



PETEE UFMG



MINICURSO MATLAB 1.0

Apostila

Minicurso de MatLAB 1.0

Curso ministrado pelos alunos do Programa de Educação Tutorial
do Curso de Engenharia Elétrica da Universidade Federal de Minas Gerais
PETEE UFMG

Belo Horizonte, Março de 2017

Sumário

1.	Breve histórico.....	4
2.	Introdução: Estrutura de dados e linguagem	4
a.	Funções Básicas	4
b.	Janelas do programa.....	6
i.	Janela de Comandos – CommandWindow.....	7
ii.	Histórico de Comandos – <i>CommandHistory</i>	8
iii.	Espaço de trabalho - <i>Workspace</i>	8
iv.	Editor de variável – <i>Variable editor</i>	9
v.	Diretório atual – <i>Currentdirectory</i>	9
vi.	Editor de texto.....	9
3.	Programação.....	10
a.	Constantes e variáveis	11
b.	Arranjos e as principais funções relacionadas.....	13
i.	Vetores e cadeias de caracteres (strings).....	13
ii.	Matrizes	18
iii.	Arranjos de mais dimensões.....	22
iv.	Funções importantes para arranjos:	23
4.	Operações e Expressões	25
a.	Operadores	25
b.	Operações vetoriais e matriciais	26
c.	Expressões e operações lógicas.....	27
d.	Constantes e Variáveis.....	29
5.	Funções.....	31
6.	Números e Matrizes Complexas.....	33
7.	Expressões simbólicas	35
8.	Estruturas condicionais	37
a.	Condição “se”	37
i.	Condição “se, caso contrário”	38
ii.	Condição “se, mas ainda se, (caso contrário)”	39
b.	Estruturas de laços e repetições.....	40
i.	Estrutura “para” ou “for”	40

ii.	Estrutura “enquanto” ou “while”	40
iii.	Comando de parada imediata ou interrupção interior – <i>break</i>	41
9.	Gráficos.....	42
a.	Gráficos numéricos bidimensionais.....	42
i.	Função <i>plot</i>	42
ii.	Função <i>fplot</i>	47
b.	Gráficos numéricos tridimensionais.....	49
i.	Função <i>meshgrid</i>	49
ii.	Função <i>surf</i>	49
iii.	Funções <i>plot3</i> , <i>mesh</i> , <i>meshc</i> , <i>meshz</i>	52
10.	Arquivos .m.....	52
11.	Polinomios	55
a.	Raízes	55
b.	Produto de polinômios	56
c.	Divisão de polinômios.....	57
d.	Expansão em frações parciais.....	58
e.	Avaliação de polinômios.....	59
f.	Ajuste polinomial (Método dos mínimos quadrados).....	60
12.	Exercícios	61
a.	Exercícios de Revisão (com solução)	61
b.	Exercícios de Fixação	65
c.	Comandos básicos do Matlab.....	67
d.	Exercícios Desafiadores	71
e.	Exercícios Extras	72
13.	Bibliografia.....	73

1. Breve histórico

A primeira versão do Matlab foi desenvolvida no final dos anos 1970 na Universidade do Novo México com o objetivo de dar acesso a pacotes para Fortran sem requerer conhecimento dessa linguagem. Então, o programa era uma ferramenta bastante primitiva que servia apenas para fazer algumas operações numéricas com matrizes. Não possuía linguagem própria, toolboxes, arquivos ".m", tampouco parte gráfica. O objetivo era apenas auxiliar estudantes em aulas de Análise Numérica e Álgebra Linear. Como se sabe, naquela época, não havia editor de texto, terminais, monitores ou discos de memória. Os programas eram escritos e lidos em cartões perfurados, nos quais a lógica binária era determinada por ter furo ou não. Após processado, a saída do programa perfurava outro cartão.

Em 1983, um engenheiro percebeu o potencial comercial da ferramenta e juntou-se ao desenvolvedor para reescrever o programa. Nessa época, já estavam aparecendo no mercado os primeiros computadores pessoais (PCs). Em 1984, a equipe reescreveu o Matlab em C utilizando um PC de 256kB de memória, sem disco rígido e a empresa Mathworks foi criada. A partir de então, o software foi gradativamente ganhando mercado e novas versões foram criadas.

No ano 2000, quando na versão 6, o Matlab foi novamente reescrito para basear-se no LAPACK, um conjunto de bibliotecas para manipulação de matrizes.

2. Introdução: Estrutura de dados e linguagem

a. Funções Básicas

O Matlab possui uma linguagem própria e homônima, que é bastante auto-suficiente, no sentido de acessar a quase todos os comandos do Matlab via linha de comando.

Para especificar melhor quando se trata do programa e quando se trata da linguagem, faremos uso do artigo definido masculino ao referirmos ao programa, no decorrer dessa apostila.

- Acessando a ajuda:

```
>> help help

HELP Display help text in Command Window.

HELP, by itself, lists all primary help topics. Each primary topic
corresponds to a directory name on the MATLABPATH.

HELP / lists a description of all operators and special characters.

HELP FUN displays a description of and syntax for the function FUN.
When FUN is in multiple directories on the MATLAB path, HELP displays
information about the first FUN found on the path.

(...)
```

Acessar a ajuda é extremamente útil para se informar sobre uso e funcionamento de funções do programa. Entretanto, observe que a ajuda exibida na janela de comando não é necessariamente a mesma do pacote de ajuda do menu do programa. Na janela de comando, são exibidas uma descrição e as formas de uso, apenas textual, além de um link para a página de referência, que é a ajuda do menu. Na página de referência, há descrições mais completas, exemplos e, eventualmente, imagens.

- Para realizar, por linha de comando, uma busca por funções que contenham certa palavra na descrição, utilizar o comando lookfor:

```
>>lookfor concatenation
horzcat          - Horizontal concatenation.
vertcat          - Vertical concatenation.
blkdiag          - Block diagonal concatenation of matrix
input arguments.
hdsCatArray      - Horizontal or vertical array concatenation.
matrxcatmask    - Matrix Concatenation block helper function.
gcat             - Global concatenation      information about
the first FUN found on the path.
```

- O símbolo % faz com que todos os caracteres que o seguem na linha sejam tratados como comentário:

```
>>fprintf 'Programei o Matlab sozinho' % Este texto é um comentário
```

- Reticências “...” fazem com que a próxima linha seja tratada como continuação da atual. Útil para uma visão mais enxuta do código.

```
>> x = 4+5 +2*3*8^2 ...
- 2 +3 +5^2 + 3^2 ...
+2*10^5 - 1000

x =

199413
```

O Matlab possui tipos de estruturas de dados. Todas as variáveis são arranjos. Vetores são arranjos $1 \times n$ e matrizes, arranjos $m \times n$. Sua estrutura de dados foi construída de tal maneira que há alta eficiência em trabalhar com operações matriciais diretamente, sendo melhor do que operar com laços para realização de operações equivalentes. Também é possível operar com arranjos multidimensionais. Há também strings, células, structs, containers e sets.

Também é possível a utilização de classes em objetos, que podem ser chamadas como no seguinte exemplo:

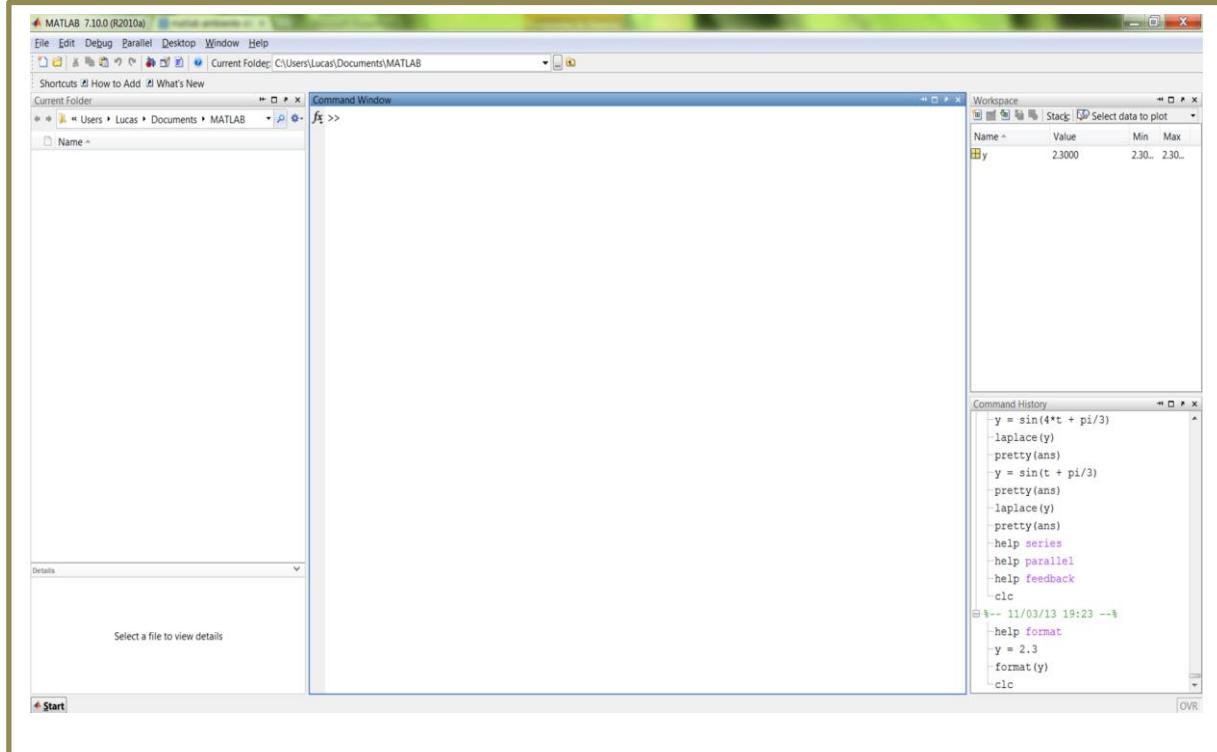
```
object.method();
```

- Utilizando uma função:

Uma função é chamada por seu nome seguido de parêntesis caso haja argumentos. O nome de funções também se diferencia por maiúsculas e minúsculas. Entretanto, se foi chamada, por exemplo, a função *alimentarogatodavovo* e, nos diretórios de trabalho, só existe a função *AlimentarOGatoDaVovo*, é executada a segunda e um aviso (*warning*) é exibido.

A maioria das funções possui argumentos, mas muitos deles são opcionais. Muitas funções aceitam uma quantidade indefinida de argumentos, como, por exemplo, funções de concatenação e de traçar gráficos, nas quais se pode entrar com vários arranjos em sequência.

b. Janelas do programa



Atualmente, as principais janelas do programa são:

i. Janela de Comandos – CommandWindow

É a janela na qual se entra com comandos, na linguagem Matlab. É mais apropriada para comandos rápidos e imediatos, já que existe o editor de texto para programas maiores.



As teclas com setas podem ser usadas para se encontrar comandos dados anteriormente, para execução novamente ou sua reedição. Por exemplo, suponha que você entre com:

`» sen(0)`

Ao apertar a tecla **Enter**, o MATLAB responde com uma mensagem de erro:

`??? Undefined function or variable sen.`

Isto acontece porque para se determinar o seno de um ângulo é necessário digitar em inglês o comando `sin`. Ao invés de rescrever a linha inteira, simplesmente pressione a tecla "seta para cima". O comando errado retorna, e você pode, então, mover o cursor para trás usando a tecla "seta para esquerda" ou o ponto de inserção com o "mouse" ao lugar apropriado para inserir a letra `i`:

`» sin(0)`

`ans =`

`0`

Note que o MATLAB chamou o resultado de `ans` (*answer= resposta*). Além das teclas com setas, pode-se usar outras teclas para reeditar a linha de comando. A seguir é dada uma breve descrição destas teclas:

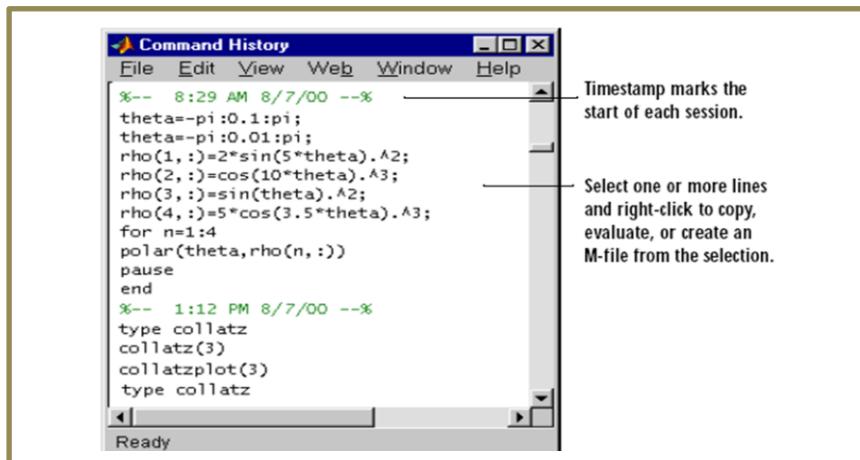
↑	<i>retorna a linha anterior</i>
↓	<i>retorna a linha posterior</i>
←	<i>move um espaço para a esquerda</i>
→	<i>move um espaço para a direita</i>
Ctrl ←	<i>move uma palavra para a esquerda</i>
Ctrl →	<i>move uma palavra para a direita</i>
Home	<i>move para o começo da linha</i>
End	<i>move para o final da linha</i>

Del apaga um caracter a direita

Backspace apaga um caracter a esquerda

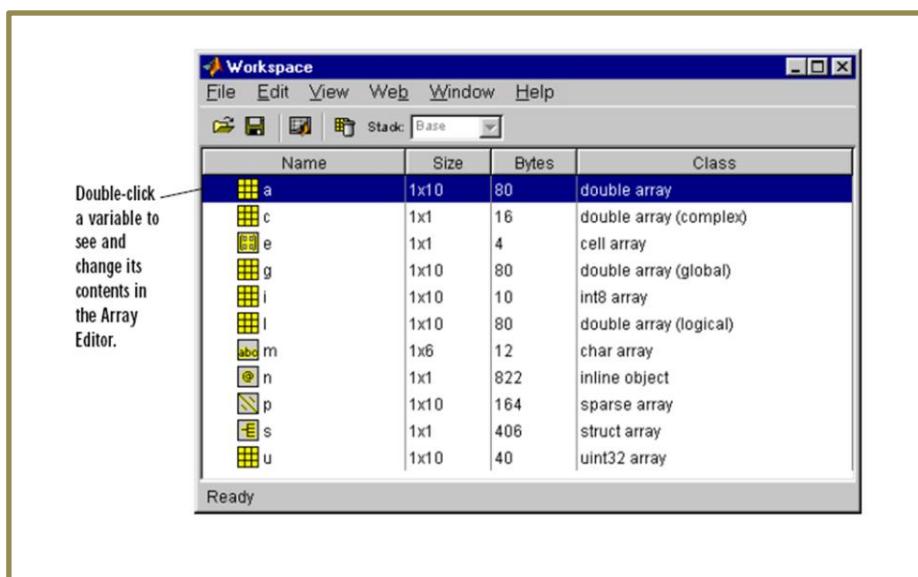
ii. Histórico de Comandos – *CommandHistory*

Exibe o histórico dos comandos entrados na Janela de Comando.



iii. Espaço de trabalho - *Workspace*

Exibe as variáveis e as estruturas existentes na memória. Ao declarar-se uma variável, ela é exibida no espaço de trabalho e, clicando-se duas vezes sobre ela, abre-se uma janela para editar seu valor sem ser pela programação.

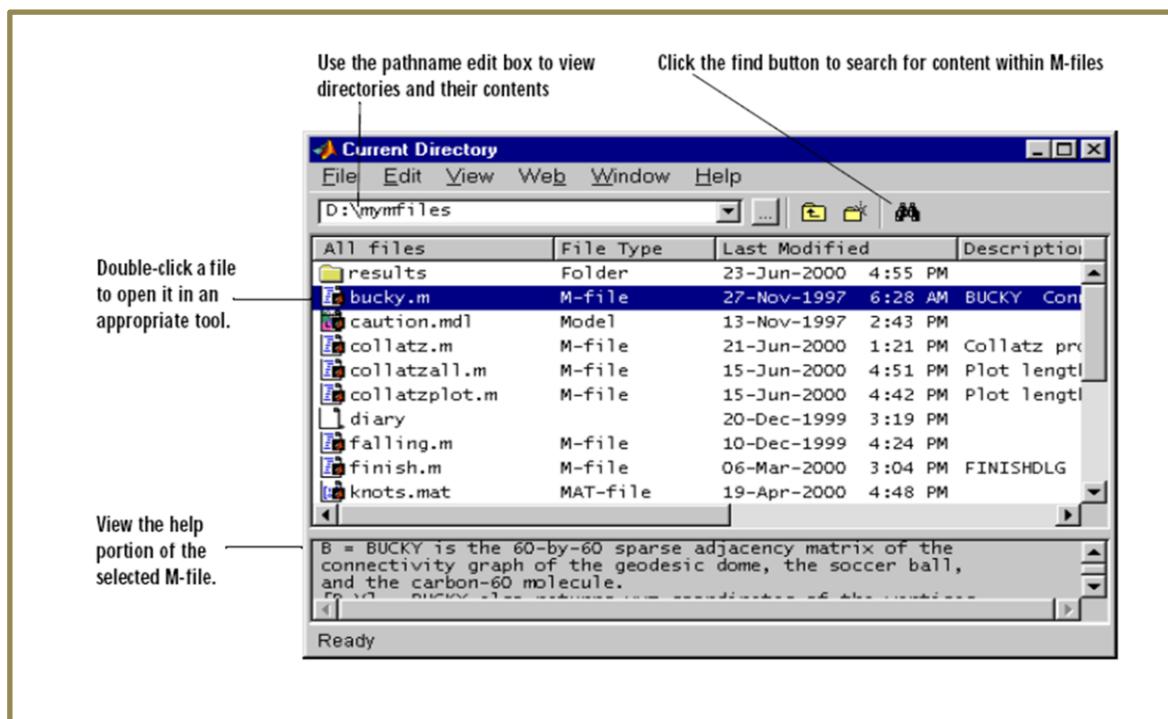


iv. Editor de variável – *Variable editor*

Exibe uma planilha em forma de tabela que permite editarem tempo real valores das variáveis.

