



---

## Introdução ao Matlab

### Parte II: Funções e elementos de programação

[Criando arquivos de lote](#)  
[Polinômios](#)  
[Elementos de programação](#)  
[Criando funções](#)

---



Este roteiro é a continuação do roteiro apresentado na aula 2 e será utilizado para acompanhar a segunda aula de laboratório de introdução ao MatLab. Nesta aula também é utilizado o programa tutorial “demat2” onde cada um dos itens apresentados aqui serão demonstrados.

### Criando arquivos de lote

Como já foi dito, o ambiente principal do Matlab é um editor de linhas. Então é mais eficiente escrever uma serie de comandos utilizando um editor de texto e salvar este arquivo de lote com extensão m (nome\_do\_arquivo.m). Para executar estes comandos, basta digitar o nome\_do\_arquivo no ambiente do Matlab.

O Matlab não reconhecerá o nome\_arquivo, caso:



O Windows pode estar configurado para colocar uma extensão de texto no final do nome do arquivo ( nome\_do\_arquivo.m.txt ).



O arquivo for criado em um diretório diferente da localização do ambiente do Matlab .

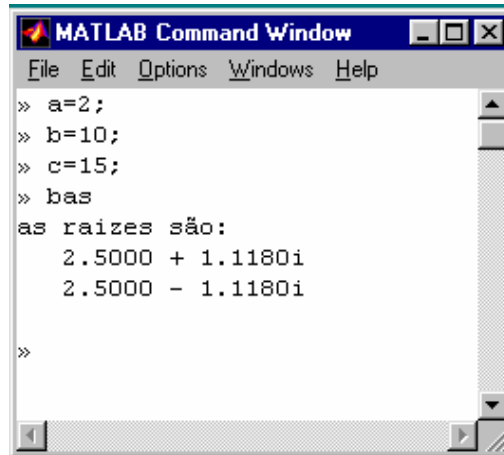
### Exemplo:

A - Escreva os comandos abaixo num arquivo lote chamado bas.m

```
%Este arquivo resolve a equação de Báscara
%  $a*x^2 + b*x + c = 0$ 
d = sqrt(b^2-4*a*c)/(2*a);
e = b/(2*a);
r1 = e + d; % 1º raiz
r2 = e - d; % 2º raiz
sol = [ r1; r2 ];
```



`disp('as raizes são:')` % mostra uma sequência de caracteres  
`disp(sol)` % mostra o conteúdo da variável `sol`  
B – No ambiente do Matlab chame o arquivo `bas.m`.



```
MATLAB Command Window
File Edit Options Windows Help
>> a=2;
>> b=10;
>> c=15;
>> bas
as raizes são:
    2.5000 + 1.1180i
    2.5000 - 1.1180i
>>
```

## Polinômios

No MatLab, um polinômio é representado por um vetor. Para criar um polinômio no MatLab, simplesmente entre com cada coeficiente do polinômio em ordem decrescente.

Exemplo 1:  $x(s) = s^4 + 3s^3 - 15s^2 - 2s + 9$

```
>> x = [1 3 -15 -2 9]
x =
    1     3   -15    -2     9
```

Exemplo 2:  $y(s) = s^2 + 1$

```
>> y = [ 1 0 1 ]
y =
    1     0     1
```

Você pode encontrar o valor do polinômio usando a função **polyval**. Para o exemplo 1, o valor resultante do polinômio para  $s = 2$  é  $-15$ .

```
>> polyval(x, 2)
ans =
   -15
```



Você também pode extrair as raízes do polinômio utilizando a função **roots**:

```
» roots(x)
ans =
-5.5745
 2.5836
 0.7860
-0.7951
```

Pode-se multiplicar e dividir polinômios, usando as funções **conv** e **deconv**. O produto de dois polinômio é feito fazendo a convolução de seus elementos.

```
» z = conv(x,y)
z =
 1  3 -14  1 -6 -2  9
```

A forma analítica do polinômio encontrada é:

$$z(s) = s^6 + 3s^5 - 14s^4 + s^3 - 6s^2 - 2s + 9$$

Dividindo-se o polinômio encontrado z por y obtém-se xx, que é exatamente o polinômio x recuperado, o resto da divisão é nulo:

```
[yy , R] = deconv(z,x)
xx =
 1  3 -15 -2  9
R =
 0  0  0  0  0  0  0
```

Resolvendo um sistema de equações lineares:  
Seja um sistema linear dado por:

$$\begin{aligned} ax + by &= p \\ cx + dy &= q \end{aligned}$$

Podemos rescrever este sistema de forma mais compacta como:

$$AX = B$$

onde:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} p \\ q \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \end{bmatrix}$$



