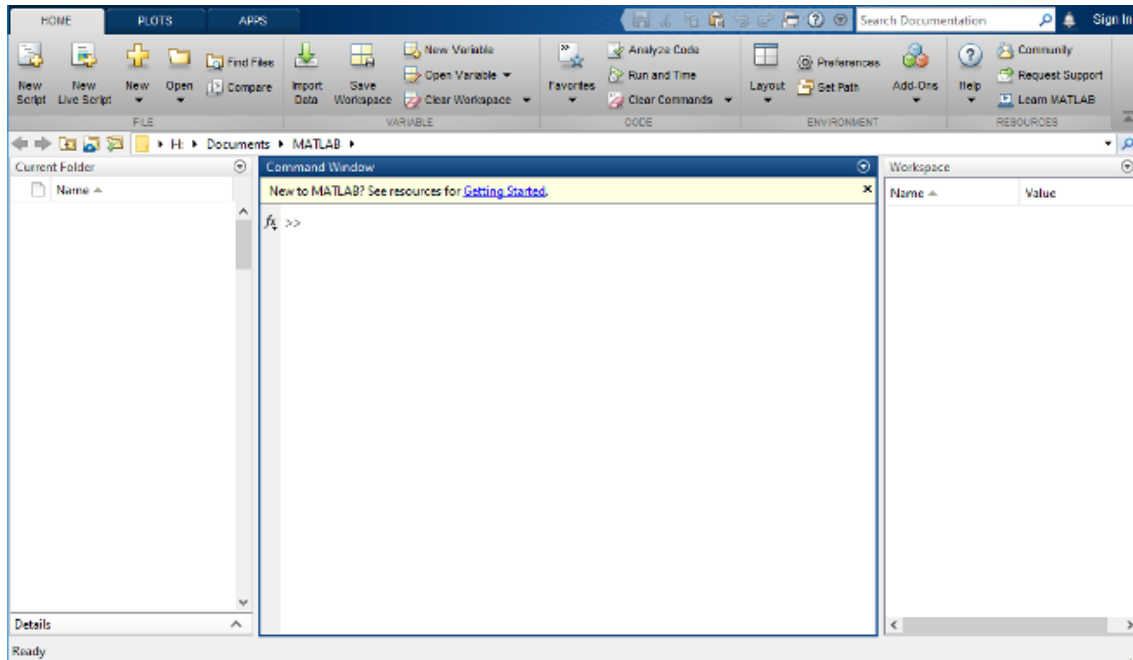


Breve Revisão de Matlab

Ambiente

Janela do Matlab.



Componentes importantes:

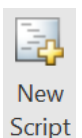
Janela de comandos (**Command Window**). Onde se introduzem comandos. O *prompt* para comandos é: “>>”. A janela de comandos deve estar sempre visível. Na janela de comandos podemos fazer operações, por exemplo `2*5+3`, `cos(pi)` ou `plot(1:3)`.

Command Window



Janela das variáveis (**Workspace**). Deve ser consultada com frequência para saber o valor ou outras propriedades das variáveis criadas. Acrescentar as colunas **size** e **class** para termos mais informação das variáveis além do seu nome e do seu valor.

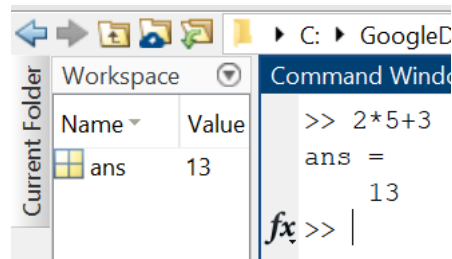
Workspace

Editor. Onde se escrevem, avaliam e se experimentam programas e funções. Evocar o editor com o botão **New Script** ou escrevendo na janela de comandos: `edit`.



