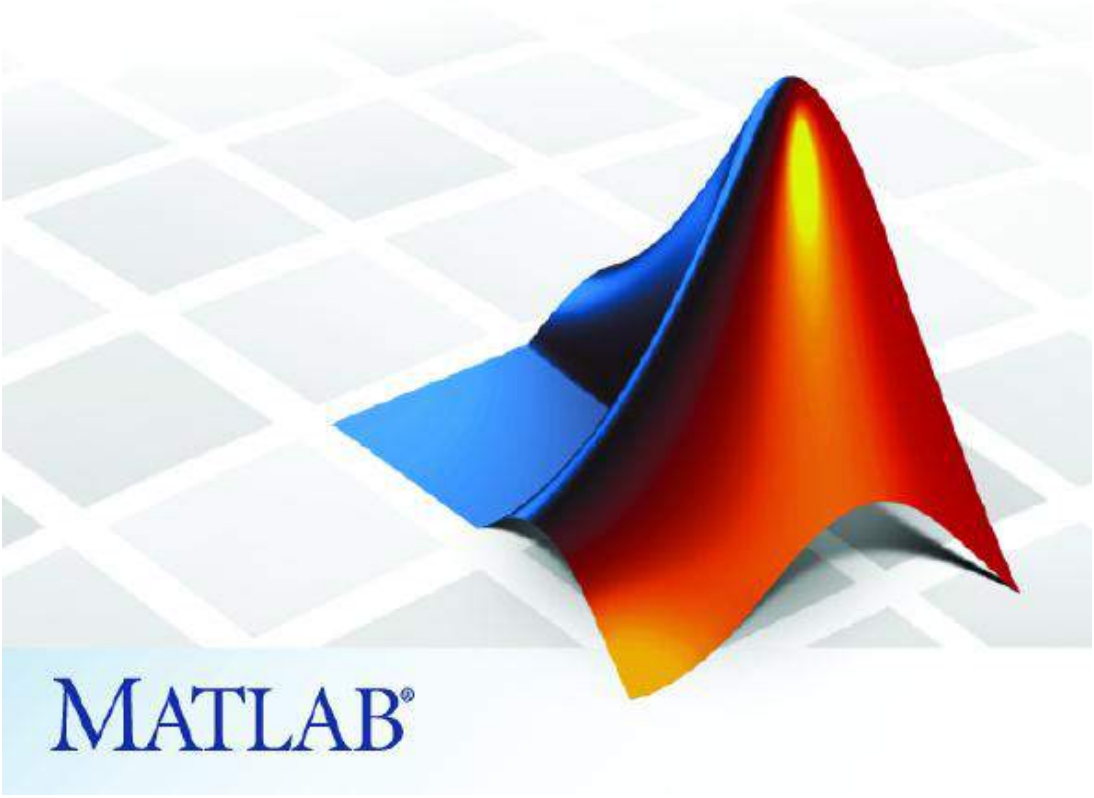
Noções Básicas de Programação em MATLAB



Alex Jenaro Becker

Daiane Medianeira Ilha da Silva

Francisco Helmuth Soares Dias

Lucélia Kowalski Pinheiro

Santa Maria, Outubro de 2010

APRESENTAÇÃO

Este trabalho trata-se da Apostila do Minicurso "**Noções Básicas de Programação em MATLAB"**, elaborado e ministrado pelos bolsistas do PET Matemática da UFSM: Daiane Medianeira Ilha da Silva e Francisco Helmuth Soares Dias, e pelos colaboradores Alex Jenaro Becker e Lucélia Kowalski Pinheiro.

A Apostila traz noções básicas do *software* matemático MATLAB :

- Ambiente de trabalho, comandos básicos, funções básicas, manipulação de matrizes;
- comandos para cálculo de limites, derivadas e integrais, cálculo de zeros de funções;
- comandos para plotagem de gráficos bidimensionais e tridimensionais,
- Noções básicas de programação em MATLAB.

Nossa intenção com a proposta do Minicurso, não é cobrir todos os tópicos do MATLAB, até por que isso seria praticamente impossível, pois nossos conhecimentos são restritos se comparados à amplitude e às abrangentes possibilidades de utilização do mesmo. Além do mais a carga horária seria insuficiente.

Pretende-se, com o Minicurso e a Apostila incentivar e motivar o estudo da ferramenta MATLAB, a partir das noções básicas que apresentaremos.

Boa aprendizagem!

SUMÁRIO

1. INTRODUÇÃO

O MATLAB (do inglês *Matrix Laboratory*) é um software de computação numérica de análise e visualização de dados. Embora seu nome signifique Laboratório de Matrizes, seus propósitos atualmente são bem mais amplos. Ele nasceu como um programa para operações matemáticas sobre matrizes, mas ao longo dos anos transformou-se em um sistema computacional bastante útil e flexível.

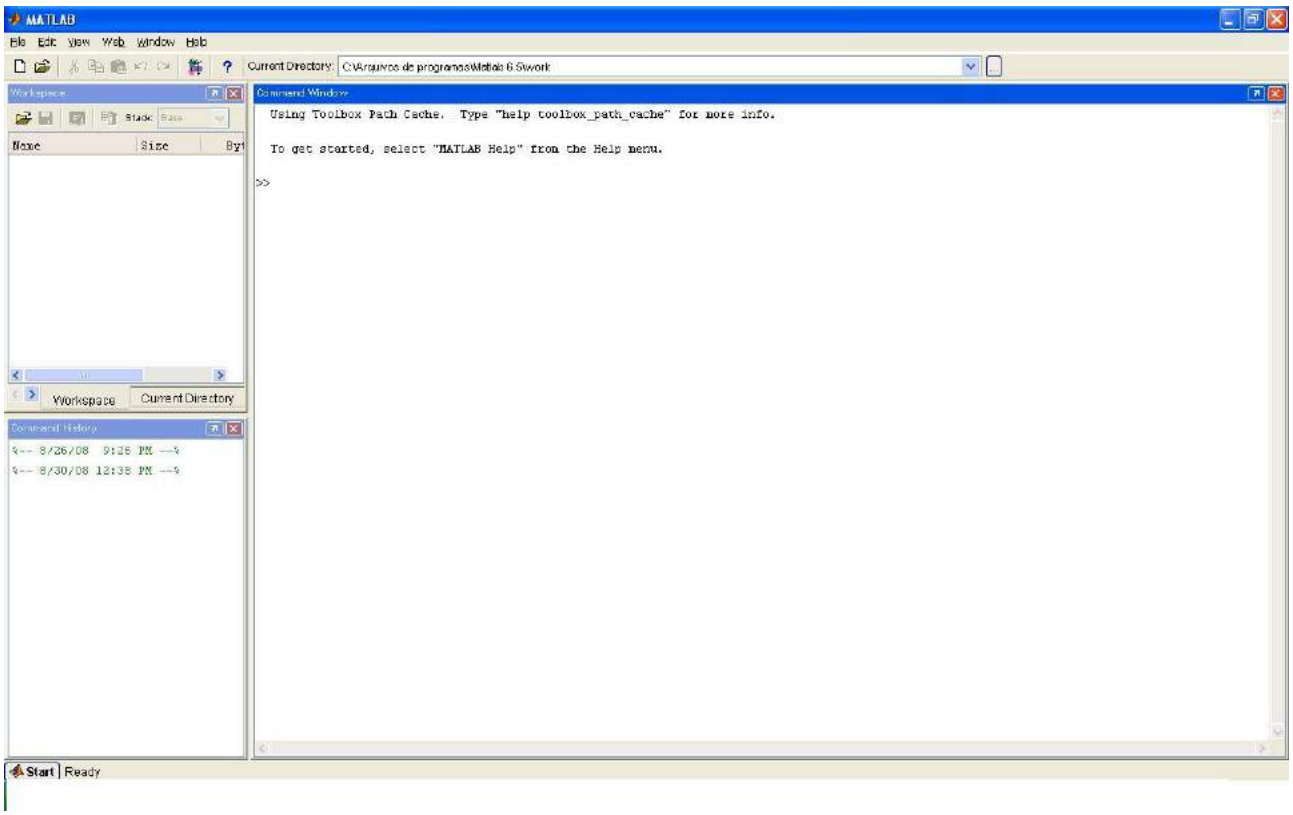
Seu ambiente de trabalho é fácil de ser utilizado, pois os problemas e soluções são escritos em linguagem matemática e não na linguagem de programação tradicional, como muitos outros softwares utilizam.

Assim o MATLAB é uma ferramenta e uma linguagem de programação de alto nível, e tem como principais funções: construção de gráficos e compilação de funções, manipulação de funções específicas de cálculo e variáveis simbólicas.

Além disso, o MATLAB possui uma grande quantidade de bibliotecas auxiliares (“Toolboxes”) que otimizam o tempo gasto para realizar tarefas, uma vez que, o usuário poderá utilizar muitas funções já definidas, poupando o tempo de criá-las. Por outro lado, infelizmente, os programas feitos são difíceis de serem executados num ambiente fora do MATLAB.

1.1 O ambiente MATLAB

O primeiro passo para iniciarmos nosso estudo do MATLAB é nos familiarizarmos com a interface do programa.

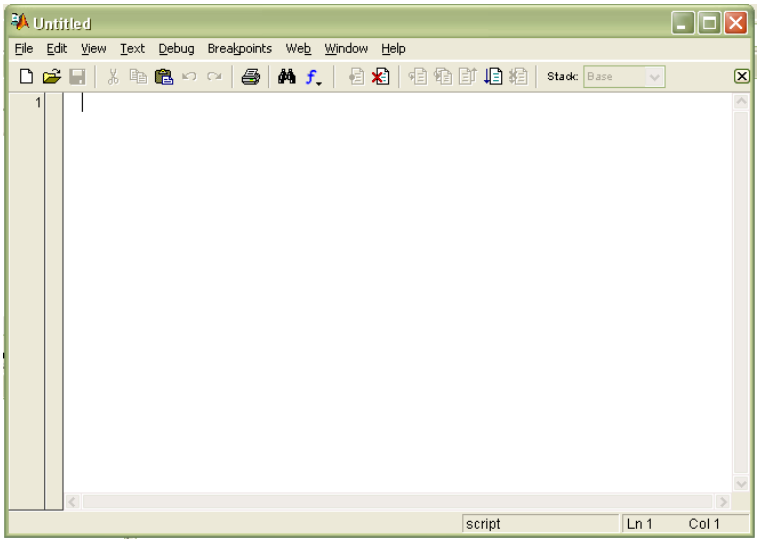


a) Command Window: Local onde as operações podem ser diretamente feitas.

b) Workspace: espaço destinado às variáveis que estão salvas na memória, onde é possível visualizar o nome, valor e classe da mesma.

c) Command History: Lista de comandos realizados, organizados por data de execução, permitindo o comando ser realizado novamente com duplo clique.

Podemos também utilizar ***M-files,*** na barra de Menus acessando a guia File>New>M-file, caso se deseje criar procedimentos de forma que estes fiquem salvos em arquivo. O MATLAB gera a seguinte janela:



Os arquivos salvos são gerados na extensão 'nomedoarquivo'.m. que são compilados utilizando-se a Command Window como espaço de comunicação de dados, de entrada e saída, entre o programa e o usuário.

Pode-se também chamar um M-file, escolhendo-se em *Current Directory* a pasta em que o mesmo está localizado. Depois de escolhido o diretório, digite na Command Window o nome do arquivo, e então a programação salva será compilada.