Se a matriz A possui inversa ou seja tem determinante diferente de zero, a solução deste sistema é:

$$X=A^{-1}B .$$

Usando a notação do MatLab temos:  $X = A \setminus B$ . O operador (  $\setminus$  ) serve para resolver sistemas de equações lineares e é diferente do operador (  $/$  ) que realiza a operação de divisão. Vamos ao exemplo:

$$2 \cdot x + 5 \cdot y = 10$$

$$3 \cdot x - 2 \cdot y = 3$$

```
» A = [ 2 5; 3 -2 ];
```

```
» B = [ 10 3]';
```

```
» X = A\B
```

```
X =
```

```
1.8421
```

```
1.2632
```

```
» X = A/B
```

```
??? Error using ==> /
```

```
Matrix dimensions must agree.
```

## Elementos de programação

O MatLab possui um conjunto de comandos para facilitar o desenvolvimento de funções ( nome\_da\_função.m ) pelo próprio usuário, tais como: declarações de controle de programa, armazenamento de dados, manipulação de arquivos, estruturas de dados e outras. Com este conjunto de comandos pode-se falar de uma linguagem de programação MatLab. A maior parte destes comandos tem muita similaridade com a linguagem C.

### O comando if

A expressão **if** usa operadores relacionais e lógicos:

Operadores relacionais	
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que
==	Igual
~=	Diferente



Operadores lógicos	
&	Lógica and
	lógica or
~	Lógica not

Exemplo:

```
» n = 10*rand;    %Gera número aleatório com distribuição uniforme entre 0 e 10
» if n <= 5
    disp('o numero é menor ou igual a 5')
else
    disp('o numero é maior que 5')
end
```

O MatLab permite usar a instrução if de maneira encadeada, com o uso do comando **elseif**.

```
» if n == 5
    disp('o numero é igual a 5')
elseif n < 5
    disp('o numero é menor que 5')
else
    disp('o numero é maior que 5')
end
```

### O comando for

Como exemplo vamos fazer a transposta de uma matriz utilizando o comando for:

```
» H=[ 1  2  3  4  5; ...
      6  7  8  9 10; ...
      11 12 13 14 15 ];

» m = 5;
» n = 3;
» for i = 1: m
    for j = 1 : n
        HT( i, j ) = H( j , i)
    end
end
» disp(HT)
```



### Comandos diversos

O comando **break** interrompe a execução de um *loop*.

O comando **return** interrompe a execução do programa.

O comando **disp** é utilizado para mostrar dados ou caracteres.

```
» b = 1000;  
» disp('b')  
b  
» disp( b )  
1000
```

O comando **error** mostra um conjunto de caracteres com um sinal característico de erro.

O comando **input** é utilizado para entrar com dados na função via valor digitado no ambiente do Matlab.

```
var = input('entre com o valor da variavel var = ')
```

Armazena na variável *var* o valor digitado no Matlab.

### Salvar arquivos de dados

O comando **save** salva em arquivo com extensão *mat* variáveis do ambiente MatLab.

```
save nome_arq variável1 variável 2 ... variável n
```

O comando **load** recupera as variáveis guardadas em *nome\_arq.mat*

```
load nome_arq
```

### Criando funções

Para criar suas próprias funções **m** devemos utilizar um editor de texto, como por exemplo o bloco de notas do windows. A primeira linha da função deve ter a seguinte:

```
function [sai_dados] = nome_função(entra_dados)
```



Como exemplo vamos implementar uma função que transforma coordenadas retangulares em coordenadas polares:

```
function[modulo, angulo] = polar1(x,y)
% converte coordenadas retangulares em coordenadas polares
% o angulo é convertido em graus

if nargin ~= 2 % testa o número de parâmetros de entrada
    error('número de parâmetros indevido')
    return
end

modulo = sqrt(x^2+y^2);
k=180/pi; %constante para converter rad em graus
angulo = k*atan(y/x)
end
```

Como fazer o help da função?

As linhas de comentário abaixo da primeira linha funcionam como o *help* da função:

```
» help polar1
converte coordenadas retangulares em coordenadas polares
o angulo é convertido em graus
```

Como testar o numero de parâmetros de entrada da função?

As variáveis internas do Matlab, **Nargin** e **Nargout**, são variáveis internas que armazenam o número de entradas e de saídas usadas na função declarada no Matlab.

Para usar a função no ambiente do MatLab deve-se digitar:

```
» x1 = 10;
» y1 = 10;
» [m , a] = polar(x1 , y1)
m =
    14.1421
a =
    45
```

Deve-se notar que as variáveis utilizadas na função são variáveis locais, ou seja só existem enquanto a função é executada. Os nomes das variáveis usados na linha de comando não precisam ser os mesmos que os adotados no *script* da função.