>> edit

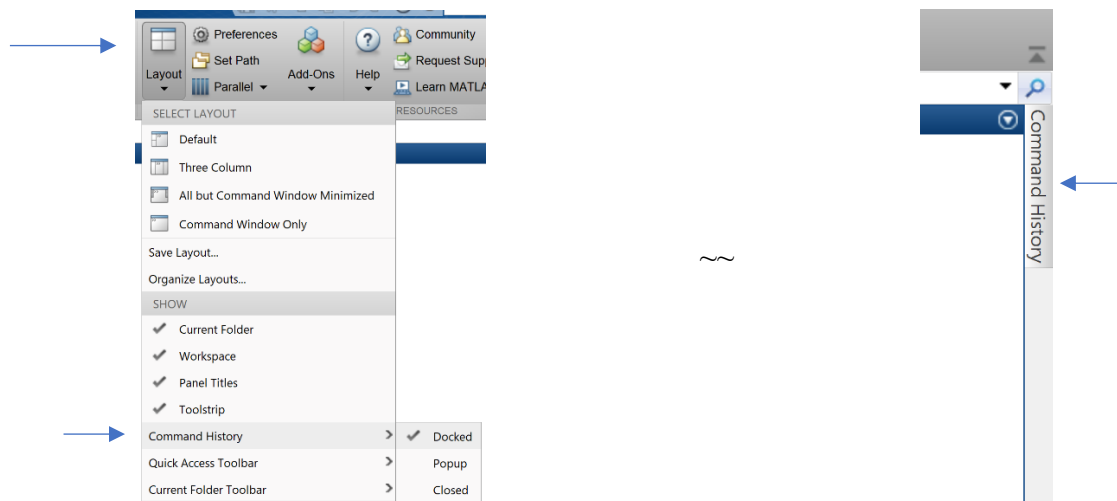
Diretoria corrente (**Current Folder**). Indica a diretoria onde se guardam os “scripts” e funções criadas com o editor. Pode usar o comando `pwd` para saber qual é. Pode encostar a janela à esquerda (com  e ) e usar esta componente apenas quando precisa.



Janela de Documentação. Ajuda em forma de browser. Usar botão [Help](#) ou escrever na janela de comandos: `doc`. Seguir os tutoriais a partir de: [Matlab::Getting Started](#).



Janela do historial de comandos anteriores ([Command History](#)). Pode criar esta janela a partir do botão [Layout](#):



E pode também encostá-la à direita (resultado na figura da direita).

Documento de referência

O Matlab fornece um conjunto de documentos de referência.

Pode obter online os documentos de ajuda em PDF através da janela de ajuda (`doc`). Clique em “[Explore MATLAB](#)” e no lado direito em “[PDF Documentation](#)”.

Matlab por temas

- 1) Variáveis.** O Matlab é uma ferramenta em que todas as variáveis são arrays multidimensionais. Uma variável pode ser um escalar (dimensão 1×1), um vetor (dimensão $1 \times N$ ou $M \times 1$), uma matriz (dimensão $M \times N$) ou um array com mais de duas dimensões. O comando `size` permite conhecer essas dimensões.

Exemplo: `z=zeros(2,3,4)`¹ cria um array (double) de dimensão $2 \times 3 \times 4$ com 4 matrizes de dimensão 2×3 : `size(z)`:

```
>> size(z)
ans =
     2     3     4
```

O carácter «'» faz a transposição (e conjugação) de uma matriz. Exemplo: `a=[1,2,3]'`

```
>> a=[1,2,3]
a =
     1
     2
     3
```

Existe uma variável especial que tem o nome `ans`: é a resposta a uma avaliação, por exemplo: `pi` ou `size(pi)` ou `2*5+3`.

- 2) Ajuda.** Usar `help <comando>` para saber o que esse comando faz, por exemplo:

```
help whos
```

Ou então `doc whos` que mostra a ajuda num novo *tab* da janela de documentação.

- 3) Indexação.** Os arrays numéricos podem ser criados com `[]` mas são indexados com `()`. Espaço ou vírgula (,) separa colunas, e ponto-e-vírgula (;) ou (enter) separa linhas.

Por exemplo: `A=[1,3;4 2]`

```
A =
     1     3
     4     2
```

A é uma matriz de dimensão 2×2 . `A(2)` é o 2º elemento da matriz (vista como um vetor coluna de colunas concatenadas) e vale 4. O Matlab armazena matrizes em memória concatenando colunas. Verificar que `a=A(:)` resulta em

```
>> a=A(:)
a =
     1
     4
     3
     2
```

As variáveis podem ser indexadas com índices inteiros a começar em 1 (o 1º elemento tem índice 1 e não 0), ou por arrays lógicos. Por exemplo:

`a=[0,1,2,3]`, `a(4)`, `a>0`, `a(a>0)` (4 instruções resultam em 4 respostas):

```
>> a=[0,3,4,2], a(4), a>0, a(a>0)
a =
     0     3     4     2
ans =
```

