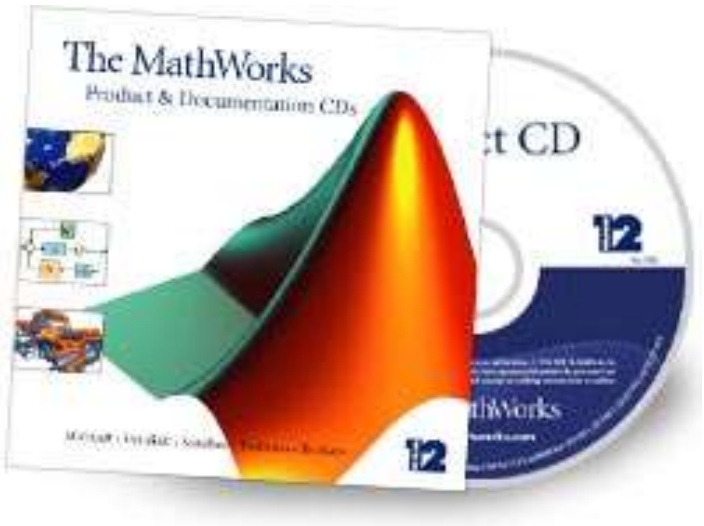
APOSTILA

MATLAB



1 INTRODUÇÃO 1

2 EXEMPLOS SIMPLES 1

3 CARACTERÍSTICAS BÁSICAS 4

3.1 Área de Trabalho do MATLAB 4

3.2 Formato dos Números 5

3.3 Variáveis 5

3.4 Comentários e Pontuações 7

3.5 Operadores Relacionais 8

4 NÚMEROS COMPLEXOS 8

5 FUNÇÕES MATEMÁTICAS 9

6 GERENCIAMENTO DE ARQUIVOS 13

6.1 Arquivos Script (*M-files*) 13

7 CONSTRUÇÃO DE VETORES 17

7.1 ENDEREÇAMENTO DE VETORES 19

8 CONTROLES DE FLUXO NO MATLAB 22

8.1 Loops for 22

8.2 Loops while 23

8.3 Estruturas if-else-end 23

9 MATRIZES 25

9.1 Operações Escalares com Matrizes no MatLab 26

9.2 Operações entre Matrizes no MatLab 27

10 MANIPULAÇÃO DE GRÁFICOS 29

10.1 Estilo de Linhas e Cores: 34

11 EXEMPLOS DE APLICAÇÃO DE VETORES NA VISUALIZAÇÃO DE SINAIS REPRESENTADOS NO DOMÍNIO DO TEMPO 35

1 INTRODUÇÃO

O MATLAB é uma ferramenta software que pode funcionar como uma simples calculadora ou até como uma linguagem de programação científica (fortran, C, etc.) para soluções de complicadas expressões algébricas. Entretanto, o MATLAB apresenta diversas vantagens em relação as calculadoras e linguagens de programação: simplicidade e uma interface gráfica bastante completa para visualização e análise dos resultados.

2 EXEMPLOS SIMPLES

Os exemplos aqui apresentados tem o objetivo de demonstrar alguns tipos de problemas que podem ser resolvidos utilizando o Matlab.

Como primeiro exemplo tem-se a utilização do Matlab como uma calculadora. Os comandos apresentados abaixo devem ser digitados diretamente no prompt (**>>**) do Matlab. O símbolo ← significa apertar a tecla **Enter**.

```
>> 3+5+2          ←
ans =
        10
```

Neste exemplo foram digitados três números intercalados pelo símbolo +. Esta expressão para o Matlab significa a soma destes três números. Ao apertar a tecla Enter (←) é apresentado o resultado desta operação que é 10. As letras **ans** significa answer, ou seja, resposta.

O próximo exemplo mostra a capacidade de atribuir um valor a uma variável e realizar operações com estas variáveis.

```
>> a = 5        ←

a =

        5

>> b = 33.50    ←

b =

        33.5000

>> a + b        ←
```

ans

38.5000

Observe que neste exemplo foram atribuídos valores para as variáveis **a** e **b** e em seguida é feita uma operação somando os dois valores.

Um comando interessante de apresentar neste momento é o **who**. Ao digitar este comando no prompt do Matlab e pressionar o Enter são apresentadas as variáveis utilizadas até o momento.

O próximo exemplo fará a visualização da função

$$y = A.sin(x + \phi)$$

A seguir estão os comandos para apresentar o gráfico desta função com A=1 e ϕ =phi=0 . Estes comandos devem ser digitados diretamente no prompt do Matlab.

```
>>A=1;          ←
>>phi = 0;      ←
>> x = 0:360;   ←
>> y = A*sin(2*pi*x/360+phi);        ←
>> plot(y,'k');grid on   ←
>>xlabel('Valores de x em graus')     ←
>>ylabel('y')   ←
>>title('Figura 1 : sin(2*pi*x/360)')  ←
```

O resultado destes comandos é a plotagem de um ciclo do seno como mostrado na figura 1. Observe a necessidade da divisão por 360 para que seja possivel o aparecimento de um ciclo completo.

Exercício : Plote um ciclo do seno como mostrado na figura 1.1 quando :

i) A = 2 e ϕ = phi = 0.

Use os comandos :

```
>>A=2;          ←
>>phi = 0;      ←
>> x = 0:360;   ←
>> z = A*sin(2*pi*x/360+phi);        ←
```

>> plot(z,'k-');grid on ←

ii) A=2 e ϕ=phi=2*pi* 45/360.

Use os comandos :

```
>>A=2;          ←
>>phi = 2*pi*45/360;
>> x = 0:360;  ←
>> v = A*sin(2*pi*x/360+phi);        ←
>> plot(v,'k');grid on   ←
```

iii) Finalmente use o comando :

>> plot(x,y,'k-.',x, z,'b-o',x, v,'r+'); grid on ←

O MATLAB oferece as seguintes operações aritméticas básicas :

Tabela 1 : Operações Aritméticas Básicas

Operação	**Símbolo**	**Exemplos**
adição, a + b	+	8+3
subtração, a – b	–	28–15
multiplicação, a.b	*	4.15*8.10
divisão, a÷b	/ ou \	64/5 ; 67\9
power, a^b	^	5^2

As expressões são executadas da esquerda para a direita com a seguinte ordem de precedência: operação de potência, seguida das operações de multiplicação e divisão, que por sua vez são seguidas pelas operações de adição e subtração. Parênteses podem ser usados para alterar esta ordem de precedências, onde as operações são executadas dos parênteses mais internos para os mais externos.

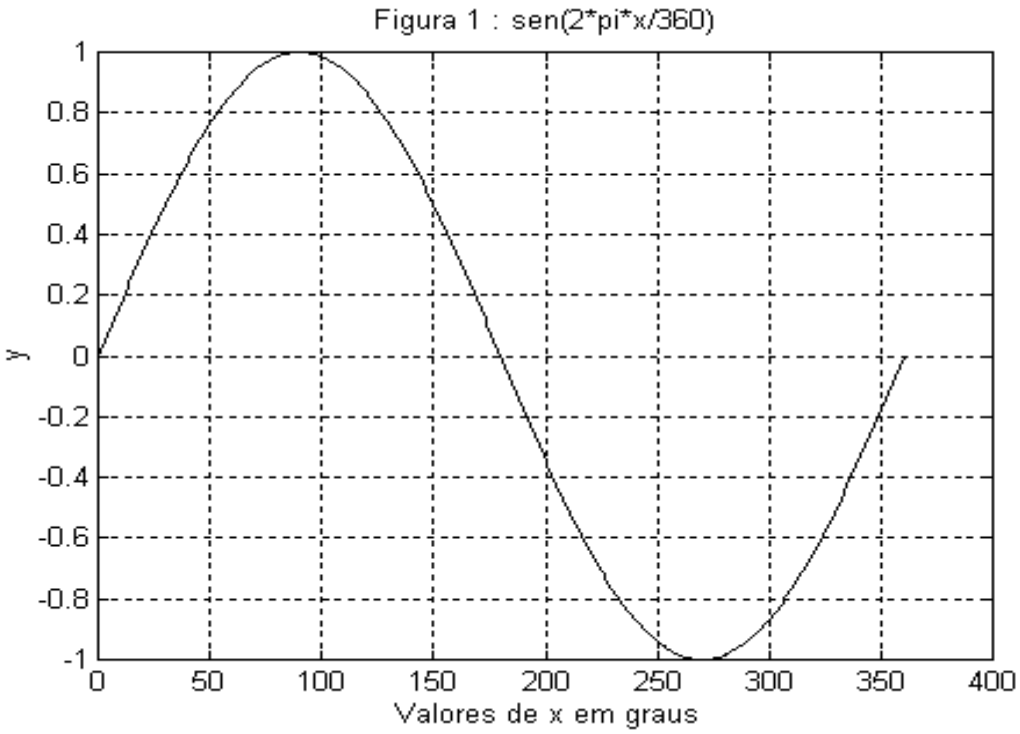


Figura 1 - Plotagem de um seno

3 CARACTERÍSTICAS BÁSICAS

O MATLAB é uma ferramenta software que tem por característica básica a simplicidade de utilização e uma poderosa interface gráfica. Como qualquer software ou linguagem de programação é necessário haver uma adaptação à ferramenta.

