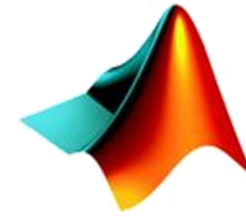




TÉCNICO
LISBOA



MATLAB

WS: INTRODUÇÃO AO MATLAB

Ricardo Trindade
Nuno Matias

2017

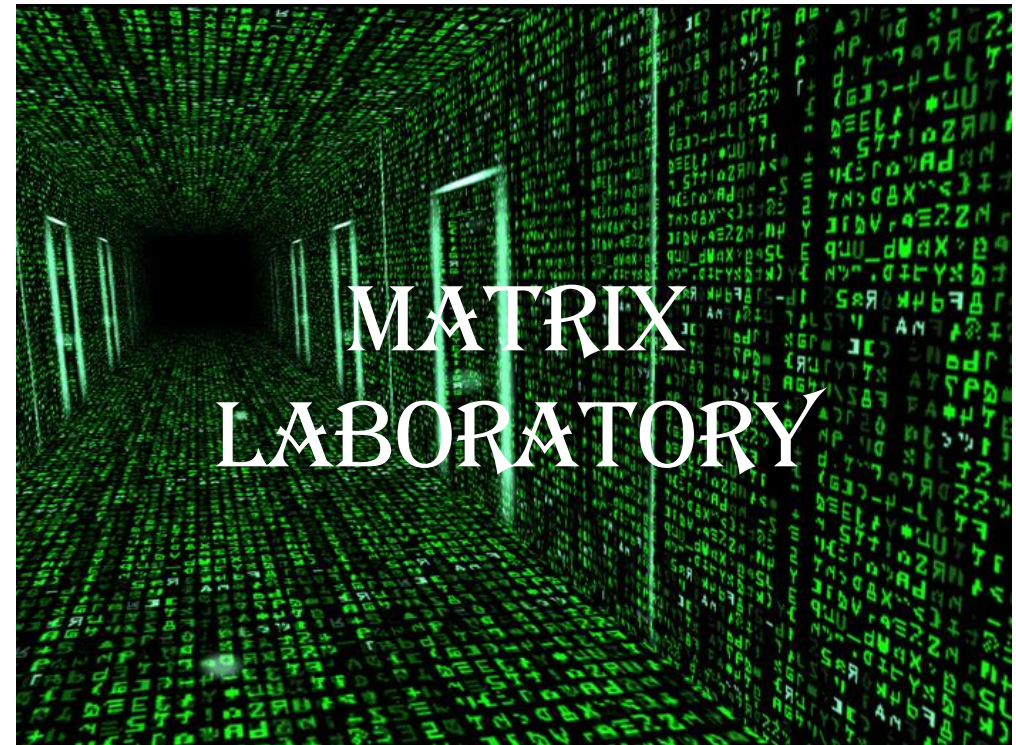
MATLAB? O que é isso?

- MATLAB = **MA**Trix **LAB**oratory
- Ferramenta computacional simples e intuitiva que se destaca das outras linguagens pela facilidade com que manuseia matrizes
- “Matrizes? Álgebra? Não sei se isso é bom sinal...”
- Não tenham medo! É mesmo simples e tem aplicabilidades em inúmeras áreas porque a informação normalmente está estruturada em matrizes! E essa informação pode ser, por exemplo, o teu sinal de ECG ou uma imagem de ressonância magnética do teu cérebro!!
- Vá, vamos lá ser campeões de MATLAB!



Conteúdos

- Janela do Matlab
- Matemática Básica
- Vetores
- Matrizes
- Representações Gráficas
- Branching
- Loops
- Funções



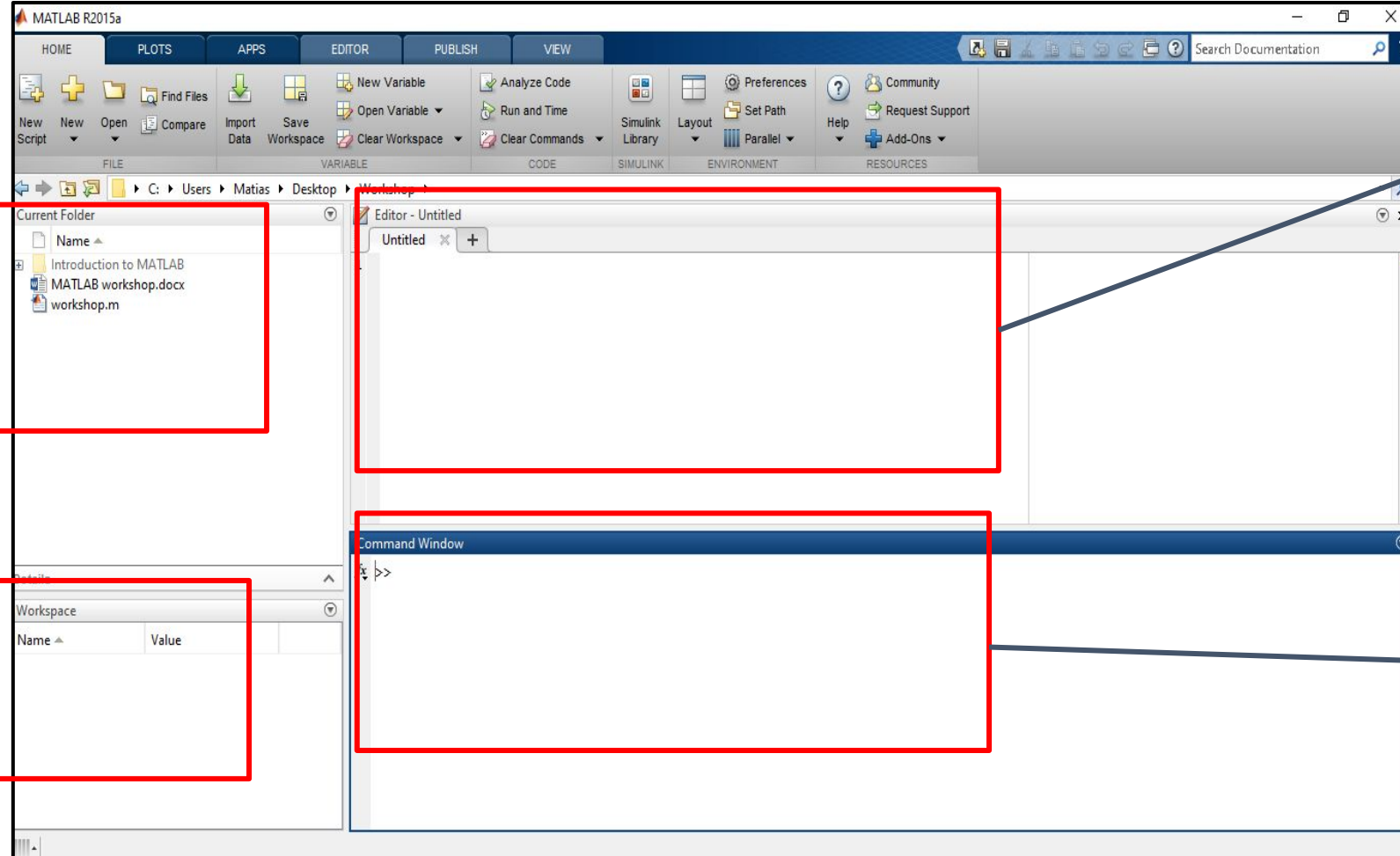
Janela do MATLAB

CURRENT FOLDER

Indica o directório onde se **encontram** e onde o MATLAB vai buscar funções ou variáveis que tenham guardadas. É importante que tudo o que querem usar esteja no directório ou então terão alguns erros!

WORKSPACE

Aqui ficam registadas as vossas variáveis. É extremamente útil pois podem ver o conteúdo das variáveis para confirmar se os valores são os que vocês querem.



EDITOR

É aqui que vão escrever os vossos scripts e funções para poderem guardar e usar mais tarde. Basicamente é aqui que vão passar o tempo a fazer o vossos projectos!

COMMAND WINDOW

Aqui executam comandos rápidos e simples. Atenção que o que fazem aqui, apesar de ficar registado na história do MATLAB, não dá para guardar em ficheiros.

Matemática Básica

| Símbolo | Operação |
|---------|---------------|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| ^ | Potenciação |

| Símbolo | Significado |
|---------|-----------------|
| pi | Número pi |
| i | Número complexo |

| Função | Significado |
|---------|----------------------------|
| sqrt | Raiz Quadrada |
| nthroot | N raiz de um número real |
| abs | Valor absoluto |
| exp | Exponencial |
| log | Logaritmo Natural (Base E) |
| log10 | Logaritmo Comum (Base 10) |
| sin | Seno (Radianos) |
| sind | Seno (Graus) |

Matemática Básica - É a tua vez!

Calcula as seguintes expressões:

$$\frac{7}{12} + (2 * 6^3) = ?$$

$$\sqrt{\frac{e^{i\frac{\pi}{2}}}{|3 + 4i|}} = ?$$

$$\sqrt[5]{\ln(2 * \sin \frac{\pi}{3})} = ?$$



HINT: Se tiveres problemas em usar alguma função experimenta a ajuda! Usa o *help* ou o *doc* seguido do nome da função (exemplo: *help sqrt*)

Matemática Básica - É a tua vez!