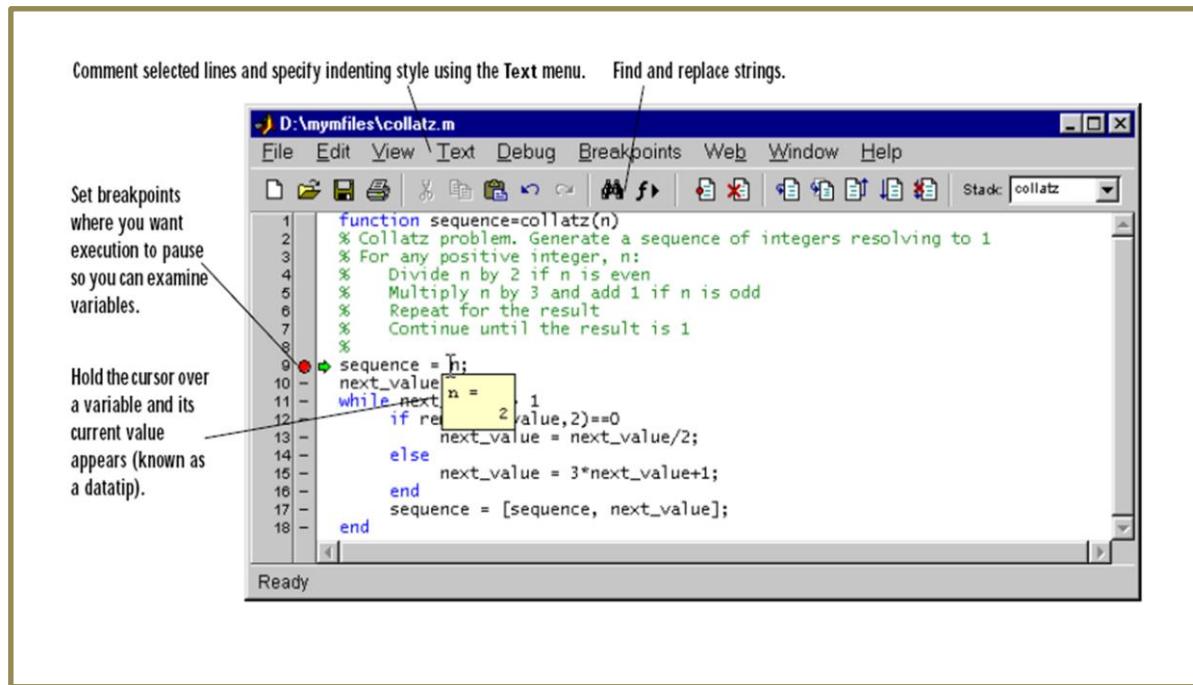
v. Diretório atual – *Current directory*

Exibe a pasta de trabalho atual que é usada como raiz para salvar e carregar arquivos. Abaixo, na janela interna *details*, são exibidos os principais detalhes do arquivo selecionado.



vi. Editor de texto

É aberto ao abrir ou criar um novo arquivo “.m”. Utilizada para construir códigos em linguagem Matlab.



3. Programação

Como foi dito anteriormente, os comandos no Matlab podem ser entrados tanto na janela de comando, quanto no editor de texto.

Na janela de comando, escreve-se uma linha de comando e aperta-se ENTER para executá-la. Caso se queira executar mais linhas, utiliza-se SHIFT+ENTER para passar para a próxima linha sem executar. A execução será feita ao pressionar-se ENTER. Podem-se também alocar vários comandos em uma mesma linha separando-os por ponto-e-vírgula ou vírgula. No Matlab, ponto-e-vírgula suprime o resultado do comando enquanto vírgula ou quebra-de-linha pura exibem o resultado ao final de um comando.

```
>>x = 2;
>>y = 2;
>>z = 2
z =
     2
>>x + y
ans = 4.0000
```

Alguns comandos básicos importantes:

- pwd – mostra o diretório de trabalho atual
- cd <caminho do diretório> – muda para o diretório do caminho.

Ex.:>> cd a;% muda para unidade “A:\”, o drive de disquete (que, hoje, está mais para uma unidade fantasma).

- clear<variável 1><variável 2> ... – apaga a(s) variável(variáveis) especificada(s) do espaço de trabalho, eliminando-a(s) da memória.
- clear – remove todas as variáveis do espaço de trabalho.
- clearvariables – faz a mesma coisa.
- clear global – remove todas as variáveis globais.
- clearfunctions – remove todas as funções compiladas de M e MEX.
- clearall – libera toda a memória apagando todas as variáveis locais e globais, funções.
- clc – limpa a janela de comando.
- close – fecha todas as figuras abertas
- who – lista todas as variáveis.
- whos – lista todas as variáveis, exibindo alguns de seus detalhes.
- format – muda a formatação numérica de exibição das variáveis. Normalmente, elas são exibidas com 4 dígitos decimais.

comando	π	resultado
format short	3.1416	4 dígitos decimais,
format bank	3.14	2 dígitos decimais,
format long	3.14159265358979	14 dígitos decimais,
format short e	3.1416e+00	4 dígitos decimais e expoente,
format long e	3.141592653589793e+00	16 dígitos e expoente,
format hex	400921fb54442d18	hexadecimal,
format +	+	+, – ou branco para valor positivo, negativo ou nulo,
format rat	355/113	aproximação racional,
format compact		suprime linhas em branco,
format loose		mantém linhas em branco,
format		atribui o padrão : short e loose.

Tabela 3.1: O comando format.

Observe que esses comandos são para linha de comando, programação (que é esse tópico 2 dessa apostila). Entretanto, podem-se realizar a maioria delas manualmente, por meio de outras janelas do programa. Por exemplo, para limpar variáveis, pode-se fazê-lo pelo menu ou pela janela *workspace*, onde é possível selecionar variáveis, acessar o *variable editor* para mudar valores, apagá-las e, até mesmo, vê-las! (como no comando Whos).

a. Constantes e variáveis

Uma constante numérica pode ser escrita em MATLAB por uma seqüência de dígitos precedida ou não por sinais de positivo (+) ou negativo (-). Decimais são indicados por ponto-final (.), já que vírgula e ponto-e-vírgula já têm uso na separação de números para criação de arranjos e o Matlab não foi programado para o Sistema Internacional (SI). Potências de 10 podem ser diretamente colocadas em constantes, simplesmente

escrevendo “e<expoente>” ou “d<expoente>”. Por exemplo, “5.33e1” é o mesmo que “53.3” e “1d-3” equivale a “0.001”.

O MATLAB é um interpretador de expressões, que analisa a expressão fornecida e, caso esteja sintaticamente correta, a avalia. Para atribuir uma expressão a uma variável, utiliza-se a forma:

<variável> = <expressão>

Em Matlab, variáveis devem ter nomes compostos por letras, números e subscrito “_”, sendo que o primeiro caractere deve ser uma letra. São permitidos até 31 caracteres em um nome. A partir daí, o restante é ignorado. Há diferenciação de maiúsculas e minúsculas. Logo,

```
>> terra = 4; % onde você planta (ou não)
>> Terra = 1000; % onde você vive (ou não)
>>TERRA = 1E5; % maníaco do caps-lock
>> terra
terra =
4.0000
```

Há constantes e variáveis usadas pelo Matlab:

- Como mencionado anteriormente, a variável de resposta “ans” (answer) possui guardada a última saída gerada de resposta que não foi atribuída a nenhuma variável, ou seja, quando se omite “=” e nome de variável. Ela não é uma árvore ou vetor de histórico de saídas (como em algumas calculadoras, como a HP 50G), portanto só possui a última saída. Ela funciona como outra variável qualquer, mas deve-se lembrar que ela muda de valor a cada comando. Em programas mais longos, é preferível (mas não obrigatório) utilizar outras variáveis auxiliares.
- A constante NaN representa um valor indefinido “não é um número” (Not a Number). Tem tratamento especial em diversas funções, sendo útil para certas aplicações. Voltaremos a apresentá-la ao discutirmos sobre matrizes e vetores.
- Número imaginário: as constantes i e j são definidas como $\sqrt{-1}$. Entretanto, pode-se redefiní-las simplesmente atribuindo-lhes outro valor. A recíproca também é possível:

```
>> u = sqrt(-1) % Adoro usar "u" como número imaginário
u =
0 + 1.0000i
```

O operador aspas simples ‘ em um número complexo retorna seu conjugado.

Observe que, na resposta, um número complexo é sempre exibido da forma $a + bi$.

- `eps` – menor número tal que, quando adicionado a 1, cria um número de ponto flutuante maior do que 1. Seu valor é $\text{eps} = 2^{-52} \cong 2,204 \times 10^{-16}$, para computadores com aritmética de ponto flutuante do IEEE.
- `realmin` – menor número positivo de ponto flutuante IEEE. Valor: $\text{realmin} \cong 2,2251 \times 10^{-308}$.
- `realmax` – maior número positivo de ponto flutuante IEEE representável. Valor: $\text{realmax} \cong 1,7977 \times 10^{308}$.
- `pi` – definida como a razão entre a circunferência e o diâmetro de um círculo. Valor: $\text{pi} = 3,14159265 \dots$

b. Arranjos e as principais funções relacionadas

Como previamente comentado, em Matlab, todas as variáveis são estruturadas como arranjos de n dimensões. A partir daí, podemos trabalhar variáveis, vetores e matrizes como arranjos de 0, 1 e 2 dimensões, respectivamente. Arranjos de mais dimensões são freqüentemente úteis, mas muitas funções não os interpretam. Em arranjos numéricos, podemos utilizar inteiros, ponto flutuante (double) e números complexos, também de ponto flutuante.

i. Vetores e cadeias de caracteres (strings)

Vetores são sequências lineares discretas de informação, nos quais cada informação ocupa uma posição. Em um vetor numérico, todos os seus elementos são valores numéricos. Em Matlab, há várias maneiras de se criar um vetor.

- Especificando cada elemento – vetor linha:

```
>> v = [4 2 5.36] % Espaço ou vírgula separam elem. na linha
v =
    4.0000    2.0000    5.3600
```

- Vetor coluna:

```
>> w = v' % Podemos obter o vetor transposto de v...
w =
    4.0000
    2.0000
    5.3600
>> % ...ou criar um vetor coluna diretamente:
>> u = [4; 2; 5.36] % pto. e vírgula separam colunas
u =
    4.0000
    2.0000
    5.3600
```

Para separar colunas, podemos, também, saltar linhas diretamente, usando shift+enter.

Observe que o operador de transposição é o mesmo de conjugado complexo. Logo, o efeito dele sobre vetores (ou matrizes) com valores complexos é de transposição e conjugado complexo, ao mesmo tempo:

```
>> x = [2+5i, 4-3i]
x =
2.0000 + 5.0000i 4.0000 - 3.0000i

>> x'
ans =
2.0000 - 5.0000i
4.0000 + 3.0000i
```

- Substituindo um elemento:

```
>> v(2) = 3
v =
4.0000 3.0000 5.3600
```

- Estendendo um vetor:

```
>> v(5) = -7
v =
4.0000 3.0000 5.3600 0 -7.0000
```

- Removendo um elemento:

```
>> v(2) = [] % "[]" indica arranjo vazio ou nulo
v =
4.0000 3.0000 5.3600
```

- Concatenando ou inserindo utilizando a função cat:

Forma da função: cat(DIM, A1, A2, A3, ...)

Dim = dimensão;

A1, A2, A3, ... = arranjos

```
>> u = [-0.6 0.03]
>> w = cat(2, [4 3 5.36], u)
w =
    4.0000    3.0000    5.3600   -0.6000    0.0300
```

- Seqüências: podemos entrar com seqüências numéricas

```
>> seq1 = (1:5) % Seqüência natural
seq1 =
    1     2     3     4     5
```

```
>> seq2 = (1:0.5:3) % Seq. com valor de passo especificado
seq2 =
    1.0000 1.5000    2.0000    2.5000    3.0000
```

```
>> seq3 = (8:-2:-3) % Seq. com valor de passo especificado
Seq3 =
    8     6     4     2     0    -2
```

Podemos usar também `linspace(<valor inicial>, <valor final>, <número de elementos>)`:

```
>> seq4 = linspace(-2, 6, 10) % Seq. com nº de elem.especific.
seq4 = -2.0000  -0.4000    1.2000    2.8000    4.4000    6.0000
```

Para sequências de 1s ou 0s, basta fazer:

```
>> seq3 = ones(1, 4) % Seqüência de 1s
seq3 =
    1     1     1     1
>> seq4 = zeros(1, 5) % Seqüência de 0s
seq4 =
    0     0     0     0     0
```

Essas funções são próprias para gerar matrizes de 2 dimensões, portanto, devemos especificar o número de linhas e de colunas. Por isso, o (1, 4) e (1, 5), respectivamente. Da mesma forma, podemos criar também seqüências de valores não numéricos NaN, seqüências de números pseudo-aleatórios seguindo certa distribuição etc..

Números pseudo-aleatórios:

- Função `rand` – gera números pseudo-aleatórios de distribuição uniforme entre 0 e 1.
- Função `randn` – gera números pseudo-aleatórios de distribuição normal com média zero e variância unitária.
- Função `randi([IMIN, IMAX], n)` – gera n números inteiros pseudo-aleatórios de distribuição uniforme em uma dada faixa. Os números podem se repetir. `IMIN` (menor valor) é opcional. Se for omitido, o menor valor passa a ser 1.
- Função `randperm` – gera uma permutação pseudo-aleatória de uma seqüência de inteiros. Nesse caso, não pode haver repetição. Por exemplo, `randperm(5)` pode ser [3 1 2 4].

Cadeias de caracteres ou strings também são vetores, já que formam uma seqüência linear de caracteres. Entretanto, não são numéricos e o Matlab as trata como uma outra estrutura. O Matlab possui várias funções para tratá-las. Podem ser encontradas ao entrar com:

```
>> help strfun
```

- Criando e manipulando uma simples cadeia de caracteres:

```
>> a = 'eu tenho um gato'
a =
eu tenho um gato
```

Como na maioria das linguagens de programação, podemos tratar letras por sua posição:

```
>>a(13) = 'p' % substituindo o 13º elemento
a =
eu tenho um pato
>>a(13) = 'a' + 17 % r é a 18ª letra, distando 17 pos. de a
a =
eu tenho um rato
```

Como não somos obrigados a saber a posição das letras em ASCII, para fazer “operações alfabéticas”, podemos utilizar letras maiúsculas ou minúsculas ou números como referência, como se observa no exemplo.

Abaixo, como referência, as tabelas de código ASCII e de ASCII estendido, que inclui acentos:

representação decimal dos caracteres																
33	!	45	-	57	9	69	E	81	Q	93]	105	i	117	u	
34	"	46	.	58	:	70	F	82	R	94	^	106	j	118	v	
35	#	47	/	59	;	71	G	83	S	95	-	107	k	119	w	
36	\$	48	0	60	<	72	H	84	T	96	'	108	l	120	x	
37	%	49	1	61	=	73	I	85	U	97	a	109	m	121	y	
38	&	50	2	62	>	74	J	86	V	98	b	110	n	122	z	
39	,	51	3	63	?	75	K	87	W	99	c	111	o	123	{	
40	(52	4	64	@	76	L	88	X	100	d	112	p	124		
41)	53	5	65	A	77	M	89	Y	101	e	113	q	125	}	
42	*	54	6	66	B	78	N	90	Z	102	f	114	r	126	~	
43	+	55	7	67	C	79	O	91	[103	g	115	s			
44	,	56	8	68	D	80	P	92	\	104	h	116	t			

Tabela 2.13: Caracteres em código ASCII.

representação decimal dos caracteres																
192	À	200	È	208	Ð	216	Ø	224	à	232	è	240	ð	248	ø	
193	Á	201	É	209	Ñ	217	Ù	225	á	233	é	241	ñ	249	ù	
194	Â	202	Ê	210	Ò	218	Ú	226	â	234	ê	242	ò	250	ú	
195	Ã	203	Ë	211	Ó	219	Û	227	ã	235	ë	243	ó	251	û	
196	Ä	204	Ï	212	Ô	220	Ü	228	ä	236	ï	244	ö	252	ü	
197	Å	205	Í	213	Ӯ	221	Ý	229	å	237	í	245	ö	253	ý	
198	Æ	206	Ї	214	Ӱ	222		230	æ	238	ї	246	ö	254		
199	Ҫ	207	Ӯ	215	Ӳ	223	ڦ	231	ڻ	239	ڻ	247	ڻ	255	ڻ	

Tabela 2.14: Caracteres em código ASCII estendido do MATLAB.

