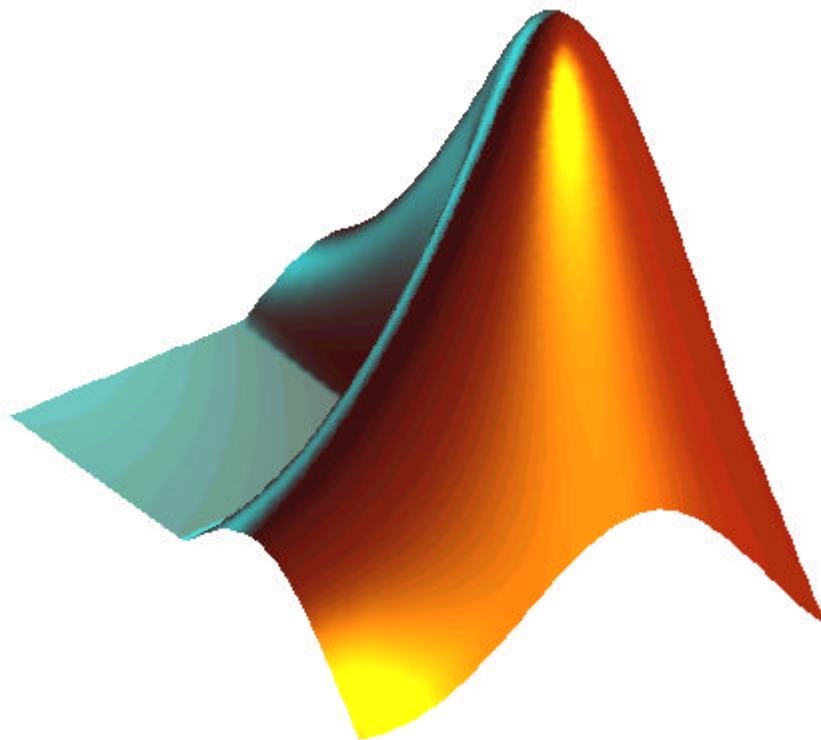


MATLAB

Curso Introdutório



Centro Brasileiro de Pesquisas Físicas

2002

Sumário

1. INTRODUÇÃO	2
1.1 O que é o Matlab?	2
2. INICIANDO	3
2.1 Variáveis	3
2.1.1 ENTRANDO COM VALORES	3
2.1.2 VARIÁVEIS PERMANENTES	4
2.2 Expressões e Comandos Básicos	4
2.2.1 SALVANDO/APAGANDO O WORKSPACE	5
2.2.2 O COMANDO MAIS IMPORTANTE	5
2.2.3 EXERCÍCIOS	6
3. FAMILIARIZANDO-SE COM MATRIZES.....	7
3.1 Manipulação de Matrizes	7
3.1.1 ELEMENTOS DE UMA MATRIZ	7
3.1.2 GERANDO VETORES	8
3.1.3 MATRIZES DENTRO DE MATRIZES	9
3.1.4 EXERCÍCIOS	10
3.2 Operações Elemento – por – Elemento	11
3.3 Operadores Lógicos e Relacionais	12
3.4 Operadores e Manipulação de Matrizes	13
3.4.1 EXERCÍCIOS	13
3.5 Algumas Funções.....	14
4. POLINÔMIOS.....	14
4.1 Representando Polinômios no MATLAB.....	14
4.2 Funções Relacionadas à Polinômios.....	14
4.2.1 EXERCÍCIOS	15
5. GRÁFICOS.....	15
5.1 Gráficos 2-D	15
5.1.2 EXERCÍCIOS	17
5.2 Gráficos 3-D	18
5.2.1 FUNÇÕES ELEMENTARES PARA GRÁFICOS 3-D.....	18
5.2.2 MESHGRID.....	18
5.2.3 MELHORANDO A APARÊNCIA.....	19
5.2.4 EXERCÍCIOS	20
6. PROGRAMAÇÃO.....	20
6.1 Controladores de Fluxo	21
6.1.1 FOR	21
6.1.2 WHILE.....	21
6.1.3 IF.....	22
6.1.4 BREAK, INPUT, PAUSE	22
6.2 Arquivos M	23
6.2.1 SCRIPTS.....	23
6.2.2 ARQUIVOS-FUNÇÃO	24
6.2.3 FUNÇÕES FUNÇÃO.....	25
6.2.4 EXERCÍCIOS	26
7. BIBLIOGRAFIA.....	26
8. RESPOSTA DOS EXERCÍCIOS.....	27

1. INTRODUÇÃO

1.1 O que é o Matlab?

MATLAB (*Matrix Laboratory*) é um software para computação numérica e visualização de alta performance, fácil de ser usado, onde os problemas e soluções são expressos quase que da mesma forma que no papel.

Seus elementos básicos são matrizes que não requerem dimensionamento. Ele permite implementar e resolver problemas matemáticos muito mais rápida e eficientemente que através de outras linguagens como C, Basic, Pascal ou Fortran.

Ainda, o MATLAB possui uma família de aplicativos específicos (*toolboxes*¹), que são coleções de funções usadas para resolver determinados problemas tais como: otimização, manipulação algébrica, redes neurais, processamento de sinais, simulação de sistemas dinâmicos, entre outros.

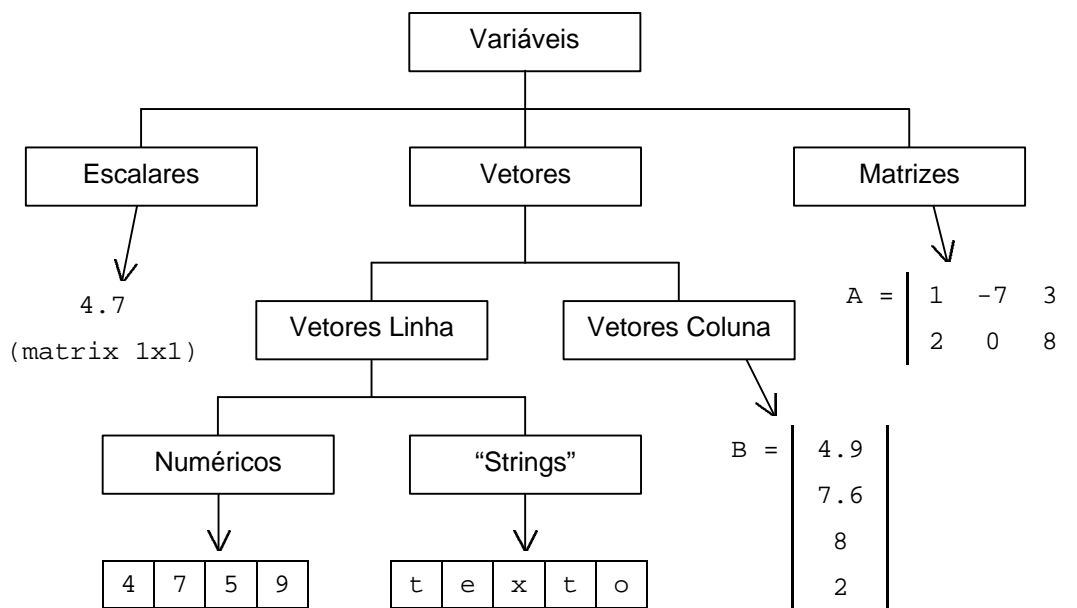
Provavelmente, a característica mais importante do MATLAB é a sua extensibilidade, que permite que engenheiros, matemáticos cientistas, e até mesmo você, contribuam para o enriquecimento.