3.1Área de Trabalho do MATLAB

A *área de trabalho* do MATLAB é onde ficam residentes os diversos *comandos* e *valores* de quaisquer *variáveis* que foram digitados na *janela de comandos*. Como aqueles comandos e variáveis estão residentes na área de trabalho do MATLAB, podem ser invocados sempre que for preciso ou desejado. Por exemplo, se quiser verificar o valor da variável **a** basta que se entre com o nome desta variável no prompt.

>> **a** ←
a =
5

Como já foi visto, se você não consegue lembrar das variáveis, pode-se obter uma lista de todas as variáveis usando-se o comando **who**

>>who ←

Your variables are :

a	b	x	A	phi
ans	y	z	v	

3.2Formato dos Números

Como default, se um resultado é inteiro, o MATLAB mostra o número como inteiro. Igualmente, quando o resultado é real, o MATLAB mostra o número com 4 dígitos a direita do ponto decimal. Se os dígitos do resultado estiverem fora desta faixa, o MATLAB mostra o resultado usando a notação científica como numa calculadora científica. Este default pode, entretanto, ser modificado usando-se o **Numeric Format** da pasta **general** na linha **Preferences** do ítem **Files** na barra de menus.

Exercício : Usando-se a variável **b** usado anteriormente, verifique a tabela de formatos numéricos abaixo :

Tabela 2 : Formatos Numéricos

Comando MATLAB	**b**	**Comentários**
format long	33.50000000000000	16 dígitos
format short e	$33.500e^{+01}$	5 dígitos mais expoente
format long e	33.50000000000000 e^{+01}	16 dígitos mais expoente
format hex	4040c00000000000	hexadecimal
format bank	33.50	2 dígitos decimais
format +	+	positivo, negativo ou zero
format rat	67/2	racional
format short	33.5000	4 dígitos decimais (formato default)

3.3Variáveis

Os nomes das variáveis devem consistir de uma única palavra, conforme as três regras abaixo :

Tabela 3 : Regras de Construção de Variáveis

Regras de Construção das Variáveis	**Comentários/Exemplos**
Variáveis em letras minúsculas e maísculas são diferentes mesmo que consistam das mesmas letras	**Items, items, itEms** e **ITEMS** são variáveis diferentes entre si no MATLAB
As variáveis podem consistir de até 19 letras	holnmbjkitkklwenohu
As variáveis devem começar com alguma letra, podendo ser seguido por quaisquer letras, dígitos ou subescrito. Caracteres de pontuação não podem ser utilizados.	how_about x512 a_b_c_d

Em adição às regras acima para formação das variáveis, as seguintes variáveis são especiais no MATLAB : **ans**, **pi**, **eps**, **flops**, **inf**, **NaN**, **i**, **j**, **nargin**, **nargout**, **realmin** e **realmax**

Tabela 4 : Variáveis Especiais

Variáveis Especiais	Significado
ans	Nome "default" da variável usado para resultados
pi	Constante igual à razão da circunferência de um círculo para o seu diâmetro
eps	O menor número tal que quando adicionado com um outro resulta em um número diferente
flops	Conta o número de operações em ponto- flutuante
inf	Indica um número infinito, p.e., 1/0
NaN	Indica que não é um número, p. e., 0/0
i (e) j	$i=j=\sqrt{-1}$
nargin	Número de argumentos de entrada usados em uma função
nargout	Número de argumentos de saída usados em uma função
realmin	O menor número real positivo utilizável
realmax	O maior número real positivo utilizável

3.4Comentários e Pontuações

Todo e qualquer texto depois do símbolo de porcentagem (%), é tomado como sendo um *comentário* :

>>n_macacos=10 % Número de macacos que vivem no bosque ←

n_macacos=

10

Vários comandos podem ser colocados em uma linha se os mesmos forem separados por *virgula* ou *ponto e vírgula* :

>>n_macacos=10, n_laranjas=20, n_uvas=1000; n_bananas=100 ←

n_macacos=

10

n_laranjas=

20

n_bananas=

100

Observe que quando é utilizado *ponto e vírgula* o MATLAB não mostra o resultado. No exemplo acima, o valor atribuído à variável **n_uvas** não foi mostrado por causa do ponto e vírgula que foi utilizado.

Observe agora o efeito dos três pontos (...) :

>>clientes=10; ←

```
>>n_uvas_por_cliente=n_uvas/...  ←

clientes  ←

>>n_uvas_por_cliente=

        100
```

Assim, os três pontos (...) indica ao MATLAB continuação da expressão matemática na próxima linha.

3.5 Operadores Relacionais

Os operadores relacionais do Matlab incluem todas as comparações comuns e são apresentados na tabela 5.

Tabela 5: Operadores relacionais

Operador	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual a
~=	Diferente de

4 Números Complexos

Os números complexos podem ser representados no MATLAB de diversas maneiras. Alguns exemplos são mostrados a seguir :

```
>>x=1-4i        % a letra "i" significa ou indica a parte imaginária  ←

x=
        1.000 - 4.000i

>>x=1-4j        % a letra "j" também pode ser usada para indicar a parte imaginária
←

x=
        1.000 - 4.000i
```

Identidade de Euler : relaciona a forma polar de um número complexo com a sua forma retangular

$$M\angle\theta \equiv M \cdot e^{j\theta} = a + bi$$
$$M = \sqrt{a^2 + b^2}$$
$$\theta = \text{theta} = \tan^{-1}(b / a)$$
$$a = M \cdot \cos\theta$$
$$b = M \cdot \sin\theta$$

No MATLAB, a conversão entre as formas polar e retangular de um número complexo utiliza as seguintes funções : **real, imag, abs** e **angle**.

```
>>x  ←

x=
        1.000 - 4.000i

>>M=abs(x)  ←

M=
        4.1231

>>theta=angle(x)*180/pi  ←

theta=-14.0362

>>a=real(x)  ←

a=
        1

>>b=imag(x)  ←

b=
        -4
```

---

5 Funções Matemáticas

Na tabela abaixo tem-se uma lista parcial das funções comuns que o MATLAB suporta.

Tabela 6 : Funções Comuns

abs(x)	Valor absoluto ou magnitude de um número complexo

acos(x)	Inverso do coseno
acosh(x)	Inverso do coseno hiperbólico
angle(x)	Ângulo de um número complexo
asin(x)	Inverso do seno
asinh(x)	Inverso do seno hiperbólico
atan(x)	Inverso da tangente
atanh(x)	Inverso da tangente hiperbólico
conj(x)	Conjugado complexo
cos(x)	Coseno
cosh(x)	Coseno hiperbólico
exp(x)	Exponencial : e^x
fix(x)	Arredondamento em direção ao zero
floor(x)	Arredondamento em direção ao menos infinito
gcd(x,y)	Máximo divisor comum dos inteiros **x** e **y**
imag(x)	Parte imaginária de um número complexo
lcm(x)	Mínimo múltiplo comum dos inteiros **x** e **y**
log(x)	Logarítmo natural
log10(x)	Logarítmo comum
real(x)	Parte real de um número complexo
rem(x,y)	Resto da divisão de **x/y**
round(x)	Arredondamento para o inteiro mais próximo
sign(x)	Função *signum*
sin(x)	Seno
sinh(x)	Seno hiperbólico
sqrt(x)	Raíz quadrada
tan(x)	Tangente
tanh(x)	Tangente hiperbólico

Uma função muito útil no estudo de princípios de comunicações, é a função retangular definida abaixo :

$$f(t) = \begin{cases} 1 & (0 < t \le \pi) \\ -1 & (\pi < t \le 2\pi) \end{cases}$$

No MATLAB, pode-se visualizar esta função a partir da função **sign(x)**. Para um melhor entendimento, inicialmente é visualizado a função -sign(t) no intervalo $-\pi \le t \le \pi$.

```
>>t=-pi:2*pi/100:pi;  % Cria-se o vetor  t com 100 elementos em  ←
>>                    % incrementos de 2*pi/100                 ←
>>y=-sign(t);
>>plot(t,y,'g+'); grid on                                       ←
>>title('Figura 2 : Função -sign(t)')                           ←
>>xlabel('t')                                                   ←
>>ylabel('-sign(t)')                                            ←
```

A figura 2 mostra a plotagem da função –sign(t), no intervalo $-\pi \leq t \leq \pi$.