Conseguiste?

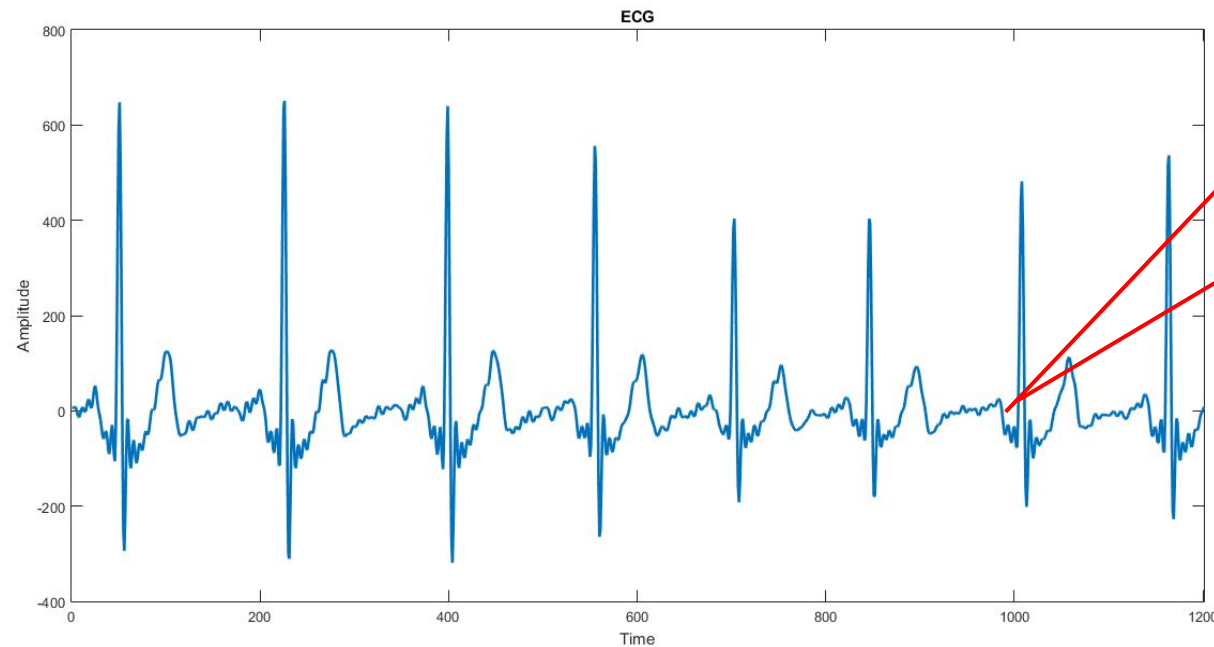
$$\frac{7}{12} + (2 * 6^3) = 432.5833$$

$$\sqrt{\frac{e^{i\frac{\pi}{2}}}{|3 + 4i|}} = 0.3162 + 0.3162i$$

$$\sqrt[5]{\ln(2 * \sin \frac{\pi}{3})} = 0.8871$$

Vectores

- Vector pode ser visto como uma matriz cuja uma das dimensões é igual a 1
- Na prática, vectores correspondem por exemplo a sinais fisiológicos ou a sinais sonoros! Olha o exemplo do ECG que corresponde a uma série finita de pontos num dado intervalo de tempo



| 209 | 210 | 211 | 212 | 213 | 214 | 215 |
|----------|----------|----------|----------|----------|----------|----------|
| -66.7477 | -55.7634 | -42.7617 | -43.2640 | -61.1532 | -82.9830 | -88.3194 |

Vectores - Como criar?

| Exemplo | Resultado | Explicação |
|----------------------|---|---|
| [1 2 3] | 1 2 3 | Vector Linha |
| [1; 2; 3] = [1 2 3]' | $\begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$ | Vector Coluna (notar a utilização do “ ; ” e do “ ' ” que devolve a transposta) |
| 1 : 10 | 1 2 3 4 5 6 7 8 9 10 | Sequência de números de um ponto a outro |
| 1 : 2 : 10 | 1 3 5 7 9 | Sequência de números dois a dois de um ponto até outro |
| 10 : -2 : 1 | 10 8 6 4 2 | Sequência de números de menos dois a menos dois de um ponto até outro |
| linspace(1, 10, 4) | 1 4 7 10 | Função que gera X número de pontos entre duas extremidades |

Vectores - Operações

| Vector - Escalar | |
|------------------|---------------|
| Símbolo | Operação |
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| .^ | Potenciação |

Exemplo:
 $[1 \ 2 \ 3] * 2 = [2 \ 4 \ 6]$

| Vector - Vector | |
|-----------------|---|
| Símbolo | Operação |
| + | Adição |
| - | Subtração |
| * | Produto Interno / Multiplicação de Matrizes |
| .* | Multiplicação de elementos |
| ./ | Divisão de elementos |
| .^ | Potenciação de elementos |

Exemplo:
 $[1 \ 2 \ 3] .* [4 \ 5 \ 6] = [4 \ 10 \ 18]$

Vectores - É a tua vez!

Descobre que vectores ou escalares tens de usar para chegar ao resultado final:

$$[12 \ 15 \ 9] / ? = [4 \ 5 \ 3]$$

$$[1 \ 2 \ 3] .* ? = [1 \ 4 \ 9]$$

$$[10 \ 2 \ 36] ./ ? = [2 \ 2 \ 2]$$

$$[1 \ 2 \ 3] * ? = 14$$

$$[3 \ 4 \ 5] .^ ? = [9 \ 16 \ 25]$$

$$? * [1 \ 2 \ 3] = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

$$[3 \ 4 \ 5] .^ ? = [27 \ 16 \ 125]$$

Vectores - É a tua vez!

Chegaste lá?

$$[12 \ 15 \ 9] / 3 = [4 \ 5 \ 3]$$

$$[10 \ 2 \ 36] ./ [5 \ 1 \ 18] = [2 \ 2 \ 2]$$

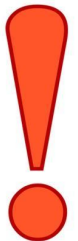
$$[3 \ 4 \ 5] .^ 2 = [9 \ 16 \ 25]$$

$$[3 \ 4 \ 5] .^ [3 \ 2 \ 3] = [27 \ 16 \ 125]$$

$$[1 \ 2 \ 3] .* [1 \ 2 \ 3] = [1 \ 4 \ 9]$$

$$[1 \ 2 \ 3] * \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 14$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} * [1 \ 2 \ 3] = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$



WARNING: Quando estiveres a fazer operações entre vectores tem em atenção que eles têm de ter o mesmo tamanho ou então terás um erro! Podes verificar o tamanho do vector usando a função *length*

Matrizes

- Agora que já sabes de vectores, matrizes é canja!
- Na matriz ambas as dimensões são diferentes de 1. Um exemplo de uma matriz é uma imagem de ressonância magnética em que cada valor da matriz corresponde à intensidade de um pixel da imagem.



| | | | | | |
|----|----|----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 8 |
| 4 | 0 | 0 | 8 | 8 | 40 |
| 0 | 0 | 0 | 32 | 32 | 81 |
| 0 | 0 | 0 | 32 | 32 | 81 |
| 0 | 16 | 16 | 77 | 77 | 77 |
| 0 | 16 | 16 | 77 | 77 | 77 |
| 0 | 60 | 60 | 89 | 89 | 69 |
| 0 | 60 | 60 | 89 | 89 | 69 |
| 36 | 81 | 81 | 73 | 73 | 105 |
| 36 | 81 | 81 | 73 | 73 | 105 |

Matrizes - Operações

- Para criar uma simples matriz pode usar o “;” entre diferentes vetores

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

- As operações entre matrizes são idênticas às dos vetores pelo que podem usar a Tabela apresentada previamente.
- Calculem os seguintes exemplos:

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

$$4*A + \frac{B}{2} = \begin{bmatrix} 8.5 & 12 & 15.5 \\ 19 & 22.5 & 26 \\ 29.5 & 33 & 36.5 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{bmatrix}$$

$$A .* B = \begin{bmatrix} 9 & 16 & 21 \\ 24 & 25 & 24 \\ 21 & 16 & 9 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$$

$$A.^2 = \begin{bmatrix} 1 & 4 & 9 \\ 16 & 25 & 36 \\ 49 & 64 & 81 \end{bmatrix}$$

Matrizes - Funções

| Função | Explicação | Exemplo |
|--------------|---|--|
| size (A) | Devolve as dimensões da matriz A | $\text{size} \left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \right) = [2 \ 3]$ |
| zeros (m, n) | Cria uma matriz de zeros com m linhas e n colunas | $\text{zeros}(2, 3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| ones (m, n) | Cria uma matriz de 1 com m linhas e n colunas | $\text{ones}(2, 3) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |
| diag (a) | Cria uma matriz diagonal cujos elementos da diagonal correspondem aos elementos do vector a | $\text{diag}([1 \ 2 \ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ |
| eye (m) | Cria uma matriz diagonal cuja diagonal é unicamente formada por 1 | $\text{eye}(3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| pinv(A) | Devolve a inversa da matriz A | $\text{pinv} \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |

Matrizes – Desafio!!