¹ Para maiores informações sobre as últimas novidades do software, consulte a página na internet da MathWorks Inc. em <http://www.mathworks.com>

2. INICIANDO

2.1 Variáveis

O MATLAB trabalha essencialmente com um tipo de variável: uma **matriz** contendo números, complexos ou não (um escalar é uma matriz 1 x 1). Em alguns casos, um tratamento especial é dado a uma matriz **1 x 1 (escalar)** ou a matrizes **1 x n** ou **n x 1 (vetores)**.



2.1.1 ENTRANDO COM VALORES

No MATLAB não é necessário que sejam declaradas as variáveis para iniciá-las, como é feito em outras linguagens de programação. Ao jogar dados numa variável, o programa aloca memória automaticamente.

A maneira mais fácil de entrar com pequena quantidade de valores é digitando diretamente os dados:

- envolva os elementos com colchetes, [];
- separe cada elemento com espaços ou vírgulas;
- use ponto-e-vírgula (;) para indicar fim da linha.

Por exemplo, para entrar com a matriz abaixo na memória do computador, e guardá-la na variável A:

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

Basta digitar:

```
» A=[1 2 3;4 5 6; 7 8 9]
```

Que a saída será:

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

OBS: Para que o computador realize a operação e não mostre a saída, basta terminar a expressão com ponto-e-vírgula (;). Isto é muito útil para evitar que o computador fique mostrando números de cálculos intermediários e para acelerar as operações.

2.1.2 VARIÁVEIS PERMANENTES

Existem algumas variáveis que são intrínsecas ao MATLAB e que não podem ser apagadas. Algumas são interessantes:

ans	Resposta mais recente, que não foi atribuída a nenhuma variável.	flops	Contador de operações matemáticas.
eps	Precisão da máquina.	NaN	Not a Number (indeterminação)
realmax	Maior número de ponto flutuante.	inf	Infinito.
realmin	Menor número de ponto flutuante.	computer	Tipo de computador.
pi	3,14159265358979	why	Resposta sucinta.
i, j	Unidade imaginária	version	Versão do MATLAB.

Para saber a precisão da máquina, basta digitar `eps`.

2.2 Expressões e Comandos Básicos

MATLAB é uma linguagem de expressão. Ele interpreta e avalia as expressões digitadas, que são geralmente na forma:

$$\text{variável} = \text{expressão}$$

Os números são em notação decimal. Pode-se criar números complexos basta escrever *i* (ou *j*) depois da parte imaginária. Alguns exemplos de números permitidos:

```
1/3          -99          .0001
9.63973     1.602E-20     6.025E23
3 + 2i      -3.1459i     3E5i
```

A precisão de todas as operações é `eps`, embora os números mostrados ocultem algumas casa decimais. Para mudar o formato de saída dos números, altere o item `numeric format` ou no menu `options`, ou então use o comando `format`.

Pode-se construir expressões com os operadores aritméticos usuais:

+	adição	/ e \	divisão
-	subtração	^	potenciação
*	multiplicação	'	matriz transposta

O MATLAB possui uma vasta gama de funções matemáticas elementares, com seno (`sin`), tangente (`tan`), logaritmo (`log10`), etc. Por exemplo, para calcular o seno de 5 e guardar na variável `x`:

```
» x=sin(5)
```

Subtraindo matrizes:

```
» A=[1 2 3;4 5 6;7 8 9];
```

```
» B=[4 5 6;1 2 3;8 7 6];
```

```
» C=A'-B
```

OBS: $X = A/B$ é a solução de $A * X = B$
 $X = B/A$ é a solução de $X * A = B$

2.2.1 SALVANDO/APAGANDO O WORKSPACE

Ao sair do MATLAB (através do comando `quit` ou `exit`) todas as variáveis do `workspace` são perdidas, a menos que sejam guardadas do comando `save`. Digitando `save`, todas as variáveis são salvas em um arquivo chamado `matlab.mat`. O comando `save nome_de_arquivo` salva as variáveis no arquivo. Já o comando:

`save nome_de_arquivo nome_da(s)_variável(is)`, salva somente as variáveis especificadas. O comando `load` carrega as informações salvas, e é análogo ao `save`.

O comando `clear nome_de_variável` apaga a variável do `workspace`. Digitar somente `clear` apaga todas as variáveis do `workspace`!

2.2.2 O COMANDO MAIS IMPORTANTE

O comando mais importante no MATLAB é o `help`, que fornece ajuda *on-line* sobre qualquer outro comando. Por exemplo, para obter ajuda sobre o comando `who`:

» help who

WHO List current variables.

WHO lists the variables in the current workspace.

WHOS lists more information about each variable.

WHO GLOBAL and WHOS GLOBAL list the variables in the global workspace.

Um comando igualmente importante é **lookfor**, que procura entre todas as funções do MATLAB a palavra-chave especificada.

» lookfor max

BITMAX Maximum floating point integer.

REALMAX Largest positive floating point number.

MAX Largest component.

2.2.3 EXERCÍCIOS

Comandos: **who**, **whos**, **eps**, **format**, **quit**, **save**, **load**, **clear**, **help**, **lookfor**.

Exercícios:

1) Armazene no *workspace* os seguintes valores:

a = 3.132;

b = -23.004;

c = 5*pi;

d = (3 5.4 7.43)

e = (-2.234 0 pi/2)

$$f = \begin{bmatrix} -9.81 \\ 0 \\ 1 \end{bmatrix}$$

$$g = \begin{bmatrix} 12e-8 \\ 4i \\ pi*i \end{bmatrix}$$

$$A = \begin{bmatrix} 0.32 & 2.5 + pi & 2 \\ 1e2 & 4 & 12 \\ 9 & 51 & 24 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 34 & 87 \\ 32 & 4.65 & 74 \\ 0 & 13 & -43 \end{bmatrix}$$

2) Verifique o resultado das seguintes das seguintes operações:

a) a + b + eps

e) g - c * f

b) c - b * (a / b)

f) A * B

c) d - e

g) a * A - B / c

d) e' + 2 * f

h) f * B

3) Verifique o resultado das seguintes operações:

a) sin(a) * log(b)

e) max(log(g + f + d'))*B

b) tan(c+eps) - asin(b)

f) sin(cos(tan(A)))

- c) $\text{mind}(d^2) - \max(e)$ g) $\text{inv}(A)$
d) $\log(f)$ h) $\text{inv}(A^t) * \cos(B)$

4) Atribua as seguintes expressões às variáveis:

- a) $3.34 * a - \pi/c$ para x
b) $\log(d + 34.0054)$ para y
c) $\log(A)$ para Z
d) $f^t * B$ para t

5) Salve as variáveis x, Z, B em um arquivo chamado exerc1.mat.

6) Saia do MATLAB, entre novamente e carregue as variáveis salvas anteriormente.

7) Apague a variável Z.