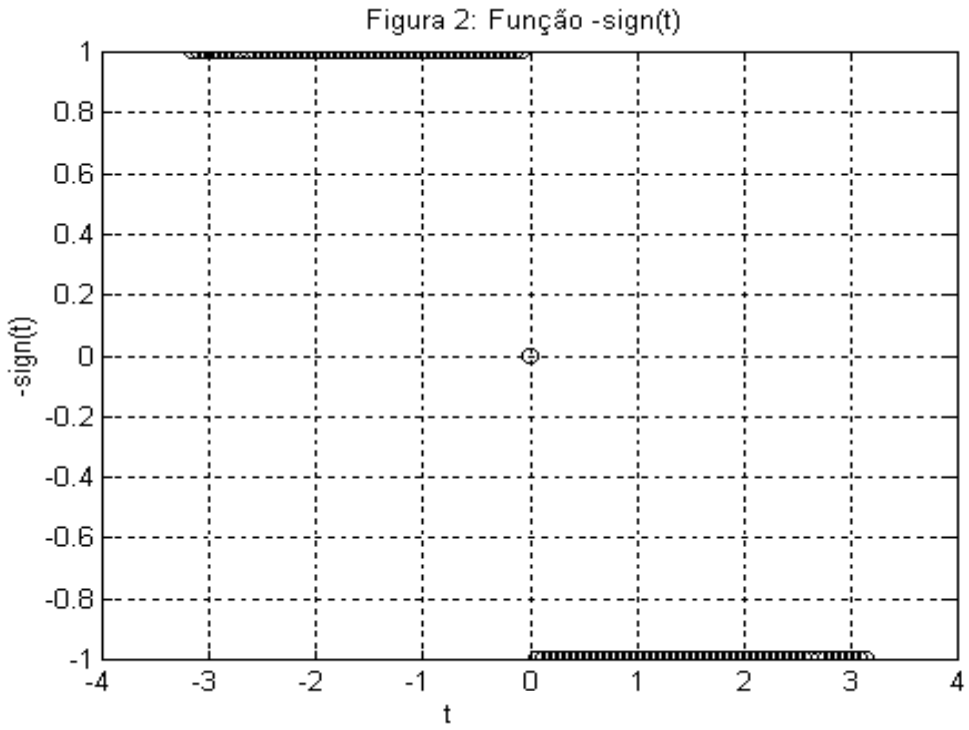


Figura 2 – Função –sign(t), no intervalo $-\pi \leq t \leq \pi$.

Comparando-se a curva obtida da função -sign(t) com a definição da função retangular f(t), conclui-se que f(t) é a função -sign(t) deslocada de π, i.e. :

$$f(t) = -\text{sign}(t - \pi), \text{ para } 0 < t < 2\pi$$

Portanto, os seguintes comandos são suficientes para reproduzir a função retangular, f(t):

```
>>t=0:2*pi/100:2pi;   % Cria-se o vetor  t com 100 elementos em        ←
>>                    % incrementos de 2*pi/100                       ←
>>y=-sign(t-pi);      ←
>>plot(t,y,'g+'); grid on                ←
>>title('Figura 3 : Função retangular f(t)')          ←
>>xlabel('t')         ←
>>ylabel('f(t)')      ←
```

Obtém-se o resultado mostrado abaixo (figura 3).

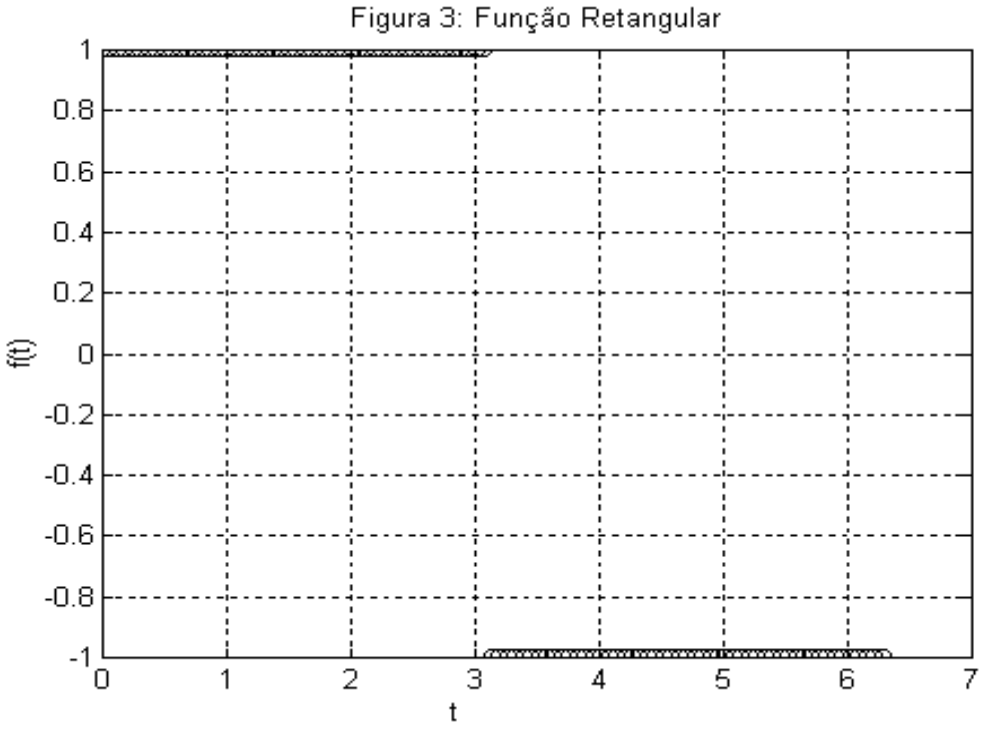


Figura 3 - Função retangular deslocada

6 Gerenciamento de Arquivos

O MATLAB possui uma série de comandos para gerenciamento de arquivos, tais como listar os nomes de arquivos, vizualizar, deletar, etc. Na tabela abaixo tem-se um resumo dos principais comandos :

Tabela 7 : Comandos para Gerenciamento de Arquivos

cd	Mostra o diretório de trabalho atual ou corrente
p=cd	Retorna para a variável **p** o diretório de trabalho corrente
cd temp	Muda para o diretório **temp**
cd ..	Muda para o diretório um nível acima
chdir	O mesmo que cd
chdir path	O mesmo que cd temp
delete test	deleta o arquivo **test.m**
dir	Lista todos os arquivos do diretório de trabalho presente
ls	Faz o mesmo que o comando **dir**
matlabroot	Retorna o caminho do diretório onde se encontra o programa MATLAB executável
path	Visualiza todos os caminhos de diretório do MATLAB
pwd	O mesmo que o comando **cd**
type test	Visualiza o arquivo *M-file* **test.m** na janela de comandos
what	Retorna uma lista de todos os *M-files* do diretório corrente
which test	Visualiza o caminho do diretório do arquivo **test.m**

6.1Arquivos Script (*M-files*)

Para resolver problemas simples, é cômodo e eficiente utilizar o MATLAB como se fosse uma calculadora, entrando-se com os comandos diretamente no prompt. Entretanto, a medida que o número de comandos aumenta, ou quando se deseja mudar o valor de uma ou mais variáveis e executar novamente os comandos, o uso do MATLAB simplesmente como calculadora torna-se ineficiente e tedioso. Nestes casos, o melhor é utilizar o MATLAB como uma linguagem de programação de alto nível, isto é, escrever um arquivo "script" ou *M-files*. Os *M-files* são os programas fontes do MATLAB e levam a extensão .**m** , por exemplo, **exemplo1.m**.