¹ O texto a cinzento (ex: `2*5+3`) identifica código que pode correr na janela de comandos.

```

      2
ans =
      1×4 logical array
      0      1      1      1
ans =
      3      4      2

```

O comando `a(0)` produz um erro, devido à indexação com 0 (texto a vermelho):

```
>> a(0)
Array indices must be positive integers or logical values.
```

Com `a=[0,1,2,3]`; `c=a==2`, `a(c)` resulta:

```
>> a=[0,1,2,3]; c=a==2; a(c)
c =
      1×4 logical array
      0      0      1      0
ans =
      2

```

- 4) Inicialização.** Um comando usual para definir (ou inicializar) uma variável numérica é o comando “zeros”. Ver `help zeros`. Exemplo: `a=zeros(3,4)`

```
>> a=zeros(3,4)
a =
      0      0      0      0
      0      0      0      0
      0      0      0      0

```

- 5) Operações básicas.** A operação de multiplicação (*) aplicada a matrizes é multiplicação matricial! Se queremos aplicar este operador a cada um dos elementos de dois arrays (de dimensões compatíveis) devemos usar “.*”. O mesmo para exponenciação (.^) e divisão (./). A soma (+) implica matrizes da mesma dimensão ou de dimensões compatíveis. Pesquise “[Compatible Array Sizes for Basic Operations](#)” e “[Array vs. Matrix Operations](#)”.

Exemplo: `n=[0,1,2,3,4]`, `2.^n`

```
>> n=[0,1,2,3,4]; 2.^n
n =
      0      1      2      3      4
ans =
      1      2      4      8     16

```

Um produto escalar é a multiplicação de um vetor linha por um vetor coluna.

Exemplo: `a=[2,3,4]`, `b=[5,6,7]'`, `a*b`

```
>> a=[2,3,4]; b=[5,6,7]'; a*b
a =
      2      3      4
b =
      5
      6
      7
ans =
     56

```

Mas `b*a` é uma matriz de dimensão 3×3:

```
>> b*a
ans =
    10    15    20
    12    18    24
    14    21    28
```

E $a.*b$ (porque têm dimensões compatíveis) é

```
>> a.*b
ans =
    10    15    20
    12    18    24
    14    21    28
```

Equivale à multiplicação ponto-a-ponto de duas matrizes repetindo 3 vezes a linha de a e 3 vezes a coluna de b . Equivale ainda a fazer explicitamente esta repetição:

```
>> repmat(a,3,1), repmat(b,1,3), repmat(a,3,1).*repmat(b,1,3)
ans =
     2     3     4
     2     3     4
     2     3     4
ans =
     5     5     5
     6     6     6
     7     7     7
ans =
    10    15    20
    12    18    24
    14    21    28
```

Uso de $*$ ou $.*$ (erros frequentes):

a) Multiplicação com escalar: é indiferente usar $*$ ou $.*$. O mesmo para divisão, soma ou subtração. Exemplo:

```
x=[-inf,-1,0,1,+inf];
f=2*x-1
```

b) Multiplicação ponto-a-ponto (vetorial): é obrigatório usar $.*$.

Exemplo: $f(x) = \frac{1}{1+e^{-x}} + x^2 + 1$

```
f = 1./(1+exp(-x))+x.^2+1 %nota: exp(x) é vetorial.
%Necessário usar "./" e ".*"
```

c) Multiplicação matricial: usar $*$.

Exemplo: distribuição normal multivariável de média $\mathbf{0}$: $f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \times |\mathbf{C}|}} e^{-\frac{1}{2}\mathbf{x}^T \mathbf{C}^{-1} \mathbf{x}}$

onde N é a dimensão de \mathbf{x} ; \mathbf{C} é a matriz de covariância e $|\mathbf{C}|=\det(\mathbf{C})$:

```
x=[-1;1], C=[3,2;2,4], N=size(x,1) %N=2
f = 1/sqrt((2*pi)^N*det(C))*exp(-0.5*x'*inv(C)*x)
%Nota: 2*pi é escalar, assim como x'*inv(C)*x, 0.5 e det(C).
```



- 6) **Supressão de resultados.** Use “;” no fim de um comando de forma a suprimir o resultado na janela de comandos. Útil principalmente para variáveis de dimensões elevadas. Exemplo: comparar `a=zeros(1,1000)` (mostra 1000 zeros na janela de comandos=, com `a=zeros(1,1000);`:

```
>> a=zeros(1,1000);
>>
```

Use “,” para separar comandos sem supressão do resultado (como na maior parte dos exemplos indicados anteriormente).