2. COMANDOS BÁSICOS

2.1 Operações Básicas

Operação	Símbolo	Exemplo
Adição	+	5 + 3
Subtração	-	23 - 12
Multiplicação	*	3,14 * 0,85

Divisão	/ ou \	56/8=8\56
Potenciação	^	5^2=25

As operações são realizadas da esquerda para a direita calculando-as conforme a ordem:

1º Potenciação;

2º Multiplicação e divisão;

3º Adição e subtração.

2.1.1 Comandos para utilização de Funções:

Comando	Descrição
abs(x)	Valor absoluto ou módulo de um número complexo
acos(x)	Arco cosseno
acosh(x)	Arco cosseno hiperbólico
angle(x)	Ângulo de um número complexo
asin(x)	Arco seno
asinh(x)	Arco seno hiperbólico
atan(x)	Arco tangent
atan2(x,y)	Arco tangente em quatro quadrantes
atanh(x)	Arco tangente hiperbólica
ceil(x)	Arredondar para inteiro na direção de mais infinito
conj(x)	Conjugado complex
cos(x)	Cosseno
cosh(x)	Cosseno hiperbólico

exp(x)	Exponencial
fix(x)	Arredonda para inteiro na direção de zero
floor(x)	Arredondar para inteiro na direção de menos infinito
imag(x)	Parte imaginária de um número complexo
log(x)	Logaritmo natural
log10(x)	Logaritmo na base 10
real(x)	Parte real de um número complexo
rem(x,y)	Resto da divisão de x por y
round(x)	Arredondar para o próximo número inteiro
sign(x)	Função sinal: retorna o sinal de um argumento.
sin(x)	Seno
sinh(x)	Seno hiperbólico
sqrt(x)	Raiz quadrada
tan(x)	Tangente
tanh(x)	Tangente hiperbólica

2.2 Comandos de ajuda

2.2.1 O comando *help*

Comando *help* é a maneira mais simples de se conseguir ajuda caso você saiba exatamente o tópico a respeito do qual você necessita de informações. Por exemplo:

```
>> help sqrt

 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also SQRTM.
```

O comando *help* funciona muito bem se você sabe exatamente o tópico sobre o qual necessita de ajuda. Já que isso não é sempre o caso, o *help* pode também levá-lo ao tópico exato que deseja digitando *help*, sem especificação do tópico.

2.2.2 O comando *lookfor*

Embora o comando *help* permita-lhe conseguir ajuda, ele pode não ser a maneira mais conveniente, a menos que você saiba o tópico exato sobre o qual necessita de informações.

Para isso, o comando *lookfor* fornece ajuda fazendo uma busca em toda primeira linha dos tópicos de ajuda e retornando aqueles que contém as palavras-chave que você especificou. O mais importante é que a palavra-chave não precisa ser um comando MATLAB. Por exemplo:

```
>> lookfor complex
ctranspose.m: %'   Complex conjugate transpose.
   COMPLEX Construct complex result from real and imaginary parts.
CONJ   Complex conjugate.
CPLXPAIR Sort numbers into complex conjugate pairs.
IMAG   Complex imaginary part.
REAL   Complex real part.
CDF2RDF Complex diagonal form to real block diagonal form.
RSF2CSF Real block diagonal form to complex diagonal form.
cplxdemo.m: %% Functions of Complex Variables
CPLXGRID Polar coordinate complex grid.
CPLXMAP Plot a function of a complex variable.
GRAFCPLX Demonstrates complex function plots in MATLAB.
ctranspose.m: % ' Complex conjugate transpose.
ctranspose.m: % ' Complex conjugate transpose.
SMOKE  Complex matrix with a "smoke ring" pseudospectrum.
```

3. MANIPULAÇÃO DE MATRIZES

O MATLAB trabalha essencialmente com um tipo de objeto: uma matriz numérica retangular podendo conter elementos complexos. Observa-se que um escalar é uma matriz de dimensão 1x1 e que um vetor é uma matriz que possui somente uma linha ou uma coluna.

O método mais fácil de introduzir pequenas matrizes no MATLAB é utilizando uma lista explícita. **Os elementos de cada linha da matriz são separados por espaços em branco ou vírgulas e as colunas separadas por ponto e vírgula, colocando-se colchetes em volta do grupo de elementos que formam a matriz com o objetivo de limitá-la.**

3.1 Matrizes elementares

Tipo de Matriz	Comando
Matriz Identidade	eye(n)
Matriz Nula	zeros(m,n)
Matriz com todos os elementos iguais a 1	ones(m,n)
Matriz Aleatória	rand(m,n)

3.2 Operações com matrizes

Operação	Comando
Transposta de uma matriz A	A't

Multiplicação por um escalar k	**K*A**
Multiplicação de duas matrizes A e B	**A.*B**
Quadrado e uma Matriz A	**A.^2**
Soma de duas matrizes A e B	**A+B**

Uma boa aplicação do MATLAB é suas funções matriciais. Dentre as mais usadas podemos citar:

COMANDO	DESCRIÇÃO
eig	Autovalores e Autovetores;
chol	Fatorização de Cholesky;
svd	Decomposição em fator singular;
inv	Inversa;
lu	Fatorização triangular LU;
qr	Fatorização ortogonal QR;
hess	Forma de Hessenberg;
schur	Decomposição de Schur;
expm	Matriz Exponencial;
sqrtm	Matriz de raiz quadrada;
poly	Polinômio característico;
det	Determinante;
size	Tamanho;
norm	Norma 1, Norma 2, Norma F, Norma Infinita;
cond	Número de condição na norma 2;
rank	Número de linhas linearmente independentes;
triu(A)	Gera uma matriz com os elementos acima da diagonal principal de A e zera os elementos que estão abaixo;
tril(A)	Gera uma matriz com os elementos abaixo da diagonal principal de A e zera os elementos que estão acima;
diag(A)	Fornece os elementos da diagonal;

diag(diag(a))	Gera uma matriz com os elementos da diagonal principal de A e com zeros nas outras posições;
flipud(A)	Coloca a matriz A de "cabeça para baixo";
fliplr(A)	Coloca a matriz da esquerda para a direita;
rot90(A)	Roda a matriz em sentido anti-horário;
reshape(A,m,n)	Retorna uma matriz m por n, cujos elementos são tomados coluna por coluna de A.

4. ZEROS DE FUNÇÕES

O MATLAB encontra zeros de funções usando o comando fzero. A função, da qual deseja-se encontrar os zeros, deve ser então escrita:

```
>> fzero('funçao', x0)
```

com x_0 chute inicial.

5. MÁXIMOS E MÍNIMOS DE FUNÇÕES

Para encontrar o mínimo de uma função usa-se o comando **fminbnd.**

Tomemos como exemplo a função $y = \text{sen}(x) - \cos(x)$:

```
>> [xmin ymin]=fminbnd('sin(x)-cos(x)',0,2*pi)

xmin =

    5.4978


ymin =

   -1.4142
```

6. CÁLCULO

6.1 Construindo funções com variáveis simbólicas

Em alguns casos precisamos utilizar uma variável simbólica, chamemos x, para definir como sendo qualquer variável do domínio, isto é, uma variável contínua. Para isso temos o comando syms. Vejamos:

```
>> syms x
```

Ou para mais de uma variável:

```
>> syms a b c
```

6.2 Limites

Para calcularmos $\lim_{x\to a} f(x)$, utilizamos o comando $limit(f(x), x, a)$.

6.2.1 Limite lateral à esquerda

Para calcular o limite lateral à esquerda utiliza-se o comando $limit(f(x), x, a, 'left')$.

6.2.2 Limite lateral à direita

Para calcular o limite lateral à direita utiliza-se o comando $limit(f(x), x, a, 'right')$.

6.3 Derivadas

Para calcularmos derivadas utiliza-se o comando $diff(f(x), x, n)$, onde n indica a ordem da derivação.

6.4 Integração

6.4.1 Integrais Indefinidas

Para calcular integrais indefinidas $\int f(x)dx$ utiliza-se o comando $int(f(x), x)$.

6.3.1 Integrais Definidas

Para calcular integrais definidas $\int_a^b f(x)dx$ utiliza-se o comando$int(f(x), x, a, b)$.

7. PLOTANDO GRÁFICOS NO MATLAB

Gráficos constituem um recurso visual poderoso para a interpretação de dados. O MATLAB dispõe de um grande número de facilidades gráficas, usadas para *plotar (gerar desenho de gráficos)* através de funções e comandos. É possível obter gráficos bidimensionais ou tridimensionais com qualquer tipo de escala e coordenada.

Alguns comandos freqüentes para *plotar* gráficos bidimensionais são:

Comando	Descrição
plot	*Plotar* linear
loglog	Gráfico em escala logarítmica
semilogx	Gráfico em escala semi-logarítmica (eixo x).
semilogy	Gráfico em escala semi-logarítmica (eixo y).
fill	Desenhar polígono 2D.
polar	Gráfico em coordenadas polar
bar	Gráfico de barras
stem	Gráfico de seqüência discreta

stairs	Gráfico em degrau
hist	Histograma.
rose	Histograma em ângulo
compass	Gráfico em forma de bússola.
feather	Gráfico em forma de pena.
fplot	Gráfico da função
comet	Gráfico com trajetória de cometa.

7.1 O comando plot

O comando *plot* é o comando mais comum para *plotagem* de dados bidimensionais.

Exemplo:

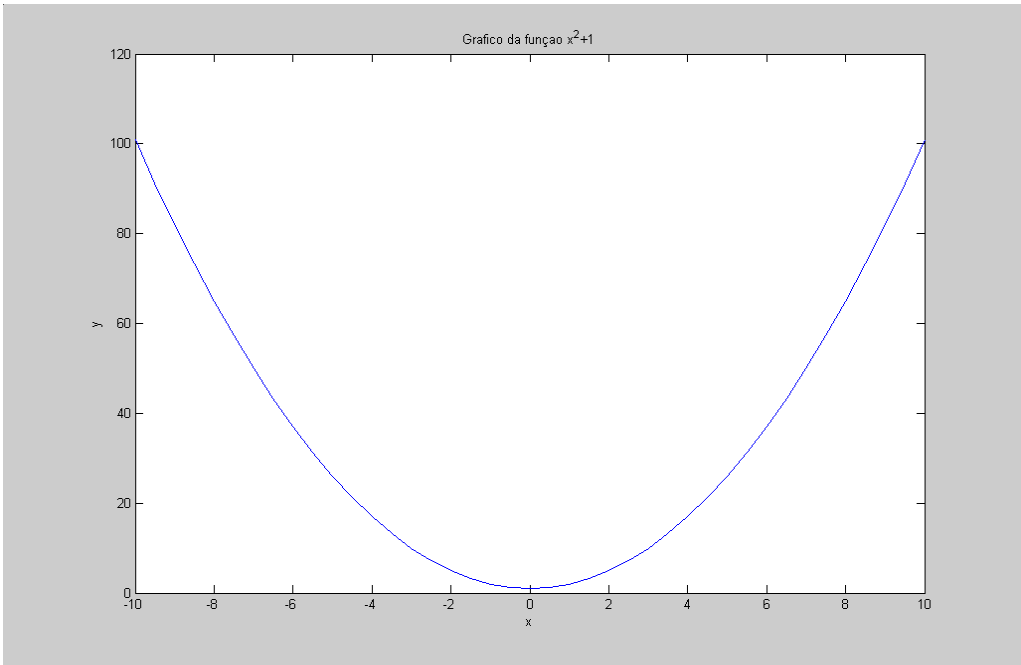
Plotar a função x^2+1

```
x= -10:0.5:10;
y=x.^2+1;
plot(x,y);
```