Cria uma matriz 6x5:

- Primeiras 5 linhas correspondem a uma matriz diagonal cujos elementos igualam 4
- Última linha é constituída apenas por elementos de valor 6



$$[4 * \text{eye}(5) ; 6 * \text{ones}(1,5)] = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 6 & 6 & 6 & 6 & 6 \end{bmatrix}$$

Vectores e Matrizes - Últimas dicas

- Como aceder a elementos específicos de um vector ou matriz?

$$a = [4 \ 5 \ 6 \ 7 \ 8 \ 9]$$

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

| | | | |
|------------------------|------------------------|------------------------|--------------------------|
| $a(2) = 5$ | $a([3 \ 6]) = [6 \ 9]$ | $a(2:4) = [5 \ 6 \ 7]$ | $a(4:end) = [7 \ 8 \ 9]$ |
| $A([1 \ 3]) = [1 \ 7]$ | $A(1,3) = 3$ | $A(1,:) = [1 \ 2 \ 3]$ | $A(:,1) = [1 \ 4 \ 7]'$ |

- Como concatenar vectores e matrizes?

$$a = [1 \ 2 \ 3]$$

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

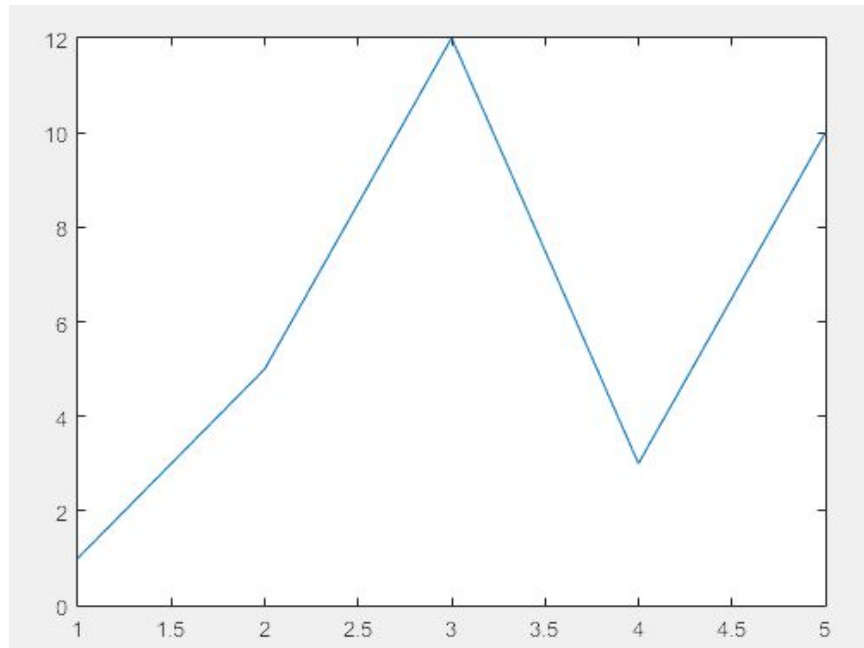
| | | |
|--|--|--|
| $[a \ a] = [1 \ 2 \ 3 \ 1 \ 2 \ 3]$ | $[A \ a'] = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 9 & 3 \end{bmatrix}$ | $[A \ A] = \begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 5 & 6 & 4 & 5 & 6 \\ 7 & 8 & 9 & 7 & 8 & 9 \end{bmatrix}$ |
| $[a ; a] = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$ | $[A ; a] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix}$ | $[A ; A] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ |

Representações Gráficas

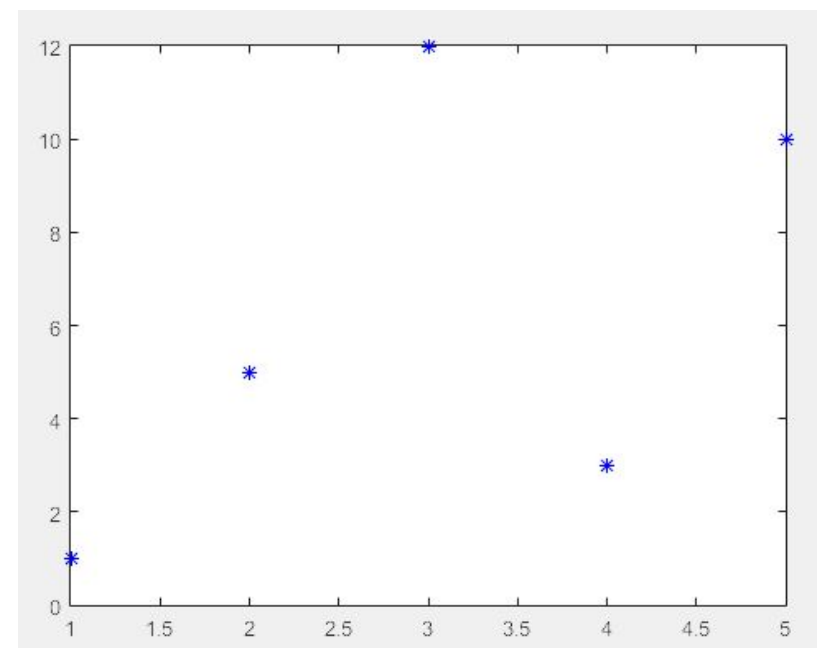
- Por vezes dão-nos um vector e queremos ver como é o seu aspecto. O que é que devemos fazer? Gráficos desses vectores!
- Para fazerem um gráfico a função que necessitam de saber é o ***plot***

Exemplos:

```
v = [1 5 12 3 10];  
plot(v);
```



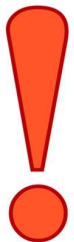
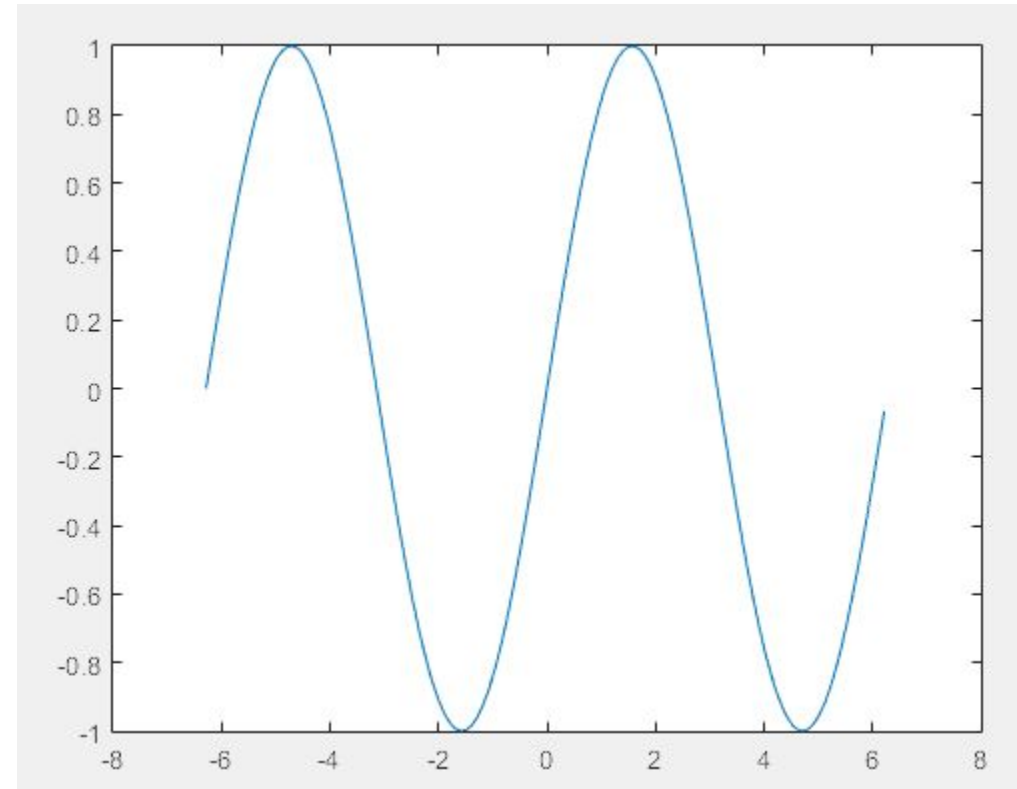
```
v = [1 5 12 3 10];  
plot(v, 'b*');
```



Representações Gráficas - Funções

```
t = -2*pi:0.1:2*pi;  
y = sin(t);
```

```
figure(1)  
plot(t,y);
```



NOTA: sempre que for necessário apresentar gráficos devem chamar a função *figure('número')* antes de fazerem *plot* para não perderem nenhum gráfico por acidente

