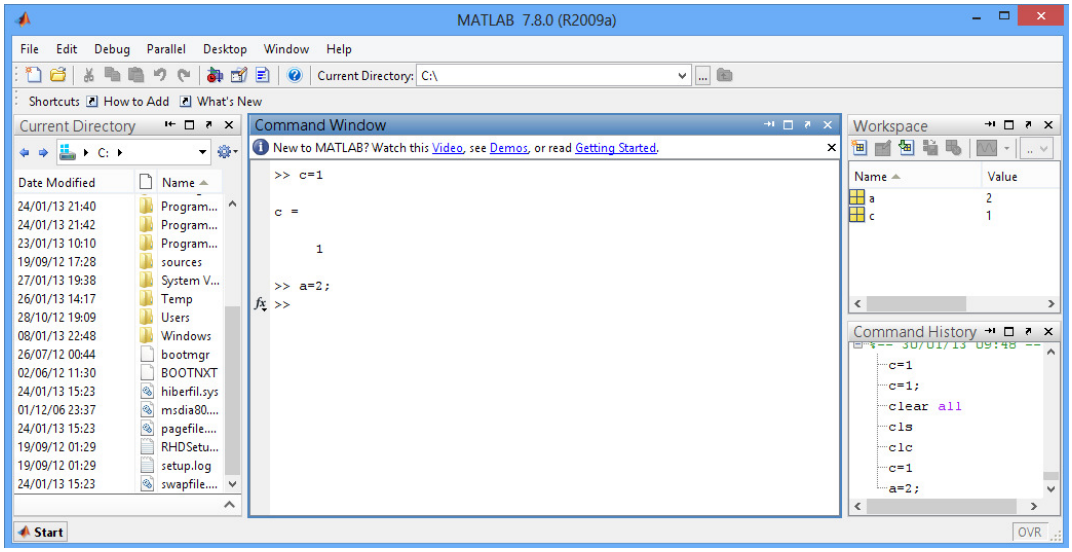
Roteiro MatLab

1. Ambiente MatLab



Principais janelas:

- ***Current Directory***: arquivos e pastas do diretório corrente
- ***Command Window***: janela para comandos e saída de resultados
- ***Workspace***: variáveis definidas (valor, tipo, dimensões)
- ***Command History***: histórico dos comandos executados

Command Window (ou prompt): janela onde escrevemos os comandos.

a) Usa-se % para colocar comentários, semelhante à linguagem C.

b) Exemplo de atribuição de variável:

```
c=1

a=2;  % Com o ponto e vírgula no final, resultado não aparece na tela
```

c) Para limpar a janela de comandos:

```
clc    % Limpa a tela, mas as variáveis continuam no workspace
```

Workspace: registra todas as variáveis definidas

a) Cuidado para não usar variáveis com valores indesejados:

```
b=8+a                  % 'a' já vale 2, conforme definido anteriormente
g = 9.7+5 / 2; 8/5     %  É atribuído 14.7 a 'g' e impresso 8/5 na tela
```

b) Para redefinir ‘a’ antes de calcular ‘b’:

```
a=5
b=8+a
```

c) Para limpar ‘a’ do workspace:

```
clear a
```

d) Para limpar tudo do workspace:

```
clear all
```

e) Somos avisados em caso de acesso a variáveis inexistentes:

```
a
??? Undefined function or variable 'a'.
```

f) Quando não é definida uma variável para armazenar o valor da expressão calculada, a resposta ficará na variável ‘ans’. Exemplos:

```
2/4          % 'ans' fica com 0.5
k=9;
k*2;         % 'ans' fica com 18
```

g) Comandos para verificar o workspace:

```
whos         % Mostra todas as variáveis
whos k       % Mostra somente a variável solicitada
```

Command History: registro de todos os comandos que efetuamos.

a) Para acessar os comandos anteriores, basta usar as teclas ↑ e ↓.

b) É oferecido autocomplementação: basta digitar o começo do comando e usar as teclas ↑ ou ↓. Exemplo:

b (clicando em ↑) aparece b=8+5

Current Directory: diretório com arquivos e pastas que podemos acessar no momento.

a) Para alterar o diretório, basta navegar nas pastas.

b) Isso também pode ser feito através de comandos no prompt:

```
pwd             % Mostra o diretório atual

cd C:\Temp      % Muda para o diretório informado
```

c) Inspeções no diretório são úteis principalmente quando se trabalha com arquivos, e não apenas com o prompt.