Podemos utilizar alguns comandos para melhorar a aparência de nosso gráfico: ***title*** (inclui um título ao gráfico), ***xlabel*** (permite que o eixo das abscissas do gráfico seja identificado), ***ylabel*** (permite que o eixo das abscissas do gráfico seja identificado).

```
title('Grafico da função x^2=1')
xlabel('x')
ylabel('x')
```

Como resultado o MATLAB nos retorna a uma janela denominada **Figure No. 1** com o seguinte gráfico:

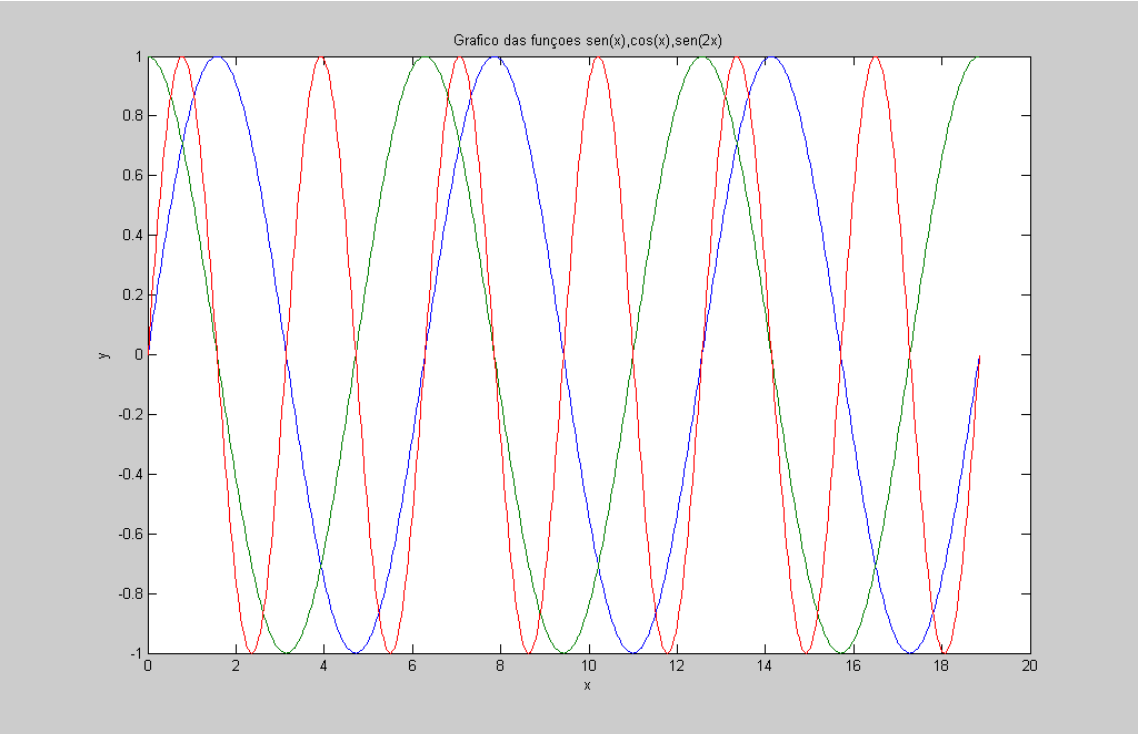


É possível desenhar mais que uma função no mesmo gráfico. Existem dois modos: um através do comando ***plot,*** e outro através do comando **hold**

Por exemplo, podemos gerar no mesmo gráfico as funções sen(x), cos(x) e sen(2x).

```
x= 0:PI/100:6*pi;
y1=sin(x);
y2=cos(x);
y3=sin(2*x);
plot(x,y1,x,y2,x,y3);
title('Grafico das funções sem(x),cos(x),sem(2x)')
xlabel('x')
ylabel('x')
```

Gerando o seguinte gráfico:



Com o comando ***hold,*** *p*ara *plotar* as funções sen(x) e cos(x), utiliza-se a seguinte síntese:

```
x= 0:PI/100:6*pi;
y1=sin(x);
y2=cos(x);
hold on
plot(x,y1);
plot(x,y2);
hold off
```

Além de títulos e designação dos eixos (funções title, xlabel e ylabel) podemos definir outras propriedades gráficas como **legendas, cores e estilos de linhas, estilos de marcadores, incluir grade**.

A **cor** e o **estilo da linha** e o **tipo de marcador** para pontos de dados na linha podem ser selecionado pelo uso de uma cadeia de caracteres de atributos após os vetores *x* e *y* na função *plot.*

Na tabela a seguir vemos os principais valores para os atributos cores, marcadores e estilos de linha.

Cor		**Marcadores**		**Estilo de Linha**	
y	amarelo	**.**	Ponto	**-**	Sólido
m	rosa(magenta)	**o**	Círculo	**:**	Pontilhado
c	azul (ciano)	**x**	X	**-.**	Ponto-traço
r	vermelho	**+**	Mais	**--**	Tracejado
g	verde	*****	Asterisco		
b	azul	**s**	Quadrado		
w	branco	**v**	Triângulo para baixo		
k	preto	**^**	Triângulo para cima		
		P	Pentágono		

Com o comando ***grid*** podemos adicionar linhas de grade no desenho do gráfico: *grid on* (para incluir) e *grid off* (para remover).

Legendas podem ser criadas por meio da função ***legend***, utilizando a seguinte estrutura:

legend('texto1', 'texto2',...,posição)

Onde na 'posição' podem ser atríbuidos os seguintes valores de posicionamento da legenda:

Valor	Significado
0	Escolha automática da melhor posição (mínimo conflito com os dados)
1	Canto superior direito
2	Canto superior esquerdo
3	Canto inferior esquerdo
4	Canto superior direito
-1	À direita do desenho

Além do título, é possível adicionar qualquer outro texto em algum lugar específico do gráfico *plotado* através do comando **text**, com a seguinte síntese:

```
text (x,y, 'texto desejado')
```

Onde x, y são as coordenadas nas quais desejamos que o texto apareça.

Com a mesma finalidade pode ser utilizado o comando **gtex**t, com a diferença de com este a posição do texto é escolhida através do mouse. Tem a síntese:

```
gtext( 'texto desejado')
```

O Comando *axis*

É possível controlar as proporções e a aparência dos eixos horizontal e vertical dos gráficos gerados pelo MATLAB através do comando *axis*.

Alguns modos principais de configuração desse comando seguem na tabela a seguir:

Comandos	***Descrição***
axis ([xmin xmax ymin ymax])	Define os valores máximos e mínimos dos eixos usando os valores dados no vetor de linha.
axis square	Torna quadrado o quadro dos eixos.
axis equal	Ajusta os incrementos de eixos para que sejam iguais nos dois eixos.
axis normal	Cancela o efeito dos dois comandos anteriores
axis off	Desliga todos os nomes de eixos, grades e marcadores. Não altera o título nem os nomes colocados pelos comandos *text* e *gtext*.
axis on	Liga nomes de eixos, marcadores e grade.

LineWidth, MarkerSize, MarkerEdgeColor, MarkerFaceColor

São propriedades do comando *plot,* através das quais podemos ajustar a cor, o estilo e o tipo de marcador para uma linha.

LineWidth: Especifica em pontos a espessura de cada linha.

MarkerSize: Especifica em pontos o tamanho do marcador.

MarkerEdgeColor: Especifica a cor do marcador ou da borda de marcadores preenchidos.

MarkerFaceColor: Especifica a cor interna dos marcadores preenchidos.

A estrutura de utilização das propriedades é a seguinte:

```
plot(x,y, 'nome da propriedade',valor,….)
```

7.1.1 Comando subplot

É possível colocar mais de um conjunto de eixos em uma mesma figura, criando assim múltiplos diagramas. Os subdiagramas são criados pelo comando *subplot*:

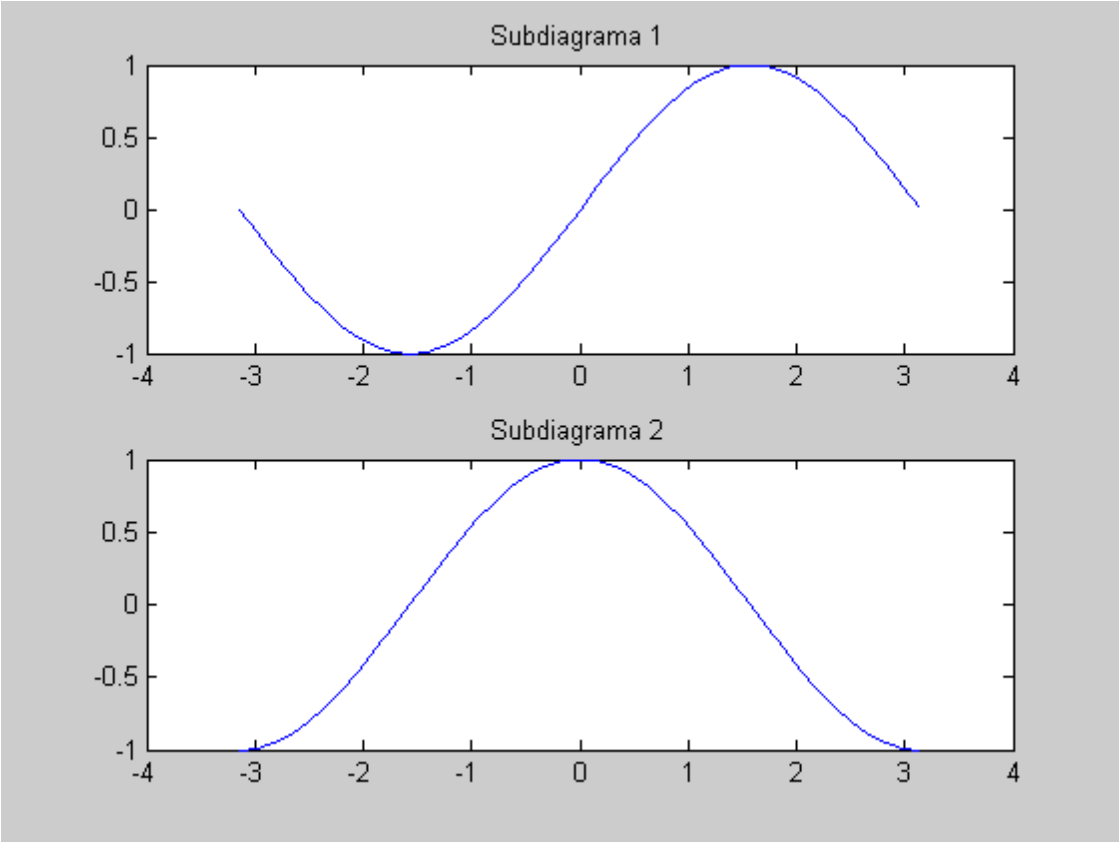
```
subplot(m,n,p)
```

Onde **m** denota o número linhas e **n** o número de colunas que se deseja dividir a janela gráfica; **p** indica qual das subdivisões vai receber o gráfico desejado.