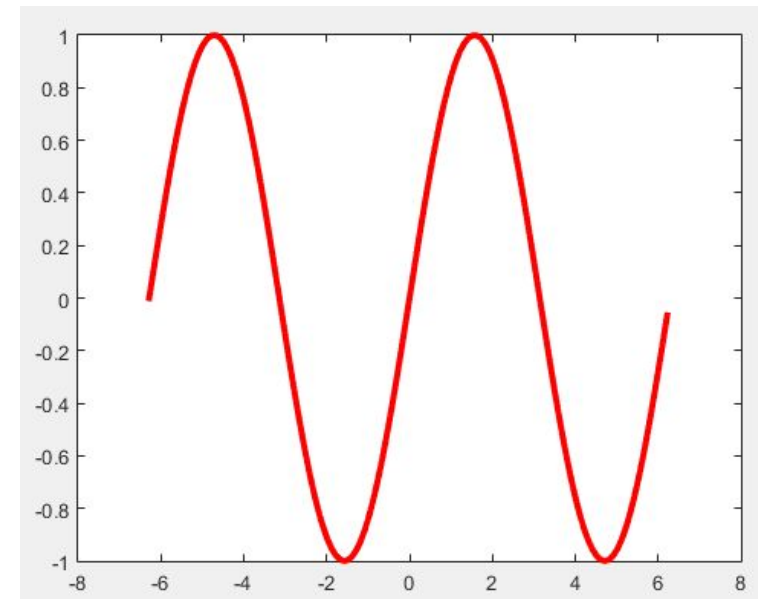
Representações Gráficas - Customização

- “Ah eu quero que o meu gráfico seja vermelho e não azul!”
- “Ah eu quero aumentar a espessura do meu gráfico para poder ver melhor”
- O Matlab tem solução para isso...

plot (t, y, OPTION, VALUE)

```
plot(t, y, 'LineWidth', 3, 'MarkerSize', 5, 'Color', 'red');
```

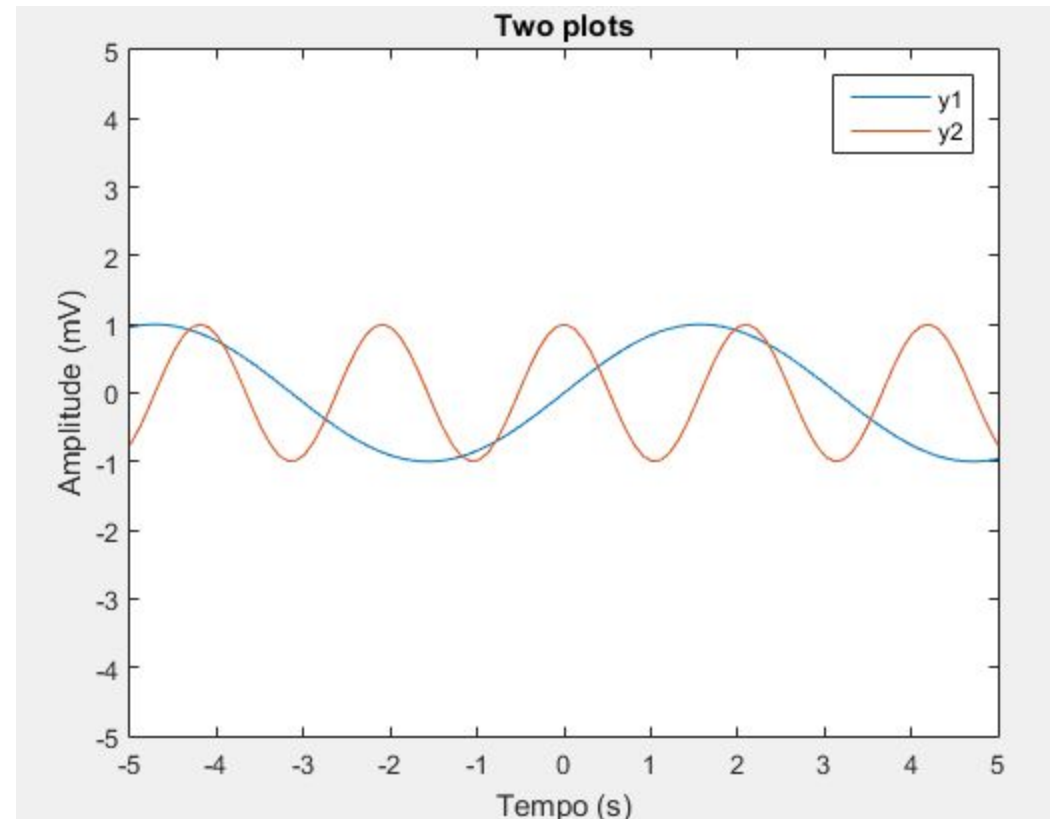
| OPTION | VALUES |
|-----------------|--------------------------------------|
| LineWidth | Positive value |
| Color | 'red', 'green', 'blue', 'black', ... |
| LineStyle | '-', '- -', ':' |
| MarkerSize | Positive value |
| MarkerFaceColor | 'red', 'green', 'blue', 'black', ... |
| MarkerEdgeColor | 'red', 'green', 'blue', 'black', ... |



Representações Gráficas – Apresentação

- Não se esqueçam de colocar sempre legendas, títulos e unidades. Pequenos pormenores como estes podem fazer a diferença!

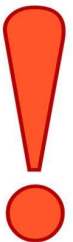
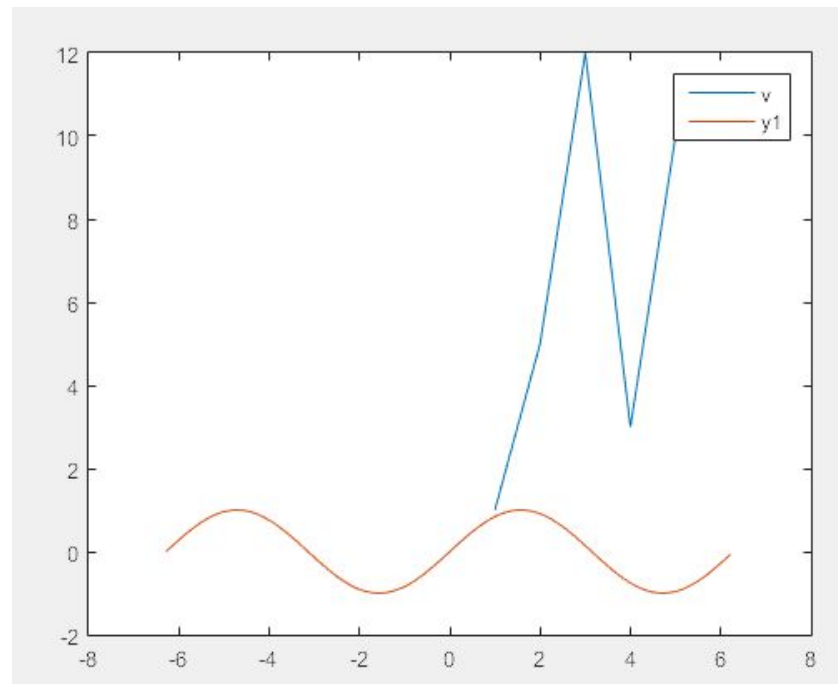
```
%Plot de funcoes
t = -2*pi:0.1:2*pi;
y1 = sin(t);
y2 = sin(3*t + pi/2);
figure(1)
plot(t,y1,t,y2);
xlabel('Tempo (s)')
ylabel('Amplitude (mV)')
title('Two plots')
axis([-5 5 -5 5])
legend('y1', 'y2')
```



Representações Gráficas – Dicas

- No slide anterior mostrámos dois gráficos na mesma figura com o comando `plot(t,y1,t,y2)`. Uma outra solução normalmente usada é a opção **'hold on ... hold off'**

```
v = [1 5 12 3 10];  
t = -2*pi*0.1:2*pi  
  
figure(2);  
plot(v);  
hold on  
plot(t, y1);  
hold off  
legend('v', 'y1');
```



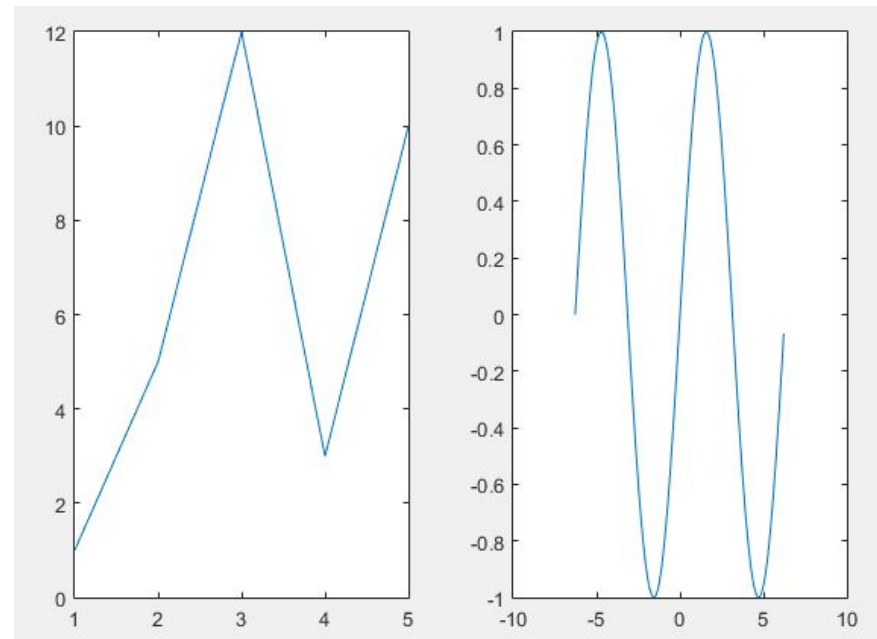
IMPORTANTE : Executar sempre o comando *hold off* depois de se ter escrito todo o código de plots. Se não o fizermos, outros gráficos indesejados serão feitos em cima da mesma figura