3. FAMILIARIZANDO-SE COM MATRIZES

3.1 Manipulação de Matrizes

3.1.1 ELEMENTOS DE UMA MATRIZ

Elementos de uma matriz podem ser qualquer expressão do MATLAB. Por exemplo:

```
» x=[-1.3 log(4.23^3) (1+2+3)/4*5]
```

Resulta em:

```
x =  
-1.3000      4.3266      7.5000
```

Elementos individuais de uma matriz podem ser referenciados com seus respectivos índices entre parêntesis. No exemplo anterior, para referenciar o segundo elemento do vetor x:

```
» x(2)  
ans =  
4.3266
```

Analogamente em uma matriz, com linha e coluna determinada tenho um elemento. Seja:

```
» x=[1 2 3;4 5 6;7 8 9]  
x =  
1      2      3  
4      5      6  
7      8      9
```



```
» x(2,3)
ans =
     6
```

Repare que a referência é sempre na forma *matriz(linha,coluna)*.

3.1.2 GERANDO VETORES

O **dois pontos** (:) é um caracter importante no MATLAB. Escrevendo:

```
» x = 1:8
```

Cria um vetor cujo primeiro elemento é 1, o último é 8 e o passo 1.

```
x =
     1     2     3     4     5     6     7     8
```

Pode-se modificar o passo:

```
» x = 1:1.5:8
```

```
x =
     1.0000     2.5000     4.0000     5.5000     7.0000
```

Os dois pontos significam *início : passo : fim*. O valor de passo pode ser qualquer número real ($\cong 0$). A notação (:) é muito útil para gerar tabelas e plotar gráficos, como veremos adiante.

```
» x=0:0.2:3;
```

```
» y=exp(-x) + sin(x);
```

```
» z=[x'y']
```

```
z =
     0     1.0000
     0.2000     1.0174
     0.4000     1.0597
     0.6000     1.1135
     0.8000     1.1667
     1.0000     1.2094
     1.2000     1.2332
     1.4000     1.2320
     1.6000     1.2015
     1.8000     1.1391
     2.0000     1.0446
     2.2000     0.9193
```

```

2.4000    0.7662
2.6000    0.5898
2.8000    0.3958
3.0       0.1909

```

3.1.3 MATRIZES DENTRO DE MATRIZES

É possível construir matrizes maiores a partir de matrizes menores. Por exemplo:

```

» A=[1 2 3;4 5 6;7 8 9];
» r=[13 32 5];
» A=[A;r]

```

```

A =
     1     2     3
     4     5     6
     7     8     9
    13    32     5

```

Seguindo o mesmo raciocínio, pode-se extrair matrizes menores a partir de uma maior. Já é sabido o comando

```

» x=A(1,3)
x =
     3

```

atribui à variável “x” o elemento da 1ª linha e 3ª coluna da matriz A. Da mesma forma que é possível atribuir um elemento de uma matriz (que é um escalar, ou seja, uma matriz 1 x 1), também é possível atribuir pedaços inteiros da mesma matriz. Por exemplo, seja a matriz A:

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 14 & 83 & 23 & 0 \end{vmatrix}$$

Pegar os elementos da 2ª linha e 3ª e 4ª colunas:

```

» A=[1 2 3 4; 5 6 7 8;9 10 11 12;14 83 23 0]

```

```

A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    14    83    23     0

```

```

» A(2,[3 4])

```

```

ans =
     7     8

```

O que aconteceu? Ao invés de passar um escalar como índice para as colunas da matriz A , passou-se o vetor $[3 \ 4]$. O MATLAB interpretou isto como sendo: pegue os elementos a_{23} e a_{24} . Um outro exemplo ainda na matriz A . Para pegar a parte selecionada.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 14 & 83 & 23 & 0 \end{bmatrix}$$

```

» A([1 2 3],[2 3])
ans =
     2     3
     6     7
    10    11

```

ou ainda um comando equivalente:

```

» A(1:3,2:3)
ans =
     2     3
     6     7
    10    11

```

Usar os dois pontos sozinhos significa todos os elementos da respectiva linha ou coluna:

```

» A(3,:)
ans =
     9    10    11    12

```

Este tipo de notação facilita enormemente a criação de programas.

3.1.4 EXERCÍCIOS

1) Sejam as matrizes abaixo:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 14 & 83 & 23 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 7.4 & \text{pi} & 0 \\ -4.01 & 2 & 3 \\ 0.1 & 10 & 0 \end{bmatrix}$$

- Atribua o elemento a_{32} à variável c
- Atribua o elemento b_{22} à variável c
- Atribua os elementos a_{11} , a_{12} e a_{13} à um vetor d
- Atribua a 3ª coluna da matriz B a um vetor d
- Atribua a 2ª linha de B à primeira linha de A . Dica: preencha os espaços restantes com 0.
- Atribua a 4ª linha de A à 2ª linha de A .

2) Gerar os seguintes vetores:

- a) x começa em 0, vai até 15, passo 1
- b) x começa em -3.4 vai até 8, passo 0.32
- c) x começa em 10, vai até 1, passo -1.23
- d) x começa em 0, vai até 15^2 , passo $10 \cdot \pi$

3) Quais destes comandos são válidos?

- a) $c = A(2,3)$
- b) $c = A[1 \ 2 \ 3]$
- c) $c = A([1 \ 2 \ 3],4)$
- d) $c = A(1:3,4)$
- e) $c = A([1:4],2)$
- f) $c = A(2,:)$
- g) $c = A(:,:)$
- h) $c = A(2:4,[1 \ 3 \ 4])$

4) Extrair das matrizes do item 1 as submatrizes selecionadas

3.2 Operações Elemento – por – Elemento

Operações elemento – por – elemento, ao contrário das operações tradicionais como multiplicação de matriz ou divisão de matriz, são operações entre elementos. Por exemplo:

```
» [1 2 3;4 5 6;7 8 9] * [1 2 3;4 5 6;7 8 9]
```

```
ans =
```

```
    30    36    42
    66    81    96
   102   126   150
```

```
» [1 2 3;4 5 6;7 8 9] .* [1 2 3;4 5 6;7 8 9]
```

```
ans =
```

```
     1     4     9
    16    25    36
    49    64    81
```

Pode-se perceber que no segundo caso p que ocorreu foi: elemento $a_{11} * b_{11}$, $a_{12} * b_{12}$, $a_{21} * b_{21}$,... Diferente do primeiro caso, onde ocorreu uma multiplicação de duas matrizes 3×3 .