- 7) **Classes de variáveis.** As variáveis podem ser de várias classes. As classes mais importantes são as seguintes:

- **double:** para reais de precisão dupla, codificados com 8 bytes por valor. Se a variável for complexa, tem um par de valores: a parte real e a parte imaginária, mas a classe é double. Exemplo: `a=1+2j`, `class(a)`, `whos`
- **char:** para caracteres. Um array de caracteres (string), é criado com ‘’. Exemplo: `s= 'abc0'`, `class(s)`
- **logic:** para valores lógicos: 1=verdade; 0=falso (podem ser convertidos implicitamente ou explicitamente a doubles). Arrays lógicos resultam da avaliação de instruções lógicas, por exemplo, `a=2,z=a==0,class(z)`. O operador “not” é “~”, por exemplo, `~0` é `true`. As variáveis podem ser indexadas com arrays lógicos.
- **cell:** variável onde cada elemento pode conter qualquer outra variável de qualquer dimensão. Os arrays de células são criados e indexados com `{}`. Exemplo: `a={'abc', 2, 'd', [1,2;3,4], true},a{4},a{5}`
Usual para armazenar *strings* de comprimentos diferentes: *cell array of strings*.

Existem muitas outras classes. Ver `help class`. Class pode também ser um comando em que o argumento é uma variável e cujo resultado é uma *string* com o nome da classe correspondente. Exemplo: `c=class(2),class(c)`

```
c =
'double'
ans =
'char'
```

- 8) **Carateres especiais em comandos.**

Pontuação: ver `help punct`.

Parenteses: ver `help paren`.

Dois pontos: ver `help colon`.

Exemplo: `a=0:4` são os 5 inteiros de 0 a 4 (mas o vetor gerado é da classe double)

```
>> a=0:4
a =
    0     1     2     3     4
```

Quando o passo é 1, este pode ser subentendido. Mas: `a=3:2:10` são os inteiros a começar em 3 com passo 2 e a terminar em 10; `a=10:-1.5:2` são os números de passo -1.5 a começar em 10 e a terminar em (ou antes de) 2:

```
>> a=10:-1.5:2
a =
 10.0000   8.5000   7.0000   5.5000   4.0000   2.5000
```

$A(:)$ são os elementos de A indexados com um só índice (coluna).

9) Ciclos e instruções condicionais.

Veja [Matlab::Programming Scripts and Functions::Control Flow](#) para se inteirar da sintaxe das instruções condicionais e dos ciclos.

Exemplo simples de ciclo: for: `for i=1:3, j=i+3, end`

```
>> for i=1:3, j=i+3, end
```

```
j =
```

```
4
```

```
j =
```

```
5
```

```
j =
```

```
6
```

Exemplo de if:

```
a=1; c=1;
```

```
if a==1
```

```
    c=10;
```

```
elseif a==0
```

```
    c=0;
```

```
else
```

```
    c=-1;
```

```
end
```

```
c
```

10) Precisão finita. O Matlab é um ambiente numérico de vírgula flutuante com precisão finita: cerca de 15 algarismos significativos com codificação “double” (formato IEEE 754)². Os cálculos encadeados propagam erros. Na comparação numérica de resultados temos de ter em conta esta precisão finita.

Por exemplo, $3*(4/3-1)$ deveria ser 1, mas devido à precisão finita e devido a $4/3$ não ter representação exata em base 2, existe uma diferença a partir da 16ª casa decimal: 0.9999999999999997779. Devemos entender o resultado de $1-3*(4/3-1)$ como sendo zero (uma vez que 15 casas decimais estão certas). A diferença para 1 é:

```
>> 1-3*(4/3-1)
```

```
ans =
```

```
2.2204e-16
```

Ver doc `eps` e/ou pesquise [Floating-Point Numbers](#).

11) Comentários. Podemos comentar as instruções colocando o carácter ‘%’ na linha comentada (aparece com cor verde no editor; vale apenas para essa linha). Exemplo:

```
1-3*(4/3-1) %esta instrução não resulta exatamente em zero,  
% mas sim em 2^(-52), pois a mantissa de “double” tem 52 bits.  
% 4/3 não tem representação binária exata: 1,01010101010101...,  
% repetindo-se na mantissa o padrão 0101 infinitamente.
```

² ver: https://en.wikipedia.org/wiki/IEEE_754

% O resultado aproximado é:

% $2^0 + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + 2^{-12} + 2^{-14} + 2^{-16}$

No editor do Matlab, no início de uma linha, "%% " marca o início de nova secção.