Representações Gráficas – Dicas

- Outro comando bastante usado é o *subplot*. Com este podem dispor vários gráficos na figura como se fosse uma matriz. Pode dar jeito quando têm muita coisa para mostrar!

subplot (nº linhas, nº colunas, nº gráfico)

```
%Plot de funcoes  
t = -2*pi:0.1:2*pi;  
y1 = sin(t);  
figure(1)  
subplot(1,2,1)  
plot(v)  
subplot(1,2,2)  
plot(t,y1);
```



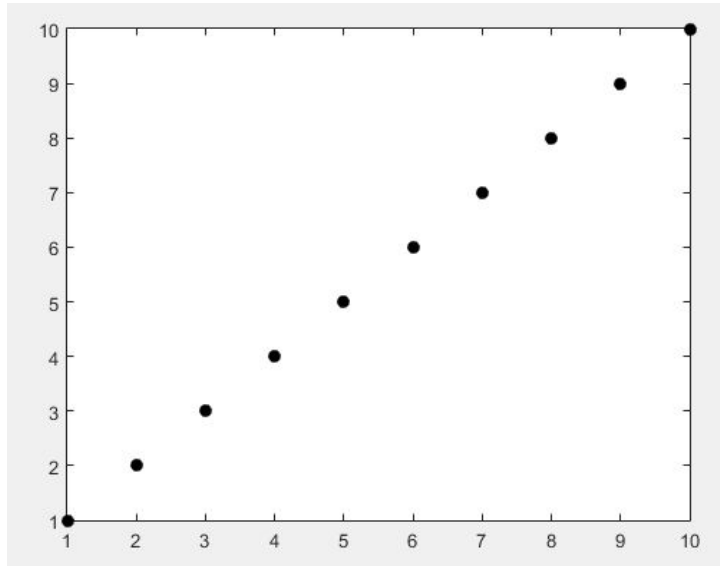
Representações Gráficas – Exercícios!

1. Gerar o gráfico de um vector de inteiros com os números de 1 a 10. Cada elemento do vector deve ser representado por um ponto de cor preta.
2. Gerar o gráfico das funções $y_1 = \sin(x)$ e $y_2 = 0.2 \cdot \cos(5x)$ com 100 e 200 pontos respectivamente. Apresenta os gráficos no intervalo $x = [0 \ 200]$ e $y = [-2 \ 2]$. Coloca também legenda nos eixos x e y e um título ao teu gosto!
3. Executa o código em baixo. O ficheiro .mat que fizeste **load** contém um sinal de ECG que fica guardado na variável *x1*. A função **findpeaks**, como o nome diz acha os picos no vector *x1* e guarda a amplitude dos mesmos na variável *pks*, e a sua localização na variável *locs*. Gera o plot que contém o sinal de ECG juntamente com os picos assinalados com pontos pretos.

```
load('ecg.mat');  
[pks,locs] = findpeaks(x1,'MinPeakHeight',300);
```

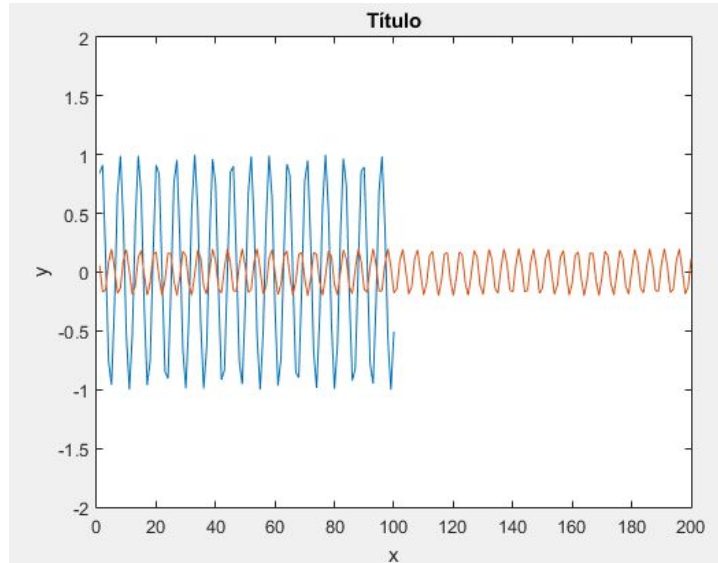

Representações Gráficas – Exercícios!

1.



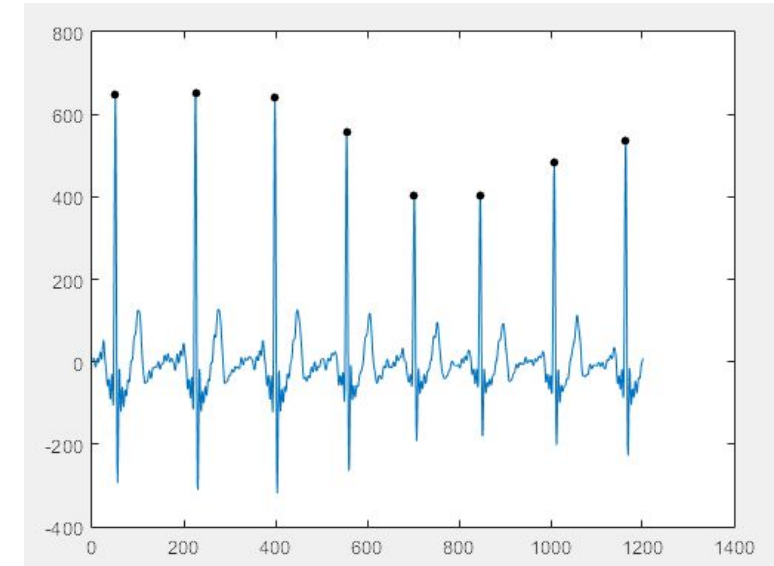
```
x = 1:10;  
figure(1)  
plot(x, 'ko', 'MarkerFaceColor', 'black')
```

2.



```
x1 = 1:100;  
x2 = 1:200;  
y1 = sin(x1);  
y2 = 0.2*cos(5*x2);  
figure(2)  
plot(x1,y1,x2,y2)  
xlabel('x')  
ylabel('y')  
title('Titulo')  
axis([0 200 -2 2])
```

3.



```
load('ecg.mat')  
[pks, locs] = findpeaks(x1, 'MinPeakHeight', 300);  
figure(3)  
plot(x1)  
hold on  
plot(locs,pks, 'ko', 'MarkerFaceColor', 'black')  
hold off
```

Fácil, não é???



Branching

- Lembram-se do **if** ? Pois é, ele também existe no Matlab com algumas novidades
- Caso a primeira condição não se verifique recorrem à segunda condição usando **else**. *“Se (if) aprender Matlab então tiro 20 a SSB, caso contrário (else) tiro 0.”*
- Também existe o **elseif** para fazer mais condições! *“Se (if) aprender bem Matlab então tiro 20 a SSB, se (else if) aprender mais ou menos Matlab então tiro 12 a SSB, caso não aprenda nada (else) desisto disto e vou dedicar-me à pesca.”*

```
u = true;  
v = false;
```

if + else

```
if u  
    disp('This is true');  
else  
    disp('This is false');  
end
```

if + elseif + else

```
if v == true  
    disp('v is true');  
elseif u == true  
    disp('u is true but not v');  
else  
    disp('nothing is true');  
end
```

Branching

- Para além do *if* existe mais uma forma de *branching* por vezes usada que é o ***switch***.
- O ***switch*** recebe uma expressão e compara cada caso com essa expressão, executando uma acção quando o caso iguala o valor da expressão

```
v = [3 4 1 0];  
switch v(1)  
    case 1  
        disp('v(1) is 1');  
    case 2  
        disp('v(1) is 2');  
    case 3  
        disp('v(1) is 3');  
    case 4  
        disp('v(1) is 4');  
    otherwise  
        disp('v(1) is something else');  
end
```

Qual dos casos é que achas que o código ao lado vai executar?

R: O terceiro caso!