2. Algumas funções oferecidas pelo MatLab

a) Comando que procura as funções disponíveis que possuem 'sqrt' no nome ou que tratem desse assunto:

```
lookfor sqrt
```

b) lookfor identity % Descobrimos que 'eye' gera matrizes identidades

c) Procurar detalhes sobre a função 'sqrt':

```
help sqrt
```

d) Outra opção é procurar no menu do programa: Help > Product Help

e) Uso do tab: permite a complementação de comandos

f) Registro dos comandos:

```
diary on - inicia o registro dos comandos e saídas no arquivo padrão 'diary'

diary off - finaliza o registro

diary  'registro1.dir' - inicia o registro dos comandos e saidas no arquivo dado como argumento.

load ''
```

3. Valores numéricos

a) Algumas constantes e valores interessantes:

```
pi              % 3.1416...

sin(pi/2)       % 1

eps     % Valor de tolerância (depende da máquina). Ex: 2^(-52) = 2.2204e-016
```

```
eps(2.3)        % Distância para o próximo real maior que 2.3

exp(1)          % 2.7183...
```

b) Caso os resultados da expressão ultrapasse o maior real representado, esse valor será informado como Inf ou -Inf (Infinito). Teste os exemplos abaixo:

```
1/(eps^100)

- 1/(eps^100)

2/0
```

c) Caso o resultado da expressão não seja um número válido, esse valor será informado como NaN (Not-a-Number). Exemplo:

```
0/0
```

d) Menor real representável:

```
realmin         % Exemplo: 2.2251e-308
```

e) Maior real representável:

```
realmax         % Exemplo: 1.7977e+308
```

f) Opção para visualizar mais casas decimais:

```
format long     % Habilita 15 casas decimais para double e 7 para single

1/3             % Faça esse teste
```

g) Opção para visualizar menos casas decimais:

```
format short    % Habilita 5 casas decimais

1/3             % Faça novamente esse teste
```

h) Faça os testes abaixo, que utilizam o tipo single e veja a quantidade de bytes usada no armazenamento das variáveis:

```
x = 5/6;

y = single(x);

whos x

whos y
```

4. Laços de repetição

a) **for** - similar ao C porém com diferenças de sintaxe:

Exemplo 1:

```
for i=1:5
    disp('Olá'); % escreve Olá
end
```

Exemplo 2:

```
 for x=1:2:11
        disp(sprintf('x=%4.2f', x));
 end
```

b) **while**

Exemplo:

```
i=10; f=1;
while i>=1
        f=f*i;
        i=i-1;
end
disp(sprintf('f=%f',f))
```

c) não há um laço do-while, mas voce pode obter mesmo efeito com o while e uma condição escolhida de modo a garantir pelo menos a primeira execução.

5. Forçar término de processamento

a) Teste o código abaixo:

```
a=1;
while (a<100)
a=2;     % Laço infinito, pois não incrementa 'a'
end
```

b) Para encerrar o processamento:

```
Ctrl c    % Repare que volta a aparecer >> no prompt
```

6. Variáveis booleanas

```
x = true                % O valor de true é 1
y = false               % O valor de false é 0
x&y                     % Operador AND
x|y                     % Operador OR
```

7. Vetores numéricos

```
a = []                  % Define o vetor 'a' como vazio
a = [5 1.5 -0.3]        % Define o vetor 'a' com 3 posições
a = [5, 1.5, -0.3]      % Neste caso, o uso de vírgula é opcional
a = [5 1.5, -0.3]       % Repare que a definição continua válida...
a(2)                    % Acesso à segunda posição de 'a' (índices começam em 1)
x = a(3)
b = 10:5:40             % Gera vetor de 10 a 40, com passo de 5
b = [10:5:40]           % Pode-se usar colchetes também
b = 10:7:40             % Gera vetor de 10 a 40, com passo de 7
b = 10:-1:40            % Retorna vetor vazio, pois não foi possível ser definido
isempty(b)              % Testa se o vetor é vazio (retorna 1) ou não (retorna 0)
u = 5:9                 % Gera vetor de 5 a 9, com passo de 1
c = linspace(10,40,7)   % Gera vetor com 7 elementos entre 10 a 40
length(c)               % Retorna o tamanho do vetor 'c'
c = linspace(10,40)     % Gera vetor com 100 elementos entre 10 a 40
```

Obs: O comando anterior gera muita saída. Para aparecer tudo de uma vez:

```
more off
c = linspace(10,40)
```