Exemplo:

Plotar as funções sen(x) e cos(x), com x= -pi: pi/20 : pi, na mesma janela mas em gráficos separados, utilizando o comando *subplot* para dividir a janela em dois subgráficos.

```
subplot(2,1,1)
x=-pi:pi/20:pi;
y= sin(x);
plot(x,y)
title('Subdiagrama 1');
subplot(2,1,2);
x=-pi:pi/20:pi;
y= cos(x);
plot(x,y)
title('Subdiagrama 2');
```



7.2 Comando fplot

Além do comando *plot* podemos graficar uma função através do comando *fplot*. Basicamente, você deve fornecer como primeiro argumento a função que pretende usar entre apóstrofes e como segundo, o intervalo sobre o qual a função será graficada.

Exemplo:

```
fplot('sin(x)',[ -pi, pi])
fplot('x^2+3', [ -1, 2])
fplot('sin(x)',[ -pi, pi])
```

7.3 Coordenadas Polares

O MATLAB tem uma função chamada *polar* que se destina ao desenho de dados usando coordenadas polares. Sua forma básica é:

polar(theta,r)

Exemplo:

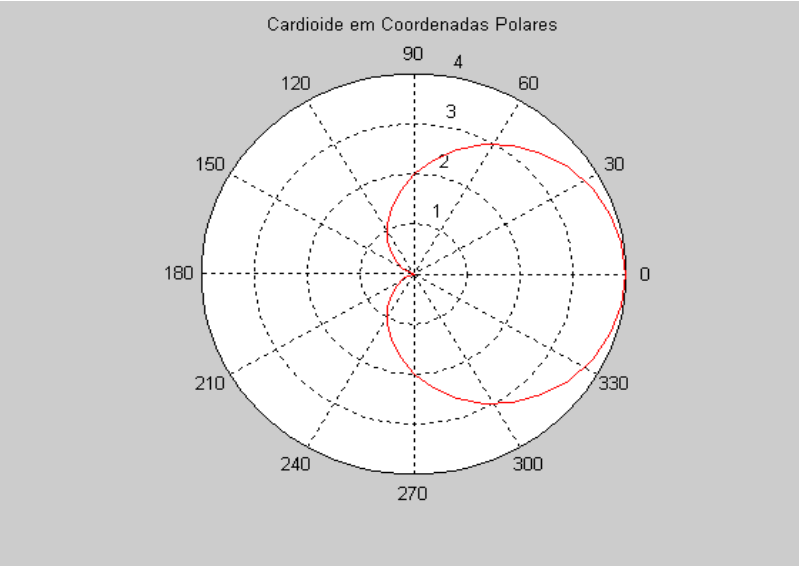
Cardióide

A Cardióide pode ser expressa através de coordenadas polares:

$$r = 2(1+\cos\theta)$$

Utilizando o comando *polar, plote* a Cardióide.

```
theta = 0:pi/50:2*pi;
r=2*(1+cos(theta));
polar(theta,r,'r-');
title('Cardioide em Coordenadas Polares');
```



7.4 Curvas Paramétricas

A forma paramétrica de uma curva plana pode ser descrita através do seguinte par de funções:

x = f(t)

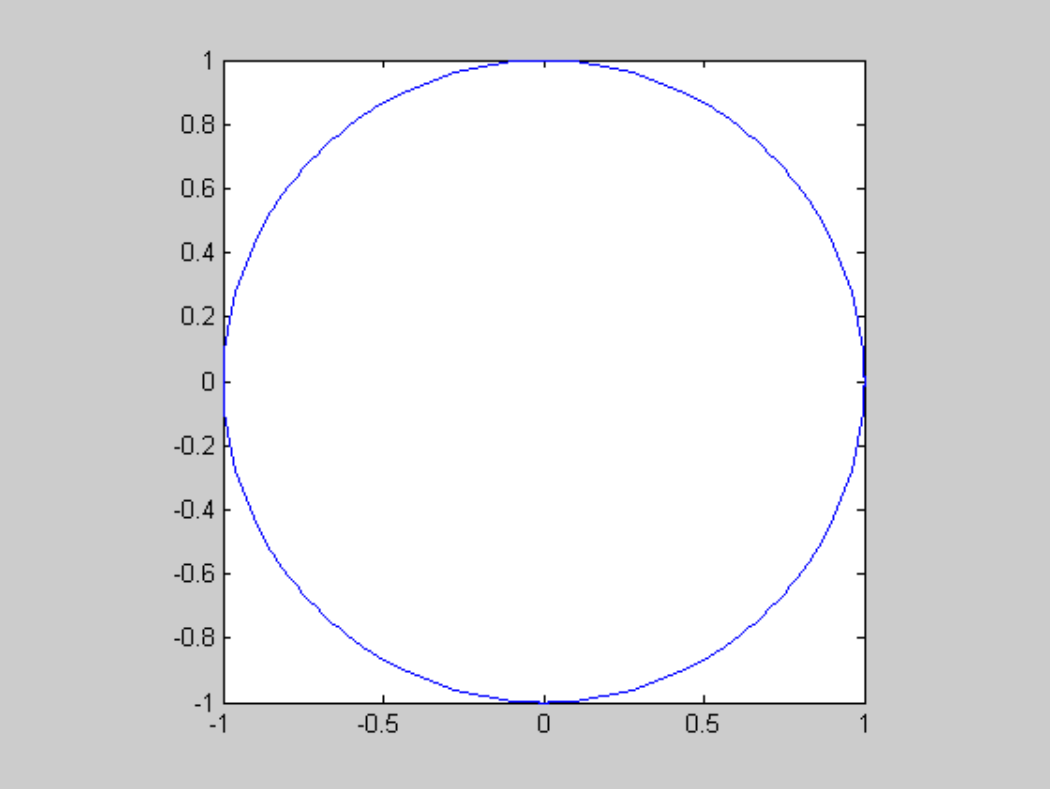
y =g (t)

Onde t é um parâmetro real que assume valores em um intervalo [a,b] e f(t) e g(t) são funções quaisquer. Quando t = a, o ponto (x_a,y_a) = (f(a),g(a)) é o início da curva e quando t= b, o ponto (x_b,y_b) = (f(b),g(b)) é o final da curva.

Exemplo:

Círculo

```
t = -6*pi:pi/100:6*pi;
x = cos(t);
y = sin(t);
plot(x,y);
axis square
```



7.5 Diagramas Bidimensionais

Além dos gráficos bidimensionais o MATLAB possui comandos que possibilitam outras formas de representação de dados tais como: gráficos de barras, gráficos do tipo pizza, gráficos de haste, diagramas de radar, entre outros.

Abaixo listamos alguns dos mais utilizados, sua síntese e descrição:

bar(x,y)	Cria um diagrama de barras verticais, sendo x o rótulo de cada barra e y a altura da barra.
barh(x,y)	Cria um diagrama de barras horizontais, sendo x o rótulo de cada barra e y o comprimento horizontal da barra.
compass(x,y)	Cria um diagrama polar, com uma seta saindo da origem em direção a cada ponto (x,y).

pie(x,y) pie(x,explode)	Cria um diagrama de pizza. Determina a porcentagem da pizza inteira que corresponde a cada valor de x, e representa pedaços de pizza desse tamanho. A matriz opcional *explode* controla se os pedaços individuais são separados ou não do restante da pizza.
stairs	Cria um diagrama de pares, com cada degrau da escada centralizado em um ponto (x,y)
hist	Cria um histograma de um conjunto de dados.

Gráficos de Barras

É um gráfico no qual cada ponto é representado por uma barra vertical ou horizontal.

Tem a seguinte síntese:

bar(x,y), para barras verticais

barh(x,y) para barras horizontais.

O vetor x representa o rótulo ou posição da barra, e o vetor y representa a altura ou comprimento da barra.

Exemplo:

Barras verticais

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

y = [5 4 6 7 2 5];

barh(x,y);

axis([0 10 0 7]);

title('Grafico de barras horizontais');
```

Da mesma forma podemos utilizar os comando *barh, compass e stairs* para gerar outras formas de diagramas.

Gráfico Pizza

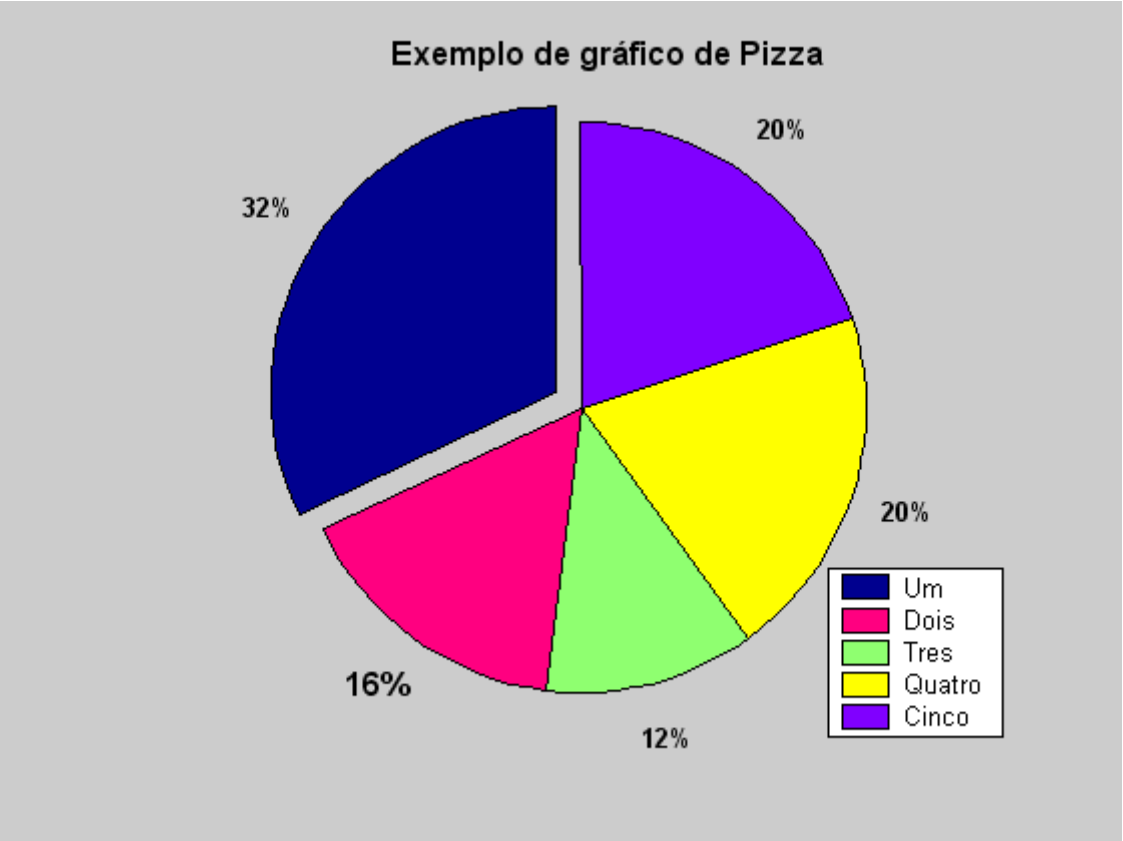
É um gráfico representado por “fatias de pizza” de tamanhos variados. Cria-se um vetor x com os dados a serem representados. Por exemplo, x=[1 2 3 4], então x(1) é 10% da pizza, x(2) é 20%, e assim por diante.

Podemos ainda acrescentar o vetor explode, que é um vetor lógico que recebe 1 ou 0, e cada elemento é associado a um elemento do vetor x. Sendo que 1 significa que a fatia da pizza correspondente é desenhada um pouco separada da pizza, e 0 a fatia permanece em sua posição normal.

Exemplo:

Construir um gráfico do tipo Pizza, onde as fatias representam 32%, 20%, 20%, 12%, 16%. Separe a fatia maior das demais fatias. Utilize também o comando *legend*.