Os caracteres "%{" e "%}" marcam comentários em bloco (como "/*" e "*/" em C).

12) Editor. É muitas vezes conveniente usar o editor do Matlab para escrever “scripts” (conjunto de comandos num ficheiro de texto) e funções (comandos definidos em ficheiro). Mas devemos estar sempre atentos à janela de comandos onde os resultados e os erros de sintaxe são indicados.

Ver [Matlab:: Programming Scripts and Functions](#).

Nos trabalhos práticos não use “scripts” separados para alíneas separadas das folhas de exercícios! É preferível usar o conceito de “secções” num único “script”: use “%% ” no começo de linha para separar secções. Ver [Matlab::Programming Scripts and Functions::Run Code Sections](#). As secções podem ser avaliadas isoladamente. Os programas podem ser depurados (*debugged*), avaliando secções ou instruções, uma a uma, ou colocando pontos de paragem (*breakpoints*) no código.

Pode também (de preferência!) seleccionar um conjunto de instruções a avaliar, no editor ou na janela de comandos, e avaliá-los (botão direito seguido de [Evaluate Selection](#) (F9 em Windows)). No exemplo seguinte a tecla F9 avalia $a*b$ de novo.

<pre>>> a=[2,3,4],b=[5,6,7]', a*b</pre>	
a =	
2	3
4	
b =	
5	
6	
7	
ans =	
56	

Evaluate Selection	F9
Open Selection	Ctrl+D
Help on Selection	F1
Function Browser	Shift+F1
Show Function Browser Button	
Function Hints	Ctrl+F1

13) Funções anónimas.

Podemos usar o conceito de funções anónimas para descrever sinais em tempo discreto. Por exemplo, para descrever no Matlab um pulso retangular de $n=-3$ até $n=1$,

$$x[n] = u[n+3] - u[n-2]$$

podemos fazer:

```
pulso = @(n)(n>=-3 & n<2)*1; %função sem nome de um só argumento: n.
```

```
%O carater @ identifica que "pulso" é uma variável da classe
```

```
%"function_handle".
```

```
%Uso:
```

```
n=-10:10; %definição do vetor n
```

```
x=pulso(n); %chamada da função com o vetor n
```

```
stem(n,x) %gráfico para sinais em tempo discreto
```

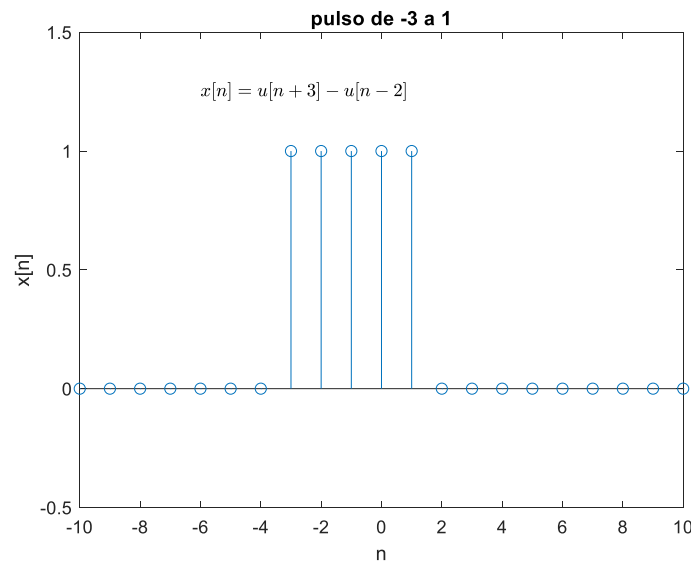
```
axis([-10,10,-0.5,1.5]) %altera os limites x,y do gráfico (decoração)
```


A multiplicação por 1 na 1ª linha de código serve para converter a classe do vetor resultado de lógica (`logical`) para real (`double`). Veja a diferença retirando “*1” do código: variável `x` fica da classe '`logical`' (figura seguinte).