Para aparecer aos poucos:

```
more on
c = linspace(10,40)
% Uma parte do resultado foi apresentada. Ao se teclar <enter>, a continuação do resultado irá sendo mostrada.
```

```
x = c(3:5)          % Gera um novo vetor com os elementos das
                    posições 3, 4 e 5 do vetor 'c'

c(5:-2:1)           % Gera um novo vetor com os elementos das
                    posições 5, 3 e 1 do vetor 'c' (ou seja, com passo -2).
                    Você também pode atribuir isso a uma nova variável.

c([4 1])            % Gera um novo vetor com os elementos das
                    posições 4 e 1 do vetor 'c'
                    Você também pode atribuir isso a uma nova variável.

v = [1.5; -3; 9]    % Gera vetor coluna (ponto-e-vírgula separa as colunas)

length(v)           % Retorna 3 (número de colunas de v)

x = v'              % Gera a transposta do vetor 'v'

y = (1:3)'          % Gera o vetor coluna [1; 2; 3]

v==y                % Compara v e y, elemento a elemento
```

8. Matrizes numéricas

```
A = [3 2 -5 ; 4 7 9]    % Gera matriz 'A' com duas linhas e 3 colunas

A(1,2)=8                % Atribui valor a uma posição da matriz

A(3,6) = 1              % Cuidado para não acessar posições inexistentes

A(1)                    % Valor guardado na posição (1,1), mas é melhor fazer A(1,1)

A(2)                    % Valor guardado na posição (2,1)

A(3)                    % Valor guardado na posição (1,2)

A([1 2]) = 10           % Altera para 10 os valores guardados nas posições (1,1) e (1,2)

B = [1 2 3; 4 5 6; 7 8 9]

B([1 3], 2)             % Valores armazenados nas posições (1,2) e (3,2)

C = B(3:-1:1, 1:3)      % Gera matriz linha por linha obedecendo a ordem: B(3,1),
                        B(3,2), B(3,3), B(2,1), B(2,2), B(2,3), B(1,1), B(1,2), B (1,3)

C = B(3:-1:1,:)         % Idem ao anterior, considerando todas as colunas existentes

X = B(1,:)              % Gera vetor correspondente à primeira linha de 'B'
```

```
D = B(1:2, [1 3])          % Gera nova matriz com os seguintes elementos de 'B': B(1,1), B(1,3), B(2,1), B(2,3)

D = B([1 2], [1 3])        % O comando anterior poderia ser assim também

E = [B C(:,2) [2 2 2]']    % Concatena a matriz 'B', a segunda coluna de 'C' e o vetor coluna [2 2 2]' (as dimensões devem ser compatíveis)

E(:,3)=[]                  % Retira de 'E' a sua terceira coluna

t = size(E)                % Retorna a dimensão da matriz, onde 't' é um vetor

[l,c] = size(E)            % Recebe a dimensão como um vetor, onde 'l' e 'c' são números

A = magic(3)               % Gera matriz cuja soma de linhas ou colunas é constante

sum(A)                     % Vetor com a soma de cada coluna de 'A'

prod(A)                    % Vetor com o produto de cada coluna de 'A'

prod(A(2,:))               % Produto da segunda linha de 'A'

max(A)                     % Vetor com o valor máximo de cada coluna de 'A'

max(A(:))                  % Maior valor guardado em 'A'

min(A)                     % Vetor com o valor mínimo de cada coluna de 'A'

sort(A)                    % Gera matriz com valores de 'A', mas com as colunas ordenadas

sort(A(:))                 % Gera vetor com valores de 'A', todos ordenados entre si

d = diag(A)                % Gera vetor com elementos da diagonal principal de 'A'. Parâmetro é matriz.

D = diag(d)                % Gera matriz nula com elementos de 'd' na diagonal principal de A. Parâmetro é matriz.

L = tril(A)                % Gera matriz nula, mas preenche triângulo inferior com elementos de 'A'

U = triu(A)                % Gera matriz nula, mas preenche triângulo superior com elementos de 'A'

Z = zeros(2)               % Gera matriz nula de ordem 2

U = ones(2,3)              % Gera matriz de dimensão 2 x 3 apenas com valores unitários

I = eye(2,4)               % Gera matriz identidade de dimensão 2 x 4

I = eye(3)                 % Gera matriz identidade de ordem 3

R = rand(3)                % Gera matriz de ordem 3 com números randômicos
```

```
inv(R)          % Gera matriz inversa de 'R'
det(R)          % Calcula determinante de 'R'
rank(R)         % Calcula o posto de 'R'
```