```
x = [32 16 12 20 20];
explode= [1 0 0 0 0];
pie(x,explode);
title('Exemplo de grafico de Pizza');
legend('Um','Dois','Tres','Quatro','Cinco',4);
```



Histogramas

Um histograma é um diagrama que mostra a distribuição de valores em um conjunto de dados. Para criar um histograma, a faixa de valores em um conjunto de dados é dividida em grupos regularmente espaçados, e o número de valores de dados que caem em cada grupo é determinado. A contagem resultante pode ser representada em um diagrama como função do número do grupo.

Algumas das sínteses para gerar histogramas são:

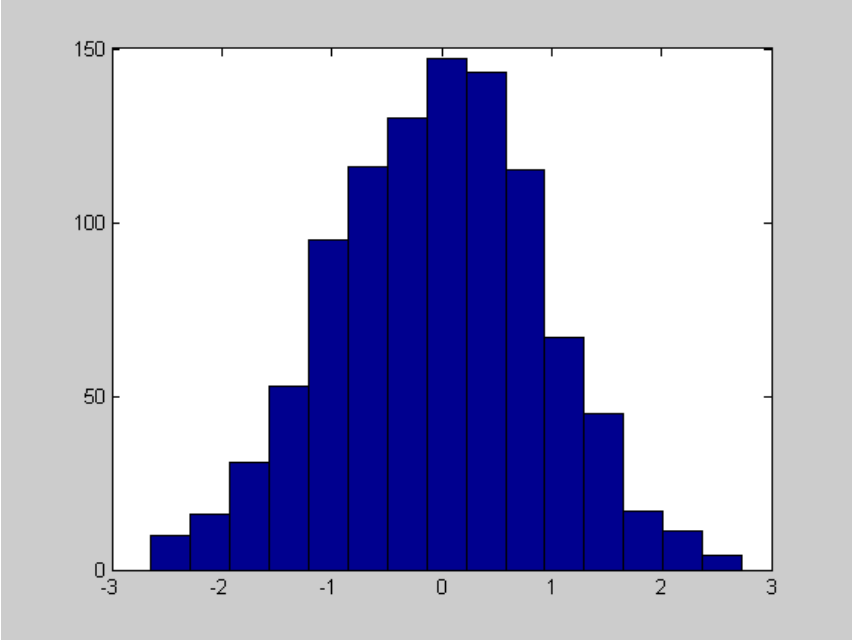
hist(y), cria um histograma com 10 grupos igualmente espaçados.

hist(y,n), cria um histograma com n grupos igualmente espaçados.

Exemplo:

Esse exemplo cria um conjunto de dados com 10.000 valores aleatórios gaussianos e gera um histograma dos dados usando 15 grupos igualmente espaçados.

```
y = randn(1000,1);
hist (y,15);
```



8. ALGORITMOS

O significado da palavra ALGORITMO é muito similar ao de uma receita, procedimento, técnica, rotina. **Um algoritmo é um conjunto finito de regras que fornece uma seqüência de operações para resolver um problema específico**. Segundo o dicionário do prof. Aurélio um algoritmo é um: "Processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, regras formais para a obtenção de resultado ou de solução de problema." Um algoritmo tem cinco características importantes:

Finitude:

Um algoritmo deve sempre terminar após um número finito de passos.

Definição:

Cada passo de um algoritmo deve ser precisamente definido. As ações devem ser definidas rigorosamente e sem ambiguidades.

Entradas:

Um algoritmo deve ter zero ou mais entradas, isto é quantidades que são lhe são fornecidas antes do algoritmo iniciar.

Saídas:

Um algoritmo deve ter uma ou mais saídas, isto é quantidades que tem uma relação específica com as entradas.

Efetividade:

Um algoritmo deve ser efetivo. Isto significa que todas as operações devem ser suficientemente básicas de modo que possam ser em princípio executadas com precisão em um tempo finito por um humano usando papel e lápis.

Para mostrar um exemplo de algoritmo considere o seguinte problema:

Dispomos de duas vasilhas com capacidades de 9 e 4 litros respectivamente. As vasilhas não tem nenhum tipo de marcação, de modo que não é possível ter medidas como metade ou um terço. Mostre uma seqüência de passos, que usando as vasilhas de 9 e 4 litros encha uma terceira vasilha de medida desconhecida com seis litros de água.

Uma possível solução é:

1. Encha a vasilha de 9 litros;
2. Usando a vasilha de 9 litros, encha a vasilha de 4 litros;
3. Despeje o que sobrou na vasilha de 9 litros (5 litros) na terceira vasilha. Observe que falta um litro para completar os seis litros;
4. Esvazie a vasilha de 4 litros;
5. Torne a encher a vasilha de 9 litros;
6. Usando a vasilha de 9 litros encha a vasilha de 4 litros;
7. Esvazie a de 4 litros;
8. Usando o que restou na vasilha de 9 litros (5 litros), encha novamente a vasilha de quatro litros;
9. Despeje o que sobrou na vasilha de 9 litros (1 litro) na terceira vasilha, que agora tem 6 litros.

8.1 Representação de Algoritmos

As formas mais comuns de representação de algoritmos são as seguintes:

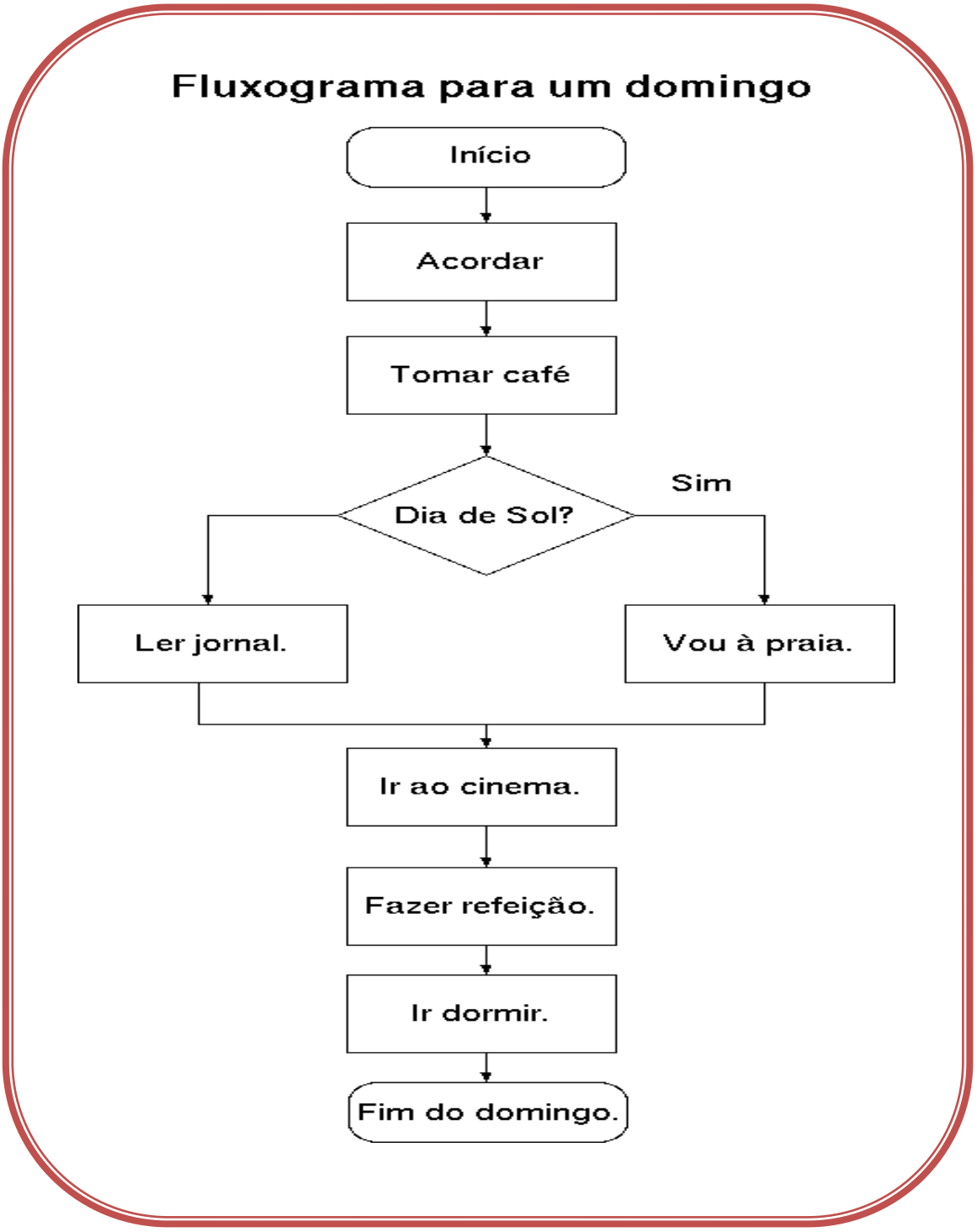
8.1.1 Linguagem Natural

Os algoritmos são expressos diretamente em linguagem natural, como no exemplo anterior. Para considerar um algoritmo que inclua decisões vamos estudar um algoritmo que nos ajude a decidir o que fazer em um domingo. Um possível algoritmo poderia ser o seguinte:

- Algoritmo de domingo.
  - Acordar.
  - Tomar o café.
  - Se estiver sol vou à praia senão leio o jornal.
  - Almoçar.
  - Ir ao cinema.
  - Fazer uma refeição.
  - Ir dormir.
- Final do domingo.

8.1.2 Fluxograma Convencional

Esta é um representação gráfica que emprega formas geométricas padronizadas para indicar as diversas ações e decisões que devem ser executadas para resolver o problema.


Fluxograma para um domingo
Início
Acordar
Tomar café
Dia de Sol?
Sim
Ler jornal.
Vou à praia.
Ir ao cinema.
Fazer refeição.
Ir dormir.
Fim do domingo.

8.1.3 Pseudo-Código

Criar programas de computador é, em geral, um trabalho difícil e cansativo, pois envolve muito a arte de pensar logicamente. Por haver muitas linguagens distintas de programação (tais como C, Pascal, Fortran, dentre outras), há uma linguagem "Universal", adotada por muitos programadores para que seus códigos sejam entendidos por programadores que conheçam qualquer linguagem.. Os códigos assim escritos são chamados de pseudo-códigos. O pseudo-código é escrito "à mão livre", ou seja, em linguagem comum, como no exemplo a seguir:

Fazer um algoritmo que pergunte que horas são e escreva uma frase de cumprimento segundo a hora inserida pelo usuário.

```
Algoritmo: Cumprimento
Variáveis: hora: Numérico

Início algoritmo

Escreva "Que horas são?"
Ler hora

Se 6<=hora<12
Escreva "Bom Dia"
Fim Se

Se 12<=hora<18
Escreva "Boa Tarde"
Fim Se

Se 18<=hora<6
Escreva "Boa Noite".
Fim Se

Fim algoritmo
```

9. M-Files: criando seus próprios programas e funções

Uma maneira simples de se fazer um programa em MATLAB é criar um arquivo texto com a lista de comandos desejados. Um programa escrito assim é chamado **script** e toda vez que for chamado efetua a lista de comandos como se eles fossem entrados sequencialmente via teclado. Por exemplo, para se calcular a distância entre dois pontos **p** e **q** em $\mathbf{R^3}$ podemos gerar um **script** com os comandos apropriados e chamá-lo sempre que o cálculo tenha que ser efetuado.

