1000000);
for k = 2:1000000
  x(k) = x(k-1) + 5;
end
toc
```

Tempo de execução = 0.011938 segundos.

Para pré-alocar matrizes podemos usar funções como: *ones* que gera uma matriz com todos os elementos iguais ao número 1, *zeros* que gera uma matriz preenchida com zeros, *eye* para criar uma matriz identidade, *magic* e *rand* para criar matrizes com números pseudoaleatórios.

Quando usamos os operadores lógicos *and* e *or*, deve-se escolher entre os operadores & e |que realizam comparações entre elementos de vetores ou os operadores && e ||que operam sobre valores escalares.

Outras considerações sobre performance:

Dividir *scripts* grandes em vários *scripts* pequenos;

Codifique funções mais simples que podem ser reutilizadas;

Use funções ao invés de *scripts*;

Se possível, use vetores em substituição a laços;

Pré-alocar variáveis;

11 Alguns Comandos

Dado A = [1 2 3;4 5 6;7 8 0] Calcular:
poly - polinômio característico
det(A) - determinante
trace(A) - traço
rank(A) - rank
roots(p) - raízes do polinômio característico
inv(A) - inversa da matriz
eig(A) - auto-valores e auto-vetores

- who → Lista as variáveis definidas até ao momento
- whos → Idêntico ao who só que é dada informação adicional (tamanho, ?)
- what → Lista dos ficheiros *.m, *.mat e*.mex
- exit,quit → Abandonar o Matlab
- ! → Executar comando do sistema operacional
- computer → Tipo de computador sobre o qual o *Matlab* está executando
- clear → Eliminar variáveis do espaço de trabalho
- control + c → Abortar uma ação executada pelo Matlab
- ans → Variável usada para armazenar o resultado da última operação
- help → Para acessar ajuda
- % → Para inserir uma linha de comentários
- pause → Espera por uma ação do utilizador

Funções Matemáticas

- abs → valor absoluto
- sqrt → raiz quadrada
- real → parte real de um número complexo
- imag → parte imaginária de um número complexo
- conj → conjugado de um número complexo
- round → converte para o inteiro mais próximo
- sign →sinal de número

- fmin → mínimo de uma função de uma variável
- fzero → zeros de uma função de uma variável
- fmins → mínimo de uma função de várias variáveis
- fzeros → zeros de uma função de várias variáveis
- sin → seno
- cos → coseno
- tan → tangente
- asin → arco seno
- acos → arco coseno
- atan → arco tangente
- sinh → seno hiperbólico
- cosh → coseno hiperbólico
- tanh → tangente hiperbólica
- log → logaritmo natural (base e)
- log10 → logaritmo (base 10)
- rem(x,y) → Resto da divisão de x/y
- exp(x) → Exponencial de x ex
- log(x) → Logaritmo natural de x na base e: ln x
- log10(x) → Logaritmo de x na base 10: log10 x

Manipulação de Matrizes

- diag → matriz diagonal
- eye → matriz identidade
- ones → matriz de uns
- zeros → matriz de zeros
- rand → matriz aleatória
- magic → matriz mágica
- tril → matriz triangular inferior

- triu → matriz triangular superior
- det → determinante de uma matriz
- rank → característica de uma matriz

Operações por colunas/linhas (vetores)

- max → máximo
- min → mínimo
- mean → valor médio
- std → desvio padrão
- median → mediana
- sort → ordenação
- sum → soma dos elementos
- prod → produto dos elementos
- diff → diferenças (aproximação da derivada)
- hist → histograma
- table1 → interpolação linear
- corr → matriz de correlação
- cov → matriz de covariân

Valores Especiais

- ans → resposta de uma expressão não atribuída a uma variável
- eps → precisão
- pi → $\pi = 3,14159...$
- inf → ∞
- Nan → indica que não é um número
- clock → relógio
- nargin → número de argumentos de entrada numa função
- nargout → número de argumentos de saída numa função

12 Erros mais comuns

1. ***index out of bounds***: Provável acesso aos elementos fora dos limites de uma matriz no *Matlab* . Exemplo: Acesso ao elemento A(1,4) em uma matriz 3 x 3.

13 Referências

*

1 - Comando help do Matlab

2 - http://www.mathworks.com/help/matlab/index.html. Acessado em fevereiro 2013.