Para escrever um programa no MATLAB, escolha **File** na barra de menu. Dentro do menu **File** escolha **New** e selecione **M-file**, como mostrado na figura 4. Abre-se, então, um editor de textos, onde pode-se escrever os comandos do MATLAB. Escreva, por exemplo, o programa abaixo :

```
%================================================
% Exemplo de programação no MATLAB
% Este exemplo plota uma função seno nas sequintes
% condições:
%               sen(x)
%               2*sen(x)
%               2*sen(x+45)
%               2*sen(x-90)
%               2*sen(2*x)
%================================================

%
x=0:360;
%
% Seno com amplitude A=1 e defasagem phi=0 graus
A=1;
phi=0;
y=A*sin(2*pi*x/360+2*pi*phi/360);

% Seno com amplitude A=2 e defasagem phi=0 graus
A=2;
phi=0;
z=A*sin(2*pi*x/360+2*pi*phi/360);

% Seno com amplitude A=2 e defasagem phi=45 graus
A=2;
phi=45;
v=A*sin(2*pi*x/360+2*pi*phi/360);

% Seno com amplitude A= 2 e defasagem  phi=-90  graus
A=2;
phi=-90;
w=A*sin(2*pi*x/360+2*pi*phi/360);

% Seno com amplitude A= 2 e defasagem  phi=0  graus
A=2;
phi=0;
u=A*sin(2*pi*2*x/360+2*pi*phi/360);

% Plotagem do resultado
plot(x,y,'k-',x,z,'k--',x,v,'k-.',x,w,'k.',x,u, 'ko')
grid
xlabel('Valores de x em graus')
ylabel('y,z,v,w e u')
title('Estudo de defasagem e amplitude de um seno')
legend('sen(x)','2*sen(x)','2*sen(x+45)','2*sen(x-90)', '2*sen(2*x)')
```

Uma vez escrito o programa, entre no menu **File** da janela do editor de textos e escollha a opção **Save as...** Nesta opção do menu, salve o programa como *prog1.m* no seu diretório de trabalho. Em seguida, feche a janela do editor de textos, o que faz com que volte à janela de comandos do MATLAB. Na janela de comandos, use o comando *cd* para ir ao diretório onde o programa *prog1.m* foi salvo. Em seguida, digite :

>>prog1 ←

O gráfico mostrado na figura 5 é obtido.

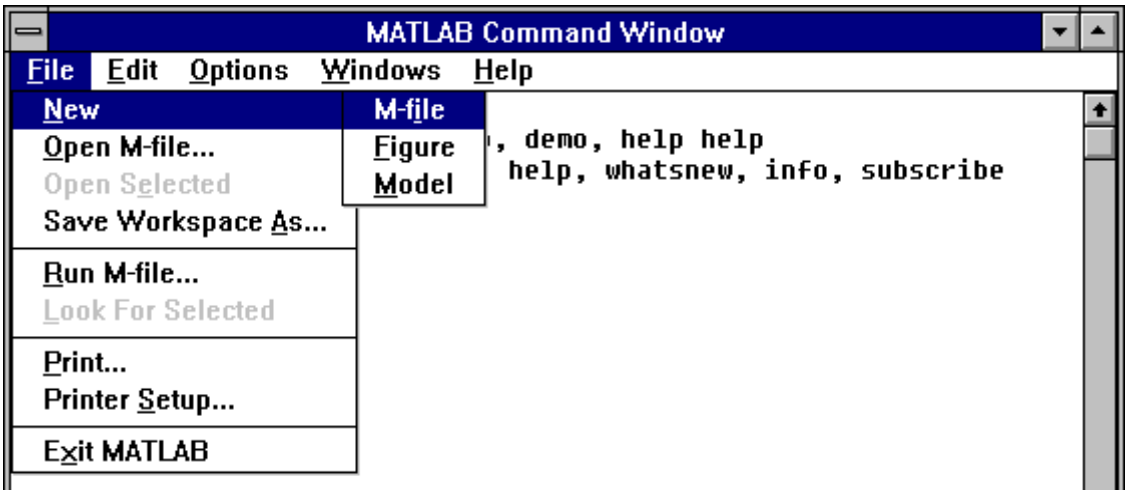


Figura 4. Procedimento para começar um novo *M-file*

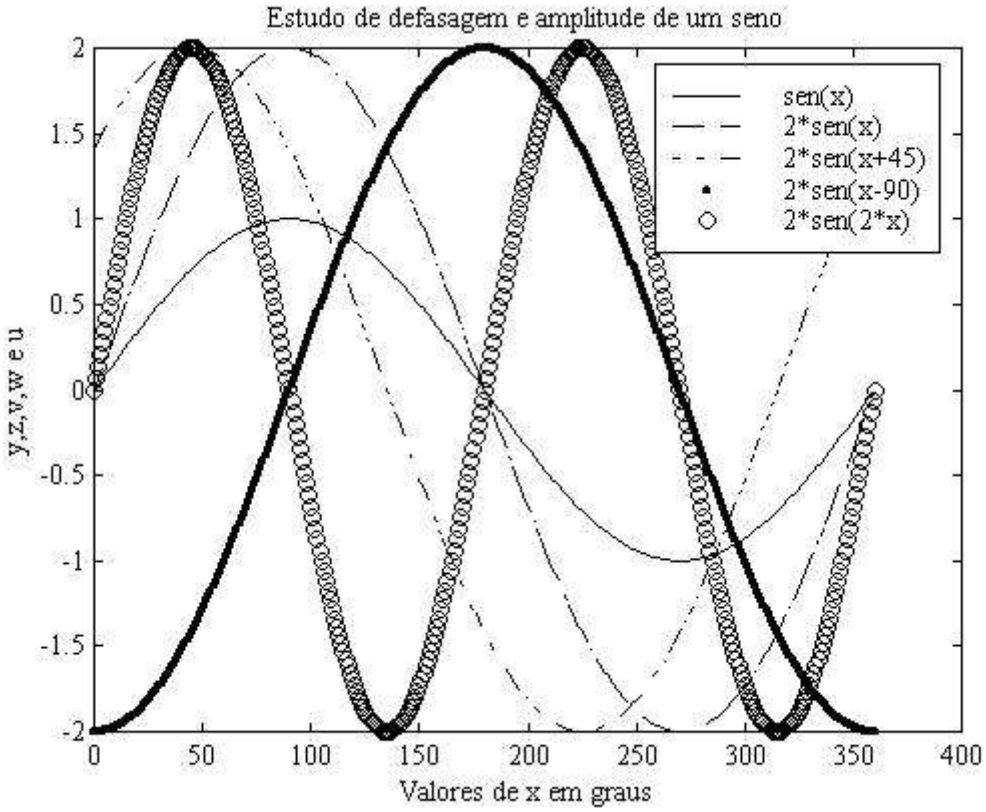


Figura 5 - Senos deslocados

Funções M-files :

Por causa da grande utilidade dos *M-files*, o MATLAB possui diversas funções que tornam os *M-files* ainda mais interessantes. Estas funções estão listadas na tabela 8 :

Tabela 8 : Funções *M-files*

echo	Ecoa cada um dos comandos do *M-file* na janela de comandos.
input	Permite entrada de dados durante a execução do programa via teclado.
pause	Faz uma pausa na execução do programa até que uma tecla qualquer seja pressionada.
pause(n)	Faz uma pausa de **n** segundos na execução do programa
disp(ans)	Visualiza os resultados sem mostrar os nomes das variáveis
waitforbottonpress	Faz uma pausa até que uma tecla do "mouse" ou do teclado seja pressionada.
keyboard	Passa o controle temporariamente para o teclado ("Type **return** to quit")

Como exemplo, escreva o seguinte programa (*M-file*) :

```
% Exemplo de utilização da função M-file "input"
%=======================================
% Programa para traçar a curva :
%
% y=A.sin(x+phi),
%
% sendo que os valores de x [rad] ; A e phi [graus] devem ser
% entrados via teclado durante a execução do programa
%
x=input('Entre com o vetor x [rad]>    ');
A=input('Entre com o valor de A>     ');
phi=input('Entre com o valor de phi [graus]>     ');
%
y=A*sin(x+2*pi*phi/360);
plot(x,y,'r'); grid on
title('Exemplo de utilização da função "input"')
xlabel('x em rad/s')
ylabel('y=A.sin(x+phi)')
```

---

7 CONSTRUÇÃO DE VETORES

Nas construções das funções implementadas até agora, utilizou-se da construção de vetores. Agora, mostrar-se-á algumas outras formas de manipular vetores no MATLAB. Na tabela 8, tem-se um resumo das diversas formas de se construir um vetor no MatLab.