```
temp = (p(1)-q(1))^2+(p(2)-q(2))^2+(p(3)-q(3))^2
distancia = sqrt(temp)
```

Esse arquivo pode ser criado com qualquer editor de texto e deve ter extensão .m (M-Files). Por exemplo, o **script** acima poderia estar no arquivo **distancia_p_q.m**. Para chamá-lo basta entrar com o nome do arquivo na janela de comandos (*Command Window*):

```
>> p=[1 0 0];
>> q=[0 1 0];
>> distancia_p_q
temp =
2
distancia =
1.44142
```

Os arquivos **script** são úteis quando se deseja efetuar uma sequência de comandos com muita freqüência. Como no exemplo anterior, os **scripts** se utilizam dos dados presentes na memória (*workspace)* para efetuar os comandos.

Uma alternativa aos arquivos **script** são os arquivos tipo **function**, que admitem parâmetros de entrada, retornam valores e possuem variáveis locais (não afetam o *workspace*). Essas características fazem com que programas escritos nesse formato atuem exatamente como os comandos nativos do MATLAB. São, portanto, uma forma de ampliar a linguagem e é um dos pontos chaves do MATLAB.

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.

EXEMPLO:

```
function distancia=calcula_distancia(x,y)
% Calcula a distancia euclidiana entre os pontos x e y (vetores do Rn)
diferença=x-y;
distancia=sqrt (diferença*transpose (diferenca));
```

Exemplo 1:

```
>> p=[1 0 0];
>> q=[0 1 0];
>> d=calcula_distancia(p,q)
```

Ou:

```
>> d=calcula_distancia([1 0 0], [0 1 0])
d=
        1.4142
```

Exemplo 2:

```
distancia=calcula_distancia([3 4], [0 0])
distancia=
        5
```

Salienta-se que essa função determina a distância entre dois vetores quaisquer, independentemente do número de componentes dos mesmos (desde que ambos vetores sejam de mesma dimensão).

O arquivo tipo **function** também tem que ter extensão **.m** e deve ter o mesmo nome usado na definição da função, no exemplo anterior o nome do arquivo deve ser **calcula_distancia.m**. Usamos o *Command Window* para executar essa função.

A seguir, tem-se uma lista com alguns comandos de programação para o controle de fluxo, bem como comandos de programação geral e interfaces com o usuário.

Break	Interrompe a execução de laços *FOR* e *WHILE* equivalente ao BREAK do C
clc, home	Limpa a tela (janela de comandos)
Clear	Limpa as variáveis da memória do MATLAB
continue	Interrompe o fluxo do programa e recomeça um loop FOR ou WHILE, já na iteração seguinte. Só faz sentido dentro de um FOR ou WHILE
Display	Exibe o nome do conteúdo de uma variável
Disp	Exibe o conteúdo de uma variável, sem mostrar o seu nome
IF	Condiciona execução de comandos

Else	Usado com o comando *IF*
Elseif	Usado com o comando *IF*
End	Usado para terminar a execução dos comandos *IF, FOR, WHILE*
Error	mostra mensagem e aborta a execução da função
errordlg	Criar uma caixa de diálogo sem erro
Eval	Excuta uma expressão MATLAB definida através de uma *string*
Feval	Executa uma função especificada por uma *string*
For	Repete comandos por um número de vezes especificado
Fprintf	Grava dados em arquivo formatado
Fscanf	Lê dados de arquivo formatado
function	Define **m-file** tipo *FUNCTION* (função)
Global	Define variáveis globais
Helpdlg	Mostra uma caixa de diálogo de ajuda (HELP)
Input	Permite requisitar (com *prompt*) fornecimento de dados pelo teclado
keyboard	Suspende a execução de uma rotina e permite que o usuário entre e execute novos comandos pelo teclado; a rotina é continuada após o usuário digitar *return*
Lasterr	Última mensagem de erro emitida pelo MATLAB
Menu	Gera um menu de escolhas para entrada do usuário
margchk	Verifica número de argumento da entrada
Pause	Pausa na execução de um programa até ser pressionada uma tecla
pause(n)	Pausa na execução de um programa de n segundos
questdlg	Cria uma caixa de diálogo de perguntas
Rbbox	Caixa para seleção de região em gráfico
Return	Causa a saída imediata de uma função
Sprintf	Grava dados formatados em uma única *string*

switch e case	É um teste para múltiplos casos
uigetfile	Caixa de diálogo para obter o nome de um arquivo existente
uiputfile	Caixa de diálogo para especificar o nome de um novo arquivo
warndlg	Cria uma caixa de diálogo para advertências
While	Repete comandos enquanto condição especificada for verdadeira

9.1 Saída de dados - Comando *format*

O modo como os números aparecerão por padrão na tela (salvo exceções comandadas pelo usuário) é imposto pelo comando *format*. A tabele abaixo indica alguns dos formatos suportados:

COMANDO	RESULTADO	EXEMPLO: $e^{2*}\pi$
Format short	4 dígitos depois de vírgula	23.2134
Format long	14 dígitos depois da vírgula	23.21340435736339
Format short e	5 dígitos com exponencial	2.321e+001
Format long e	15 dígitos com exponencial	2.321340435736330e+001
Format short g	5 dígitos totais	23.213
Format long g	15 dígitos totais	23.2134043573634
Format hex	Formato hexadecimal	403736ª1aaffb31c

Format bank	Valores bancários	23.21
Format rat	Aproximação por racionais	3807/164
Format +	Exibe comente o sinal	+

9.2 Saída de dados - Função *disp*

O comando *disp* mostra o valor de variáveis, assim como se pode usá-lo para combinar frases com variáveis alfanuméricas. Para que isso aconteça, é interessante combinar o comando *disp* com o comando *num2str* ou *int2str* que convertem valores numéricos em valores alfanuméricos. O comando *num2str* converte qualquer número (ou matriz) em uma cadeia de caracteres, mantendo o formato dos elementos, já o comando *int2str* converte primeiro os valores em inteiros, para só então transformá-los em caracteres.

EXEMPLO: Mostrar o valor de e^{13}.

```
frase  =  [‘O valor de e^13 é:’num2str(exp(13))];

Disp(frase)
```

9.3 Saída de dados - Saída *fprintf*

O comando *fprintf* é um dos métodos mais simples de saída de dados. Com ele é possível combinar frases com variáveis numéricas de dimensão 1, ou seja, um escalar ou um elemento da matriz. Vejamos como ele funciona com o seguinte exmplo:

EXEMPLO: Atribuir dois valores às variáveis x e y e mostrá-las junto com uma frase qualquer:

```
x=1;

y=7;

fprintf('O x vale %d, enquanto y vale %d',x,y)
```

Acima, o que está entre aspas aparecerá para o usuário, os itens onde aparece %d serão substituídos pelas variáveis, respeitando-se a ordem em que aparecem. Ainda, %d significa que só aparecerá a parte inteira de x e y. Existem, ainda, os seguintes comandos:

%d	Exibe o valor como inteiro
%e	Exibe o valor no formato exponencial
%f	Exibe o valor em ponto flutuante
%g	Escolhe o mais curto entre ponto flutuante e exponencial

Ainda, se colocarmos *n* dentro das aspas temos uma mudança de linha.

EXEMPLO: Escrever uma saudação ao usuário:

```
fprintf(' Olá.\n Obrigado por freqüentar o minicurso do MATLAB.\n Esperamos que seja bastante útil.')
```

9.4 Saída de dados - Interação com o usuário através do comando *input*

Falamos bastante até aqui em variáveis e saídas de dados, porém é necessário que dados possam ser inserido por usuários (que não seja o programador) e que possamos trocar nossas variáveis sem alterar o código do programa. Por exemplo, para calcular as raízes de um polinômio é necessário que o programa conheça o polinômio. Mas, para cada polinômio diferente teremos que mudar o código-fonte do programa?

Esse problema é resolvido pelo comando *input*. A cada vez que o programa for rodado ele mesmo pedirá as variáveis, logo não será necessário mudar o código original do programa e haverá uma maior interação entre o programa e o usuário (não necessariamente um programador). Ainda dentro do comando *input* podemos colocar uma frase identificando que variável deve ser inserida. Vejamos como funciona através do seguinte exemplo:

EXEMPLO: Pedir que o usuário entre com os coeficientes de um polinômio do segundo grau ax^2+bx+c:

```
disp('Este programa lerá valores para os coeficientes a, b e c de um polinômio do segundo grau. ')

a=input('Insira o valor de a: ');

b=input('Insira o valor de b: ');

c=input('Insira o valor de c: ');

fprintf('O polinômio definido pelo usuário é %dx^2 + %dx + %d',a,b,c)
```

EXEMPLO: Fazer com que o usuário calcule as raízes de um polinômio de segundo grau:

```
disp('Este programa irá calcular as raízes de um polinômio de segundo grau. ')
a=input('Entre com o primeiro coeficiente :');
b=input('Entre com o segundo coeficiente :');
c=input('Entre com o terceiro coeficiente :');
delta = b^2 - 4*a*c;
x1=(-b + sqrt(delta))/(2*a);
x2=(-b - sqrt(delta))/(2*a);
fprintf('As raízes são: %d e %d',x1,x2)
```

9.5 Saída de dados - Operadores lógicos

Às vezes é necessário colocar mais de um comando ao mesmo parâmetro (principalmente no laço *if* que será visto posteriormente), para isso existem os comando lógicos dados na seguinte tabela:

&	E lógico
/	OU lógico
xor	OU exclusivo lógico
~	NÃO lógico

10. Condicionais e laços

Uma estrutura condicional permite a escolha do grupo de comandos a serem executados quando uma dada condição for satisfeita ou não, possibilitando desta forma alterar o fluxo natural de comandos. Esta condição é representada por uma expressão lógica.

A estrutura de repetição (ou laço) faz com que uma sequência de comandos seja executada repetidamente até que uma dada condição de interrupção seja satisfeita.

10.1 A estrutura condicional *if-end*

A estrutura condicional mais simples do MATLAB é:

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

Se o resultado da expressão lógica <condição> for verdadeiro então a lista de <comandos> será executada. Se o resultado for falso, os <comandos > não serão executados.

EXEMPLO: Fazer um programa que peça um número ao usuário e retorne o valor da raiz quadrada desse número.

```
a=input('Digite um valor para o qual se queira saber a raiz quadrada:');
if (a>=0)
   Raizquadrada=sqrt(a)
end
```

10.2 A estrutura condicional *if-else-end*

No caso de haver duas alternativas, uma outra estrutura condicional deve ser usada:

```
if <condição>
  <comandos 1>
else
  <comandos 2>
end
```

Se a expressão lógica <condição > for verdadeira então a lista <comandos 1> será executada. Se <condição> for falsa então será executada a lista <comandos 2>.

EXEMPLO: Faça um programa que leia o ano de nascimento de uma pessoa e o ano atual e diga a idade da pessoa. O programa deve verificar se o ano de nascimento é válido ou não.