Branching – Operadores relacionais e lógicos

| Operador Relacional | Símbolo |
|---------------------|---------|
| Maior | > |
| Menor | < |
| Igual | == |
| Diferente | ~= |
| Maior ou igual | >= |
| Menor ou igual | <= |

Exemplos

```
3 > 4.% false or 0
5 < 6.% true or 1
3 == 3.% true or 1
4 ~= 4.% false or 0
3 >= 0.% true or 1
3 <= 0.% false or 0
```

| Operador Lógico | Símbolo/Função |
|-----------------|----------------|
| AND | && |
| OR | |
| NOT | ~ |

Exemplos

```
(3 > 4) && (3 < 4) % false or 0
(5 < 6) || (3 > 4) % true or 1
~(3 == 3) %false or 0
```



Não te esqueças dos parênteses!

Branching – Operadores relacionais e lógicos (vectores!)

Com os vectores $x = [0 \ 1 \ 2 \ 3]$ e $y = [0 \ 4 \ 5 \ 0]$ executem as seguintes expressões:

| Expressão | Resultado |
|------------|-------------------|
| $x > y$ | $[0 \ 0 \ 0 \ 1]$ |
| $x < y$ | $[0 \ 1 \ 1 \ 0]$ |
| $x == y$ | $[1 \ 0 \ 0 \ 0]$ |
| $x \& y$ | $[0 \ 1 \ 1 \ 0]$ |
| $x \mid y$ | $[0 \ 1 \ 1 \ 1]$ |



Nota: Repara que com vectores é $\&$ e \mid e não $\&\&$ e $\mid\mid$

Loops

- Já relembrámos o **if** agora só te falta lembrar o **while**!
- O **while** executa a acção enquanto a condição permanecer verdadeira.
- Tem atenção aos ciclos infinitos!

```
u = 10;  
[- while u > 0  
    disp([int2str(u) ' seconds to the final countdown!']);  
    u = u - 1;  
end
```

- Outra forma de fazer loop que é bastante usada em programação é o **for**.
- O **for** percorre um vector e executa a acção até chegar ao fim desse vector.

```
[- for u = 1:10  
    disp(['Daqui a ' int2str(u) ' anos terei ' int2str(u+19) ' anos'])  
end
```



NOTA: no **for** não precisas de dizer que queres reduzir/aumentar a tua variável

NOTA: nos loops tens a opção de “escondê-los” clicando no botão (-). Usa-o para esconder linhas e ser mais fácil de ler o teu código!

Loops – Valerá sempre a pena?

- Em vez de encheres o teu código cheio de loops porque não fazes simples operações com matrizes e vectores?? O Matlab serve mesmo para isso!
- Perde algum tempo no papel e tenta escrever os teus problemas em forma vectorizada

Exemplo

Variáveis (vectores)

```
A = rand(10000000, 1);  
B = rand(10000000, 1);
```

Loop

```
res = 0;  
for i = 1:length(A)  
    res = res + A(i)*B(i);  
end
```

Operação vectorial

```
A'*B;
```



Qual das formas gostas mais? A segunda claro!

Mais rápido!

Mais elegante!

Talvez faça a diferença na
nota final!

Loops – Desafio!!

Escreve um script que conte o número de zeros de um vector :

- Primeiro em loop
- Depois em forma vectorial (dica: investiga a função *find*)
- Vê também a duração de cada uma usando as funções *tic....toc*



Vector `x = [ones(1,10) zeros(1,10) 5*ones(1,10)];`

Loop

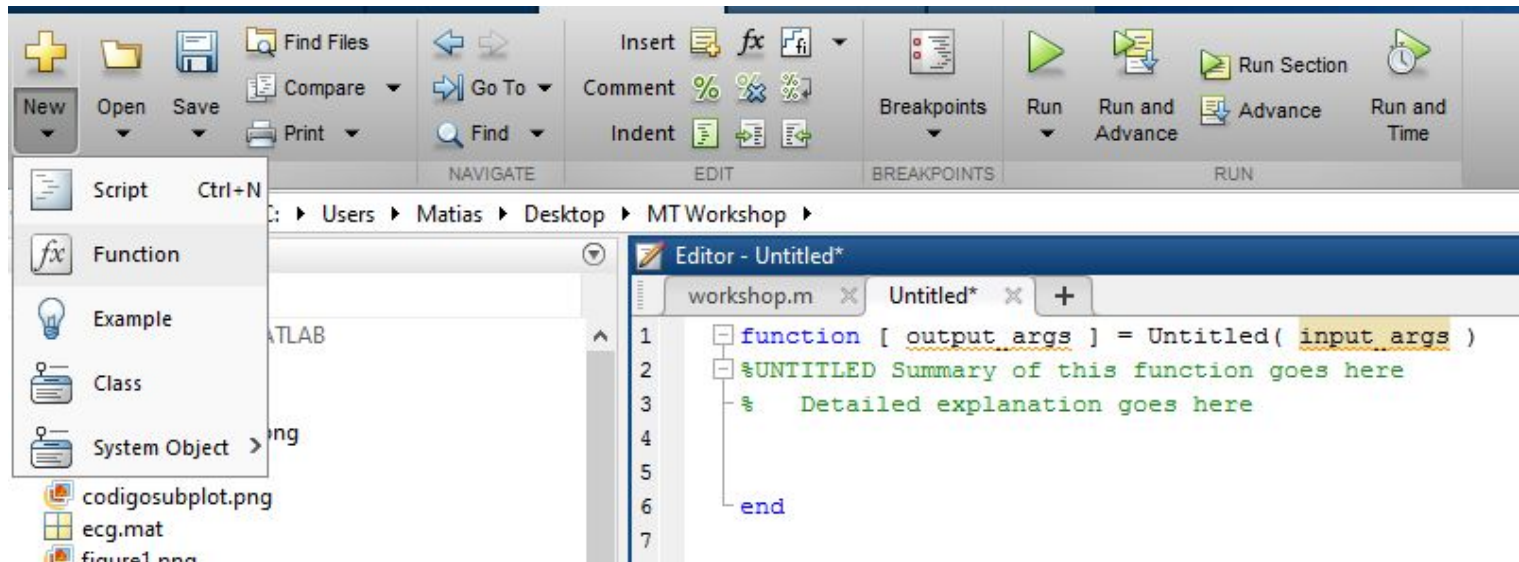
```
count = 0;
for i = 1:length(x)
    if(x(i) == 0)
        count = count + 1;
    end
end
```

Operação vectorial

```
count = length(find(x == 0));
```


Funções

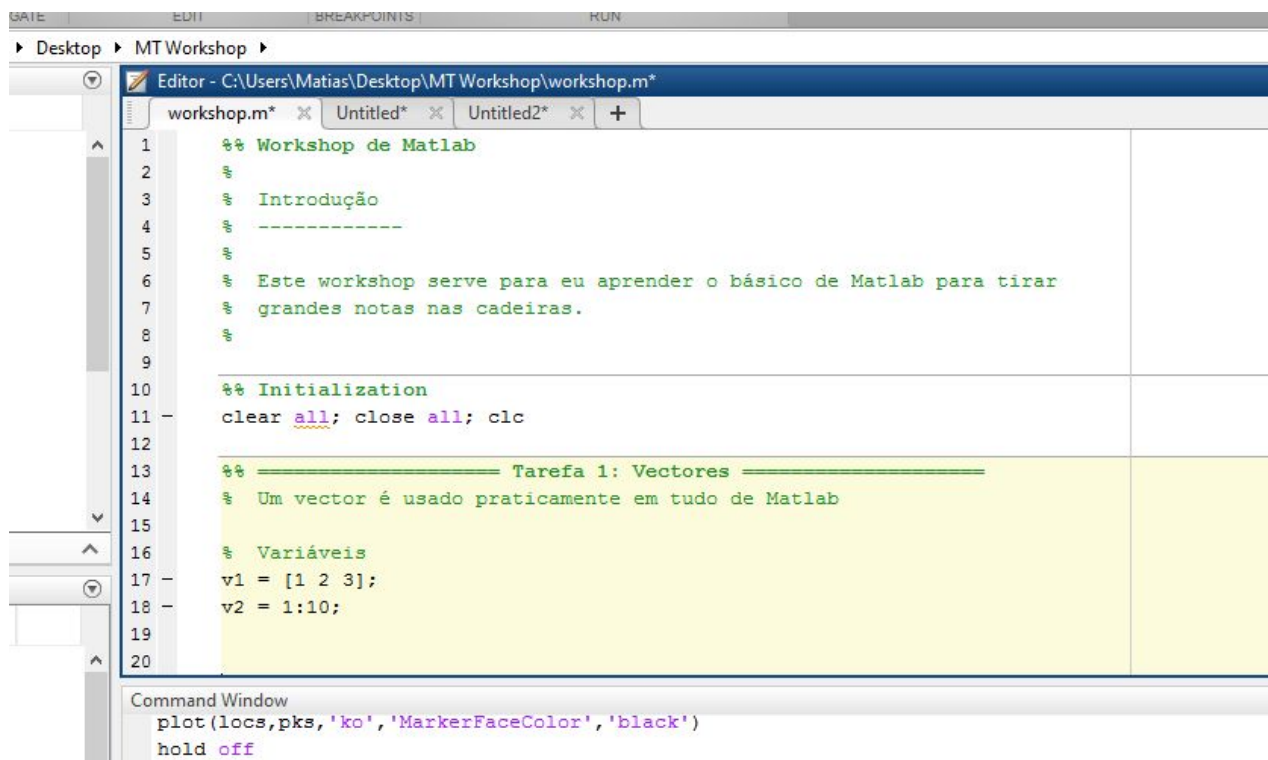
- Como já devem ter percebido, o Matlab tem imensas funções (*sqrt*, *zeros*, *find*) mas normalmente nos vossos projectos têm de construir as vossas próprias funções!