Exemplo de uso, com a função *setstr*:

```
>> m = [70 117 110 231 227 111]
m =
    70    117    110    231    227    111
>> fun = setstr(m)
fun =

```

Função

Cadeias de caracteres podem ser interpretadas como constantes numéricas pela função *str2num* (*stringtonumber*), como abaixo:

```
>> texto = '1.892e03 + 5684d-01i'
texto =
1.892e03 + 5684d-01i

>> x = str2num(texto)
x =
1.8920e+003 +5.6840e+002i
```

O contrário pode ser feito pela função *num2str (numbertostring)*. Tal função pode ser de grande utilidade para, por exemplo, exibir valores em rótulos de gráficos, que suportam cadeias de caracteres como entrada.

```
>>v = 1.123623746;
>> ['velocidade = ', num2str(v), 'm/s']
ans =

velocidade = 1.1236m/s
```

Muitas outras funções, muitas delas encontradas em toolboxes, precisam de valores ou expressões em forma de cadeias de caracteres, que podem incluir valores numéricos. Portanto, em termos de importância, as *strings* do MATLAB vão bastante além de apenas textos e palavras que aparecerão na tela de execução.

ii. Matrizes

Matrizes são seqüências retangulares discretas de informação, nas quais cada informação ocupa uma posição. Em uma matriz numérica, todos os seus elementos são valores numéricos. Em Matlab, há várias maneiras de se criar um matriz, mas não há diferenças entre criação de matrizes e vetores.

- Especificando cada elemento:

```
>> A = [4 2 3; -9 -3 6] %
A =
4      2      3
-9     -3      6
```

- Transpondo:

```
>> B = A' % Matriz transposta
B =
    4     -9
    2     -3
    3      6
```

- Substituindo um elemento:

```
>> A(1, 2) = 0 % Linha 1, coluna 2. O mesmo que A(3) = 0
A =
    4     0     3
   -9    -3     6
```

Lembrar que o operador de transposição ‘ também faz o conjugado complexo dos valores.

- Estendendo uma matriz:

```
>>A(3, 8) = 0 % A matriz é redimensionada para receber o elem
A =
    4     0     3     0     0     0     0     0
   -9    -3     6     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

- Substituindo um conjunto de elementos ou submatriz:

```
>>A(8, 3) = 0 % A matriz é redimensionada para receber o elem
A =
    4     0     3     0     0     0     0     0
   -9    -3     6     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Observe que podemos utilizar dois-pontos “:” para selecionar um trecho extenso.

- $i:j$ – seleciona de i até j em um vetor ou subvetor de uma matriz. (i, j inteiros.)
- $i:\text{passo}:j$ – seleciona de i até j pulando passo .
- $:$ - seleciona um vetor ou subvetor por completo.

Veja:

```
>>A(1:3, 5:6) = B % B criada anteriormente
A =
    4     0     3     0     4    -9     0     0
   -9    -3     6     0     2    -3     0     0
     0     0     0     0     3     6     0     0
```

- Usando ":" para substituir uma linha:

```
>>A(3, :) = 4*ones(1,8) % Todas as colunas da linha 3
A =
    4     0     3     0     4    -9     0     0
   -9    -3     6     0     2    -3     0     0
    4     4     4     4     4     4     4     4
```

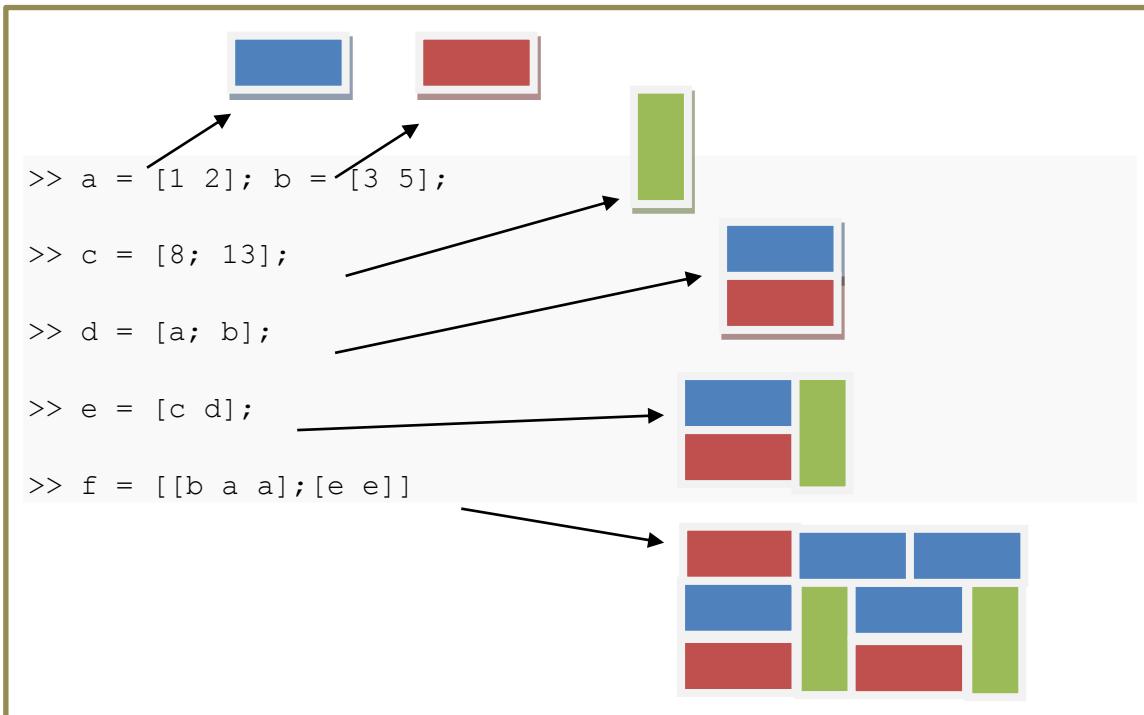
- O mesmo vale para selecionar:

```
>> C = A(:, 3:5) % Todas as colunas da linha 3
C =
    3     0     4
    6     0     2
    4     4     4
```

- Removendo linhas ou colunas:

```
>> C(:, 2) = [] % "[]" indica um arranjo vazio
C =
    3     4
    6     2
    4     4
```

- Concatenação direta:



- Concatenando ou inserindo utilizando a função *cat*:

Forma da função: `cat(DIM, A1, A2, A3, ...)`

`Dim` = dimensão;

`A1, A2, A3, ...` = arranjos

```

>> A = [3 6; 4 6];
>> B = [0 2; 5 9];
>> C = cat(1, A, B)% O mesmo que C = (A; B)
C =
    3     6
    4     6
    0     2
    5     9

```

```

>> C = cat(2, A, B)% O mesmo que C = (A, B)
C =
    3     6     0     2
    4     6     5     9

```

- Seqüências numéricas:

Funcionam da mesma forma que as vistas para vetores.

Exemplo: gerando uma matriz 3x3 de 5s.

```
>> A = 5*ones(3) % O mesmo que A = 5*ones(3, 3)
A =
    5     5     5
    5     5     5
    5     5     5
```

- Matriz identidade:

Para criá-la utilizamos a função “*eye(n)*”:

```
>> I3 = eye(3) % Matriz identidade 3x3
I3 =
    1     0     0
    0     1     0
    0     0     1
```

iii. Arranjos de mais dimensões

Podemos criar arranjos de n dimensões para estruturar melhor certos dados.

- Arranjo de 3 dimensões:

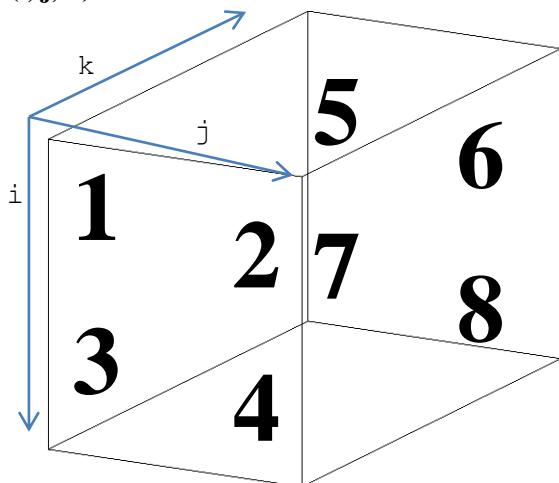
```
>> A = [1 2; 3 4]; B = [5 6; 7 8]; % Duas matrizes
>> C = cat(3, A, B)% Criando o arranjo de 3 dimensões
C(:,:,1) =
    1     2
    3     4
C(:,:,2) =
    5     6
    7     8
```

Observe que o arranjo acima possui 3 dimensões e é da forma 2x2x2, sendo composto por submatrizes.

Por exemplo, a submatriz $C(:,:,1)$ é $[3, 6; 4, 6]$.

Podemos fatiar esse arranjo cúbico em outras submatrizes:

$C(i, j, k)$:



```
>> D = C(:,1,:)% Sub-arranjo é criado com i e k
D(:,:,1) =
```

```
1
```

```
3
```

```
D(:,:,2) =
```

```
5
```

```
7
```

```
>>D = cat(2, C(:,1,1), C(:,1,2))% Na forma de matriz
```

```
D =
```

1	5
3	7

iv. Funções importantes para arranjos:

- **Size** – gera um vetor com o tamanho do arranjo em linhas, colunas, etc.

```
>>Tamanho_C = size(C) % Arranjo C do exemplo acima
Tamanho_C =
```

```
2      2      2
```

- ***Det*** – calcula o determinante de uma matriz:

```
>> A = [2 6; 5 9] %
>>det_A = det(A)
det_A =
-12
```

- ***Trace*** – Calcula o traço da matriz:

```
>>tr_A = trace(A)
tr_A =
11
```

- ***Sum*** – gera um arranjo de dimensão $n-1$ com a soma de todos os elementos de um arranjo:

```
>> somala100 = sum(1:100)
Somala100 =
5050
>>soma_mat = sum([1 2; 3 4])
soma_mat =
4     6
>> soma_arr3D = sum(C) % C do exemplo de arranjos de 3dim
Soma_arr3D(:,:,1) =
4     6
Soma_arr3D(:,:,2) =
12     14
```

Da mesma forma, as seguintes funções podem ser usadas:

- ***Mean*** – arranjo $n-1$ com a média aritmética de cada vetor
- ***Std*** – arranjo $n-1$ com o desvio padrão de cada vetor
- ***Prod*** – arranjo $n-1$ com o produto dos elementos de cada vetor
- ***Max*** – arranjo $n-1$ com o maior elemento de cada vetor
- ***Min*** – arranjo $n-1$ com o menor elemento de cada vetor
- ***Sort*** – arranjo $n-1$ com cada vetor ordenado

4. Operações e Expressões

a. Operadores

O MATLAB oferece as seguintes operações aritméticas básicas:

operação	expressão	operador	exemplo
adição	$a + b$	+	$1+2$
subtração	$a - b$	-	$5.1-4.7$
multiplicação	$a \times b$	*	$6*9.98$
divisão	$a \div b$	/ ou \	$6/7 \quad 5\backslash 3$
potenciação	a^b	^	2^10

Operações aritméticas básicas do MATLAB

```
>> 2 + 3/4*5
ans =
5.7500
>>
```

A ordem nas expressões segue a ordem matemática - potência, seguida da multiplicação e da divisão, que por sua vez são seguidas pelas operações de adição e subtração. Parêntesis podem ser usados para alterar esta ordem. Neste caso, os parêntesis mais internos são avaliados antes dos mais externos.

```
>> x = 2+3*4/2
x =
8
>> y = (2+3)*4/2
y =
10
```

Ordem de precedência geral de operadores no MATLAB:

ordem de precedência	operadores
1 ^a	$^ \cdot \cdot ^ \cdot ^ \cdot ^ \cdot$
2 ^a	$* / \backslash \cdot \cdot / \backslash$
3 ^a	$+ - \sim +(\text{unário}) -(\text{unário})$
4 ^a	$: > >= < <= == \sim =$
5 ^a	$ \&$

Ordem de precedência das operações aritméticas e lógicas

Pode-se alterar a precedência de operadores em determinada expressão por meio do uso de parênteses.

b. Operações vetoriais e matriciais

Seja $a = [a_1 \ a_2 \ \dots \ a_n]$, $b = [b_1 \ b_2 \ \dots \ b_n]$ e c um escalar		
operação	expressão	resultado
adição escalar	$a+c$	$[a_1+c \ a_2+c \ \dots \ a_n+c]$
adição vetorial	$a+b$	$[a_1+b_1 \ a_2+b_2 \ \dots \ a_n+b_n]$
multiplicação escalar	$a*c$	$[a_1*c \ a_2*c \ \dots \ a_n*c]$
multiplicação vetorial	$a.*b$	$[a_1*b_1 \ a_2*b_2 \ \dots \ a_n*b_n]$
divisão à direita	$a./b$	$[a_1/b_1 \ a_2/b_2 \ \dots \ a_n/b_n]$
divisão à esquerda	$a.\backslash b$	$[b_1/a_1 \ b_2/a_2 \ \dots \ b_n/a_n]$
potenciação	$a.^c$	$[a_1^c \ a_2^c \ \dots \ a_n^c]$
	$c.^a$	$[c^a_1 \ c^a_2 \ \dots \ c^a_n]$
	$a.^b$	$[a_1^b \ a_2^b \ \dots \ a_n^b]$

Operações vetoriais básicas

Observe que as operações de multiplicação, potência e divisão acima são ponto a ponto, por isso, precedidas por um ponto na expressão. Isso, para que o Matlab trate os arranjos elemento por elemento na respectiva posição. As operações sem ponto são operações matriciais, como se verá abaixo.

Seja c um escalar e $A = [a_{11} \ a_{12} \ \dots \ a_{1n}; \ a_{21} \ a_{22} \ \dots \ a_{2n}; \ \dots; \ a_{m1} \ a_{m2} \ \dots \ a_{mn}]$ $B = [b_{11} \ b_{12} \ \dots \ b_{1n}; \ b_{21} \ b_{22} \ \dots \ b_{2n}; \ \dots; \ b_{m1} \ b_{m2} \ \dots \ b_{mn}]$		
operação	expressão	resultado
adição escalar	$A+c$	$a_{ij} + c$
adição matricial	$A+B$	$a_{ij} + b_{ij}$
multiplicação escalar	$A*c$	$a_{ij} * c$
multiplicação por elemento	$A.*B$	$a_{ij} * b_{ij}$
multiplicação matricial	$A*B$	AB
divisão por elemento	$A./B$	a_{ij}/b_{ij}
divisão à esquerda	$A.\backslash B$	b_{ij}/a_{ij}
potenciação	$A.^c$	a_{ij}^c
	A^c	A^c
	$c.^A$	$c^{a_{ij}}$
	$A.^B$	$a_{ij}^{b_{ij}}$

Operações matriciais básicas

Comparação entre operações matriciais e operações ponto a ponto:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> B = [10\4 3*3 sqrt(9); 10:-5:0; linspace(40,100,3)];
>> C = A.*B
C =
    0.4000    18.0000    9.0000
    40.0000   25.0000     0
   280.0000   560.0000  900.0000
>> D = A*B
D =
    140.4000   229.0000  303.0000
    291.6000   481.0000  612.0000
    442.8000   733.0000  921.0000
>> A_cubo = A^3 % Equivalente a A_cubo = A*A*A
A_cubo =
    468576      684
    106213051548
    1656        2034        2412
```

c. Expressões e operações lógicas

Note que (=) é usado para atribuição de um valor a uma variável, enquanto que (==) é usado para comparação de igualdade. No MATLAB os operadores relacionais podem ser usados para comparar vetores de mesmo tamanho ou escalares. O resultado de uma relação ou de uma expressão lógica é verdadeiro ou falso; contudo, no MATLAB o resultado é numérico, sendo que 1 significa verdadeiro e 0 significa falso. Por exemplo:

```
>> 5>8
ans =
    0
>> 5==5
ans =
    1
```

operador relacional	descrição
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
~=	diferente de

Operadores relacionais do MATLAB

```
>> u = 1:10
>> v = u > 6
v =
    0     0     0     0     0     0     1     1     1     1
```

- Atenção: a comparação é feita por elemento:

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

ans =

0     0     0
1     1     1
0     1     0
```

- Útil: remover linhas ou colunas sob certa condição:

Removendo linhas nas quais a 1^a coluna de A é zero.

```
>> A = [1 -7; 0 0; -2 -6; 0 0; 0 8; -3 0; 1 2]
A =

    1 -7
    3 -8
   -2 -6
    3 4
    3     8
   -3     0
    1     0
>> B = A(A(:,1) ~= 3, :)
% Remove as linhas que recebem 0
B =

    1 -7
   -2 -6
   -3     0
    1     2
```

Observe a remoção das linhas que contêm 3 na primeira coluna de A. À variável B, atribuiu-se a matriz A, mas selecionando apenas as linhas de A que satisfazem à condição dada – 1^a coluna diferente de 3 – . A seleção é feita pelos 1s ou 0s que aparecem no vetor de condição. Da mesma forma, a condição poderia ter sido checada em outro vetor qualquer do espaço de trabalho, desde que satisfaça à condição de mesmo tamanho – número de elementos.