As operações elemento por elemento são:

Símbolo	Operação
.*	multiplicação
./ ou \.	divisão
.^	potenciação

3.3 Operadores Lógicos e Relacionais

Existem seis operadores relacionais no MATLAB. São eles:

Símbolo	Operador
<	menor que
<=	menor ou igual que
>	maior que
>=	maior ou igual que
==	igual
~=	não igual

O resultado da comparação é 1 se verdadeiro e 0 se falso. por exemplo:

```
» 2 + 2 == 4
ans =
    1
```

No caso de comparação com matrizes, o resultado será uma matriz de 0 e 1:

```
» x=[2 3 4;5 2 7;9 2 7]
x =
     2     3     4
     5     2     7
     9     2     7
» x>4
ans =
     0     0     0
     1     0     1
     1     0     1
```

Para os operadores lógicos, tem-se:

Símbolo	Operador
&	e
	ou
~	não

Existem ainda algumas funções que são úteis com os operadores lógicos, com **any** ou **all**. Por exemplo:

```
» x = [1 2 3 4 5 6]
x =
     1     2     3     4     5     6
```

```
» any(x>5)
ans =
     1

» all(x>5)
ans =
     0
```

3.4 Operadores e Manipulação de Matrizes

Pode-se usar os vetores de 0 e 1, geralmente criados a partir de operações de comparação, como referência para matrizes. Seja a matriz A descrita anteriormente. para atribuir os elementos de A que satisfazem determinada comparação a uma outra variável, faz-se:

variável = A(comparação)

Por exemplo:

```
» b = A(A>5)
b =
     9
    14
     6
    10
    83
     7
    11
    23
     8
    12
```

3.4.1 EXERCÍCIOS

Comandos: **any**, **all**.

Exercícios:

- 1) Seja o vetor $x = 1:10$. Verifique as afirmativas:
 - a) $(x.^2) == (x.*x)$
 - b) $\text{any}((x.^3)>(3*x))$
 - c) $\text{all}((x./(x+1))>((x+1)./(x+1.5)))$
 - d) $A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9];$
 $A^2 == A.^2$

- 2) Extrair a partir do vetor x do exercício anterior:
- a) elementos maiores que 5
 - b) elementos diferentes de 3 e 5
 - c) elementos iguais a 2 5 8 9
 - d) elementos cujo resto da divisão por 3 é 1. Dica: utilize a função **rem**.

3.5 Algumas Funções

Eis abaixo alguns exemplos de funções básicas no MATLAB. Qualquer dúvida em como usá-las basta usar o comando **help**,

exp	e	poly	polinômio característico
log	logaritmo natural	det	determinante
log10	logaritmo base 10	abs	módulo
find	índice de matriz	sqrt	raiz quadrada
max	máximo valor	real	parte real de número complexo
min	mínimo valor	imag	parte imaginária de número complexo
mean	média aritmética	conj	conjunto de número complexo
std	desvio padrão	round	arredondar

4. POLINÔMIOS

4.1 Representando Polinômios no MATLAB

O MATLAB representa polinômios como vetores de uma linha, contendo os coeficientes ordenados em ordem decrescente. Por exemplo, o polinômio:

$$x^3 - 6x^2 - 72x - 27$$

é representado da forma

```
» p=[1 -6 -72 -27]
```

```
p =
```

```
1    -6   -72  -27
```

4.2 Funções Relacionadas à Polinômios

roots	raízes do polinômio	deconv	divisão
polyval	valor do polinômio no ponto	polyder	derivadas
conv	multiplicação	polyfit	melhor curva

Exemplo de ajuste de curva:

```
» x=1:7;
» y=[1.2 1.6 2.3 2.8 3.9 4.5 5.6];
» [x' y']
ans =
    1.0000    1.2000
    2.0000    1.6000
    3.0000    2.3000
    4.0000    2.8000
    5.0000    3.9000
    6.0000    4.5000
    7.0000    5.6000
» faj=polyfit(x,y,1);
» faj=polyval(faj,x);
» plot(x,y,'+blue',x,faj,'black')
```

4.2.1 EXERCÍCIOS

1) Sejam os polinômios $p = x^4 - 3x^2 + 5x - 30$ e $q = 2x^4 - 7x^3 + 2x - 15$. Calcule:

- | | |
|-----------------|--------------------------------------|
| a) $p \times q$ | e) 1ª derivada de p |
| b) $p \div q$ | f) 1ª derivada de p no ponto $x = 3$ |
| c) $p(2)$ | g) 1ª derivada de $p \div q$ |
| d) raízes q | |

5. GRÁFICOS

O MATLAB proporciona técnicas sofisticadas para visualização de dados. Ele trabalha essencialmente com **objetos gráficos**, tais como linhas e superfícies, cuja aparência pode ser controlada através de **propriedades de objeto**. Entretanto, como o MATLAB possui uma vasta gama de funções para gráficos que automatizam essas propriedades, na maioria das vezes não será necessário lidar com estes objetos.

5.1 Gráficos 2-D

A função básica para desenhar gráficos em duas dimensões é a função **plot**. Quando esta função recebe um conjunto de ponto x e y, ela desenha-os em um **plano cartesiano**. Por exemplo, seja o conjunto de pontos abaixo:

Para plotar o gráfico $y = f(x)$, primeiro cria-se um vetor x contendo os valores de x , e depois um vetor y com os valores de y . Então chama-se a função **plot**, que é usada da seguinte maneira:

x	$y = x^2$
0	0
1	1
2	4
3	9
4	16
5	25

```
» x=[0 1 2 3 4 5];
```

```
» y=x.^2;
```

```
» plot(x,y)
```

Lembrando que para definir o vetor x pode-se usar os comandos mostrados na seção 3.1.2 (“Gerando Vetores”) e colocar a função diretamente num dos parâmetros da função **plot**. Por exemplo:

```
» plot(x,y)
```

```
» x=[0:5];
```

```
» plot(x,x.^2)
```

O MATLAB criará uma janela com a figura do gráfico (vide figura 1). Na verdade, a função **plot** recebe um número variável de argumentos. Sua forma mais geral é $plot(x_1, y_1, jeito_1, x_2, y_2, jeito_2, \dots, x_n, y_n, jeito_n)$. Ou seja, você pode traçar mais de uma curva no mesmo gráfico. O argumento *jeito* representa as várias opções para o gráfico, que pode ser qualquer um dos seguintes *strings*²:

y	amarelo	w	branco	+	cruz
m	roxo	k	preto	-	sólida
c	azul claro	--	tracejada	*	estrela
r	vermelho	.	ponto	:	pontilhada
g	verde	o	círculo	-.	traço ponto
b	azul	x	x		

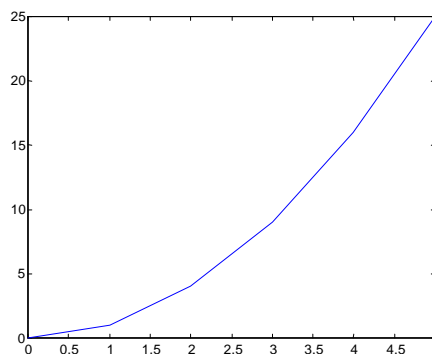


Figura 1 – Gráfico $f(x) = x^2$

² Um *string* é uma seqüência de caracteres que o MATLAB interpreta como um texto. Os *strings* são sempre denotados entre apóstrofes.

Seja a função $y = e^{-x} \cdot \sin(x)$. Como seria seu gráfico no intervalo $[0; e\pi]$?