```
an=input('Digite o ano de nascimento da pessoa:');
aa=input('Digite o ano atual:');
if ((an<0)|(an>2010))
   fprintf('Ano de nascimento invalido.');
else I=aa-an;
   fprintf('A  idade da pessoa eh %d anos.',I);
end
```

10.3 A estrutura condicional *if-elseif-else-end*

Quando houver mais de duas alternativas, a estrutura vista anteriormente **if-else-end** torna-se:

```
if <condição 1>
  <comandos 1>
elseif<condição 2>
  <comandos 2>
elseif <condição 3>
  <comandos 3>
    .
    .
    .

else
  <comandos 0>
end
```

A lista <comandos 1> será executada se a <condição 1> for verdadeira, já a lista <comandos 2> será executada se a <condição 2> for verdadeira e assim para as outras condições. Se nenhuma das condições for verdadeira então <comandos 0> será executada.

Quando a primeira <condição> for satisfeita e os <comandos> executados, a estrutura if-elseif-else-end será abandonada, ou seja, o controle do processamento será transferido para o comando imediatamente após o end.

EXEMPLO: Faça um programa tal que o usuário insira um valor para a variável x e em seguida mostre se x é maior, menor ou igual a zero.

```
x=input('Digite um valor para x:');
if x<0
   fprintf('x<0');
elseif x==0
   fprintf('x=0');
else
   fprintf('x>0');
end
```

EXEMPLO: Faça um programa que permita calcular os valores para a função:

$$f(x) = \begin{cases} 1, se\ x < -1 \\ x^2, se\ -1 \leq x \leq 1 \\ -x + 2, se\ x > 1 \end{cases}$$

```
function a=program1(x)
x=input('Digite um valor x para o qual se deseja avaliar a funçao:');
if x<-1
   a=1;
elseif (x>=-1)&(x<=1)
   a=x^2;
else
   a=-x+2;
end
```

EXEMPLO: Faça um programa que verifique se um número dado pelo usuário é positivo, se sim determinar se o mesmo é par ou ímpar:

```
n=input('Insira um numero qualquer:');
if n<0
   fprintf('O numero dado eh negativo.');
elseif rem(n,2)==0
   fprintf('O numero dado eh positivo e par.');
else
   fprintf('O numero dado eh positivo e impar.');
end
```

10.4 O laço *for*

A estrutura *for* permite que um grupo de comandos seja repetido um número específico de vezes definido pelo programador. Através do *for*, podemos criar uma variação de elementos, o que nos dá a vantagem de economizar tempo. Sua sintaxe é:

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

Acima, *variável* é a variável-de-controle que recebe o valor de cada coluna do vetor <*arranjo*> e, para cada conteúdo que receba, executa o corpo do *for*. Assim, o número de repetições da lista <*comandos* > é igual ao número de elementos no vetor <*arranjo*>. A variável-de-controle não pode ser redefinida dentro da estrutura *for*.

O laço *for* é o controlador de fluxo mais simples e usado na programação MATLAB. Analisando o seguinte exemplo:

```
for i=1:5
    X(i)=i^2;
end
disp(X)
```

pode-se notar que o laço *for* é dividido em três partes:

- A primeira parte (**i=1**) é realizada uma vez, antes de o laço ser inicializado.
- A segunda parte é o teste ou condição que controla o laço, (**i<=5**). Esta condição é avaliada; se verdadeira, o corpo do laço (**X(i)=i^2**) é executado.
- A terceira parte acontece quando a condição se torna falsa e o laço termina.

O comando *end* é usado como limite inferior do corpo do laço.

EXEMPLO: Faça um programa que crie uma matriz A_{1x15} pedindo que o usuário insira cada um dos 15 valores.

```
A=zeros(1,15);
for i=1:15
   fprintf('Insira um elemento (1,%d):',i);
   A(1,i)=input('');
end
disp(A)
```

OBSERVAÇÃO: Veja que aqui ocupa muito espaço se usarmos o comando *input* 15 vezes.

EXEMPLO: Faça um programa que pergunte ao usuário quantas linhas e quantas colunas ele quer que uma matriz A tenha. Em seguida, pedir que ele insira cada valor da matriz. Além disto, verificar quantos elementos menores que zero tem nesta matriz.

```
l=input('Digite um valor para o numero de linhas da matriz A:');
c=input('Digite um valor para o numero de colunas da matriz A:');
A=zeros(l,c);
b=0;
for i=1:l
   for j=1:c
        fprintf('Insira o numero (%d,%d):',i,j);
        A(i,j)=input('');
        if A(i,j)<0
           b=b+1;
        end
   end
end
disp(A);
fprintf('A matriz A possui %d numeros(s) menor(es) que zero.',b);
```

EXEMPLO: Faça um programa que diga todas as possibilidades de que, no lançamento de dois dados, a soma dos valores de cada dado seja igual a 7.

```
for d1=0:6
   for d2=0:6
      if (d1+d2==7)
         fprintf('\n O valor do 1º dado pode ser %d e o valor do 2º dado pode ser %d pois,
%d+%d eh 7',d1,d2,d1,d2);
      end
   end
end
```

EXEMPLO: Faça um programa que plote uma função retangular f(n) tal que 0<n<2π e f(n)=1, se 0<n<π e f(n)=-1, se π<n<2π.

```
x=linspace(0,2*pi,100); % Criou-se 100 amostras entre 0 e 2*pi
for n=1:100
   if x(n)<=pi
      f(n)=1;  %Faz f(t)=1 para 0<t<=pi,i.e., as primeiras 50 amostras de f(t) são iguais a
1
   else
      f(n)= -1; % Faz f(t)=-1 para pi<t<=2*pi, i.e., as últimas 50 amostras de f(t) são
iguais a 1
   end
end
plot(x,f,'r.'); % plota o grafico, r=cor:vermelha e .:estilo do ponto: ponto
grid on; % adiciona linhas de grade no desenho do grafico
title('Função retangular'); % Coloca um titulo ao gráfico
xlabel('t em radianos'); % nomeia o eixo x
ylabel('f(t)'); % nomeia o eixo y
```

EXEMPLO: Faça um programa que plote 360 pontos de um período da função $y=\text{sen}(\frac{2\pi x}{360})$.

```
for x=1:360
   y(x)=sin(2*pi*x/360);
end
plot(y)
xlabel('x em graus');
ylabel('f(x)');
```

EXEMPLO: Faça um programa que construa duas matrizes A e B dadas pela lei $A_{ij}=i+j$ e $B_{ij}=i-j$ e calcule a soma das duas.

```
for i=1:8
   for j=1:8
      A(i,j)=i+j;
      B(i,j)=i-j;
   end
end
C=A+B;
A, B, C
```

10.5 O laço *while*

O laço *while*, ao contrário do *for*, repete um grupo de comandos um número indefinido de vezes, até obtermos uma resposta satisfatória ou até que o usuário mande interromper o programa. Sua sintaxe é:

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

Enquanto a expressão lógica <condição> for verdadeira a lista <comandos> será repetida.

No laço *while* apenas a condição é testada. Por exemplo, na expressão:

```
a = 1; b = 15;
while a<b,
   clc
   a = a+1
   b = b-1
   pause(2)
end
disp('fim do loop')
```

A condição **a<b** é testada. Se ela for verdadeira o corpo do laço, será executado. Então a condição é testada novamente, e se verdadeira o corpo será executado novamente. Quando o teste se tornar falso o laço terminará, e a execução continuará no comando que segue o laço após o *end*.

Ao contrário do laço *for*, que executa um conjunto de comandos um número fixo de vezes, o laço *while* executa um conjunto de comandos um número indefinido de vezes. Os comandos entre as instruções *while* e *end* são executadas enquanto todos os elementos na *expressão* forem verdadeiras.

EXEMPLO: Faça um programa que faça um cadastro da idade e do peso de um cidadão. Ao final pergunte ao usuário se quer continuar (1) ou parar (0) o programa.

```
i=1;
a=1;
while a==1
   fprintf('Insira a idade do cidadao %d:',i);
   M(1,i)=input('');
   fprintf('Insira o peso do cidadao %d:',i);
   M(2,i)=input('');
   i=i+1;
   a=input('Continuar? (1) para sim ou (0) para nao.');
end
disp(M)
```

EXEMPLO: Faça um programa que leia um número e calcule o seu fatorial. O programa deve exibir um erro caso o número seja negativo.

```
i=1;
prod=1;
n=input('Digite um valor n para o qual se deseja saber o fatorial:');
if n<0
   error('n deve ser nao negativo.')
else
   while i<=n
      prod=prod*i;
      i=i+1;
   end
   disp(prod)
end
```

EXEMPLO: Faça um programa que plote o gráfico da função $y=a*x^2+b*x+c$, no intervalo $x_v - 5 < x < x_v + 5$.

```
clear % limpa variáveis e funções da memória (RAM)
clc % limpa a tela
aux=1;
while aux==1;
   clc % limpa a janela de comandos, posiciona o cursor no início da tela
   a=input('a=');
   b=input('b=');
   c=input('c=');
   xv=-b/2*a;
   x=(xv-5):0.01:(xv+5); % definição do domínio
   y=a*x.^2+b*x+c;
   plot(x,y);
   figure(1) % Mostra a janela grafica numero 1
   pause
   clc % limpa a tela
   close % fecha a figura
   aux=input('Plotar outro gráfico? (1 para sim, 0 para não)==>'); % Pergunta se
quer plotar um outro gráfico. Apenas quando for digitado "0" o programa termina,
caso contrário continua indefinidamente o loop.
end
if aux==0
   fprintf('fim do loop')
end
```

10.6 O comando *break*

A estrutura *while* permite que um grupo de comandos seja repetido um número indeterminado de vezes. No entanto, a condição de interrupção é testada no início da estrutura. Em várias situações em programação se faz necessário interromper a execução da repetição verificando a condição no interior da estrutura e não no seu início. O comando *break* interrompe a execução de uma estrutura *while* ou *for* e

transfere a execução para o comando imediatamente seguinte ao *end* . Em repetições aninhadas, o *break* interrompe a execução apenas da estrutura mais interna.

Uma repetição com condição de interrupção no interior pode ter a forma:

```
while 1
   <comandos 1>
   if <condição>
   break
   end
  <comandos 2>
end
```

A estrutura *while* é executada indefinidamente a princípio, pois a condição do *while* é sempre verdadeira. Contudo, quando a *<condição>* do *if* for satisfeita o comando *break* será executado causando a interrupção da repetição *while*.

EXEMPLO: Faça um programa que calcule a média de no máximo 5 números usando o comando *break*.

```
acumulador = 0;
n = 0;
while n<5,
  numero = input('Digite um numero: ');
  n = n+1;
  acumulador = acumulador + numero;
  x = input('\n Continua? (1) para sim e (0) para nao: ');
  if x == 0
    break
  end
end;
media = acumulador/n;
disp('A média dos números é: ');
disp(media);
```

10.7 O comando *switch*

O comando ***switch*** executa certas afirmações baseando-se no valor de uma variável ou expressão. É usado quando se tem opções de escolha. Sua forma básica é:

```
switch <expressão>
 case <valor1>
  <afirmações>
 case <valor2>
  <afirmações>
 ...
 otherwise
  <afirmações>
end
```

Apenas um *CASE* é executado e a seqüência de execução do programa prossegue após o *END*. Se nenhum dos *CASEs* concordar com a *expressão* do *switch*, então o caso *OTHERWISE* é executado (se existir). Apenas as instruções entre o *CASE* concordante e o próximo *CASE*, *OTHERWISE* ou *END* são executadas. Por exemplo, se o primeiro caso for verdadeiro os outros casos não são executados.

EXEMPLO: Faça um programa que leia um dos números -1, 0 ou 1 e diga se é negativo, positivo, zero ou nenhum dos outros casos.