Tabela 9. Construção de Vetores

x=[2 2*pi sqrt(2) 2-3j]	Cria um vetor **x** contendo os elementos especificados
x=primero : último	Cria um vetor **x** começando com o valor **primeiro**, incrementando-se de 1(um) em 1(um) até atingir o valor **último** ou o valor mais próximo possível de **último**
x=primeiro:incremento:último	Cria um vetor **x** começando com o valor **primeiro**, incrementando-se do valor **incremento** até atingir o valor **último** ou o valor mais próximo possível de último
x=linspace(primeiro, último, n)	Cria um vetor **x** começando com o valor **primeiro** e terminado-se no valor **último**, contendo **n** elementos
x=logspace(primeiro, último, n)	Cria um vetor com os elementos espaçado logaritmicamente, começando-se com o valor $10^{\textbf{primeiro}}$ e terminando-se no valor $10^{\textbf{último}}$, contendo **n** elementos

Exemplos :

i) Uso da construção de vetor : **x=[2 2*pi sqrt(2) 2-3j]**

>> x=[8 6 8.10 5-6j] ←

x=

8.0000 6.0000 8.1000 5.0000-6.0000i

---

ii) Uso da construção de vetor : **x=primero : último**

>> x=1:10.5 ←

x=

1 2 3 4 5 6 7 8 9 10

---

iii) Uso da construção de vetor : **x=primeiro:incremento:último**

>> x=1:0.5:10.5 ←

x =

Columns 1 through 7

1.000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000

Columns 8 through 14

4.5000 5.0000 5.5000 6.0000 6.5000 7.0000 7.5000

Columns 15 through 20

8.0000 8.5000 9.0000 9.5000 10.0000 10.5000

iv) Uso da construção de vetor : **x=linspace(primeiro, último, n)**

```
>> x=linspace(1,10.5,5)        ←

x=

        1.0000    3.3750    5.7500    8.1250    10.5000
```

v) Uso da construção de vetor : **x=logspace(primeiro, último, n)**

```
>> x=logspace(0,2,5)   ←

x=

        1.0000    3.1623    10.0000    31.6228    100.00
```

7.1ENDEREÇAMENTO DE VETORES

Seja :

```
>> x=linspace(0,360,10)        ←

x=

        0    40    80    120    160    200    240    280    320    360

>> y=sin(2*pi*x/360)   ←

y=

  Columns  1  through   7

        0    0.6428    0.9848    0.8660    0.3420    -0.3420    -0.8660
```

Columns 1 through 10

-0.9848 -0.6428 0.0000

No exemplo acima, os vetores de entrada *x* e de saída *y* possuem 11 elementos cada um. No MatLab, cada um dos elementos de um vetor podem ser acessados através de seu ***índice*** que identifica cada uma das colunas. Por exemplo :

>> x(3)% Acessa o terceiro elemento de *x* ←

ans =

80

>> x(5)% Acessa o quinto elemento de *x* ←

ans =

160

O MatLab também permite o acesso em blocos dos elementos de um vetor, como ilustrado nos exemplos a seguir :

>> x(1:5) ←

ans =

0 40 80 120 160

Neste exemplo, a expressão entre parêntesis " 1:5 " , diz para acessar os elementos de 1 a 5 do vetor *x*.

>> x(2:2:8) ←

ans =

40 120 200 280

A expressão entre parêntesis no comando acima, “ 2:2:8” , diz para que acesse os elementos do vetor *x*, começando-se do 2o. elemento e, a partir deste, contando-se de 2 em 2 até atingir o 8o. elemento.

>> x(2:2:7) ←

ans =

40 120 200

Este exemplo é igual ao anterior, mas como depois do 6o. elemento vem o 8o. elemento (pois a contagem é feita de 2 em 2), o 7o. elemento não é acessado.

>> y(10:-2:1) ←

ans =

0.0000 -0.9848 -0.3420 0.8660 0.6428

A expressão entre parêntesis no comando, “ 10:-2:1” , diz para que acesse os elementos do vetor *y*, começando-se do 10o. elemento e, a partir deste, contando-se de 2 em 2 no sentido decrescente até atingir o 1o. elemento.

>> y([8 5 10 1]) ←

ans =

-0.9848 0.3420 0.0000 0

Os elementos do vetor *y* são acessados na ordem indicada pelo vetor de *índices* [8 5 10 1], isto é : *y*(8), *y*(5), *y*(10) e *y*(1).

8 Controles de Fluxo no MatLab

8.1Loops for

Os loops **for** permitem que um conjunto de comandos seja repetido por um número de vezes fixo e pré-definido. A forma geral do loop **for** é :

```
for x=n1:n2
        comandos
end
```

Exemplo 1 : O vetor x=[0 36 72 108 144 180 216 252 288 324] pode ser construído com as seguintes instruções :

```
>> x(1)=0;  ←
>> for n=2:10  ←
        x(n)=x(n-1)+36;  ←
end  ←
>> x  ←

x =

   0   36   72  108  144  180  216  252  288  324
```

Isto é, a primeira instrução diz : para n igual a 2 até 10, execute todas os comandos até a instrução de *end*. No primeiro ciclo do *for*, n=2, no segundo n=3 a assim por diante, até n=10. Depois do ciclo para n=10, o loop *for* termina e os comandos após a instrução *end* são executados, como é o caso da apresentação dos resultados em x.

Exemplo 2 : Plote 360 pontos de um período da função y=sen(2*π*x/360) mostrado na figura 1 usando o loop *for*.

```
>>for x=1:360  ←
   y(x)=sin(2*pi*x/360); ←
end  ←
```

```
>>plot(y) ←
```

8.2Loops while

Ao contrário do loop *for*, que executa um conjunto de comandos um número fixo de vezes, o loop *while* executa um conjunto de comandos um número indefinido de vezes. A forma geral do loop *while* é :

```
while expressão
        Comandos
end
```

Os comandos entre as instruções *while* e *end* são executadas enquanto todos os elementos na *expressão* forem verdadeiras.

Exemplo : Construa o vetor y = [64 32 16 4 2 1] , usando o loop while

---

```
>>eps=128; ←
>>n=0; ←
>>while eps>1   ←
    eps=eps/2; ←
    n=n+1; ←
    y(n)=eps; ←
end  ←
>>y   ←
```

8.3Estruturas if-else-end

Em diversas situações, as seqüências de comandos tem de ser executadas condicionalmente, com base em um teste relacional. Isto pode ser resolvido por meio de alguma variação da estrutura *if-else-end.* A estrutura *if-else-end* mais simples é :

```
if   expressão
        Comandos
end
```

Os comandos entre as instruções *if* e *end* são executados se todos os elementos na *expressão* forem Verdadeiros (diferentes de zero).

Exemplo 1 :

```
» custo=5;
» bananas=10;
» if  bananas>5
        custo=0.1*custo;
end
» custo

custo =

   0.5000
```

No exemplo acima, a expressão *bananas* > 5 é verdadeira, assim o comando :

custo=0.1* custo

foi executado, de modo que o novo valor de custo é igual a 0.5.

Exemplo 2 :

```
custo=5;
» bananas=5;
» if   bananas>5
        custo=0.1*custo;
end
» custo

custo =

    5
```

Neste exemplo, a expressão *bananas* > 5 é falsa, assim o comando :

custo=0.1* custo

não foi executado. Assim o custo continua igual a 5.

Exemplo 3 : Plote a função retangular da figura 3 utilizando-se a estrutura *if-else-end*.

```
%
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, 'ko'); grid on
title('Figura II.3: Função retangular')
xlabel('t em radianos')
ylabel('f(t)')
```

9 MATRIZES

Os vetores vistos até agora tratam-se de *vetores linha*, pois, possuem apenas 1(uma) *linha* com várias colunas. Também pode-se obter *vetores coluna*, isto é, vetores com apenas 1(uma) *coluna* e várias linhas :

>> c=[1; 5; 6; 8; 10] ←

c =

1
5
6

```
        8
        10
```

Se um vetor passa a consistir de várias linhas e colunas, denominamo-lo de *Matrizes* :

```
>> m=[1  4  5  6;  5  10  3  20]        % Este comando cria uma matriz  2×4   ←

m=
        1       4       5       6
        5       10      3       20
```

Portanto, um vetor linha é um caso particular de uma matriz $1 \times N$, e um vetor coluna é um caso particular de matriz $N \times 1$.

9.1Operações Escalares com Matrizes no MatLab

Uma operação de adição, subtração, multiplicação e divisão de uma matriz com um valor escalar, é obtida simplesmente aplicado-se a respectiva operação em cada um dos elementos da matriz :

```
>> m=[1  4  5  6;  5  10  3  20]        % Matriz  2×4 ←

m=
        1       4       5       6
        5       10      3       20

>> m-2          ←

ans=

        -1      2       3       4
        3       8       1       18
```

No exemplo acima, de cada elemento da matriz *m* subtraiu-se o valor 2.