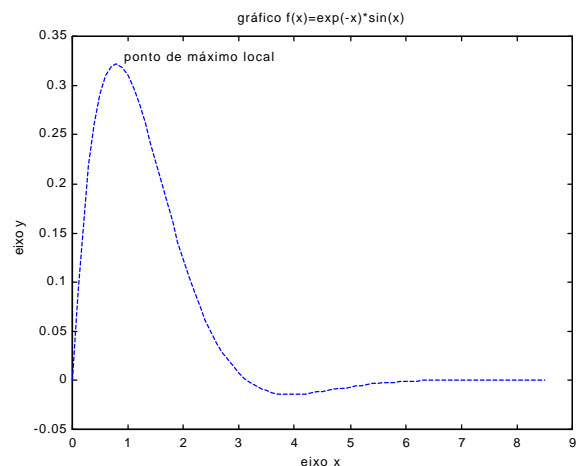
Para desenhá-lo, primeiro cria-se um vetor “x” do tamanho do intervalo desejado com um passo suficientemente pequeno para que a curva do desenho seja suave (um passo 0.1 neste caso é suficiente). Depois cria-se o vetor “y = f(x)” (ou coloca-se diretamente a função no parâmetro da função plot), e plota-se o gráfico com o comando plot. Alguns detalhes podem ser acrescentados:

title	título do gráfico	xlabel	nome do eixo x
text	escreve no local especificado	ylabel	nome do eixo y
gtext	escreve texto no usando mouse	grid	desenha linhas de grade
semilogx	gráfico mono-log em x	semilogy	gráfico mono-log em y
loglog	gráfico di-log	axis	intervalo dos eixos no gráfico

```

» x=[0:0.1:exp(1)*pi];
» y=exp(-x).*sin(x);
» plot(x,y,'--b');
» title('gráfico f(x)=exp(-x)*sin(x)');
» xlabel('eixo x');
» ylabel('eixo y');
» gtext('ponto de máximo local');

```



O gráfico da figura 2 foi gerado a partir dos comandos acima.

Figura 2 – Exemplo do uso dos vários comandos relacionados aos gráficos

5.1.2 EXERCÍCIOS

Comandos: **plot**, **title**, **xlabel**, **ylabel**, **axis**, **gtext**, **grid**, **semilogx**, **loglog**

Exercícios:

1) Plote o gráfico das seguintes funções, no intervalo especificado:

a) $y = x^3 - 5x + 2$, $x \in [-20; 20]$

c) $y = \cos(e^x)$, $x \in [0; 20]$

b) $y = \sin(x) \cdot \cos(x)$, $x \in [-2\pi; \pi]$

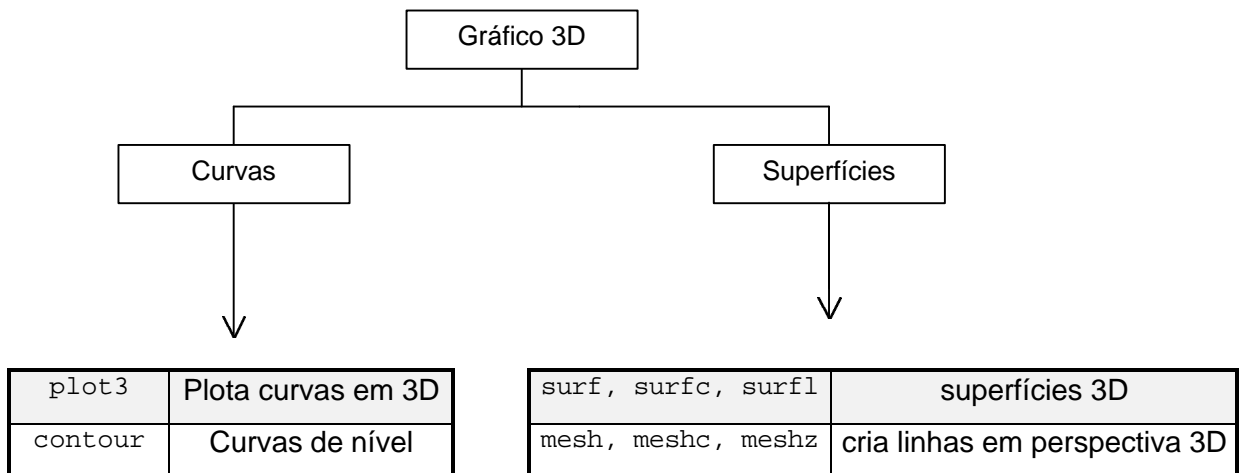
d) $y = d(x^3 - 5x + 2)/dx$, $x \in [0; 10]$

2) Plote em escala mono-log ou di-log as funções acima, acrescentando elementos como título, nome aos eixos, etc.

5.2 Gráficos 3-D

5.2.1 FUNÇÕES ELEMENTARES PARA GRÁFICOS 3-D

O MATLAB cria uma variedade de funções para gráficos em 3 dimensões. Entre elas:



Por exemplo:

```
» t=0:pi/50:10*pi;
» plot3(sin(t),cos(t),t);
```

Gera a figura 3. Todos os outros comandos de escala, título, nome aos eixos continuam valendo (nome ao eixo z: **zlabel**).

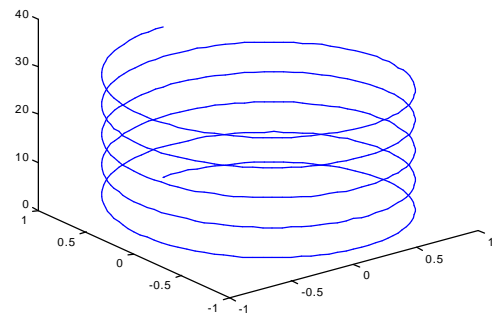


Figura 3 – Gráfico de uma hélice circular

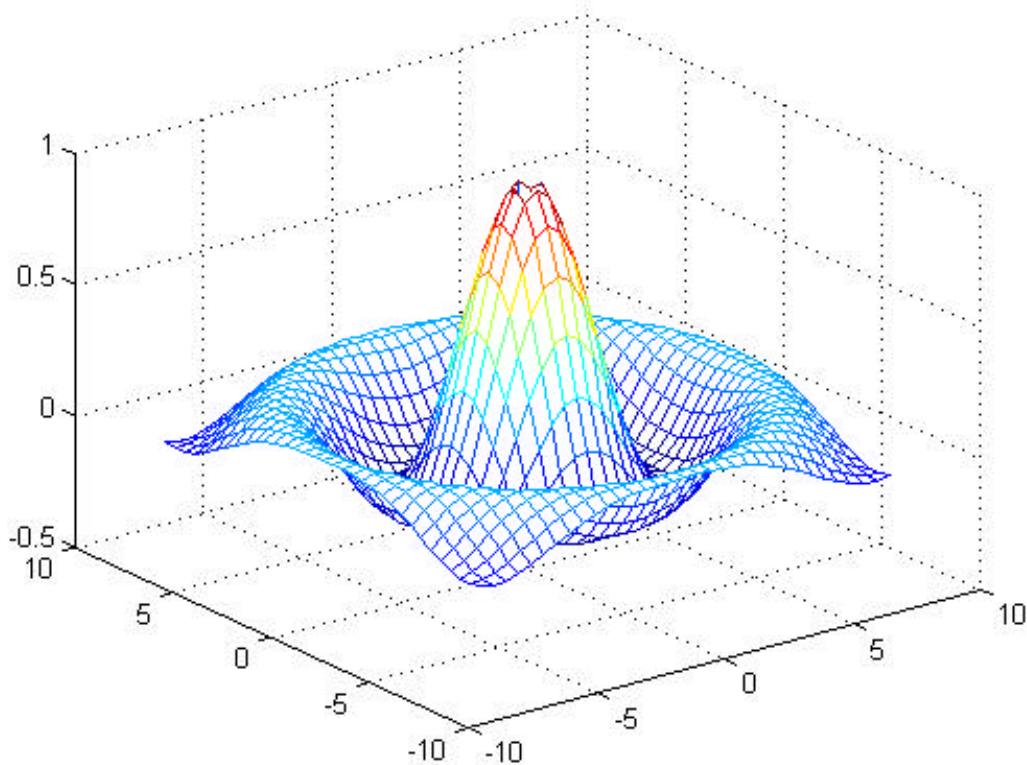
5.2.2 MESHGRID

O MATLAB define uma superfície do tipo *mesh* (rede) pelas coordenadas Z sobre um plano x-y. Superfícies tipo *mesh* são úteis para visualizar matrizes demasiadamente grandes para serem mostradas na forma numérica, ou para plotar funções de duas variáveis.