```
var=input('Entre com um dos valores:-1, 0 ou 1 ==> ');
switch var
case –1
  disp('Número um negativo')
case 0
  disp('Zero')
case 1
  disp('Número um positivo')
otherwise
  disp('outro valor')
end
```

EXEMPLO:

```
var=input('Entre com um valor ==> ');
switch var
case 1
  disp('1')
case {2,3,4}
  disp('2 ou 3 ou 4')
case 5
  disp('5')
otherwise
  disp('outro valor')
end
```

11. Vetorização

O MATLAB é um *software* desenvolvido para trabalhar com matrizes. Quando se está construindo uma rotina de cálculo e se deseja utilizar uma ferramenta de *loop*, onde o objetivo é calcular uma expressão diversas vezes, é muito mais eficiente trabalhar utilizando o conceito de operações matriciais ou vetoriais.

Para demonstrar o que queremos dizer, vamos apresentar uma rotina simples, implementada de dois modos, onde se evidencia a diferença de eficiência entre uma implementação utilizando *loops* sua versão vetorial.

	x=1;
	for k=1:100
	  clc
	  y(k)=log10(x);
	  x=x+0.01;
	  y(k);
	  pause(1);
	end
		x:1:0.01:100;
		y=log10(x);

TEMPO COMPUTACIONAL	0.1900	0.0600

Utilizando a forma vetorizada, para este exemplo, tem-se uma redução de 70% no tempo computacional.

12. Reserva de espaço para variáveis

A reserva de espaço em memória para as variáveis faz-se simplesmente preenchendo a variável a ser utilizada com zeros, de forma a que a sua dimensão não volte a ser alterada.

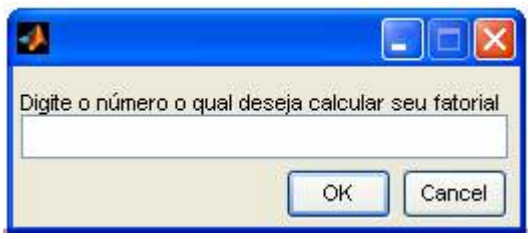
13. Caixa de Diálogos

As caixas de diálogos possibilitam a construção de programas com interface mais amigável. Por exemplo, ao invés de utilizarmos a função ***input*** podemos usar a função ***inputdlg*** (entrada de dados com caixa de diálogos), a qual também permite que se insira dados através do teclado.

EXEMPLO 1: Faça um programa que calcule o fatorial de qualquer número.

```
i=1;
prod=1;
b={'Digite o número o qual deseja calcular seu fatorial'};
Numero=inputdlg(b)
n=str2num(char(Numero(1)));
if n<0
disp('ERRO: O número deve ser não negativo');
else
while i<=n
prod=prod*i;
i=i+1;
end
fprintf('O fatorial de %d é %d',n,prod);
end
```

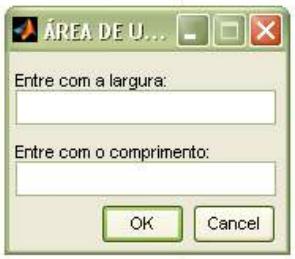
OBS: Neste programa aparecerá uma janela onde o usuário irá informar o número o qual deseja calcular o fatorial.



EXEMPLO 2: Faça um programa tal que o usuário insira a largura e o comprimento de um retângulo e logo após calcule e escreva sua área.

```
aviso={'Entre com a largura:' ,'Entre com o comprimento:'}
titulo='ÁREA DE UM RETÂNGULO';
resposta=inputdlg(aviso,titulo);
larg=str2num(char(resposta(1)));
comp=str2num(char(resposta(2)));
disp(' **************')
disp(' * ÁREA *')
disp(' **************')
area=larg*comp;
disp(area)
```

OBS: Depois de digitado o algorítmo, se colocarmos o programa para rodar, aparecerá a seguinte janela, onde o usuário irá informar a largura e o comprimento do retângulo.



14. Métodos Numéricos

A idéia central destes métodos é partir de uma aproximação inicial para a raiz e em seguida refinar essa aproximação através de um processo iterativo.

Para facilitar a utilização do programa para encontrar raiz através de métodos numéricos, deve-se criar arquivo na janela M-FIRE, para seguinte utilização no **Command Window** do MatLab.

14.1 Método da Bisseção

DESCRIÇÃO: Seja *f(x)* contínua no intervalo [a, b] e *f(a) . f(b) < 0.*

Dividindo o intervalo [a, b] ao meio, obtém-se xô, havendo dois subintervalos, [a, xo] [xo, b], a ser considerados. Se f(xo) = 0, então δ = xo (raiz encontrada), caso contrário, a raiz estará no subintervalo onde a função tem sinais opostos nos pontos extremos, ou seja, se *f(a). f(xo) < 0* , então, δ Є (a, xo); senão *f(a) . f(xo)* > 0 e δ Є (xo, b). O novo intervalo [a_1, b_1] que contém δ é dividido ao meio e obtém-se o ponto x_1. O processo se repete até que obtenha uma aproximação para a raiz exata δ, com a tolerância desejada.

ALGORÍTMO:

```
ao=input('Digite o 1° valor do intervalo:  ');
bo=input('Digite o 2° valor do intervalo:  ');
Ep=input('Insira o valor para o erro:  ');
if f(bo)*f(ao) > 0,
    disp('Não tem raiz nesse intervalo')
else
    while (bo-ao) > Ep,
        x=(ao+bo)/2;
        if f(x)*f(ao) > 0,
            ao=x;
        else
            bo=x;
        end;
    end;
    y =(ao+bo)/2;
    disp('A raiz é   ')
    disp(y);
end
```

EXEMPLO 1: Calcule a raiz positiva para *f(x)* = $x^2 - 5$ com € ≤ 0,00001.

```
>> f = inline('x^2-5');
>> bissec
Digite o 1° valor do intervalo:  0
Digite o 2° valor do intervalo:  3
Insira o valor para o erro:  0.00001
A raiz é
    2.2361
```

OBS: bissec é o arquivo anteriormente digitado na extensão M-FIRE.

EXEMPLO 2: Calcule a raiz aproximada para *f(x)* = $xlogx$ para € ≤ 0,01 e com o intervalo inicial [0,3].

```
>> f = inline('x*log(x)');
>> bissec
Digite o 1° valor do intervalo:  0
Digite o 2° valor do intervalo:  3
Insira o valor para o erro:  0.01
A raiz é
    0.0029
```

14.2 Método de Newton

DESCRIÇÃO: Seja *f(x)* uma função contínua no intervalo [a, b] e δ o seu único zero neste intervalo; as derivadas *f'(x) (f'(x)≠0) e f''(x)* devem também ser contínuas. Encontra-se uma aproximação x_0 para a raiz δ e é feita uma expansão em série de Taylor para *f(x)=0.* É condição suficiente para a convergência do Método de Newton que: *f'(x) e f''(x)* sejam não nulas e preservem o sinal em (a, b) e x_0 seja tal que $f(x_0).\ f''(x_0) > 0.$

ALGORÍTMO:

```
xo=input('Digite o valor de xo:  ');
E=input('Insira o valor para o erro:  ');
x1=xo-f(xo)/g(xo);
while (x1-xo)>E
    xo=x1;
x1=xo-f(xo)/g(xo);
end
disp('A raiz é   ');
disp(x1)
```

EXEMPLO 1: Achar a raiz de $f(x) = x^3 - 5x^2 + x + 3$, com € ≤ 0,0001 e $x_0 = -2$.

```
>> f = inline ('x^3-5*x^2+x+3');
>> g = inline ('3*x^2-10*x+1');
>> newton
Digite o valor de xo:  -2
Insira o valor para o erro:  0.0001
A raiz é
   -0.6458
```

Neste, utilizamos o arquivo newton, digitado na extensão M-FIRE.

EXEMPLO 2: Encontre a raiz de $f(x) = 2x^3 + logx - 5$, pelo método de Newton, com € ≤ 10^{-3}. Sabe-se que δ Є [1,2].

```
>> f = inline('2*x^3+log(x)-5');
>> g = inline('6*x^2+(1/x)');
>> newton
Digite o valor de xo:  2
Insira o valor para o erro:  0.001
A raiz é
    1.5227
```

14.3 Método da Secante

DESCRIÇÃO: É muito semelhante ao Método de Newton, mas substitui o cálculo das derivadas pelo cálculo de uma razão incremental.

Geometricamente, corresponde a substituir o papel da tangente, no método de Newton, por uma secante (de onde vem o nome). Isto significa que vamos precisar sempre de dois pontos para determinar, o que implica que tenhamos que considerar duas iteradas iniciais, que designamos por x_0 e x_1. É condição suficiente para a convergência do Método da Secante que: *f'(x) e f''(x)* sejam não nulas e preservem o sinal em (a, b), e $f(x_0)\ .f\ ''(x_0) \geq 0$ e também $f(x_1)\ .f\ ''(x_1) \geq 0$ para qualquer x_0 e x_1 em

(a, b).

ALGORÍTMO:

```
xo=input('Digite o valor de xo:  ');
x1=input('Digite o valor de x1:  ');
E=input('Insira o valor para o erro:  ');
while (x1-xo)>E
x2=x1-((f(x1)*(x1-xo))/(f(x1)-f(xo)));
xo=x1;
x1=x2;
end
disp('A raiz é:    ');
disp(x2);
```

EXEMPLO 1: Considerando $f(x) = x^2 + x - 6$; $x_0 = 1$ e $x_1 = 2$. Encontre a raiz da $f(x)$ com € ≤ 0,001.

```
>> f = inline('x^2+x-6');
>> secante
Digite o valor de xo:  1
Digite o valor de x1:  2
Insira o valor para o erro:  0.001
A raiz é:
     2
```

EXEMPLO 2: Considerando $f(x) = x^3 - 9x + 3$; $x_0 = 0$ e $x_1 = 1$. Encontre a raiz da $f(x)$ com $\epsilon \leq 5 \times 10^{-4}$.

```
>> f = inline('x^3-9*x+3');

>> secante

Digite o valor de xo:  0

Digite o valor de x1:  1

Insira o valor para o erro:  0.0005

A raiz é:

    0.3750
```

REFERÊNCIAS BIBLIOGRÁFICAS

PET MATEMÁTICA UFSM; **Noções Básicas de Utilização e Programação em MATLAB**. Santa Maria, 2008.

CHAPMAN, S.J.; **Programação em MATLAB para engenheiros**. Tradução técnica: Flávio Soares Correa da Silva, São Paulo, Thomson Learning, 2006.

FALCÃO, M. I; **Iniciação ao MATLAB.** Universidade do Minho, 2001, Disponível em: https://repositorium.sdum.uminho.pt/handle/1822/1480

HANSELMAN, D; LITTLEFIELD, B; **MATLAB – Versão Estudante – Guia do Usuário – Versão 4.** MAKRON Books do Brasil. São Paulo, 1997.

PET ENGENHARIA ELÉTRICA UFSM; **Introdução ao MATLAB**. Santa Maria, 2007. Disponível em: http://w3.ufsm.br/petee/

PORTUGAL, R.; **MATLAB para leigos e desinteressados: uma introdução simples com exemplos banais.** Disponível em: *www.ime.unicamp.br/~cheti/intmatlab.pdf*