Outro exemplo :

```
>> 2*m-2        ←

ans=

     0     6     8    10
     8    18     4    38
```

Cada elemento da matriz foi multiplicado por 2 e, posteriormente, subtraiu-se 2.

```
y=(1:5)*0      % Cria um vetor linha com todos os elementos iguais a 0(zero)   ←

y=

     0     0     0     0     0
```

9.2Operações entre Matrizes no MatLab

As operações entre matrizes requer que as mesmas tenham as mesmas dimensões, e as operações de adição, subtração, multiplicação e divisão são aplicados *elemento-por-elemento* :

```
>> g=[1 2 3 4; 5 6 7 8; 9 10 11 12]    % g é uma matriz 3×4 ←

g=

     1     2     3     4
     5     6     7     8
     9    10    11    12


>> h=[1 1 1 1; 2 2 2 2; 3 3 3 3]        % h é  uma  outra matriz  3×4        ←

h=

     1     1     1     1
     2     2     2     2
     3     3     3     3
```

>> g+h % adiciona a matriz *g* com a matriz *h* ←

ans =

2	3	4	5
7	8	9	10
12	13	14	15

>> 2*g-h % Multiplica a matriz *g* por 2 e subtrai a matriz *h* do resultado ←

ans=

1	3	5	7
8	10	12	14
15	17	19	21

>> g.*h % Multiplica a matriz *g* pela matriz matriz *h* elemento por elemento ←

ans=

1	2	3	4
10	12	14	16
27	30	33	36

>> g./h % Divide a matriz *g* pela matriz matriz *h* elemento por elemento ←

ans=

1.0000	2.0000	3.0000	4.0000
2.5000	3.0000	3.5000	4.0000
3.0000	3.3333	3.6667	4.0000

>> g % Chama a matriz *g* novamente ←

g =

1	2	3	4
5	6	7	8
9	10	11	12

>> w=[1 2; 3 4; 5 6; 7 8] % Cria uma matriz 4×2 ←

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

>> g*w        %  Multiplicação da matriz g pela matriz w        ←

ans =

    50    60
   114   140
   178   220
```

Observe que a operação g.*h é diferente da operação g*h. Enquanto que a primeira executa uma mutiplicação elemento-por-elemento de duas matrizes, a segunda executa uma multiplicação entre duas matrizes.

10 Manipulação de Gráficos

Nesta secção mostrar-se-ão alguns comandos úteis para manipulação de gráficos. Pode-se adicionar curvas a um gráfico já plotado usando o comando hold.

```
% Exemplo de utilização do comando hold
%=======================================
% Geração da curva sin(x) :
%
%
x=linspace(0,2*pi,30);
y= sin(x);
plot(x,y,'r'); grid on
title('Exemplo de utilização do comando hold')
xlabel('x em rad/s')
ylabel('y=sin(x)')

%Agora mantém-se acurva e acrescenta-se a curva do cosseno
z= cos(x);
hold on
plot(x,z,'g');
ylabel('y1=sin(x); y2=cos(x)')
hold off
```

O resultado é apresentado na figura 6.

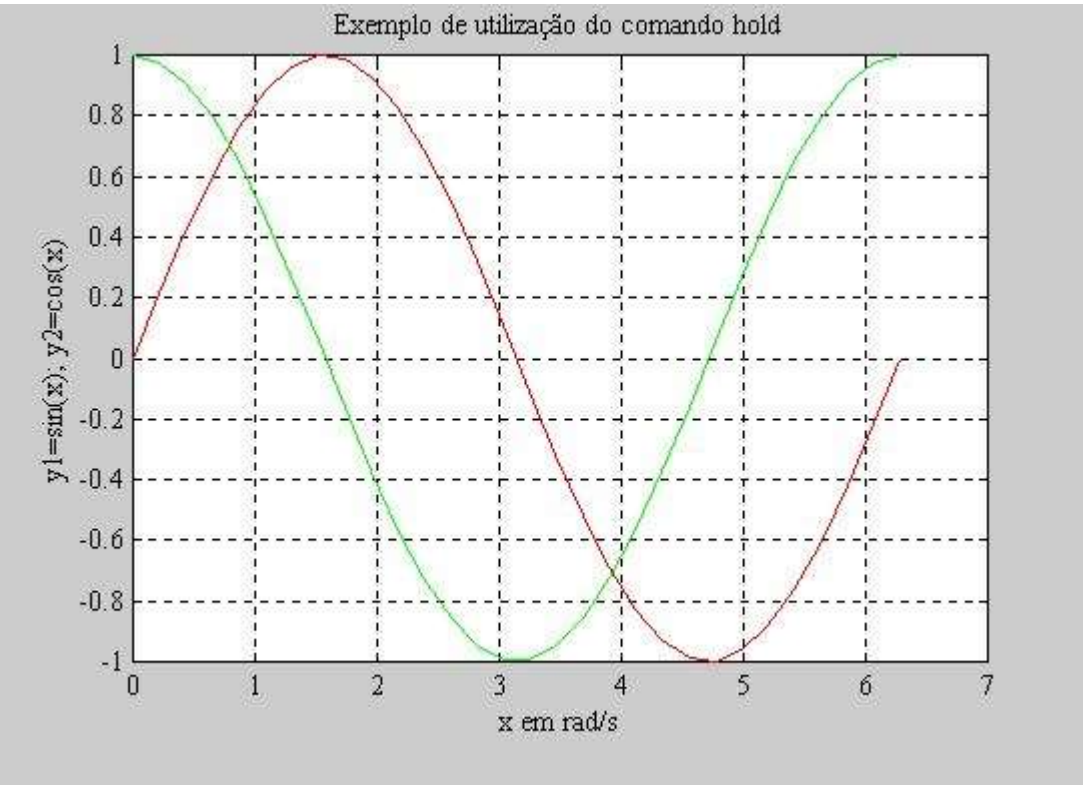


Figura 6: Exemplo do comando hold.

Pode-se utilizar o comando subplot(m,n,p) para subdividir a janela de figura em uma matriz m-por-n de áreas de plotagem e escolher a p-ésima área para ser ativa. O exemplo seguinte ilustra a utilização deste comando.

```
% Exemplo de utilização do comando subplot(m,n,p)
%=========================================

% Geração das curvas
%
x=linspace(0,2*pi,30);
y= sin(x);
z=cos(x);
a=2*sin(x).*cos(x);
b=sin(x)./(cos(x)+eps);          %o eps é uma variável especial do MatLab utilizada para
                                 %se  evitar divisão por zero
subplot(2,2,1); %ativa o subplot à esquerda superior dentre os 4 subplots
plot(x,y);
grid on
title('sin(x)')
ylabel('y=sin(x)')
```