O primeiro passo para plotar uma função de 2 variáveis $z=f(x,y)$ é gerar matrizes X e Y contendo linhas e colunas repetidas, respectivamente, para funcionarem como o domínio da função. A função **meshgrid** transforma o domínio especificado por dois vetores x y em duas matrizes X e Y. Essas matrizes então são usadas para avaliar a função de 2 variáveis. Por exemplo, seja a função:

$$f(x,y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$$

```
» plot3(sin(t),cos(t),t);  
» [X,Y]=meshgrid(-8:0.5:8,-8:0.5:8);  
» r= sqrt(X.^2+Y.^2)+eps;  
» Z=sin(r)./r;  
» mesh(X,Y,Z)
```



5.2.3 MELHORANDO A APARÊNCIA

É possível especificar o ponto de vista no qual se enxerga a figura usando o comando **view**. O comando **view** recebe dois argumentos. O primeiro é a rotação em graus no plano xy , e o segundo é a elevação em graus do ponto de vista. O padrão é `view(-37.5,30)`.

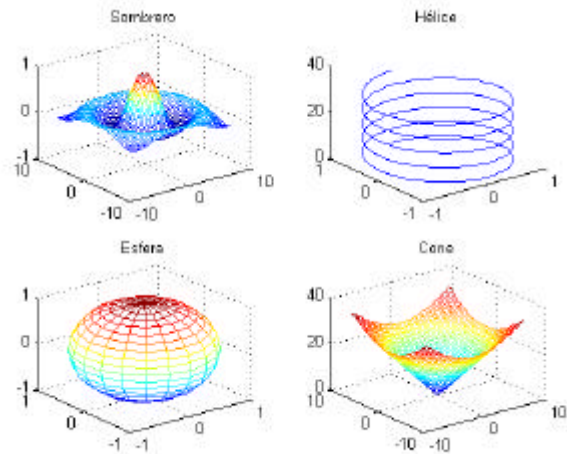
Também é possível colocar vários tipos de gráficos em uma mesma figura, usando o comando **subplot**. Por exemplo, `subplot(m,n,p)` quebra a figura em m por n subgráficos e seleciona o p -ésimo como o atual. Mais detalhes ver help on-line.

É possível ainda mudar o mapa de cores dos gráficos. Para tal, utilize a função **colormap**.

```

» subplot(221)
» mesh(X,Y,Z)
» title('Sombrero')
» subplot(222)
» plot3(sin(t),cos(t),t);
» title('Hélice')
» subplot(223)
» mesh(a,b,c)
» title('Esfera')
» subplot(224)
» mesh(X,Y,3*sqrt(X.^2+Y.^2))
» title('Cone')

```



5.2.4 EXERCÍCIOS

Comandos: **plot3**, **mesh**, **contour**, **surf**, **meshgrid**, **view**, **subplot**, **colormap**

1) Plote as seguintes funções no intervalo especificado:

a) $f(x,y) = x^2 + y^2$, $x, y \in [-5;5]$

e) $f(x,y) = (x + y)/(x - y)$, $x, y \in [-10;10]$

b) $f(x,y) = (1 - x^2 - y^2)^{1/2}$, $x, y \in [-0.5;0.5]$

f) $f(x,y) = x \cdot y / (x^2 + y^2)$, $x, y \in [-10;10]$

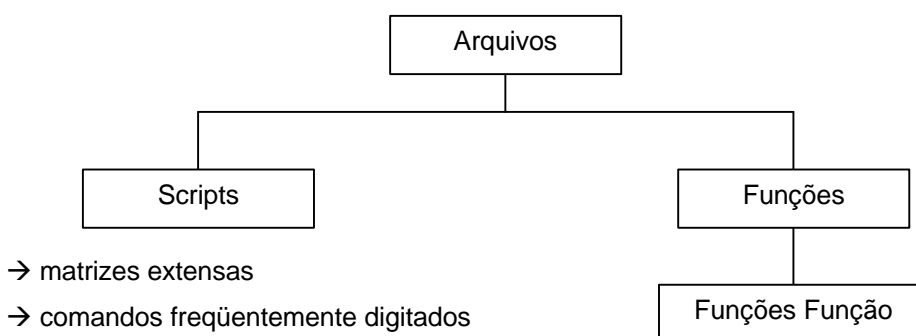
c) $f(x,y) = x \cdot y$, $x, y \in [0;1]$

g) $f(x,y) = \text{sen}(x/2) \cdot \text{cos}(y/3)$, $x, y \in [-\pi;\pi]$

d) $f(x,y) = \text{arctg}(x^2 + y^2)$, $x, y \in [-8;8]$

2) Plote os seis primeiros gráficos do item anterior na mesma figura.

6. PROGRAMAÇÃO



Outra grande vantagem do MATLAB é a facilidade para criar programas, da mesma forma que linguagens como o C, BASIC ou Fortran.

6.1 Controladores de Fluxo

6.1.1 FOR

O laço **for** permite um comando, ou grupo de comandos, repetir-se um número determinado de vezes. A forma geral de um laço **for** é:

```
for variável = expressão
    comandos
end
```

Um laço **for** é sempre terminado com um **end**.

Por exemplo,

```
» for i = 1:10
v(i)=3*i;
end
```

Cria um vetor “v” com 10 elementos:

```
» v
v =
```

```
3     6     9    12    15    18    21    24    27    30
```

6.1.2 WHILE

O laço **while** permite que o comando seja repetido um número indefinido de vezes, enquanto uma condição lógica seja satisfeita. A forma geral do laço **while** é:

```
while (expressão verdadeira)
    comandos
end
```

Assim, como no caso de **for**, **while** precisa de um **end** para indicar o fim do laço. O exemplo abaixo calcula o fatorial de n enquanto $n! < 10^{100}$.

```
» n=1;
```

```
» while prod(1:n)<1.e100
n=n+1;
end
» n
n =
    70
```

6.1.3 IF

O comando **if** estabelece caminhos alternativos para a fluência do programa. Sua forma geral é:

```
if condição verdadeira
    comandos
elseif condição 2 verdadeira
    comandos
else
    comandos
end
```

A primeira condição é sempre escrita com *if*, a última com *else*, e todas as intermediárias com *elseif*.

```
» a=round(10*rand(1));
» if a>5
b=3*a;
elseif a<5;
b=a/3;
else
b=a;
end
```

6.1.4 BREAK, INPUT, PAUSE

Além dos controladores de fluxo tradicionais, existem mais algumas funções que são bastante úteis para programação. Entre elas:

break – termina um laço

input – recebe dados através do teclado. Por exemplo, $n = \text{input}(\text{'Entre com valor: '})$ atribui o valor digitado no teclado à variável n .

pause – pausa na execução do programa, até que qualquer tecla ser digitada. *Pause(n)* dá uma pausa de n segundos.