Operadores lógicos permitem a ligação ou negação de relações lógicas. Em MATLAB:

operador lógico	Descrição	uso
&	e	conjunção
	ou	disjunção
~	não	negação

Operadores lógicos do MATLAB

- Uso de uma relação lógica sob mais de uma condição:

```
>> u = 1:10
>> v = (u <= 3) & (u > 6)
v =
    1     1     1     0     0     0     1     1     1     1
>> w = ~v
w =
    0     0     0     1     1     1     0     0     0     0
```

d. Constantes e Variáveis

O MATLAB faz cálculos simples e científicos como uma calculadora. Para tal, os comandos devem ser digitados diretamente no *prompt* (>) do MATLAB, já que este se trata de um software interativo. Por exemplo:

```
>> 3*25 + 5*12
ans =
135
```

Observe que no MATLAB a multiplicação tem precedência sobre a adição. Uma constante numérica no MATLAB é formada por uma sequência de dígitos que pode estar ou não precedida de um sinal positivo (+) ou negativo (-) e pode conter um ponto decimal (.). Esta sequência pode terminar ou não por uma das letras e, E, d ou D, seguida de outra sequência de dígitos precedida ou não de um sinal de (+) ou de (-). Esta segunda sequência é a potência de 10 pela qual a primeira sequência deve ser multiplicada. Por exemplo,

» *1.23e-1*

significa 0,123.

O formato em que uma constante numérica é mostrada no MATLAB segue, como opção *default*, os seguintes critérios: se um resultado é inteiro, o MATLAB mostra o número como inteiro; quando o resultado é real, o MATLAB mostra o número com 4 dígitos à direita do ponto decimal; se os dígitos do resultado estiverem fora desta faixa, o MATLAB mostra o resultado usando a notação científica. Este default pode, entretanto, ser modificado utilizando-se o **Numeric Format** do item **Options** na barra de menus.

Comando	Formato	Comentário
<code>format short</code>	33.5000	4 dígitos decimais (default)
<code>format long</code>	33.50000000000000	16 dígitos
<code>format short e</code>	3.3500e+001	5 dígitos mais expoente
<code>format long e</code>	3.35000000000000e+001	16 dígitos mais expoente
<code>format hex</code>	4040c00000000000	Hexadecimal
<code>format bank</code>	33.50	2 dígitos decimais
<code>format +</code>	+	positivo, negativo ou zero
<code>format rat</code>	67/2	Racional

Alternativamente, você pode usar **variáveis** para armazenar informação. Por exemplo:

```
>> q1=3, p1=25, q2=5, p2=12
```

```
q1 =
```

```
3
```

```
p1 =
```

```
25
```

```
q2 =
```

```
5
```

```
p2 =
```

```
12
```

```
>> total=q1*p1+q2*p2
```

```
total =
```

```
135
```

Primeiro, criamos quatro variáveis, q1, p1, q2 e p2, atribuindo a elas os seus valores respectivos. Observe que o sinal de igual (=) aqui significa atribuição. O que estiver à direita do sinal de igual é “colocado” na variável que estiver à esquerda. Finalmente, criamos uma variável chamada total que recebeu o total da compra.

Regras de construção das variáveis	Comentários/Exemplos
Variáveis com letras minúsculas e maiúsculas são diferentes, mesmo que consistam das mesmas letras.	Total, total, TOTAL e ToTaL são variáveis diferentes.
As variáveis podem consistir de até 19 caracteres	Sdtf65erkjh3448bafg
As variáveis devem começar com uma letra e pode ser seguida de letras, números ou subscrito ().	Var_2 X34 a_b_c

As variáveis podem ser alteradas a qualquer momento, bastando apenas atribuir-lhe algum novo valor.

Alguns nomes especiais devem ser evitados no momento da criação de uma variável, pois elas são variáveis predefinidas do MATLAB. Estas são:

pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, 2^{-52}
realmin	Smallest floating-point number, 2^{-1022}
realmax	Largest floating-point number, $(2-\epsilon)2^{1023}$
Inf	Infinity
NaN	Not-a-number

5. Funções

O MATLAB tem uma série de funções científicas pré-definidas. A palavra função no MATLAB tem um significado diferente daquele que tem na Matemática. Aqui, função é um comando, que pode ter alguns argumentos de entrada e alguns de saída. Algumas dessas funções são intrínsecas, ou seja, não podem ser alteradas pelo usuário. Outras funções estão disponíveis em uma biblioteca externa distribuídas com o programa original (MATLAB TOOLBOX), que são na realidade arquivos com a extensão ".m" criados a partir das funções intrínsecas.

A biblioteca externa (MATLAB TOOLBOX) pode ser constantemente atualizada à medida que novas aplicações são desenvolvidas. As funções do MATLAB, intrínsecas ou arquivos ".m", podem ser utilizadas apenas no ambiente MATLAB.

Abaixo o exemplo do uso da função *sqrt()* e *acos()*:

```
>> x = sqrt(2)/2

x =
0.7071

>> y = acos(x)

y =
0.7854

>> y_graus = y*180/pi

y_graus =
45
```

Abaixo segue uma lista de funções numéricas básicas:

função	Descrição	função	Descrição
trigonométricas			
acos	arco co-seno	cos	co-seno
acosh	arco co-seno hiperbólico	cosh	co-seno hiperbólico
acot	arco co-tangente	cot	co-tangente
acoth	arco co-tangente hiperbólica	coth	co-tangente hiperbólica
acs	arco co-secante	csc	co-secante
acsch	arco co-secante hiperbólica	csch	co-secante hiperbólica
asec	arco secante	sec	secante
asech	arco secante hiperbólica	sech	secante hiperbólica
asin	arco seno	sin	seno
asinh	arco seno hiperbólico	sinh	seno hiperbólico
atan	arco tangente	tan	tangente
atanh	arco tangente hiperbólica	tanh	tangente hiperbólica
atan2	arco tangente de 4 quadrantes		
exponenciais			
exp	exponencial	log10	logaritmo decimal
log	logaritmo natural	sqrt	raiz quadrada
complexas			
abs	valor absoluto	imag	parte imaginária do complexo
angle	ângulo de fase	real	parte real do complexo
conj	complexo conjugado		
numéricas			
ceil	arredonda em direção a $+\infty$	lcm	mínimo múltiplo comum
fix	arredonda em direção a 0	rem	resto de divisão
floor	arredonda em direção a $-\infty$	round	arredonda em direção ao inteiro mais próximo
gcd	máximo divisor comum	sign	sinal

Funções matemáticas elementares

```
>> A = [0:5:10; cos(10*pi) 2 8; 4 sinh(2) 5\9];
>> B = [(6-8) ];
```

Comandos de busca e ajuda

Como dito anteriormente, é possível utilizar o comando help para obter maiores informações sobre uma determinada função, desde de que seja conhecida seu nome.

```
>> help acos
acos    Inverse cosine, result in radians.
acos(X) is the arccosine of the elements of X. Complex
results are obtained if ABS(x) > 1.0 for some element.

See also cos, acosd.

Overloaded methods:
codistributed/acos
gpuArray/acos

Reference page in Help browser
doc acos
```

O Comando *lookfor* provê assistência pela procura através de todas as primeiras linhas dos tópicos de auxílio do MATLAB e retornando aquelas que contenham a palavra-chave especificada. O interessante deste comando é que a palavra chave não precisa ser um comando do MATLAB.

```
>> lookfor integrate
polyint                                - Integrate polynomial analytically.
findslobj                             - creates and manages the Simulink/Stateflow Integrator
intdump                               - Integrate and dump.
fnint                                 - Integrate a function.
```

Apesar da palavra *integrate* não ser um comando do MATLAB, ela foi encontrada na descrição de 3 comandos.

6. Números e Matrizes Complexas

Números complexos são permitidos em todas operações e funções no MATLAB. Os números complexos são introduzidos usando-se as funções especiais **i** e **j**.

Exemplo:

```
>> z1 = 3 + 4*i
z1 =
3.0000 + 4.0000i

>> z2 = 4 + j*5
z2 =
4.0000 + 5.0000i

>> z3 = 3.3*exp(i*.771)
z3 =
2.3668 + 2.2996i
```

Identidade de Euler

Ela relaciona a forma polar de um número complexo com a sua forma retangular.

$$M\angle\theta \equiv M * e^{j\theta} = a + j * b$$

Onde,

$$M = \sqrt{a^2 + b^2}$$

$$\Theta = \tan^{-1} \frac{b}{a}$$

$$a = M * \cos \theta$$

$$b = M * \sin \theta$$

No MATLAB, a conversão entre as formas polar e retangular de um número complexo utiliza as seguintes funções:

- *real*: parte real de um número complexo
- *imag*: parte imaginária de um número complexo
- *abs*: calcula o valor absoluto ou módulo de um número complexo
- *angle*: calcula o ângulo de um número complexo

Exemplo:

```
» x=1-4i
x =
1.0000 - 4.0000i
» a=real(x)
a =
1
» b=imag(x)
b =
-4
» M=abs(x)
M =
4.1231
» theta=angle(x)*180/pi
theta =
-75.9638
```

7. Expressões simbólicas

No MATLAB, é possível manipularmos expressões que além de números e variáveis numéricas, contêm também variáveis simbólicas. Por exemplo, iremos simplificar a expressão $\sin^2 x + \cos^2 x$.

```
>> syms x
>> simplify(sin(x)^2 + cos(x)^2)

ans =
1
```

Primeiro precisamos dizer ao MATLAB que x é uma variável simbólica, depois pedimos para simplificar a expressão que envolve x usando a função *simplify*.

Uma vez definido que x é uma variável simbólica, podemos definir expressões que envolvem esta variável. Por exemplo, dadas as funções $f(x) = 2x^2 + 3x - 5$ e $g(x) = x^2 - x + 7$ podemos fazer uma série de operações algébricas envolvendo estas funções.

```
>> syms x
>> f = 2*x^2 + 3*x - 5; g = x^2 - x + 7;
>> f+g

ans =
3*x^2 + 2*x + 2

>> f*g

ans =
(x^2 - x + 7)*(2*x^2 + 3*x - 5)

>> expand(ans)

ans =
2*x^4 + x^3 + 6*x^2 + 26*x - 35

>> f/g

ans =
(2*x^2 + 3*x - 5)/(x^2 - x + 7)

>> expand(ans)

ans =
(3*x)/(x^2 - x + 7) - 5/(x^2 - x + 7) + (2*x^2)/(x^2 - x + 7)
```

O MATLAB pode realizar operações mais avançadas sobre expressões simbólicas. Por exemplo, a função *compose* calcula a composição das funções $f(x)$ e $g(x)$ e $f(g(x))$, a função *finverse* encontra a inversa funcional de uma expressão e a função *subs* substitui uma variável por um número (ou variável) em uma expressão.

```
>> syms x
>> f = 1/(1-x^2); g = sin(x);
>> compose(f,g)

ans =
-1/(sin(x)^2 - 1)

>> compose(g,f)

ans =
-sin(1/(x^2 - 1))

>> finverse(g)

ans =
asin(x)

>> finverse(f)

ans =
((x - 1)/x)^(1/2)
```

Também podemos resolver funções algébricas através do comando *solve*.

```
>> syms a b c x
>> solve( a*x^2 + b*x + c)

ans =
-(b + (b^2 - 4*a*c)^(1/2))/(2*a)
-(b - (b^2 - 4*a*c)^(1/2))/(2*a)

>> pretty(ans)

+-      2      1/2   +
|   b + (b  - 4 a c)   |
| - -----   |
|   2 a           |
|   2      1/2   +
|   b - (b  - 4 a c)   |
| - -----   |
|   2 a           |
+-                  +-
```

A função *pretty*, usada no exemplo anterior, exibi o resultado em uma forma mais “bonita”.

Segue abaixo um resumo das funções para manipulação de expressões algébricas:

```
diff(f) - calcula a derivada de f.
compose(f, g) - determina a composta  $f(g(x))$ .
expand(expr) - expande uma expressão expr.
finverse(expr) - determina a inversa funcional da expressão expr.
pretty(expr) - exibe a expressão expr numa forma mais bonita.
simple(expr) - procura encontrar uma forma mais simples de escrever uma expressão expr.
simplify(expr) - simplifica a expressão expr.
solve(expr) - acha a(s) solução(es) da equação expr = 0.
subs(expr, x, a) - substitui na expressão expr a variável x por a.
syms x y z a b - define as variáveis simbólicas x, y, z, a e b.
```

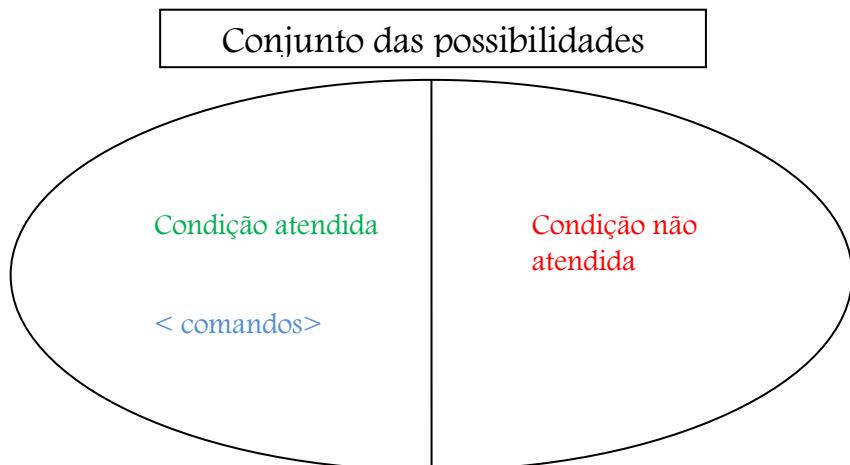
Existem várias outras funções para manipulação de expressões algébricas. Você pode obter informações sobre elas digitando *help symbolic*. Uma função interessante que mostra as capacidades do MATLAB em tratar com funções matemáticas é a *funtool*, que é uma calculadora para funções.

8. Estruturas condicionais

a. Condição “se”

Sendo a estrutura mais simples do MATLAB, é da forma:

```
if <condição>
    <comandos>
end
```



Se o resultado da expressão lógica <condição> for verdadeiro (valor 1), a lista <comandos> será executada. Caso seja falso (valor 0), será ignorada e a execução saltará para a linha end.

Caso a expressão de condição seja do tipo $[a, b] == [c, d]$, na qual teremos uma resposta dupla da forma $[a == b, c == d]$, que poderá ser $[0, 0]$, $[0, 1]$, $[1, 0]$ ou $[1, 1]$, haverá uma relação de “e” ou “and”, ou seja, só haverá execução caso “ $a == b$ ” e “ $c == d$ ”.