```
%
subplot(2,2,2); %ativa o subplot à direita superior dentre os 4 subplots
plot(x,z);
grid on
title('cos(x)')
ylabel('y=cos(x)')
%
subplot(2,2,3); %ativa o subplot à esquerda inferior dentre os 4 subplots
plot(x,a);
grid on
title('2sin(x)cos(x)')
xlabel('x em rad/s')
ylabel('y=2*cos(x)*sin(x)')
%
subplot(2,2,4); %ativa o subplot à direita inferior dentre os 4 subplots
plot(x,b);
grid on
title('sin(x)/cos(x)')
xlabel('x em rad/s')
ylabel('y=sin(x)/cos(x)')
```

O resultado é apresentado na figura 7.

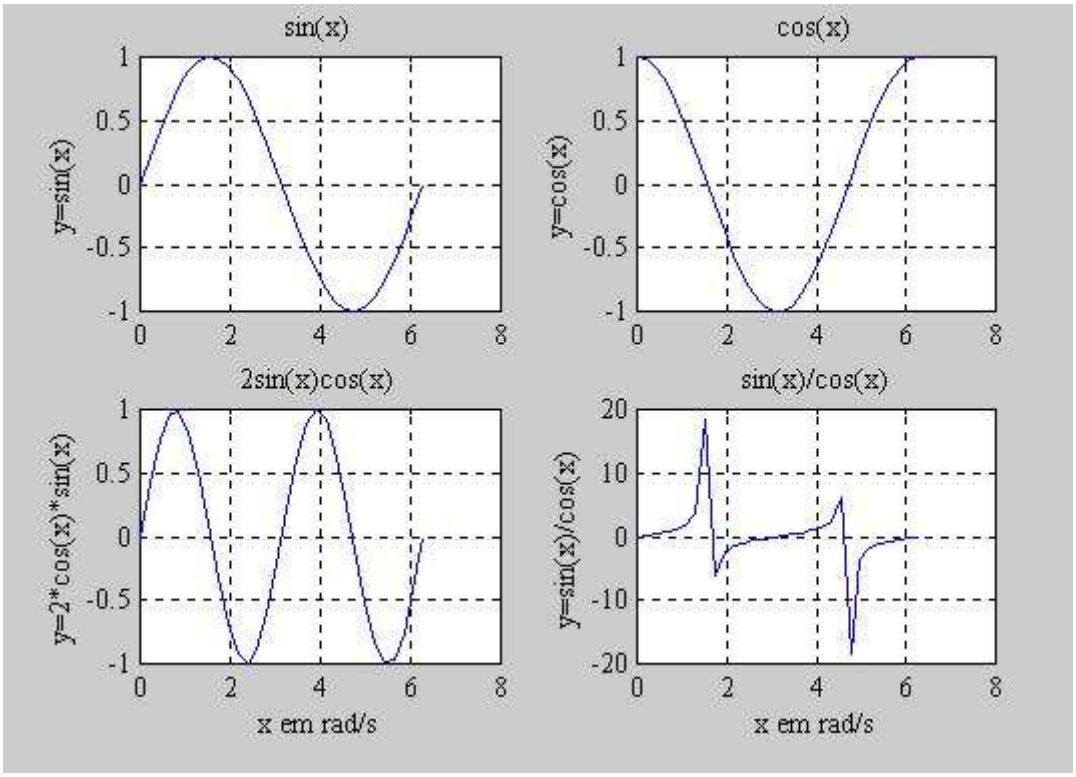


Figura 7: Utilização do comando subplot.

O uso do comando subplot(1,1,1) retorna ao modo default de utilização da janela de figura.

Duas outras formas úteis de se construir gráficos são utilizando-se os comandos hist e stem. O Comando hist(y) desenha um histograma com 10 bins para os dados de um vetor y. hist(y,n) cria um histograma com n bins. hist(y,x), onde x é um vetor cria um histograma usando os bins especificados no vetor x. O exemplo seguinte ilustra o uso do hist.

```
% Exemplo de gráficos usando hist
%=======================================
x=-2.9:0.2:2.9;          %gera um vetor com os bins a serem usados em um dos gráficos
y= randn(5000,1);        %gera 5000 números aleatórios
subplot(3,1,1);
hist(y);                 %gera um histograma com 10 bins
title('Histograma com 10 bins')
subplot(3,1,2);
hist(y,20);              %gera um histograma com 20 bins
title('Histograma com 20 bins')
subplot(3,1,3);
hist(y,x)                %gera um histograma com os bins dados por x
title('Histograma com bins dados por x')
```

O resultado é dado pela figura 8.

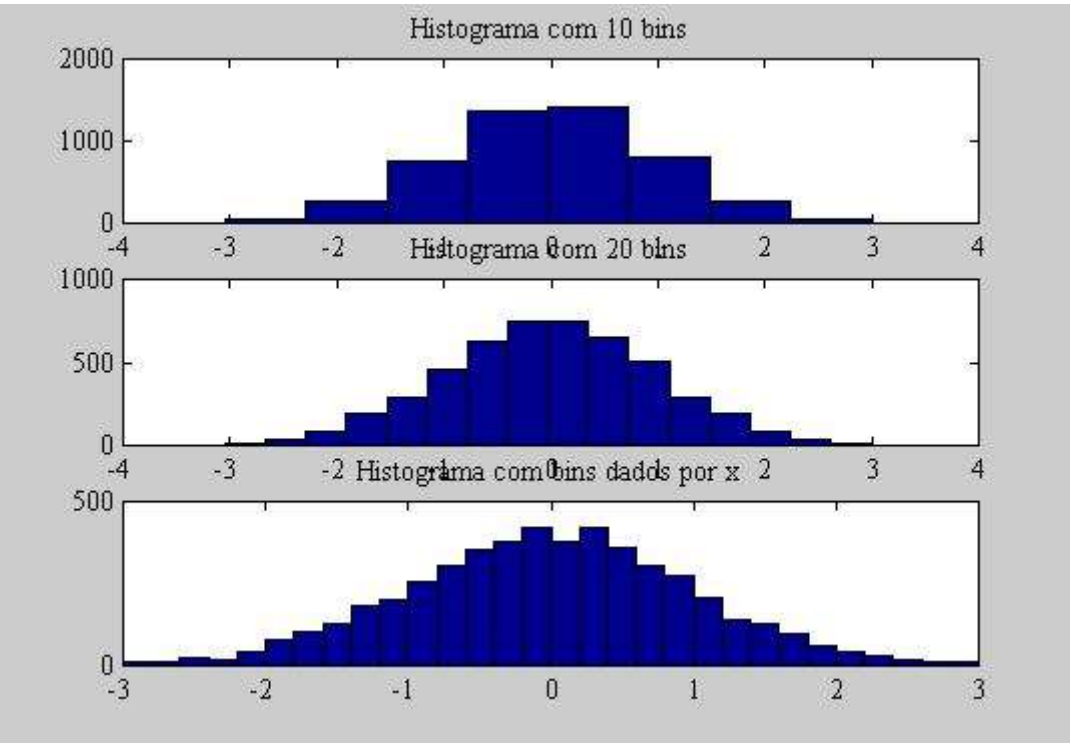


Figura 8: Exemplo de Histogramas.

Para a representação de seqüências discretas, é útil a plotagem de gráficos com stem. stem(y) plota os dados do vetor y. stem(x,y) plota os valores do vetor y dados por x. O exemplo seguinte ilustra o uso do stem.

```
% Exemplo de gráficos usando stem
%=====================================
x=1:50;                 %gera um vetor com os pontos do vetor y
y= randn(50,1);         %gera 50 números aleatórios
subplot(1,2,1);
stem(y);                %gera um gráfico com os 50 pontos do vetor y
title('stem(y)')
subplot(1,2,2);
stem(x,y,':');          %gera um gráficos com os pontos do vetor y dados por x
title('stem(x,y)')
```

O resultado é dado pela figura 9.

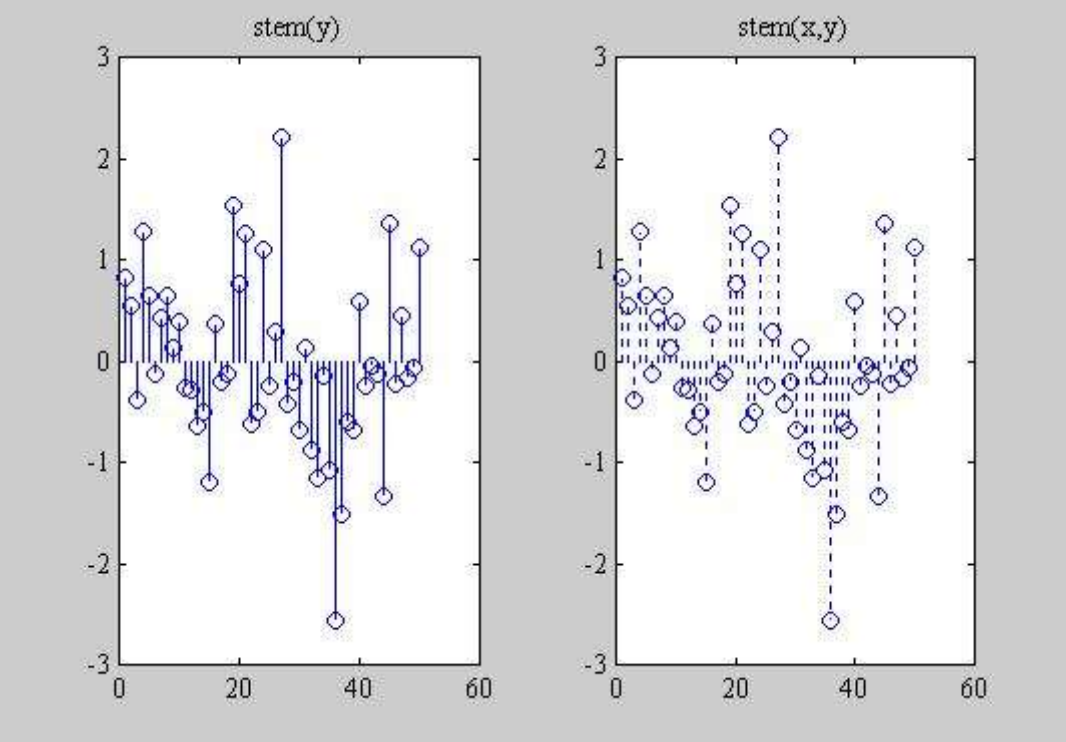


Figura 9: Exemplo de construção de gráficos com stem.

10.1 Estilo de Linhas e Cores:

Nos exemplos anteriores, utilizamos diferentes estilos de linhas e cores. Os estilos de linhas e as cores podem ser especificados nos comandos de plotagem como um argumento do tipo caracter string (entre apóstrofes – por exemplo, ´c´), consistindo de 1,2 ou 3 caracteres. A tabela 11 apresenta estes caracteres.

Tabela 11: Estilos de linhas e cores

Símbolo	Cor	Símbolo	Estilo de Linha
y	amarelo	.	pontos
m	magenta	o	círculos
c	cian	x	marcas x
r	vermelho	+	marcas mais
g	verde	*	marcas asterístico
b	azul	-	linha sólida
w	branco	:	linha pontilhada
k	preto	-.	linha com traço e ponto
		--	linha com segmentos de traço

11 EXEMPLOS DE APLICAÇÃO DE VETORES NA VISUALIZAÇÃO DE SINAIS REPRESENTADOS NO DOMÍNIO DO TEMPO

Exemplo 1 : Escreva um M-file que visualize o sinal f(t) representada no domínio do tempo por :

$$f(t) = \begin{cases} 1 & (0 < t \leq \pi) \\ -1 & (\pi < t \leq 2\pi \end{cases}$$

Resolução : Na seção I.3.6, esta mesma função foi visualizada através da utilização da função sign(x). Neste exemplo, será utilizado somente o conceito de vetores :