- Reparem que uma função foi criada no Editor. Ela está a vossa espera para vocês colocarem os *inputs*, os *outputs* e tudo o que vocês quiserem! O céu é o limite!!

(Dicas para Scripts e Funções)

- Quando estiverem a escrever o vosso código lembrem-se que os professores (e mesmo vocês!) têm de conseguir saber o que está lá. Sejam ORGANIZADOS!!



```
1 %% Workshop de Matlab
2 %
3 % Introdução
4 % -----
5 %
6 % Este workshop serve para eu aprender o básico de Matlab para tirar
7 % grandes notas nas cadeiras.
8 %
9
10 %% Initialization
11 clear all; close all; clc
12
13 %% ===== Tarefa 1: Vectores =====
14 % Um vector é usado praticamente em tudo de Matlab
15
16 % Variáveis
17 v1 = [1 2 3];
18 v2 = 1:10;
19
20
```

```
Command Window
plot(locs,pks,'ko','MarkerFaceColor','black')
hold off
```

- O comando '%' serve para comentar o código
- O comando '%%' serve para criar divisões. Podem correr cada divisão clicando 'Ctrl+Enter'
- É hábito começar cada script com **clear all** (apaga as variáveis); **close all** (apaga as figuras); **clc** (apaga a Command Window)
- Terminem as linhas com ';' para evitar *outputs* indesejados
- Dêem nomes intuitivos às variáveis e funções
- Documentem bem o código e a função (informação dos *inputs* e *outputs*)

Funções – Fazer e usar uma função

Experimentem fazer a função abaixo. Esta devolve a soma de cubos de dois números.

```
function [ z ] = sum_of_cubes( x,y )  
%sum_of_cubes Recebe dois numeros x e y  
%Devolva a soma de x^3 com y^3 na variavel z  
  
z = x^3 + y^3;  
  
end
```

Gravem a função. *SUPER IMPORTANTE*: o ficheiro tem de ter o mesmo nome que dão à função senão já sabem o que dá. ERRO!



Agora usem a vossa primeira função! Vão à CommandWindow e usem-na com dois números.

```
Command Window  
  
>> res = sum_of_cubes(2,3)  
  
res =  
  
35
```

Funções – Additional stuff

- A vossa função pode ter quantos *inputs* e *outputs* vocês entenderem
- Podem “proteger” a vossa função para que receba exactamente o que querem. O *return* interrompe a função.
- Podem imprimir texto (*strings*) com valores sempre que quiserem. O *fprintf* é uma função que reconhece símbolos ‘%d’ e substitui pela variável que vem a seguir (**Exemplo: %d -> z**)



DICA: Com o *fprintf* podem mostrar o valor com o número de casas decimais que quiserem!

```
function [ z, p ] = sum_of_cubes( x, y )
%sum_of_cubes Recebe dois números x e y
%Devolve a soma de x^3 com y^3 na variável z e o dobro desse valor

z = x^3 + y^3;
p = 2*z;

end
```

```
function [ z ] = sum_of_cubes( x,y )
%sum_of_cubes Recebe dois numeros x e y
%Devolva a soma de x^3 com y^3 na variavel z
if(length(x) == 1 && length(y) == 1)
    z = x^3 + y^3;
else
    disp('Esta função apenas recebe numeros e não matrizes');
    return;
end
end
```

```
function [ z ] = sum_of_cubes( x,y )
%sum_of_cubes Recebe dois numeros x e y
%Devolva a soma de x^3 com y^3 na variavel z
if(length(x) == 1 && length(y) == 1)
    fprintf('O valor de x^3 é %d \n', x^3);
    fprintf('O valor de y^3 é %d \n', y^3);
    z = x^3 + y^3;
    fprintf('O valor de z é então %d \n', z);
else
    disp('Esta função apenas recebe numeros e não matrizes');
    return;
end
end
```

Funções – Additional stuff

Se forem ao *help* da função *mean* podem observar que ela pode receber um *input* e executar um comando ou receber dois *inputs* e executar outro comando à mesma. Mas na função que criámos antes isso não é possível!!! Por enquanto...

O Matlab tem esta função com nome estranho chamada *nargin* que serve mesmo para isso. Ela devolve o número de inputs que a função recebe.

```
function [ z ] = sum_of_cubes( x, y )
%sum_of_cubes Recebe dois números x e y
%Devolve a soma de x^3 com y^3 na variável z e o dobro desse valor

if nargin == 2
    z = x^3 + y^3;
elseif nargin == 1
    z = x^3 + x^3;
end
end
```

Funções – Exercícios!!

1. Implementa uma função que receba um vector e devolve o número de zeros desse vector
2. Implementa uma função que recebe um vector e devolve o elemento máximo desse vector (bonus para implementações na forma vectorial)
3. Implementa uma função que recebe uma matriz e devolve o elemento máximo dessa matriz (bonus para implementações na forma vectorial)

Funções – Exercícios!!

1.

```
function [ count ] = count_zeros( v )  
%count_zeros Recebe um vector v  
%Devolve o número de zeros desse vector  
  
count = length(find(v == 0));  
  
end
```

2.

```
function [ maxim ] = max_value_vec( v )  
%max_value_vec Recebe um vector v  
%Devolve o elemento máximo desse vector  
  
maxim = max(v);  
  
end
```

3.

```
function [ maxim ] = max_value_mat( M )  
%max_value_vec Recebe uma matriz M  
%Devolve o elemento máximo dessa matriz  
  
maxim = max(M(:));  
  
end
```



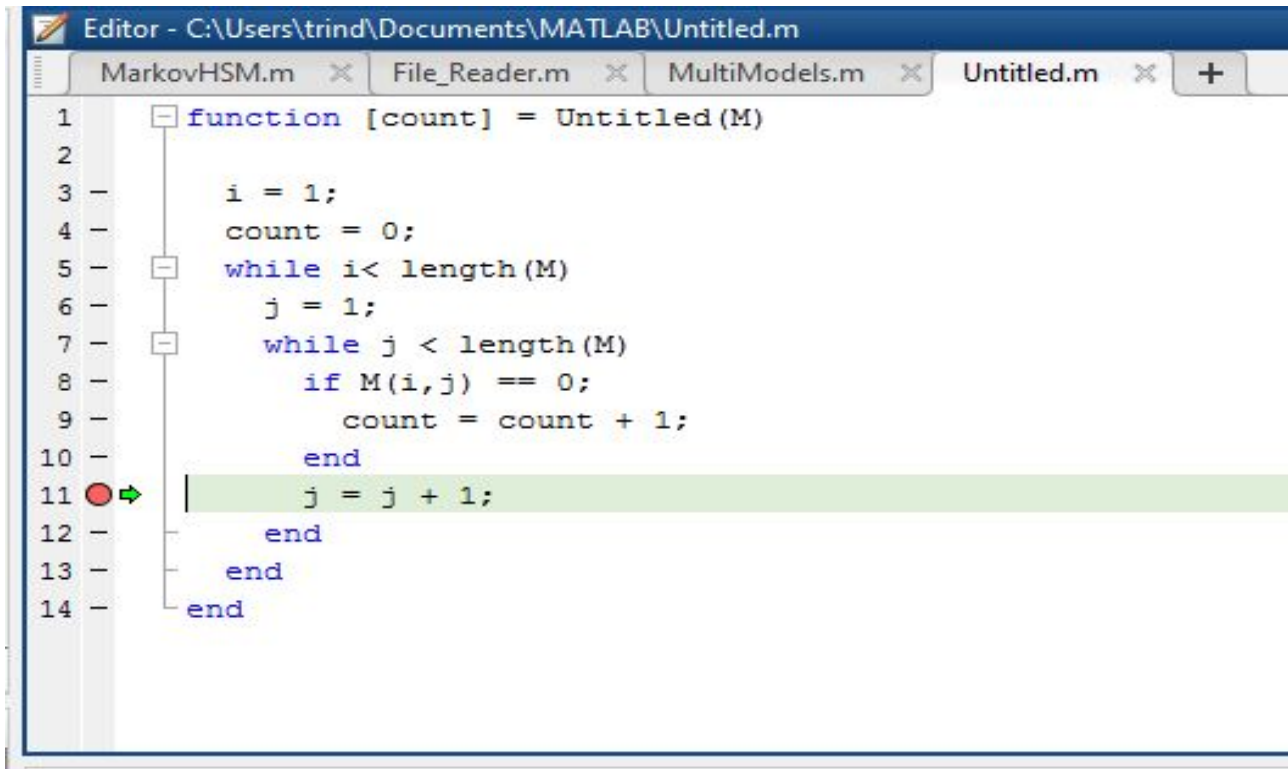
LIKE A BOSS

Funções - Debug

Há algo de errado nesta função, descubra o que é...

```
function [count] = zeros_da_matriz(M)
    i = 1;
    count = 0;
    while i < length(M)
        j = 1;
        while j < length(M)
            if M(i,j) == 0
                count = count + 1;
            end
        end
        j = j + 1;
    end
end
```


Funções - Debug



```
1 function [count] = Untitled(M)
2
3     i = 1;
4     count = 0;
5     while i < length(M)
6         j = 1;
7         while j < length(M)
8             if M(i,j) == 0;
9                 count = count + 1;
10            end
11            j = j + 1;
12        end
13    end
14 end
```

- O problema está em que o valor de i nunca é incrementado logo o ciclo nunca termina!
- Em casos em que não esteja a fazer sentido o comportamento do programa podem sempre recorrer ao modo de debug do Matlab e colocarem **breakpoints** nas linhas que quiserem.
- Depois basta executarem a função/script e o código pára em cada um dos **breakpoints**. Assim podem consultar o valor de cada uma das variáveis na altura em que o código parou de ser executado

Funções Anónimas - IMPORTANTE PARA MC!