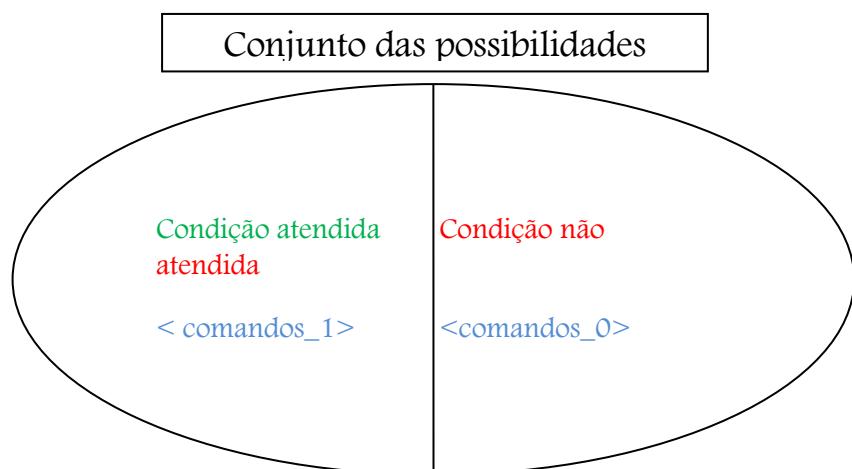
- Exemplo, utilizando entrada input:

```
>> a = input('entre com o valor de a: ');
>> b = input('entre com o valor de b: ');
>> c = input('entre com o valor de c: ');
>>if a == b & b == c
    fprintf('triangulo equilátero\n')
end
```

i. Condição “se, caso contrário”

Como a condição “se”, mas permite adicionar uma seqüência alternativa de comandos caso a condição não seja atendida:

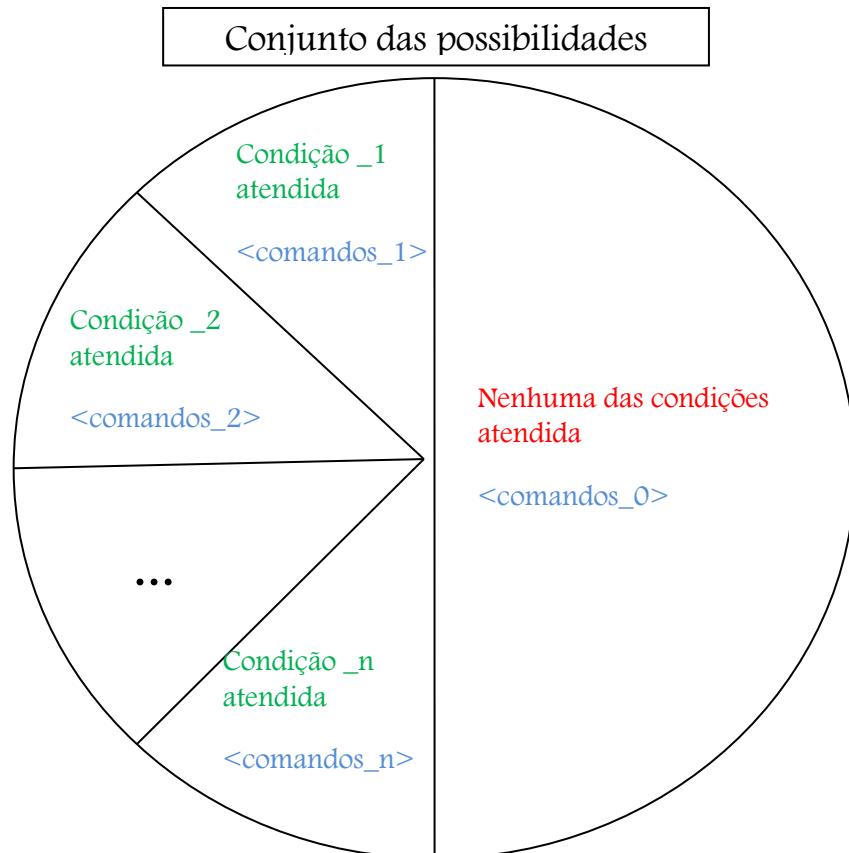
```
if    <condição>
    <comandos_1>
else
    <comandos_0>
end
```



ii. Condição “se, mas ainda se, (caso contrário)”

Como a condição “se”, mas permite adicionarsequências alternativas de comandos caso outras condições sejam atendidas ou ainda se nenhuma das condições seja atendida:

```
if      <condição_1>  
       <comandos_1>  
elseif   <condição_2>  
       <comandos_2>  
...  
elseif   <condição_n>  
       <comandos_n>  
  
else  
       <comandos_0>  
end
```



b. Estruturas de laços e repetições

i. Estrutura “para” ou “for”

Faz com que um grupo de comandos seja executado uma sequência de vezes:

```
for <variável> = <arranjo>
    <comandos>
end
```

<variável> é uma variável de controle que assumirá todos os subarranjos de ordem $n-1$ contidos no arranjo. Geralmente, esse arranjo é um vetor linha, de tal forma que a variável de controle assume valores numéricos.

- Exemplo de uso:

```
>> s = 0;
>> n = input('Entre com o número de termos: ');
>> for k = 1:10
    s = s + k;
end
```

O código acima calcula a soma de uma Progressão Aritmética: $\sum_{k=1}^n k$. O valor n é de entrada.

ii. Estrutura “enquanto” ou “while”

Mantém um laço de repetição de comandos até que uma condição seja satisfeita:

```
while <condição>
    <comandos>
end
```

Enquanto a condição for verdadeira (1), a lista de comandos será repetida. Ao parar, o código segue da linha *end*.

- Exemplo: para qual valor de n a soma $\sum_{k=1}^n k$ atinge pelo menos 1.000.000?

```
>> s = 0;
>> while s <= 1000000
        k = k + 1;
        s = s + k;
    end
>> s, k% exibe o valor de s e o de k
s =
    1000350
k =
    1414
```

- Determinando a precisão dos números de ponto flutuante no computador:

```
>> n = 0; Epsilon = 1;
while 1 + Epsilon > 1
    n = n + 1;
    Epsilon = Epsilon/2;
end
n, Epsilon, eps

n =
53
Epsilon =
1.1102e-016
ans =
2.2204e-016
```

O código acima faz: enquanto for percebida a diferença entre a soma de 1 + épsilon e 1, épsilon é continuamente dividido ao meio. Quando atinge o limite de imprecisão, 1 + épsilon e 1 é visto como o mesmo valor.

iii. Comando de parada imediata ou interrupção interior – *break*

Geralmente é utilizado dentro de condições “se” que estão, por sua vez, internas a um laço. Pára o laço mais interno, jogando a execução para a linha de seu respectivo fim (end).

Quase sempre, laços “for” e “while” podem ser intercambiados. O uso do comando “break” dentro de um “if” pode verificar a condição de parada desejada, fazendo com que um while seja trocado por um “for”.

- Exemplo genérico:

```
while 1
    <comandos_1>
        if <condição de parada>
            <comandos_if>
            break;
        end
    <comandos_2>
end
```

O laço “while1” é infinito, pois 1 é uma condição sempre verdadeira. Então, a parada será feita dentro do “se”, sob sua condição.

- Exemplo com for: cálculo do valor de PI com algoritmo de Monte Carlo.

```
>>pi_anterior = 0; pi_calc = 0; p_interno = 0;
>>for p = 1:1E5 % se não convergir em 100000 iterações, parar
    x = rand; y = rand;
    if(x^2 + y^2 < 1)
        p_interno = p_interno + 1;
    end
    pi_anterior = pi_calc;
    pi_calc = 4*p_interno/p;
    if (pi_calc - pi_anterior< 1d-4&pi_calc - pi_anterior> 0)
        break;
    end
end
```

O laço “*for*” faz 100000 iterações, mas, caso seja detectada uma variação muito pequena de um passo para outro (segundo “*if*”), o comando *break* interrompe o laço e já se considera convergência suficiente.

9. Gráficos

O Matlab possui algumas funções para traçar gráficos de diversos tipos, em duas e três dimensões, permitindo alteração de parâmetros, como tipo ou cor de linhas.

a. Gráficos numéricos bidimensionais

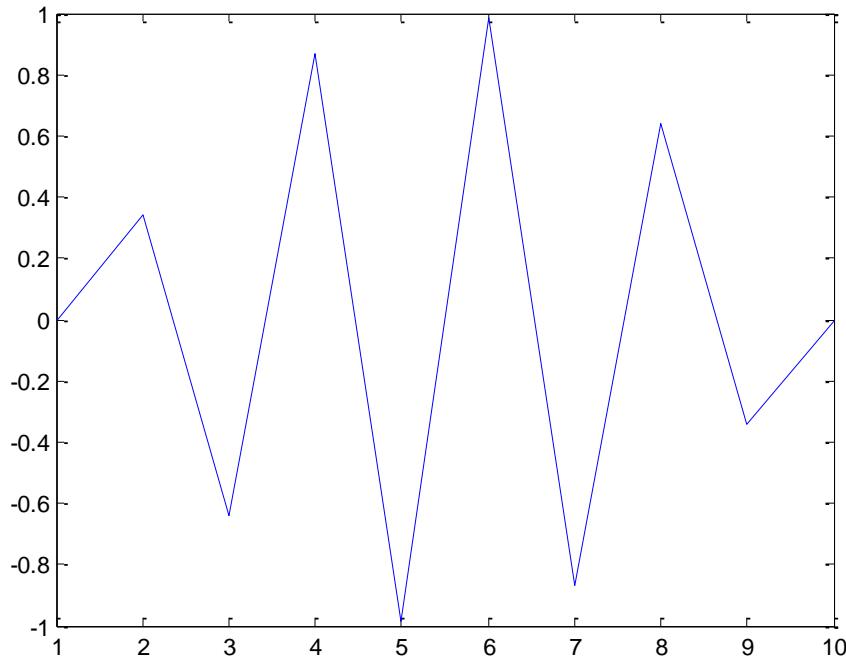
i. Função plot

```
plot(x1,y1,'<tipo de linha 1>', x2,y2,'<tipo de linha 2>',...)
```

símbolo	cor	símbolo	estilo de linha
y	amarela	.	ponto
m	lilás	o	círculo
c	turquesa	x	marca x
r	vermelho	+	mais
g	verde	*	asterisco
b	azul	-	linha sólida
w	branco	:	linha pontilhada
k	preto	-.	linha de traço e ponto
		--	linha tracejada

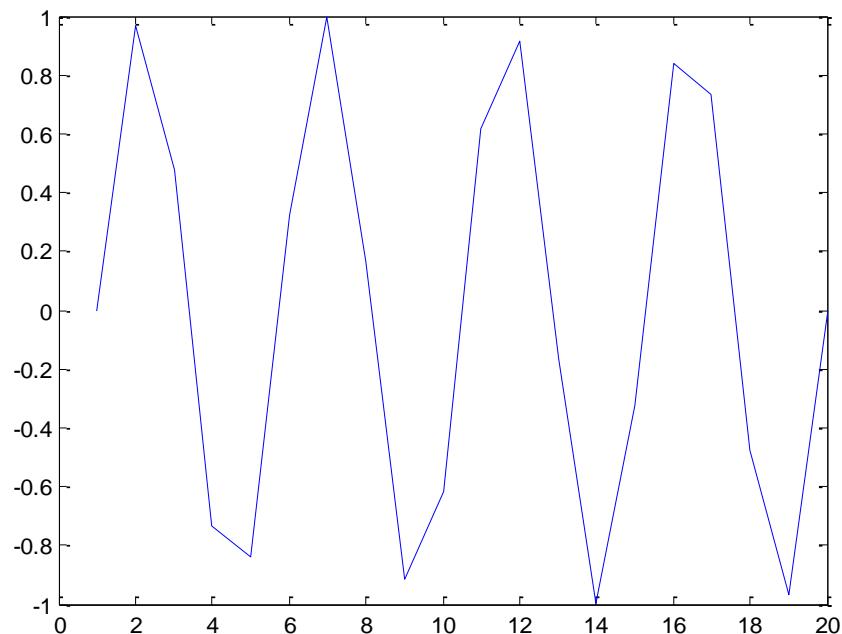
Tabela 2.15: Alguns tipos de linha da função *plot*.

```
>> x = linspace(0, 8*pi, 10);% 10 amostras ou divisões
>> y = sin(x);
>> plot(y); % Traça y versus os índices de y
```

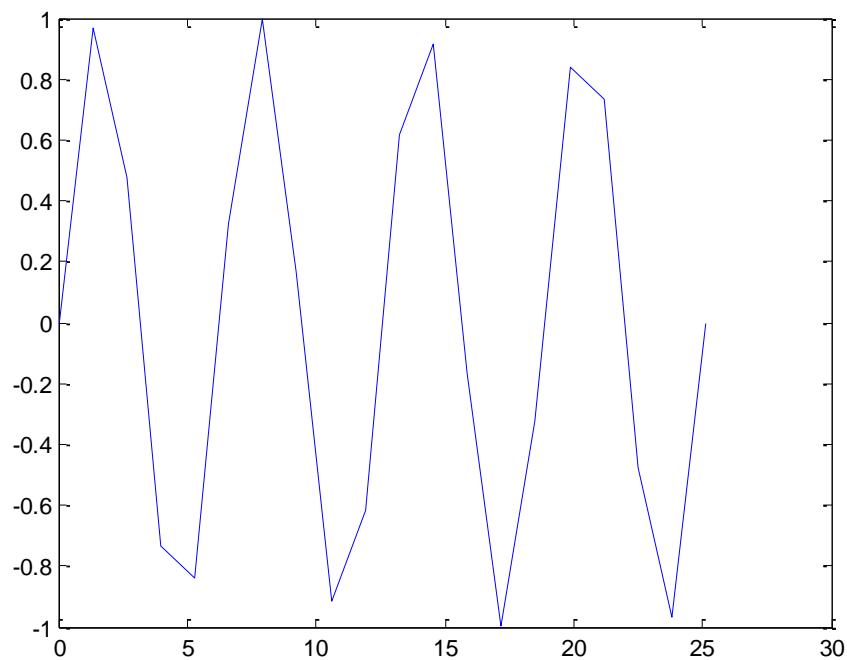


Perceba a deformação do sinal causada pela baixa taxa de amostragem. Conforme a frequência máxima e a taxa de amostragem, pode acontecer falseamento de sinal (ou *aliasing*) visível no gráfico.

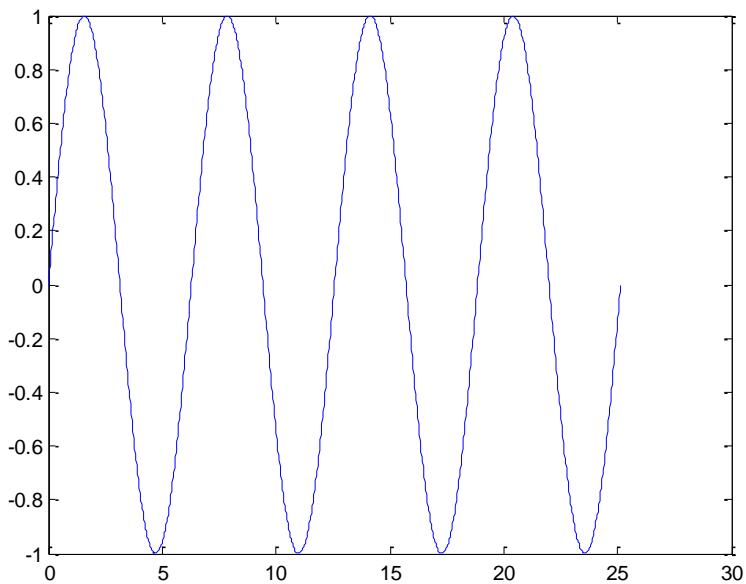
```
>> x = linspace(0, 8*pi, 20);% 20 amostras ou divisões
>> y = sin(x);
>> plot(y);
```



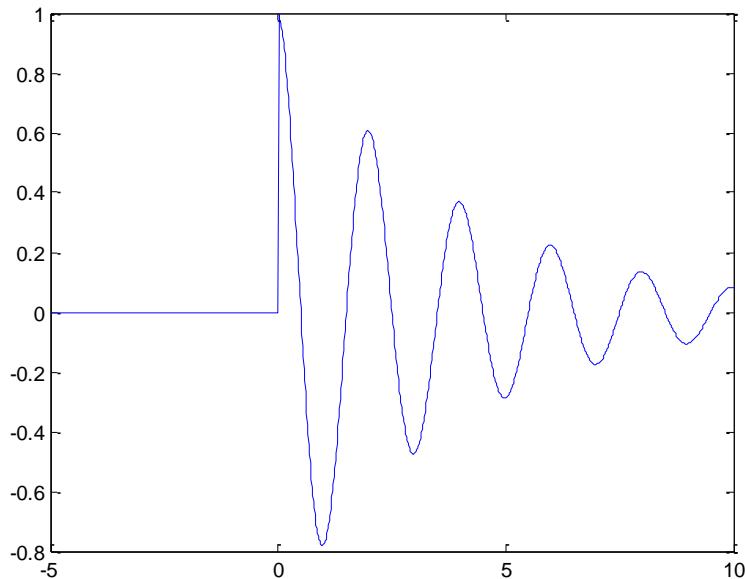
```
>> plot(x, y); % Traça y versus x
```



```
>> x = linspace(0, 8*pi, 1000); % 1000 amostras ou divisões  
>> y = sin(x);  
>> plot(x, y);
```



```
>> t = linspace(-5, 10, 1000); % 1000 divisões
>> u = t > 0; % Função degrau
>> x = u.*exp(-t/4).*cos(pi*t);
>> plot(t, x);
```

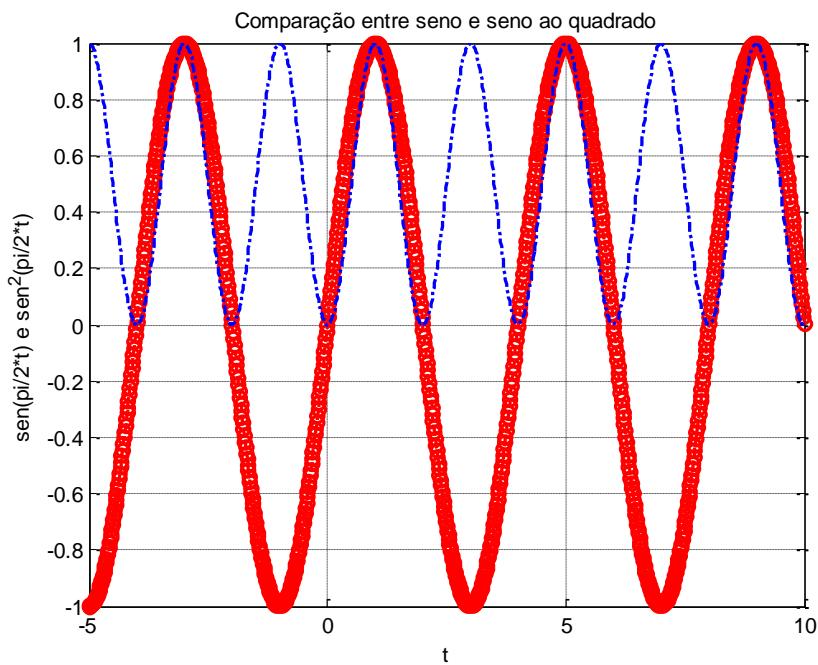


- Rótulos (Labels) nos gráficos por linha de comando:

comando	descrição
<code>xlabel('<texto>')</code>	escreve <code><texto></code> abaixo do eixo das abscissas;
<code>ylabel('<texto>')</code>	escreve <code><texto></code> ao lado do eixo das ordenadas;
<code>title('<texto>')</code>	escreve <code><texto></code> no alto da figura;
<code>text(x_i, y_i, '<texto>')</code>	escreve <code><texto></code> na posição (x_i, y_i) ;
<code>gtext('<texto>')</code>	escreve <code><texto></code> na posição indicada pelo <i>mouse</i> .