```
%
x=linspace(0,2*pi,100)        % Criou-se 100 amostras entre 0 e 2*pi
%
f=(1:50)*0+1                  % Faz  f(t) = 1 para 0 < t  <= pi , i.e., as
%                             primeiras 50 amostras de f(t) são
%                             iguais a 0
f(51:100)=(1:50)*0 - 1        % Faz  f(t) = -1 para pi < t <=2*pi, i.e., as
%                                últimas 50 amostras de f(t) são
%                                iguais a 1
plot(x,f, 'ko'); grid on
title('Figura II.3: Função retangular')
xlabel('t em radianos')
ylabel('f(t)')
```

Obter-se-á o resultado mostrado na figura II.3

Exemplo 2 : Escreva um M-file que visualize o sinal f(t) representada no domínio do tempo por :

$$f(t) = \begin{cases} 0 & t \leq 0 \\ t & 0 < t \leq 1 \\ -t/2 + 3/2 & 1 < t \leq 3 \\ 0 & t > 3 \end{cases}$$

Resolução :

```
%
t=linspace(0,5,51);                  % Cria 51 amostras entre 0 e 5
%                                       espaçados de 0.1
```

```
f(1:11)=t(1:11);                    % f(t) = t            para 0<t<=1
f(12:31)=-t(12:31)/2 + 3/2;         % f(t)=-t/2 + 3/2 para 1<t<=3
f(32:51)=(32:51)*0;                         f(t)=0              para t>3
%
plot(t,f, 'ko'); grid on
title('Figura II.4 : Função Triangular')
xlabel ('Tempo')
ylabel('f(t)')
```

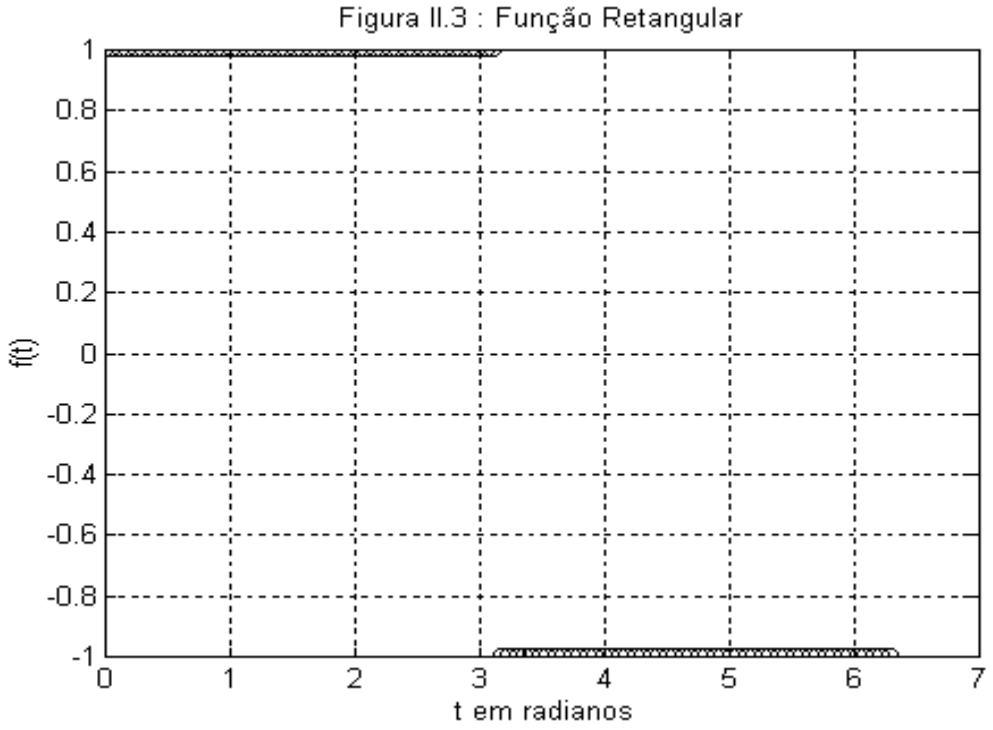


Figura 2.3 - Função retangular

Exercício : Faça as modificações necessárias no M-file do exemplo 2 de modo a visualizar 101 amostras da função f(t) no intervalo de 0 a 5.

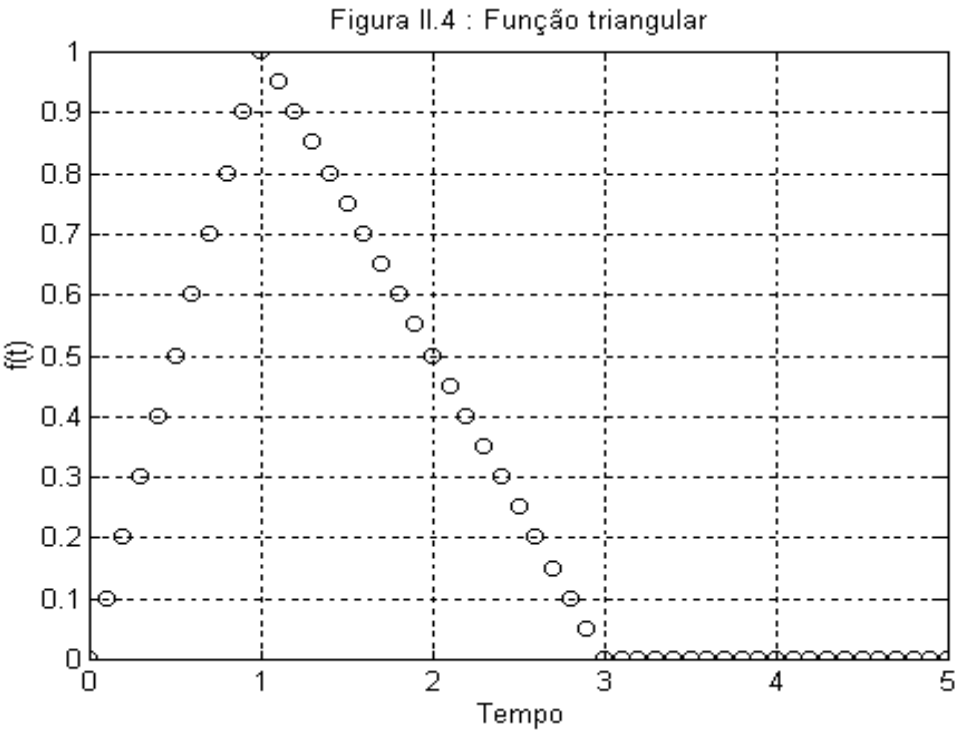


Figura 2.4 - Função triangular