- “Funções anónimas? Então, ninguém quer saber das pobres coitadas?”
- Estas funções não são criadas no Editor mas apenas numa linha.
- Normalmente usadas para definir uma **expressão matemática** e serem usadas como **inputs** em funções a sério (não anónimas)

Exemplo: Soma de Cubos

$$f = @(x, y) (x^3 + y^3)$$

Inputs Expressão

Comparem agora *sum_of_cubes(2,3)* com *f(2,3)*. Deu igual certo?



Mega-Desafio!!!

➤ Implementa o método de Newton-Rhapson em Matlab que recebe como *inputs*:

- Uma função f
- A sua derivada df
- A aproximação inicial x_0
- Uma tolerância de erro.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$e_{n+1} = |x_{n+1} - x_n|$$

➤ E devolve as aproximações com os erros em cada iterada.

➤ Experimentem a função com a expressão $e^x + x$, aproximação inicial igual a 0.5 e tolerância de erro $0.5 * 10^{-5}$

Dicas: Determina primeiro uma aproximação inicial $n = 1$ e só depois entras no método iterativo. Só páras o método iterativo quando o erro for menor que a tolerância que deste!



Mega-Desafio!!

Função

```
function [ x, ex ] = newton_ws( f, df, x0, tol)
%newton_ws Recebe uma função f, a sua derivada df, uma aproximação inicial
%x0 e uma tolerância de erro tol
%Devolve a aproximação da raiz x assim como a estimacão do erro ex em cada
%iteração
%
x(1) = x0 - (f(x0)/df(x0));
ex(1) = abs(x(1)-x0);
k = 2;
while (ex(k-1) >= tol)
    x(k) = x(k-1) - (f(x(k-1))/df(x(k-1)));
    ex(k) = abs(x(k)-x(k-1));
    k = k+1;
end
end
```

Aplicação da Função

```
>> f = @(x) (exp(x)+x);
>> df = @(x) (exp(x)+1);
>> x0 = 0.5;
>> tol = 0.5*10^-5;
>> [x, ex] = newton_ws(f,df,x0,tol)

x =

    -0.3112    -0.5544    -0.5671    -0.5671    -0.5671

ex =

    0.8112    0.2432    0.0127    0.0000    0.0000
```

Leitura de ficheiros

- O Matlab permite a leitura de ficheiros como imagens, audio, txt, xls entre outros. Abaixo ficam exemplos de como usar algumas dessas funções

```
a = csvread('acceleration.csv');  
im = imread('len_gray.jpg');  
[song, fs] = audioread('Hallelujah.wav');  
player = audioplayer(song, fs);  
play(player);  
stop(player);
```

- csv significa comma-separated-values. Estes ficheiros têm a seguinte estrutura (val1, val2, val3..., valn \n). Onde \n significa nova linha. Ao executar o comando **csvread** o ficheiro é carregado para uma matriz.
- O comando **imread** lê a imagem especificada e coloca-a numa matriz.
- O mesmo sucede para ficheiros áudio mas para o comando **audioread**.

Exercícios!

1. Faz download do ficheiro **xr.mat** e carrega-o para o Matlab usando o comando **load**. Este ficheiro contém uma matriz que deves manipular de modo a tornar num vector **v** de dimensão $[N \times 1]$. Escreve um script que faça esta operação (dica: função *reshape*). Usa o comando **soundsc(v, 44100)** para descobrires o segredo desta matriz.



2. Escreve uma função **vector2String** que converte um vector de inteiros para uma **sequência de nucléotidos** de acordo com o seguinte mapeamento { 0:A, 1:C, 2:T, 3:G }. Experimentem fazer sem e com a função `containers.Map`

3. Faz o download dos 3 ficheiros **Acceleration.csv** do github. Estes ficheiros são registos de um acelerómetro de 3 eixos colocado no pulso. Cada linha do csv segue a seguinte estrutura ax, ay, az, timestamp. O timestamp corresponde ao momento em que a amostra foi adquirida.

- a) Escreve um script que **lê os ficheiros csv e carrega os dados para o MATLAB**. Podes colocar os dados dos ficheiros em matrizes separadas (dica: função *csvread*) ou todas numa **cell** (uma cell corresponde a uma estrutura que funciona como uma matriz mas que em cujos elementos podes inserir quaisquer tipos de dados como um inteiro, uma string ou mesmo um vector ou matriz).
- b) Escreve uma função que recebe uma das matrizes anteriores e faz a extracção de **features** destes sinais. As **features** devem ficar numa matriz em que cada coluna corresponde às **features** de um sinal. As **features** a extrair são a **média do sinal, desvio padrão, mediana, correlação entre todas as direcções, percentage de número de amostras positivas e percentage de número de amostras negativas**.

Soluções!

1.

```
%% 1
load('xr.mat')
v = reshape(xr,size(xr,1)*size(xr,2),1);
soundsc(v, 44100)
```

Com função
containers.Map



2.

```
function [ x ] = vector2String( v )
%Receives an integer vector v and converts its contents to the output
%vector x with the mapping {0:a, 1:c, 2:t, 3:g}

x = '';
s = 'acgt';

for i = 1:length(v)
    x = s(v(i) + 1);
    x = [x; x];
end

end
```

```
function [ x ] = vector2StringMap( v )
%Receives an integer vector v and converts its contents to the output
%vector x with the mapping {0:a, 1:c, 2:t, 3:g}

key = { 0, 1, 2, 3};
value = {'A','C','T','G'};
newMap = containers.Map(key,value);
x = '';

for i =1:length(v)
    x = [newMap(v(i)); x];
end

end
```


3.a

```
%Easy way (for few files)
a1 = csvread('Acceleration (1).csv');
a2 = csvread('Acceleration (2).csv');
a3 = csvread('Acceleration (3).csv');

%General way (suitable for lot of files)
%Variable to store the data
fulldata = cell(1);
%\
f = filesep;
%Take all the files from your directory that start with 'Acceleration'
D = dir([pwd, f, 'Acceleration*']);
%Count the number of files that start with 'Acceleration'
num = length(D(not([D.isdir])));
%Loop through all the files and save each in the cell structure
for i=1:num
    filename_a = sprintf('Acceleration (%d).csv',i);
    fulldata{i,1} = csvread(filename_a);
end
```

3.b

```
function [ v ] = extraction( a )
%Receives acceleration matrix a and returns vector v with features from a

%Mean
u = mean(a(:,1:3));

%Standard Deviation
sd = std(a(:,1:3));

%Median
med = median(a(:,1:3));

%Correlations
xy = corr(a(:,1),a(:,2));
xz = corr(a(:,1),a(:,3));
yz = corr(a(:,2),a(:,3));

%Percentage of positive and negative samples
positive = [ sum(a(:,1)>0)/length(a) sum(a(:,2)>0)/length(a) sum(a(:,3)>0)/length(a) ]*100;
negative = [ sum(a(:,1)<0)/length(a) sum(a(:,2)<0)/length(a) sum(a(:,3)<0)/length(a) ]*100;

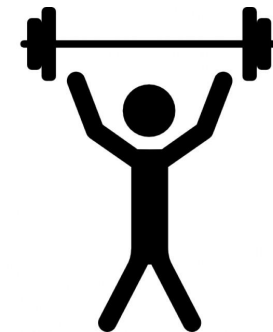
v = [u sd med xy xz yz positive negative];

end
```

Dicas Importantes (Again)!

1. Se não sabes como usar uma função usa o *help* ou o *doc*!
2. Ao manipulares vectores/matrizes atenção aos tamanhos!
3. Atenção à diferença de operação de elementos (*.**) com operações vectoriais (***)!
4. Mostra gráficos apresentáveis! Não metas para lá só uma linha aleatória!
5. Documentar o vosso código é importante!

6. TREINEM!



Se tiverem dúvidas....



...pergunte ao Google. Google knows.

PARABÉNS POR COMPLETARES O WORKSHOP!