9. Strings

```
a = 'MATLAB'        % Usa-se aspas simples para declarar strings ou
                    caracteres isolados
c = 'e'
b = 'strings'
d = [a ' manipula ' b]          % Concatenação de strings
```

10. Operações com vetores, matrizes e escalares

Considere as variáveis abaixo nos exemplos a seguir:

```
a = 1:5, b=10:10:50, c=2
a+c             % Soma 'c' a cada elemento de 'a'
a/b             % Retorna a(1)/b(1)
a./b            % Divide a(1)/b(1), a(2)/b(2), ..., a(5)/b(5).
                  Operação é realizada elemento a elemento.
a.\b            % Divide b(1)/a(1), b(2)/a(2), ..., b(5)/a(5).
                  Operação é realizada elemento a elemento.
a.^c            % Eleva cada elemento de 'a' à potência 'c'
c.^a            % Eleva 'c' a cada elemento de 'a'
```

Considere as novas variáveis abaixo:

```
A=[1 2 3; 4 5 6; 7 8 9], B=[11 12 13; 14 15 16; 17 18 19], c=3
A*B             % Multiplicação convencional de matrizes
A.*B            % Multiplica elementos de mesma posição
A*c             % Multiplica cada elemento de 'A' por 'c' (equivale a A.*c)
A.^c            % Eleva cada elemento de 'A' à potência 'c'
A^c             % Equivalente a A*A*...*A , 'c' vezes
```

```
c.^A                % Eleva 'c' a cada elemento de 'A'
```

Considere as novas variáveis abaixo:

```
a = 1:5, b=sqrt(a)
```

```
[a;b]'              % Gera matriz concatenando o vetor 'a' (na primeira linha)
                    com o vetor 'b' (na segunda linha), e depois faz a transposta

d=a>=c              % Verifica se cada elemento de 'a' é maior ou igual a 'c',
                    retornando 1 (true) ou 0 (false) em cada elemento

f=(a>3)&(a<=8)      % Verifica se cada elemento de 'a' está no intervalo (3,8]
```

11. Gráficos

a) Inicialmente, é preciso definir os pontos do gráfico:

```
x = linspace(-8,8,50)

y =sin(x)

z=cos(x)
```

b) Fazer um gráfico por vez:

```
plot(x,y)    % Aparece o gráfico com os pontos (x,y)

plot(x,z)    % Aparece o gráfico com os pontos (x,z), mas apagando o anterior
```

c) Desenhar as linhas de grade na janela do gráfico:

```
grid         % Aparecem linhas verticais e horizontais tracejadas
```

d) Mostrar gráficos juntos:

```
plot(x,y)    % Aparece o gráfico com os pontos (x,y)

hold on      % Deixa o gráfico anterior gravado na tela

plot(x,z)    % Aparece o gráfico com os pontos (x,z), agora junto ao anterior

hold off     % A partir de agora novos gráficos irão sobrepor os anteriores
```

e) Mostrar gráficos juntos usando um só comando:

```
plot(x,y,'*',x,z,'o')          % Mostra pontos (x,y) com '*' e (x,z) com 'o'

plot(x,y,'-*',x,z,'-o')        % Mostra os pontos, unindo-os

plot(x,y,'-*',x,z,'-.o')       % Agora, com linha tracejada para (x,z)
```

f) Há mais recursos relacionados à apresentação de gráficos (cor, legenda, etc.). Basta procurar no Help.

12. Arquivos

a) Criação de arquivo de comandos:

- Através do menu do programa: File > New > Blank M-file

- Ou diretamente com o botão New M-file da barra de ferramentas

A extensão de arquivos MatLab é ".m".

Qualquer conjunto de comandos pode originar um arquivo.

Salve o arquivo no diretório desejado:

- Com "Save", arquivo é gravado no "Current Directory"

- Com "Save as", é possível definir outro diretório

b) Exemplo de programa com entrada fornecida pelo usuário:

```
disp('Vamos resolver um sistema linear 2x2')
a11 = input('Digite a11: ');
a12 = input('Digite a12: ');
a21 = input('Digite a21: ');
a22 = input('Digite a22: ');
A = [a11 a12; a21 a22];
b11 = input('Digite b11: ');
b12 = input('Digite b21: ');
b = [b11; b12];
x = A\b
```

c) Para rodar o programa:

- Clique no "Play" da janela com o arquivo .m.

Obs: Se o diretório do arquivo .m for diferente do "Current Directory", receberá um aviso para atualizar o diretório. Basta confirmar!

- Outra opção é digitar o nome do arquivo (com ou sem a extensão .m) no prompt do MatLab.