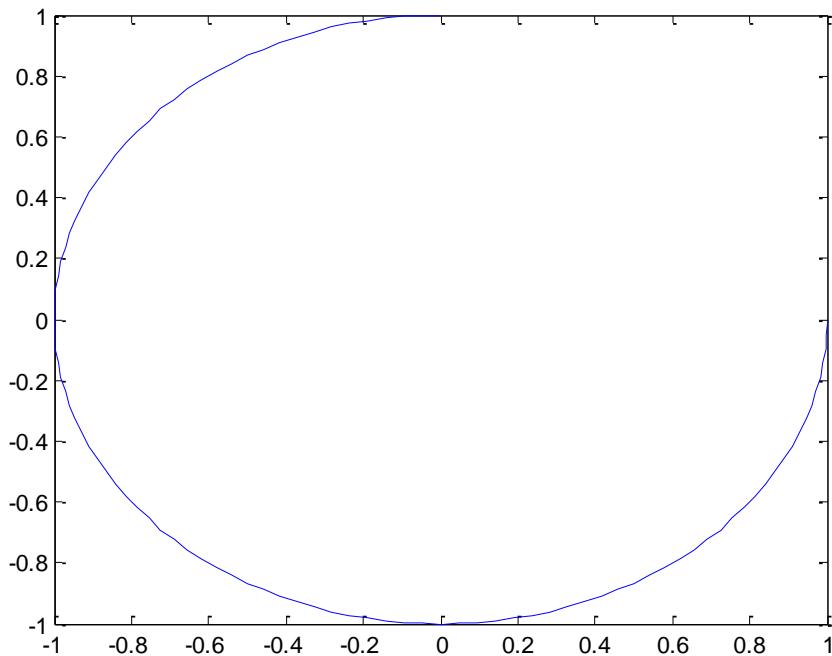
Tabela 2.16: Funções para identificação de gráficos.

```
>> t = linspace(-5, 10, 1000); %
>> x = sin(pi/2*t)
>> y = cos(pi/2*t)
>> plot(t,x,'r--o', t,y,'-.', 'linewidth', 1.5);
>> title('Comparação entre seno e seno ao quadrado');
>> xlabel('t');
>> ylabel('sen(pi/2*t) e sen^2(pi/2*t)');
>> grid;
```

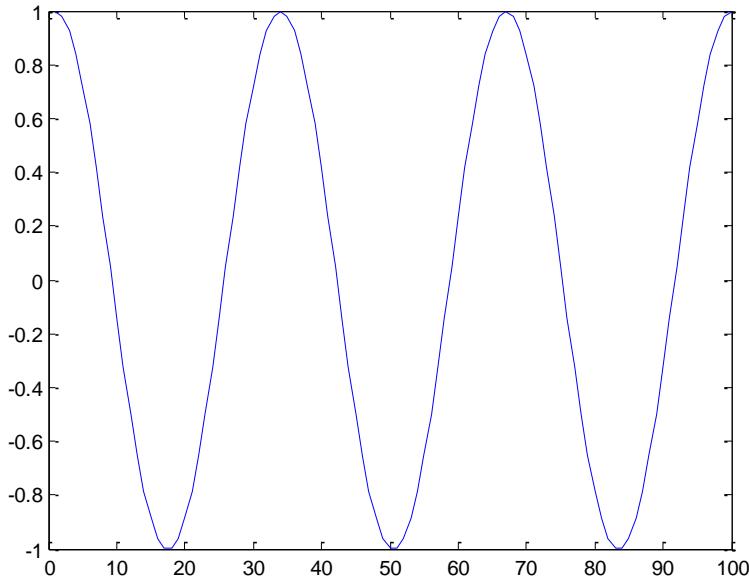


- Podemos naturalmente usar números complexos dentro da função plot:

```
>> t = linspace(0, 6*pi/4, 100) ;
>> x1 = [exp(j*t)', (exp(j*4*t)+exp(-j*4*t))/2];
>> plot(x1(:,1))
```



```
>> t = linspace(0, 6*pi/4, 100) ;
>> x1 = [exp(j*t)', (exp(j*4*t)+exp(-j*4*t))/2];
>> plot(x1(:,2))
```



ii. Função fplot

Essa função permite com que entremos diretamente com a função dada como uma string:

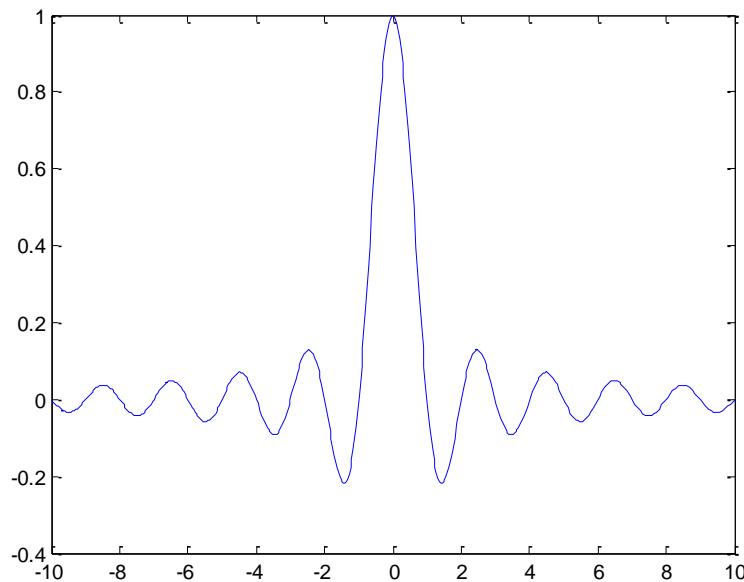
```
>> fplot('<função>', [Xmín, Xmáx])
```

Ou, opcionalmente:

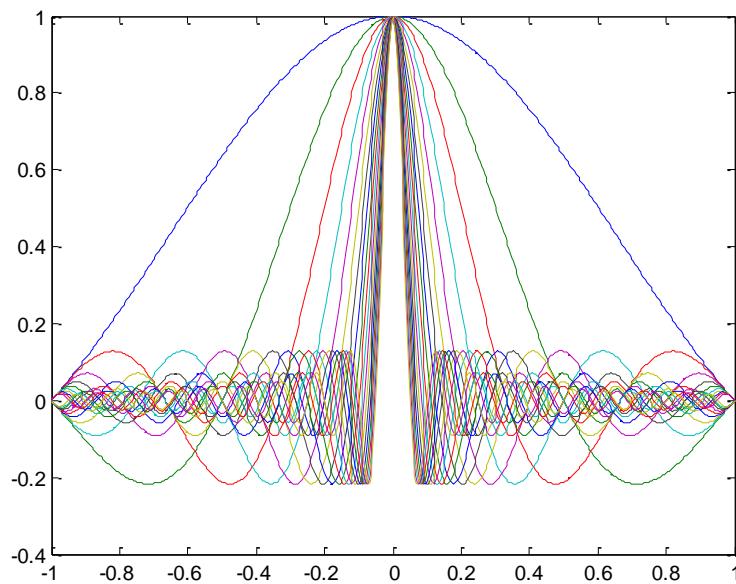
```
>> fplot('<função>', [Xmín, Xmáx, Ymín, Ymáx], '<tipo de linha>')
```

- Exemplo – traçando a função sinc diretamente:

```
>> fplot('sinc(x)', [-10 10])
```



```
>> k = 1:20;
>> k_str = num2str(k); % Observe a importância dessa função
>> f = ['sinc([' k_str, ']*x)'];
>> fplot(f, [-1 1])
```



Além dessas, existem várias outras funções para gráficos bidimensionais, como para gráficos em coordenadas polares, eixos logarítmicos, etc.. Para mais informações, digite

```
>>help graph2d;
```

b. Gráficos numéricos tridimensionais

Tal como para os gráficos 2D, podemos acessar a lista de ajuda para gráficos 3D pelo comando

```
>> help graph3d;
```

Aqui, veremos as principais dessas funções, de maneira que sejam suficientes para, facilmente, adquirir certa independência.

i. Função meshgrid

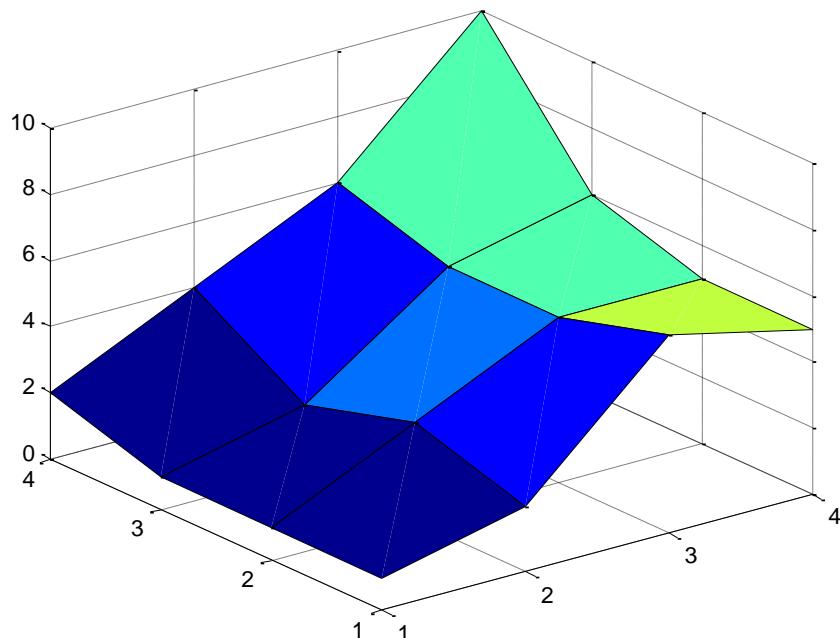
Usada para criar matrizes ideais para vários gráficos tridimensionais. Substitui o uso de laços (loops).

ii. Função surf

Traça gráficos 3D em forma de malha, representando o valor em cada ponto de uma matriz. Forma de entrada: `surf(<matriz>);`

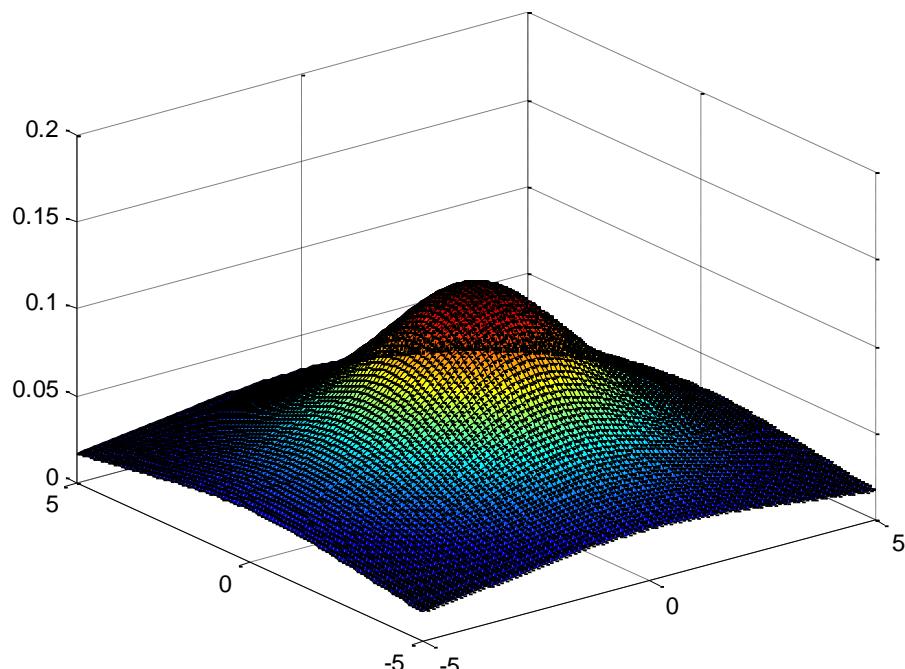
- Traçando uma superfície:

```
>> surf([1 2 6 5; 1 3 5 5; 1 2 5 6; 2 4 6 10]);
```

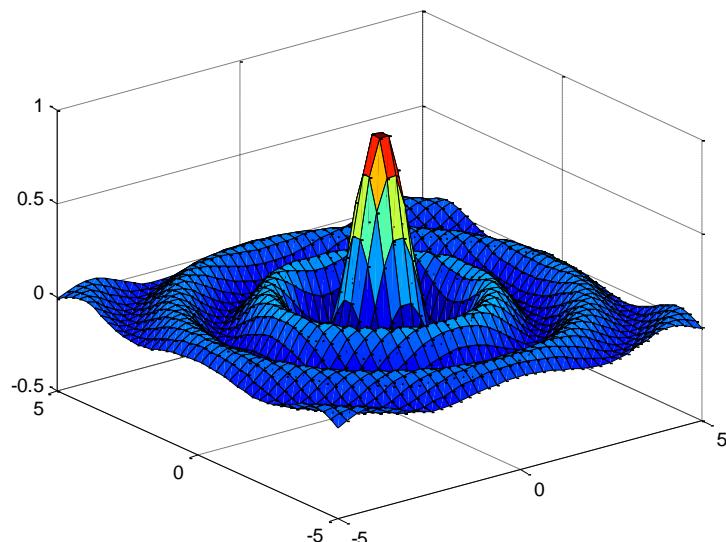


- Utilizando a função meshgrid para definir uma malha:

```
>> x = linspace(-5,5,100); y = linspace(-5,5,100);
>> [X, Y] = meshgrid(x, y);
>> Z = 1./((X^2-1)+(Y^2-1));
>> Z = 1./((X.^2-1)+(Y.^2-1)+10);
>> surf(x,y,Z);
```

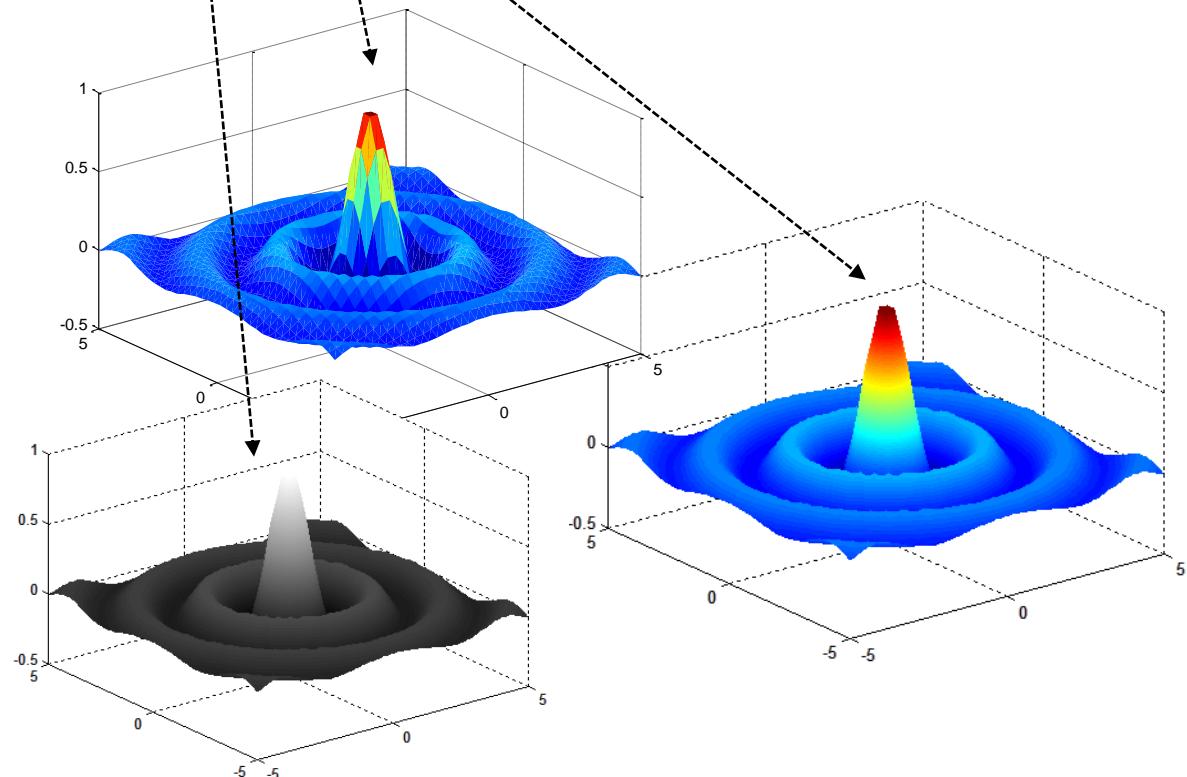


```
>> x = linspace(-5,5,40); y = linspace(-5,5,40);
>> [X, Y] = meshgrid(x, y);
>> Z = sinc(sqrt(X.^2+Y.^2));
>> surf(x,y,Z);
```



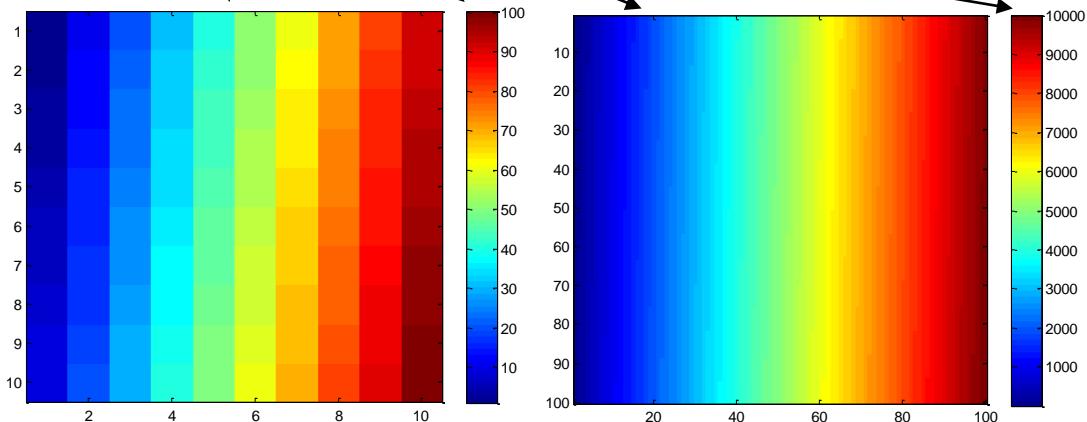
- Mudando o visual do gráfico:

```
>> shading faceted  
>> shading flat  
>> shading interp  
>> colormap(gray)
```



- Mudar e visualizar mapas de cores:

```
>> mat = reshape(1:100,10,10);
>> mat2 = reshape(1:10000,100,100);
>> imagesc(mat);
>> colorbar
>> imagesc(mat2);
>> colorbar
```



A função *imagesc(<mapa>)* mostra o mapa de cores, a função *colorbar* exibe a barra lateral com valores e a função *colormap(<mapa>)* aplica o mapa à figura aberta.