6.2 Arquivos M

Quando uma linha de comandos é digitada no MATLAB, ele imediatamente processa e devolve o resultado. Porém, é possível executar seqüências de comandos, que podem ser guardados em arquivos. Arquivos que contém comandos do MATLAB são chamados arquivos M porque possuem extensão *.m*.

Um arquivo M é formado por uma seqüência de comandos ou de referência para outros arquivos. Eles podem ser criados a partir de qualquer editor de texto (como por exemplo, o *Notepad* do Windows), e são arquivos de texto comuns. Existem dois tipos distintos de arquivos: *Scripts* e *Funções*. Existe ainda uma classe especial de funções chamada *funções função*.

6.2.1 SCRIPTS

Os arquivos *script* automatizam uma seqüência de comandos. Quando um *script* é chamado, o MATLAB simplesmente executa os comandos contidos no arquivo. *Scripts* são úteis para entrar com matrizes muito extensas (pois erros de digitação podem ser facilmente corrigidos) e comandos que seriam digitados freqüentemente.

O exemplo abaixo foi escrito do editor de texto *Notepad*, e calcula os primeiros números de Fibonacci, mostrando o resultado em um gráfico.

```
% Arquivo M que calcula os primeiros
% números de Fibonacci
f = [1 1];
i=1;
while f(i) + f(i+1) < 1000
    f(i+2) = f(i) + f(i+1)
    i=i+1;
end
plot(f)
```

O símbolo “%” significa comentários. Tudo que estiver na linha após “%” não será considerado pelo MATLAB.

Estando este arquivo salvo em um dos diretórios *path* do MATLAB com uma extensão *.m*, toda vez que seu nome for digitado, todas as linhas de comandos acima serão executadas.

OBS: Para saber o *path* do MATLAB, use o comando **path**. Com este comando é possível inclusive alterar o *path* original. Lembre-se de salvar o arquivo com uma extensão *.m*, com a opção *salvar como tipo: todos os arquivos (*.*)*.

6.2.2 ARQUIVOS-FUNÇÃO

Uma *função* difere de um *script* já que argumentos podem ser passados para a função, e as variáveis criadas e manipuladas na função são locais para a mesma.

Na primeira linha de um arquivo função deverá aparecer a palavra *function*, definirá o nome da função.

```
function y = escal(a,b)
% ESCAL Produto escalar de dois vetores
%
% ESCAL retorna um vetor que é o resultado do produto
% escalar de dois vetores.
% Os dois vetores devem ser do mesmo tamanho
if size(a) ~= size(b)
    error('Erro: vetores não tem mesmo tamanho');
end
y=sum(a.*b);
```

Salve este texto como um arquivo **escal.m** no **path** do MATLAB. Crie dois vetores de mesmo tamanho e chame a função *escal*. A resposta será o produto escalar de dois vetores.

OBS: As primeiras linhas de comentários, que começam na 2ª linha do arquivo são tratadas pelo *help on-line* como a explicação da função, sendo que a 2ª linha é usada pelo comando *lookfor*. Por exemplo,

```
function y=próximo(x)
% PRÓXIMO Número consecutivo
% PRÓXIMO(x) retorna o próximo número natural depois de x.
%
% Veja também ANTERIOR, BLABLABLABLA
```

Quando dor digitado **help próximo** todas as linhas de comentários vão aparecer. E ao

digitar **lookfor próximo**, o MATLAB irá procurar todas as funções que contenham esta palavra, e mostrar suas 2^{as} linhas.

6.2.3 FUNÇÕES FUNÇÃO

A *função função* (*Function Functions*) é uma classe especial de função do MATLAB, que ao invés de receber variáveis numéricas como argumento (como no item 6.2.2) recebem strings que são nomes de funções.

Abaixo algumas funções função e suas utilidades:

fplot	gráfico de uma função	fzero	raiz de uma função de uma variável
quad	integração numérica	fmin/fmins	mínimo de uma função

As funções matemáticas são representadas por arquivos-função. Por exemplo, seja a função representando a velocidade de uma partícula:

$$v(t) = \sin(t) \times t^2 + 8t + 1$$

Para plotar o gráfico da velocidades em função do tempo existem duas opções:

- criar um vetor “t” do tamanho desejado, criar um vetor $v = \sin(t) \cdot t.^2 + 8 \cdot t + 1$, e usar a função plot com v e t;
- criar um arquivo função com a função “v(t)” e usar a função fplot.

A vantagem da 2ª opção é que o MATLAB escolherá pontos suficientemente espaçados para que a curva seja suave. Crie um arquivo chamado *velocid*:

```
» !notepad velocid.m
function v=velocid(t)
% VELOCID velocidade de uma
% partícula num instante t
%
v=sin(t).*t.^2+8*t+1;
```

Agora usando o comando **fplot**:

```
» fplot('velocid',[0,12])
```

Da mesma forma, para avaliar o espaço percorrido no mesmo intervalo de tempo (integral da função), ou quando a partícula está parada ($v=0$), basta usar as outras funções:

```
» x=quad('velocid',0,12)
x =
453.2952
```

```
» t2=fzero('velocid',10)
t2 =
    10.3261
```

6.2.4 EXERCÍCIOS

- 1) Crie *scripts* para as seqüências de comandos
 - a) item 5.1.1
 - b) item 5.2.2
 - c) item 6.2.1
 - d) que geram números primos de 2 a 1000

- 2) Crie uma função que:
 - a) calcule a média de um vetor
 - b) calcule o produto de duas matrizes
 - c) diz se um número é positivo ou negativo, par ou ímpar
 - d) dados T e v , a e b calcula a pressão de um gás de Van der Waals

- 3) Crie um *script* que plote o gráfico $p \times v$ de um gás de Van der Waals cujas constantes a e b sejam determinadas ao carregar o *script* e cuja a temperatura seja digitada pelo usuário. dica: use a função criada no exercício 2-d.

- 4) Plote o gráfico do item 2-d usando *fplot*. Encontre o mínimo da função através de *fmin*. Assuma v , a , b constantes quaisquer.

- 5) Calcule a integral de:
 - a) $f(x) = \text{sen}(x)$ entre 0 e 1
 - b) $f(x) = x^2 - 6x + 7$ entre -3 e 3

7. BIBLIOGRAFIA

- **MATLAB User's Guide**, The MathWorks Inc.
- TODESCO, José Leomar, **Curso Introdutório de MATLAB**, UNIVERSIDADE FEDERAL DE SANTA CATARINA, 1995.