Workspace		Current Folder	
Name	Value	Size	Class
n	1x21 double	1x21	double
pulso	@(n)(n>=-3&n<2)	1x1	function_handle
x	1x21 logical	1x21	logical

Nas figuras podemos colocar textos em abcissa, ordenada e título, além de podermos colocar texto nos eixos a começar em pontos particulares. O texto pode ser interpretado como sendo TeX (por omissão) ou LaTeX³. Por exemplo:

```
xlabel('n'); ylabel('x[n]'); title('pulso de -3 a 1');
text(-6,1.25,'$x[n]=u[n+3]-u[n-2]$', 'interpreter', 'latex')
```



Imaginemos agora o sinal

$$y[n] = r[n+3] - 2r[n-1] + r[n-5]$$

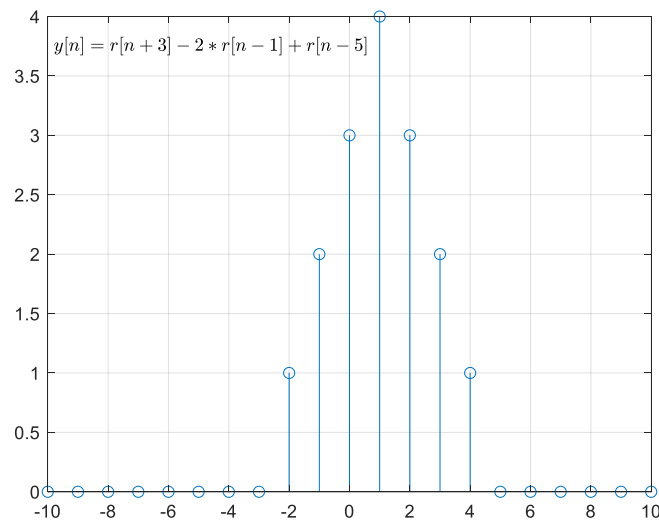
onde $r[n]$ é uma rampa:

$$r[n] = \begin{cases} n, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

Podemos ver que se trata de um pulso triangular usando uma função anónima:

```
rampa=@(n) (n>=0).*n; %rampa=0 para n<0 e rampa=n para n>=0
n=-10:10;
y=rampa(n+3)-2*rampa(n-1)+rampa(n-5);
stem(n,y); grid;
text(-9.8,3.75,'$y[n]=r[n+3]-2*r[n-1]+r[n-5]$', 'interpreter', 'latex')
```

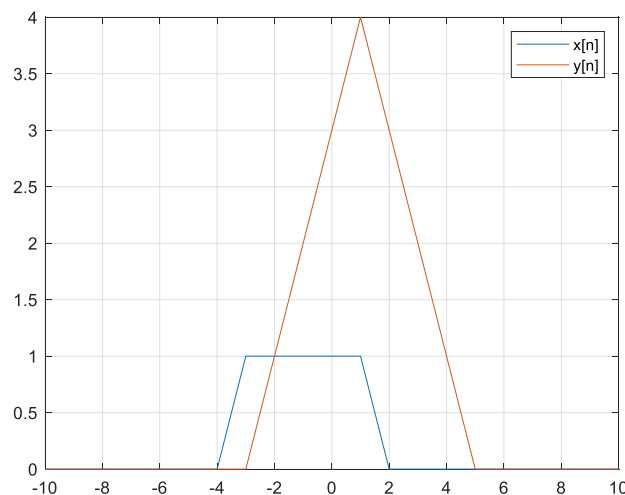
³ <https://www.latex-project.org/>



14) plot versus stem.

Para sinais de grande dimensão, muitas vezes é preferível usar “plot” em vez de “stem” (uma vez que o gráfico ficaria demasiado sobrecarregado com linhas e bolas). Em Matlab os gráficos são sempre definidos com pontos discretos, mas o comando “plot” traça segmentos de reta entre pontos, o que dá a aparência de um sinal em tempo contínuo. Quando queremos ver duas funções num só gráfico, a cor das linhas muda, e podemos identificar as linhas usando uma legenda. Para o caso dos dois sinais anteriores, podemos fazer:

```
plot(n,x,n,y),grid %gráfico de duas linhas
legend({'x[n]', 'y[n]'}) %legenda para identificar os sinais
```



De salientar que é necessário interpretar este gráfico como sendo equivalente aos dois gráficos anteriores.

Podemos também repetir os dois gráficos, sobrepondo pontos nas linhas:

Processamento Audiovisual – LCED – 2º Semestre – Revisão Matlab

```
plot(n,x,n,y,n,x,'.',n,y,'. '),grid %gráfico com pontos sobrepostos  
legend({'x[n]', 'y[n]', 'x[n]', 'y[n]'})
```