- Para customizar um mapa de cor, devemos fazer uma matriz $n \times 3$ com os valores dos canais vermelho, verde e azul (RGB) normalizados de 0 a 1.

```
>> mapa = zeros(256,3);
>> mapa(:,2) = (0:255)/255;
>> mapa(129:256, 1) = (0:2:254)/254;
>> imagesc(mapa); % visualizando o mapa
```

iii. Funções **plot3**, **mesh**, **meshc**, **meshz**

Figuras, subplots, hold e plots com matrizes como argumentos:

```
>> theta = linspace(0, pi, 1000);
>> X = sin((1:20)')*theta;
>> X = X./((sin((1:20)/20))'*ones(1,1000));
>> plot(theta, X);
>> figure(2)
>> subplot(2,2,1)
>> scatter() % variação do plot - pontos
```

10. Arquivos .m

Para resolver problemas simples, é cômodo e eficiente utilizar o MATLAB como se fosse uma calculadora, entrando-se com os comandos diretamente no prompt. Ou seja, cada linha de comando é introduzida na Janela de Comandos e processada imediatamente. Entretanto, à medida que o número de comandos aumenta, ou quando se deseja mudar o valor de uma ou mais variáveis e executar novamente os comandos, o melhor é utilizar o MATLAB como uma linguagem de programação, ou seja, utilizar o MATLAB para executar seqüências de comandos armazenadas em arquivos de roteiro (script). Esses arquivos que contêm as declarações do MATLAB são chamados arquivos ".m" (ou M-files), como, por exemplo, exemplo1.m. Esses M-files são os programas fontes do MATLAB e consistem de seqüências de comandos normais do MATLAB, possibilitando incluir outros arquivos ".m" escritos no formato texto (ASCII). Para escrever um programa (ou arquivo .m) no MATLAB, escolha File na barra de menu. Dentro do menu File escolha New e selecione M-file. Abre-se, então, um editor de textos, onde pode-se escrever os comandos do MATLAB. Para editar um arquivo já existente, selecione a opção Open M-File, a partir do menu File. Os arquivos podem, também, ser editados fora do MATLAB utilizando qualquer editor de texto. Segue um exemplo abaixo:

```

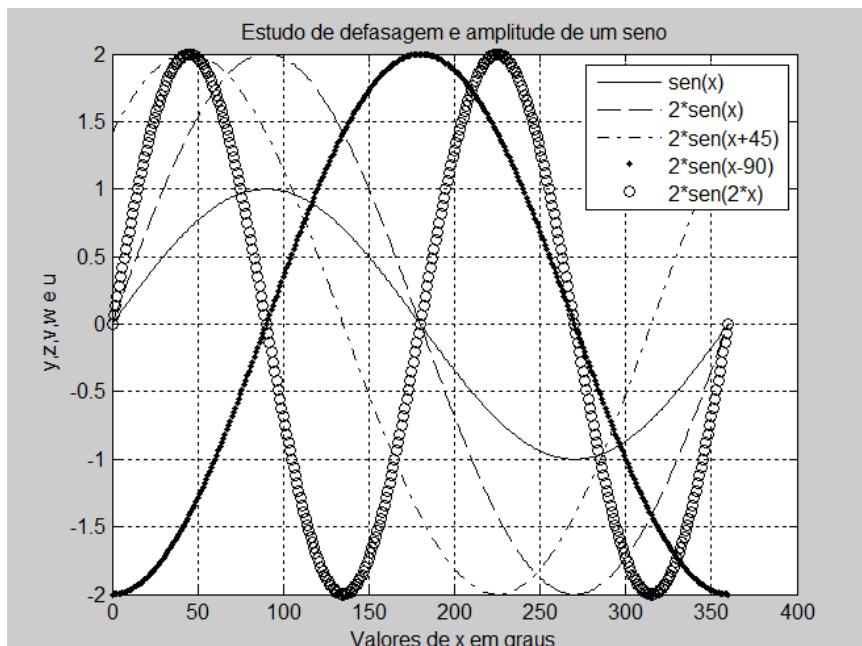
1 %=====
2 % Exemplo de programação no MATLAB
3 % Este exemplo plota uma função seno nas seguintes
4 % condições:
5 % sen(x)
6 % 2*sen(x)
7 % 2*sen(x+45)
8 % 2*sen(x-90)
9 % 2*sen(2*x)
10 %=====
11 %
12 - x=0:360;
13 %
14 % Seno com amplitude A=1 e defasagem phi=0 graus
15 - A=1;
16 - phi=0;
17 - y=A*sin(2*pi*x/360+2*pi*phi/360);
18 % Seno com amplitude A=2 e defasagem phi=0 graus
19 - A=2;
20 - z=A*sin(2*pi*x/360+2*pi*phi/360);
21 % Seno com amplitude A=2 e defasagem phi=45 graus
22 - phi=45;
23 - v=A*sin(2*pi*x/360+2*pi*phi/360);
24 % Seno com amplitude A= 2 e defasagem phi=-90 graus
25 - phi=-90;
26 - w=A*sin(2*pi*x/360+2*pi*phi/360);
27 % Seno com amplitude A= 2 e defasagem phi=0 graus
28 - phi=0;
29 - u=A*sin(2*pi*2*x/360+2*pi*phi/360);
30 % Plotagem do resultado
31 - plot(x,y,'k-',x,z,'k--',x,v,'k-.',x,w,'k.',x,u, 'ko')
32 - grid
33 - xlabel('Valores de x em graus')
34 - ylabel('y,z,v,w e u')
35 - title('Estudo de defasagem e amplitude de um seno')
36 - legend('sen(x)', '2*sen(x)', '2*sen(x+45)', '2*sen(x-90)', '2*sen(2*x)')

```

Uma vez escrito o programa, entre no menu File da janela do editor de textos e escolha a opção Save as... Nesta opção do menu, salve o programa como prog1.m no seu diretório de trabalho. Em seguida, volte à janela de comandos do MATLAB e use o comando cd ou a opção Set Path...do menu File para ir ao diretório onde o programa prog1.m foi salvo. Em seguida, digite o nome do arquivo (sem a extensão .m) para executar o programa:

>>prog1

O gráfico obtido segue abaixo:



11. Polinomios

No MATLAB, um polinômio é representado por um vetor-linha contendo os coeficientes do polinômio em ordem decrescente. Por exemplo, o polinômio de 4º grau: $x^4 - 3 * x^3 + 10 * x^2 - 7 * x - 48$ é representado pelo vetor:

$$p = [1 \ -3 \ 10 \ -7 \ -48].$$

a. Raízes

Para calcular as raízes de um polinômio utiliza-se o comando *roots*:

```
>> p = [1 -3 10 -7 -48];
>> r = roots(p)

r =
    0.8995 + 3.3273i
    0.8995 - 3.3273i
    2.6984 + 0.0000i
   -1.4973 + 0.0000i
```

Se forem conhecidas as raízes de um polinômio, então o comando *poly* retorna os coeficientes desse polinômio:

```
>> poly(r)

ans =

1.0000   -3.0000   10.0000   -7.0000  -48.0000
```

Observe que neste exemplo, os valores são os coeficientes do vetor p . Entretanto, se o argumento de `poly()` for uma matriz quadrada, então este comando retorna os coeficientes do polinômio característico dessa matriz.

Exemplo:

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

pp =

1.0000   -15.0000   -18.0000   -0.0000
```

Os valores do vetor pp correspondem aos coeficientes do polinômio característico da matriz $[A]$. Lembrando que, para este exemplo, o polinômio característico é calculado como segue:

$$\det([A] - \lambda[I]) = \begin{vmatrix} 1-\lambda & 2 & 3 \\ 4 & 5-\lambda & 6 \\ 7 & 8 & 9-\lambda \end{vmatrix} = \lambda^3 - 15\lambda^2 - 18\lambda$$

Ou seja, os valores do vetor pp são, de fato, os coeficientes do polinômio característico de $[A]$ como mostrado no cálculo anterior. Assim, para se obter os autovalores de $[A]$, basta calcular as raízes de pp :

```
>> lambda = roots(pp)

lambda =

16.1168
-1.1168
-0.0000
```

b. Produto de polinômios

Se tivermos dois polinômios y_1 e y_2 e quisermos calcular o produto $y_1 * y_2$, utiliza-se o comando `conv`:

```

>> y1 = [1 1 9]
y1 =
    1     1     9
>> y2 = [2 -4 0 1]
y2 =
    2    -4     0     1
>> conv(y1,y2)
ans =
    2    -2    14   -35     1      9

```

O resultado da operação `conv(y1,y2)` contém os coeficientes do polinômio:

$$(x^2 + x + 9) \cdot (2x^3 - 4x^2 + 1) = 2x^5 - 2x^4 + 14x^3 - 35x^2 + x + 9$$

c. Divisão de polinômios

A divisão de polinômios é feita com o comando `deconv`:

```

>> z1 = [1 2 1 -3 1]
z1 =
    1     2     1    -3     1
>> z2 = [-1 1 3]
z2 =
    -1     1     3
>> [q,r] = deconv(z1,z2)
q =
    -1    -3    -7
r =
    0     0     0    13    22

```

No caso da divisão do polinômio $x^4 + 2x^3 + x^2 - 3x + 1$ por $-x^2 + x + 3$, o resultado é o polinômio $-x^2 - 3x - 7$ e o resto é $13x + 22$.

A operação de divisão de polinômios é também denominada deconvolução polinomial.

d. Expansão em frações parciais

A expansão em frações parciais é comumente utilizada em várias aplicações, tais como as transformadas de Laplace e de Fourier, ou qualquer outro problema que envolva razão de polinômios. Como exemplo considere o seguinte polinômio:

$$\frac{B(s)}{A(s)} = \frac{10s^3 + 80s^2 + 177s + 95}{s^3 + 8s^2 + 19s + 12}$$

Aplicando a função *residue* obtemos:

$$\frac{B(s)}{A(s)} = \frac{9}{s+4} + \frac{-7}{s+3} + \frac{-2}{s+1} + 10$$

No matlab:

```
>> B = [10 80 177 95]

B =
    10     80    177     95

>> A = [1 8 19 12]

A =
    1      8     19     12

>> [res,pol,k] = residue(B,A)

res =
    9.0000
   -7.0000
   -2.0000

pol =
   -4.0000
   -3.0000
   -1.0000

k =
    10
```

A sintaxe da função *residue* é a seguinte:

[r,p,k] = residue(num,den),

Sendo *num* o numerador e *den* o denominador. Os vetores coluna *r*, *p* e *k* são, respectivamente, o resíduo, o polo e o termo direto.

Não por outra razão, o cálculo das frações parciais também é chamado de resíduos. Por outro lado, a aplicação da função *residue* com os argumentos contendo os vetores *r*, *p* e *k* permite o cálculo dos vetores *den* e *num*:

[num,den] = residue(r,p,k),

Exemplo:

```
>> r = [2; -0.5]

r =
    2.0000
   -0.5000

>> p = [1.0; 0.5]

p =
    1.0000
    0.5000

>> k = 1

k =
    1

>> [n,d] = residue(r,p,k)

n =
    1     0     0

d =
    1.0000   -1.5000    0.5000
```

De modo que:

$$F(s) = \frac{2}{s-1.0} - \frac{0.5}{s-0.5} + 1 = \frac{s^2}{s^2 - 1.5s + 0.5}$$

e. Avaliação de polinômios

Utilizando a função *polyval* podemos avaliar um determinado polinômio numericamente.

Exemplo:

```
>> x = 0:0.5:2;
>> p = [2 -5 8 -10 40];
>> y = polyval(p,x)

y =
    40.0000    36.5000    35.0000    36.2500    44.0000
```

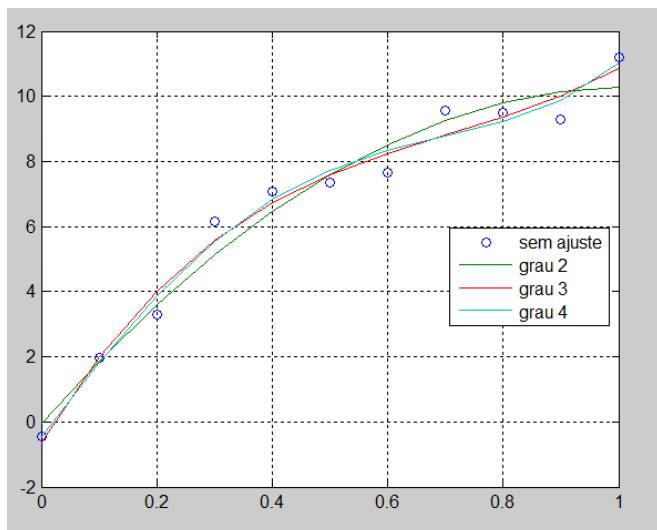
f. Ajuste polinomial (Método dos mínimos quadrados)

O Ajuste ou regressão polinomial pode ser executado no Matlab através do comando *polyfit(x,y,n)*, no qual x e y são vetores contendo os valores serem ajustados e n é o grau do polinômio de ajuste. A função *polyfit* retorna um vetor que contém os coeficientes do polinômio de ajuste.

No Exemplo seguinte, vamos calcular o ajuste polinomial de 2º, 3º e 4º graus e traçar o gráfico comparativo entre as funções de ajuste (representados por gráficos de linhas contínuas) e o dos valores a serem ajustados. Note o uso da função *polyval* para converter os coeficientes dos polinômios de ajuste em valores numéricos para o traçado gráfico.

```
>> x = 0:0.1:1;
>> y = [-0.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2];
>> pp2 = polyfit(x,y,2);
>> pp3 = polyfit(x,y,3);
>> pp4 = polyfit(x,y,4);
>> yp2 = polyval(pp2,x);
>> yp3 = polyval(pp3,x);
>> yp4 = polyval(pp4,x);
>> plot(x,y,'o',x,yp2,x,yp3,x,yp4)
>> grid on
```

```
>> legend('sem ajuste', 'grau 2', 'grau 3', 'grau 4')
```



12. Exercícios

a. Exercícios de Revisão (com solução)

1. Calcule as seguintes expressões:

$$\begin{aligned}x &= 3 \times 2^2 - (3 + 4)^3 \\y &= x - \frac{2}{3 \times x} + x^{\frac{1}{2}} + 11 \\z &= y * x - \frac{y}{x + 3 * y} + x^y\end{aligned}$$

2. Dados os vetores $w = [1 \ 2 \ 3]$ e $z = [8 \ 9]$
- Crie o vetor wz tal que $wz = [2 \times w, -z]$
 - Faça com que o segundo elemento do vetor w tenha o valor -10
 - Mostre o terceiro elemento de w

3. Verifique o que acontece se você fizer

```
X = [1 + 3 * i, 2 - 2 * i]
>> X'
```

4. Crie um vetor X com valores crescentes de 1.0 até 2.0 com espaçamento de 0.2.

5. Crie as seguintes matrizes:

- Matriz formada apenas de zeros 2x4
- Matriz formada apenas por 1's
- Matriz identidade 4x4
- Matriz composta de forma randômica com distribuição normal

6. Crie uma matriz A com o seguinte conteúdo:

16.0	3.0	2.0	13.0
5.0	10.0	11.0	8.0
9.0	6.0	7.0	12.0
4.0	15.0	14.0	1.0

Crie a matriz B formada pela concatenação de A , da forma: $[A \ A+32; \ A+48 \ A+16]$

7. Dada a matriz X :

$x =$

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

- a. Encontre a soma das colunas da matriz
- b. Encontre a diagonal principal de X
- c. Encontre o determinante de X
- d. Encontre a matriz inversa de X
- e. Encontre os autovalores (raízes do polinômio característico)
- f. Encontre o posto de X (numero de linhas ou colunas LI)
- g. Encontre o polinômio característico de X

8. Determine os autovalores e autovetores da matriz:

$$A = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 4 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

Respostas dos Exercícios de Revisão**1.**

```
>> x = 3*2^2 - (3+4)^3
x =
-331
>> y = x - 2/(3*x) + x^(1/2) + 11
y =
-3.2000e+02 + 1.8193e+01i
>> z = y*x - y/(x + 3*y) + x^y
z =
1.0592e+05 - 6.0220e+03i
```

2.**a.**

```
>> w = [1 2 3]; z = [8 9];
>> wz = [2*w, -z]
wz =
2      4      6      -8      -9
```

b.