8. RESPOSTA DOS EXERCÍCIOS

2.2.3

1)

```
» f=[-9.81;0;1]
```

f =

```
-9.8100
```

```
0
```

```
1.0000
```

```
» g=[12e-8; 4i; pi*i]
```

g =

```
0.0000
```

```
0 + 4.0000i
```

```
0 + 3.1416i
```

```
» A=[0.32 2.5+pi 2; 1e2 4 12; 9 51 24]
```

A =

```
0.3200    5.6416    2.0000
```

```
100.0000    4.0000    12.0000
```

```
9.0000    51.0000    24.0000
```

```
» B=[5 34 87;32 4.65 74; 0 13 -43]
```

B =

```
5.0000    34.0000    87.0000
```

```
32.0000    4.6500    74.0000
```

```
0    13.0000   -43.0000
```

2)

a) -19.8720

b) 12.5760

c) 5.2340 5.4000 5.8592

d) -24.8540

```
0
```

```
3.5708
```

e) 1.0e+002 *

```
1.5410
```

```
0 + 0.0400i
```

```
-0.1571 + 0.0314i
```

f) 1.0e+003 *

```
0.1821    0.0631    0.3593
```

```
0.6280    3.5746    8.4800
```

```
1.6770    0.8552    3.5250
```

g)

```
0.6893    15.5050    0.7254
```

```
311.1628    12.2320    32.8730
```

```
28.1880    158.9044    77.9055
```

h) -49.0500 -350.5400 -896.4700

3)

a) 0.0301 + 0.0301i

b) 1.5708 - 3.8283i

c) $\min(d.^2) - \max(e)$

```
7.4292
```

d) 2.2834 + 3.1416i

```
- Inf
```

```
0
```

e) 1.0e+002 *

```
0.0959+0.1571i    0.6523+1.0681i    1.6690+2.7332i
```

```
0.6139+1.0053i    0.0892+0.1461i    1.4196+2.3248i
```

```
0
```

```
0.2494+0.4084i    -0.8249-1.3509i
```

f) 0.8108 0.6696 -0.5450

```
0.7396    0.3906    0.7205
```

```
0.7830    0.5804   -0.5095
```

g) 0.1739 0.0113 -0.0201

```
0.7723    0.0035   -0.0661
```

```
-1.7064   -0.0116    0.1897
```

h) -1.0128 -1.7442 -0.7156

```
-0.0055   -0.0203    0.0006
```

```
0.1288    0.1933    0.0825
```

4)

a) x = 10.2609

b) y = 3.6111 3.6739 3.7241

c) Z =

```
-1.1394    1.7302    0.6931
```

```
4.6052    1.3863    2.4849
```

```
2.1972    3.9318    3.1781
```

d) t=f*B

5) save exerc1.mat x Z B

6) load exerc1.mat

7) clear Z

3.1.4

1) a) c=A(3,2) c=10

b) c=B(2,2) c=2

c) d=A(1,[1 2 3]) d = 1 2 3

d) d=B(:,3)

```
d = 0
```

```
3
```

```
0
```

e) A(1,:)= [B(2,:),0]

f) A(2,:)=A(4,:)

2)

a) x=0:1:15

b) x=-3.4:0.32:8

- c) `x=10:-1.23:1`
 d) `x=0:10*pi:15^2`

3)

- a) válido
 b) inválido, sugestão: `» c=A([1 2 3],:)`
 c) válido
 d) válido
 e) válido
 f) válido
 g) válido, resulta na própria matriz A
 h) válido

4) `t=A([1 2],[2 3])` `t=B(1,:)`

3.4.1

a) 1 1 1 1 1 1 1 1 1 1

b) 1

c) 0

d)

0 0 0

0 0 0

0 0 0

2)

a) `t=x(x>5)`

b) `t=x(x~=3 & x~=5)`

c) `t=x(x==2 | x==5 | x==8 | x==9)`

d) `t=x(rem(x,3)==1)`

4.2.1

1)

a) `» conv(p,q)`

2 -7 -6 33 -110 204 55 -135 450

b) `» deconv(p,q)`

0.5000

d) `polyval(p,2)`

-16

e) `» roots(q)`

3.5850

0.5709 + 1.1745i

0.5709 - 1.1745i

-1.2268

f) `» polyval(polyder(q),3)`

g) `» polyder(deconv(p,q))`

5.1.2

1) a) `x=[-20:20];`

`y=x.^3-5*x+2;`

`plot(x,y)`

b) `x=[-2*pi:0.1pi];`

`y=sin(x).*cos(x)`

`plot(x,y)`

d) `x=0:0.1:10;`

`y=polyder([1 0 -5 2]);`

`y=polyval(y,x);`

`plot(y)`

2) Para aplicar escala usando mono-log ou di-log ao gráfico basta usar os comandos `semilogx(...)`, `semilogy(...)` ou `loglog(...)`.

5.2.4

1) a)

`x=[-5:0.5:5];`

`y=x;`

`[x,y]=meshgrid(x,y);`

`z=x.^2 + y.^2;`

`mesh(z)`

b)

`x=[-0.5:0.1:0.5];`

`y=x;`

`[x,y]=meshgrid(x,y);`

`z=sqrt(1 - x.^2 - y.^2);`

`mesh(z)`

c)

`x=[0:0.1:1];`

`y=x;`

`[x,y]=meshgrid(x,y);`

`z=x.*y;`

`mesh(z)`

d)

`x=[-10:0.5:10];`

`y=x;`

`[x,y]=meshgrid(x,y);`

`z=atan(x.^2 + y.^2);`

`mesh(z)`

e)

`x=[-10:0.5:10];`

`y=[-10:0.5:10];`

`[x,y]=meshgrid(x,y);`

`z=(x+y)./(x-y);`

`mesh(z)`

f)

`x=[-10:0.5:10];`

`y=[-10:0.5:10];`

`[x,y]=meshgrid(x,y);`

`z=(x.*y)./(x.^2-y.^2);`

`mesh(z)`

g)

`x=[-pi:0.1:pi];`

`y=x;`

`[x,y]=meshgrid(x,y);`

`z=sin(x/2).*cos(y/3);`

`mesh(z)`

6.2.4

```

1) a) x=0:5;
      y=x.^2;
      plot(x,y)
      end

b) x=[-10:0.5:10];
   y=[-10:0.5:10];
   [x,y]=meshgrid(x,y);
   z=x.^2 + y.^2;
   mesh(z)
   end

c) % Arquivo M que calcula os primeiros
   % números de Fibonacci
   f=[1 1];
   i=1;
   while (f(i) + f(i+1))<1000
       f(i+2)=f(i) + f(i+1);
       i=i+1;
   end
   plot(f)

d)

2) a) funçiofn y=media(x)
      y=sum(x)/length(x)

b) function y=produto(a,b)
      y=a*b

c) function y = teste(x)
      if rem(x,2)==0
          disp('O número é par')
      else
          disp('O número é impar')
      end
      if abs(x)~=x
          disp('O número é negativo')
      else
          disp('O número é positivo')
      end

d) function y=pressao(t,v,a,b)
      y=(8.31*t)./(v-b)-a./v.^2;

3) clear
   clc
   v=[1:0.3:10];
   a=input('a=');
   b=input('b=');
   t=input('t=');
   plot(v,pressao(t,v,a,b))
   end

5) a)

```

```

» quad('sin',0,1)
ans =
    0.4597

b)
» !notepad poli.m
function y=poly(x)
y=x.^2-6*x+7
» quad('poli',-3,3)
ans =
    60

```