Obs: Só funcionará se o "Current Directory" for justamente o diretório onde o arquivo se encontra.

- Agora teste com o sistema: x+y=4 e x-y=2:

a11 = 1; a12 = 1; a21 = 1; a22 = -1; b11 = 4; b22 = 2

Obs: O resultado final (valor de 'x') aparecerá no "Command Window", pois não tem ponto-e-vírgula no comando `x = A\b` do arquivo.

- Podemos depurar o programa: basta inserir breakpoints e rodá-lo.

- Podemos verificar o valor das variáveis no decorrer da execução: através do workspace, ou no Command Window (se elas estiverem sendo impressas), ou passando o mouse sobre a variável no próprio programa.

13. Funções em arquivos

a) Novas funções devem ser declaradas em arquivos.

O **nome do arquivo** não precisa ser igual ao **nome da função**, mas convém fazer desse modo, pois ela será chamada pelo **nome do arquivo**.

O **nome do arquivo** não pode ser igual ao nome de alguma função já existente no MatLab. Ex: Se o arquivo *normafun.m* abaixo for chamado *norm.m*, ocorrerá conflito com a função *norm* do MatLab. Veja o erro que aparece:

```
Warning: Function C:\...\norm.m has the same name as a MATLAB
builtin. We suggest you rename the function to avoid a
potential name conflict.
```

b) Exemplo de function (arquivo normafun.m):

```
function norm = normafun(y)
  n = length(y);
  soma = 0;
  for i=1:n
      soma = soma + y(i)^2
  end
  norm = sqrt(soma);
```

c) Testando a função no prompt:

w = [1 1 3 5]

n2 = normafun(w)

Obs1: Para esconder os valores de *soma*, basta colocar ponto-e-vírgula no final do comando que calcula essa variável.

Obs2: Não podemos rodar a função pelo "Play" do arquivo, pois não estaríamos fornecendo o argumento de entrada da função.

d) Cada arquivo pode conter mais de uma função. Nesse caso, as demais funções são usadas como auxiliares da primeira.

```
function norm = normafun(y)   % Função principal: deve ser
                              a primeira declarada
  n = length(y);
  soma = 0;
  teste();
  k = teste2(n)
  for i=1:n
      soma = soma + y(i)^2
  end
  norm = sqrt(soma);
```

```
function teste     % Função auxiliar sem entrada nem saída
  disp 'Testando';      % Escreve na tela uma mensagem

function y = teste2(x)  % Função auxiliar
  y = x + 1;      % Esse y é diferente do y de normafun
```

e) Alterar a função para retornar duas informações:

```
function [norm, n] = normafun(y)
```

Obs: Basta mudar a assinatura da função, pois 'n' já está sendo definido lá dentro.

Para testar no prompt:

[n2, n] = normafun(w)

Obs: Na chamada da função, podemos usar variáveis com os mesmos nomes das de dentro da função (ex: 'n'), mas são outras!

f) Após fechar o arquivo, caso queira reabrir:

- Clique no arquivo dentro do "Current Directory"

- Ou execute o comando edit nome_arquivo

14. Passagem de parâmetros por valor

a) MatLab trabalha com passagem de parâmetro por valor, até em caso de vetores. Exemplo (arquivo teste.m):

```
function teste (x)
  x(2) = 99;
  end
```

Execute no prompt:

w = [1 1 3 5]

teste(w);

w % Verifique que 'w' não foi alterado

b) Se quisermos realmente alterar o parâmetro enviado, podemos atualizá-lo com o retorno da função. Ex:

```
function x = teste (x)
  x(2) = 99;
  end
```

Execute no prompt:

```
w  = [ 1 1 3 5]

w = teste(w);

w                    % Verifique agora que 'w' foi alterado
```

15. Passagem de funções como parâmetro

a) Exemplo de função (arquivo calcula.m):

```
function x = calcula (f,a,b)
   x = f(a,b);
   end
```

b) Exemplo de programa (arquivo exemplo.m):

```
fun1 = @(x,y) x + y;  % @(x,y) define fun1 com 2 parâmetros
fun2 = @(x,y) x - y;  % idem para fun2
x = calcula(fun1,2,3) % Passando fun1 como parâmetro
y = calcula(fun2,2,3) % Passando fun2 como parâmetro
```

Obs: Esse código também pode ser executado no prompt.

c) Use breakpoints no código: coloque-os na 3ª e 4ª linha e execute passo a passo, verificando valores de 'x' e 'y' no workspace.