```
>> w(2) = -10
w =
1     -10      3
```

c.

```
>> w(3)
ans =
3
```

3. O operador ‘ representa o conjugado de um número complexo:

```
x =
2.0000 + 2.0000i
>> x'
ans =
2.0000 - 2.0000i
```

4.

```
>> x = 1:0.2:2
x =
1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

5.

```
>> R = rand(3)
R =
0.8147    0.9134    0.2785
0.9058    0.6324    0.5469
0.1270    0.0975    0.9575

>> E = eye(4,4)
E =
1     0     0     0
0     1     0     0
0     0     1     0
0     0     0     1

>> u = ones(2,4)
u =
1     1     1     1
1     1     1     1

>> Rn = randn(3)
Rn =
2.7694    0.7254   -0.2050
-1.3499   -0.0631   -0.1241
3.0349    0.7147    1.4897
```

6.

```
B = [A A+32; A+48 A+16]
```

```
B =
16     3     2    13     48     35     34     45
 5    10    11     8     37     42     43     40
 9     6     7    12     41     38     39     44
 4    15    14     1     36     47     46     33
 64    51    50    61     32     19     18     29
 53    58    59    56     21     26     27     24
 57    54    55    60     25     22     23     28
 52    63    62    49     20     31     30     17
```

7.

- a. sum(X)
- b. diag(X)
- c. det(X)
- d. inv(X)
- e. eig(X)
- f. rank(X)
- g. Poly(X)

8.

```
>> [Autovetores,Autovalores] = eig(A)

Autovetores =
0.4082    0.7071    0.5774
0.4082   -0.7071    0.5774
-0.8165         0    0.5774

Autovalores =
2.0000         0         0
      0    2.0000         0
      0         0    8.0000

>> poly(A)
ans =
1.0000   -12.0000   36.0000  -32.0000

>> A = [4 2 2; 2 4 2; 2 2 4]
A =
4     2     2
2     4     2
2     2     4

>> rank(A)
ans =
3
```

b. Exercícios de Fixação

1. Crie 2 matrizes **A** e **B**, 3X4 inteiros.
2. Crie em **C** a transposta de **A**.
3. Gere **D**, somando **A** e **B**.
4. Gere **E**, subtraindo **B** de **A**.
5. Gere **F** com a multiplicação, elemento a elemento, de **A** por **B**.
6. Gere com a função **rand** uma matriz **G**, real 5X5.
7. Gere em **H** a soma das colunas de **F**.
8. Adicione 5 a cada elemento de **B**.
9. **Apague todas as variáveis usadas até este momento.**

10. Gere um vetor inteiro **H**, com o valor inicial 6 e valor máximo 100, com variação de 6 entre os elementos.
11. Gere um vetor **I**, subtraindo 2 de cada um dos valores de **H**.
12. Gere um vetor **J** com a multiplicação, elemento a elemento, de **H** por **I**.
13. Crie duas matrizes reais **A** e **B**, a primeira 5X4 e a segunda 4X5.
14. Crie uma matriz **K**, com o produto matricial de **A** por **B**.

15. Gere em M a soma dos elementos de H.

16. Digite as seguintes linhas e observe os resultados:

a) $X = [2 \ 7 \ 9 \ 7; 3 \ 1 \ 5 \ 6; 8 \ 4 \ 2 \ 5]$

$Y = X(:, 2:2:end)$

Resp: Atribui colunas pares de X para Y

b) $W = X(1:2:end, :)$

Resp: Atribui linhas ímpares de X para W

17. Gere C com as linhas pares de A.

18. Gere D com as colunas ímpares de B.

19. Gere novamente A sem as 3 últimas linhas.

20. Gere novamente B sem a última coluna.

21. Armazene em E a média das colunas de D.

22. Armazene em F a média dos elementos de H.

23. Apague todas as variáveis usadas até este momento.

24. Digite as seguintes linhas, observando os resultados:

a) $x = [3 \ 15 \ 9 \ 12 \ -1 \ 0 \ -12 \ 9 \ 6 \ 1]$

$x(x > 0) = 0$

Resp: Valores positivos de x para zero

b) $x = [3 \ 15 \ 9 \ 12 \ -1 \ 0 \ -12 \ 9 \ 6 \ 1]$

$x(\text{rem}(x, 3) == 0) = 3$

Resp: Valores que são múltiplos de 3 para 3

c) $x = [3 \ 15 \ 9 \ 12 \ -1 \ 0 \ -12 \ 9 \ 6 \ 1]$

$x(\text{rem}(x, 2) == 0) = [x(\text{rem}(x, 2) == 0) * 5]$

Resp: Multiplicar os valores que são pares por 5

d) $x = [3 \ 15 \ 9 \ 12 \ -1 \ 0 \ -12 \ 9 \ 6 \ 1]$

$y = x(x > 10)$

Resp: Extrair os valores maiores que 10 para um vetor y

e) $x = [3 \ 15 \ 9 \ 12 \ -1 \ 0 \ -12 \ 9 \ 6 \ 1]$

$\text{media_de_x} = \text{mean}(x)$

$x(x < \text{mean}(x)) = 0$

Resp: Valores que são menores que a media de x para zero

25. Apague as variáveis y e media_de_x

Realize cada uma das quatro tarefas a seguir com o x do item 24.

26. Zere os valores negativos.

27. Divida por 2 os valores que são múltiplos de 2.

28. Some 10 aos valores ímpares.

29. Calcule a média de x e transformar em negativos os valores que são maiores que essa média.

30. Apague todas as variáveis do espaço de trabalho.

31. Crie com a função **rand** a matriz real **mat**, 7X5, e multiplique por 10.

Sobre **mat** realize os itens que seguem.

- 32.** Calcule a soma das colunas de **mat**.
- 33.** Calcule a soma das linhas de **mat**.
- 34.** Calcule a média dos valores de **mat**.
- 35.** Calcule o somatório dos valores de **mat**.
- 36.** Encontre o maior valor de **mat**.
- 37.** Encontre o menor valor de **mat**.

c. Comandos básicos do Matlab

1 - Execute os seguintes comandos e interprete os resultados

```
a = 2500/20  
a = 2500/20;  
b = [1 2 3 4 5 6 7 8 9]  
c = [1 2 3 ; 4 5 6 ; 7 8 9]  
c = [c ; [10 11 12]]  
c(2,2) = 0  
l = length(b)  
[m,n] = size(b)  
[m,n] = size(c)  
who  
whos  
clear  
who  
b = l + 2 + 3 + 4 + ...  
5 + 6 - 7  
x = 1 : 2 : 9  
x = (0.8 : 0.2 : 1.4);  
y = sin(x)  
help sin  
dir  
a = 2^3  
a = 4/3  
format long  
a = 4/3  
format short  
clear  
a=[1 2 3 ; 4 5 6 ; 7 8 9];  
b = a'  
c = a + b  
c = a - b  
a(l,:) = [-1 -2 -3]
```

```

c = a(:,2)
c = a(2:3, 2:3)
x = [- 1 0 2];
y = [-2 -1 1]';
x*y
c = x + 2
a = [1 0 2; 0 3 4 ; 5 6 0];
size(a)
b = inv(a);
c = b*a
c = b/a
c = b\ a
clear a b x y
whos

```

2- A instrução seguinte abre o arquivo notas.dry e grava todas as instruções digitadas na sequência

```

diary notas.dry
x = [1 -2 3]
y = [4 3 2]
z = x.*y
z = x.^y
y.^2
diary off % Encerra a gravação da instrução diary em notas.dry
dir
type notas.dry
clear
help diary
help sqrt

```

3- Trabalhando com números complexos

```

i = sqrt(-1)
a = [1 2;3 4] + i*[5 6;7 8]
realz = real(z)
imagz = imag(z)
modz = abs(z)
fasez = angle (z)

```

4- Multiplicação de polinômios

```

x3 = (x^2 + 3x + 2).(x^2 - 2x + 1)
x3 = conv([1 2 3],[1 -2 1]) % Como ele faz isto?

```

5- Determinação das raízes de um polinômio

```
roots([1 3 2])
roots([1 -2 1])
roots(x3)
```

6- Utilitários para matrizes

```
a = eye(4)
a = rand(5)
help rand
b = [2 0 0;0 3 0;0 0 -1];
d = det(b)
l = eig(b)
help det
help eig
clear
```

7- Recursos de gravação (armazenagem) de dados

```
help save
help load
a = [1 2 3 4 5 6 7 8];
b = a*2;
c = a - 1;
save arquivo 1 a b c
dir
clear
whos
load arquivo 1
whos
% Em que arquivo estão gravados os vetores a, b e c?
clear
```

8- Recursos gráficos

```
y = [0 2 5 4 1 0];
plot(y)
help pi
t = 0:.4:4*pi
y = sin(t)
z = cos(t);
plot(t, y, '.', t, z "-.")
title('Funções')
xlabel("t")
ylabel("Seno e Cosseno")
text(3, 0.5, 'Seno')
```

```
% Após o próximo comando, selecione a posição que deseja colocar o texto 'Cosseno' com
% o mouse
gtext('Cosseno')
```

9- Ajuste de curvas de dados experimentais

```
t = (-1:.1:1);
x = t.^2;
xr = x+0.2(rand(size(x))-0.5);
figure(1); plot(t, xr, 'g*')
p = polyfit(t, xr, 2)
xa = polyval(p, t);
figure(1); plot(t, xr, 'g*', t, xa)
% Após a próxima instrução, clique em dois pontos do gráfico, e os valores
% das coordenadas serão retornados em [x,y]
[x, y] = ginput(2)
```

10- Programando com o Matlab

```
% Abra um arquivo a partir do Matlab (File, New, M-File)
% e você estará trabalhando no Bloco de Notas (Notepad) do Windows.
% Digite os seguintes comandos e grave o arquivo com o nome
% testel.m, no diretório de usuários (alunos).
n = 3 ;
m = 3;
for i = 1:m
    for j= 1 : n
        a(i, j) = i + j;
    end;
end
disp('Matriz A')
disp(a) %final do programa testel.m
```

11- Criando uma sub-rotina

```
% Abra outro arquivo, salvando-o com nome de teste2.m
% Digite os seguintes comandos neste arquivo
v = 1:1:10;
m = media(v);
s = sprintf('\n A média é: %4.2f', m);
disp(s);
% final do programa teste2.m
```

Agora crie o seguinte arquivo, com o nome de *media.m*

```
function x = media(u)
% function x = media(u) calcula a média do vetor u, colocando o resultado em x
```

```

x = sum(u)/length(u);
% final da subrotina media.m
% Na linha de comando do Matlab, digite:
teste2
echo on
teste2
echo off

```

12- Criando um programa exemplo de gráfico 3D

```

% Abra outro arquivo, salvando-o com nome de teste3.m
% Digite os seguintes comandos neste arquivo
clear
n = 30;
m = 30;
for i = 1:m
    for j = 1:n
        a(i,j) = sqrt(i+j);
    end
end
b = [a+0.5 a'-0.5;
(a.^2)/5 ((a'-0.1).^2)/2];
mesh(b)

```

d. Exercícios Desafiadores

- 1- Calcule a raiz da equação $f(x)=x^3-9x+3$ pelo Método da Bisseção no intervalo $I=[0,1]$ com $\epsilon=10^{-3}$ e número máximo igual a 15.
- 2- Calcule as raízes da equação $f(x) =x^3-9x+3$ pelo Método de Newton-Raphson nos intervalos $I1=(-4,-3)$, $I2=(0,1)$ e $I3=(2,3)$ com $\epsilon =10^{-3}$ e com número máximo de iterações igual a 10.
- 3- Resolva o sistema linear abaixo usando o Método de Eliminação de Gauss e compare com o resultado obtido pelo MATLAB.
- 4- Usando Fatoração LU resolva o sistema linear mostrado no exercício 3 e compare com os resultados obtidos pelo MATLAB (comando lu).
- 5- Uma grande placa de 300 mm de espessura ($k=37,25$ kcal / m $^{\circ}\text{C}$) contém fontes de calor uniformemente distribuídas ($q= 9 \times 105$ kcal / h m³). A temperatura numa face é 1000°C e calor é transferido para essa superfície, $q(0)$, a 2500 kcal/hm². Escreva um programa para determinar a distribuição de temperatura em regime permanente na placa plotando os resultados. A placa deve ser dividida em fatias iguais e deve ser um dado de entrada do programa.
- 6- Uma chaminé de tijolos de 60 m de altura, com um diâmetro de 1,80 m, tem uma camada de tijolos refratários ($k = 0,9$ kcal / h m $^{\circ}\text{C}$) de 110 mm de espessura e uma parede externa de tijolos de alvenaria ($k = 0.5$ kcal / h m $^{\circ}\text{C}$) que varia linearmente de uma espessura de 600 mm na base até uma espessura de 200 mm no topo. O coeficiente de transmissão de calor entre o gás da chaminé e a parede é 50 kcal / h m² $^{\circ}\text{C}$, e entre a parede externa e o ar é 15 kcal / h m² C. Se o gás da chaminé está a 300 $^{\circ}\text{C}$ e o ar está a 4 $^{\circ}\text{C}$, calcule numericamente a perda de calor da chaminé dividindo-a em pedaços que representem um anel circular com raio crescente. Calcule a resistência térmica total e plote os resultados para intervalos que vão de 1 a 20.

e. Exercícios Extras

Exercício 1 - Faça um programa que desenhe unia pirâmide 3D. Utilize o `mesh()`.

Exercício 2 - Copie o gráfico de uma senóide para um arquivo texto do “WORD”. Siga os seguintes passos: 1º após ter gerado o gráfico, faça `print-dmeta` (no MATLAB); 2º Pressione `ALT-TAB` até entrar no “WORD” ou então abra o “WORD”; 3º Posicione o cursor no local do texto onde o gráfico deva entrar; 4º Digite `Ctrl-V`; 5º Ajuste a escala vertical do gráfico com o editor de gráficos do “WORD”.

Exercício 3 - Repita o exercício 2 com `-dbitmap` no lugar de `-dmeta` e compare o tamanho (em Kb) dos dois arquivos texto.

Exercício 4 - Resolva o circuito dado na figura abaixo (encontre i_1 e i_2) utilizando a inversão de matrizes do MATLAB. Faça um programa para isto. Adote $R_1 = 5\Omega$, $R_2 = 10\Omega$, $R_3 = 5\Omega$, $R_4 = 15\Omega$, $R_5 = 20\Omega$, $V_1 = 10,0\text{ V}$, $V_2 = 20,0\text{ V}$.

Resp.: $i_1 = 0,01026\text{ A}$ e $i_2 = 0,4615\text{ A}$.

Exercício 5 - Supondo que a fonte V_2 esteja em curto, ou seja, $V_2 = 0,0\text{ V}$, quais os valores de i_1 e i_2 ?

Resp.: $i_1 = 0,4103\text{ A}$ e $i_2 = -0,1538\text{ A}$.

Exercício 6 - Gere um vetor com N elementos aleatórios. Escreva uma função que tenha como entrada o vetor, e retome o índice e o valor do maior elemento do vetor, utilizando o comando `if`.

Exercício 7 - Escreva um programa (utilizando o comando `while`) que aceite entradas numéricas pelo teclado. Os valores devem ser números entre 0 e 5, e caso o usuário digite algum valor fora deste intervalo, o programa é encerrado.

Exercício 8 - Em uma sala estão 8 pessoas, reunidas em uma mesa circular. Cada uma escolhe um número aleatoriamente e pega o seu número e soma com os números das pessoas ao lado, a sua esquerda e direita. Passa-se as 8 somas para você, que estava fora da reunião. Como você descobre o número que cada pessoa escolheu ? Utilize o MATLAB.

13. Bibliografia

- http://www.mathworks.com/company/newsletters/news_notes/clevescorner/jan06.pdf
- Reginaldo de Jesus Santos, "Introdução ao Matlab," Departamento de Matemática, ICEX, UFMG
<http://www.mat.ufmg.br/~regi>
- Frederico Ferreira Campos Filho, "Apostila de Matlab," Departamento de Ciência da Computação, ICEX, UFMG
- Grupo PET, "Curso de MATLAB," Engenharia Elétrica - UFMS
<http://www.del.ufms.br/tutoriais/matlab/apresentacao.htm>
- Frederico F. Campos, Introdução ao MATLAB
- B.P. Lathi, Sinais e Sistemas lineares