Workshop MATLAB - IEEE

2ª sessão Teórica - João Inácio

Índice

- Plots
- Loops
- Controlo de fluxo
- Funções
- Animações

Plots

Plots - Na última sessão vimos...

- `plot(x1, y1, lineSpec1, x2, y2, lineSpec2, ...)`
- Os pares `(x1, y1)`, ..., devem ter o mesmo tamanho.
- Podemos fazer o *plot* de duas funções na mesma janela usando a *keyword* `hold on`
- O argumento `lineSpec` é usado para definir o modo de como a linha desse *plot* vai aparecer. Para as combinações possíveis visitar: https://www.mathworks.com/help/matlab/ref/linespec.html
- Podemos dividir a janela dos *plots* usando `subplot(xdim, ydim, pos)`

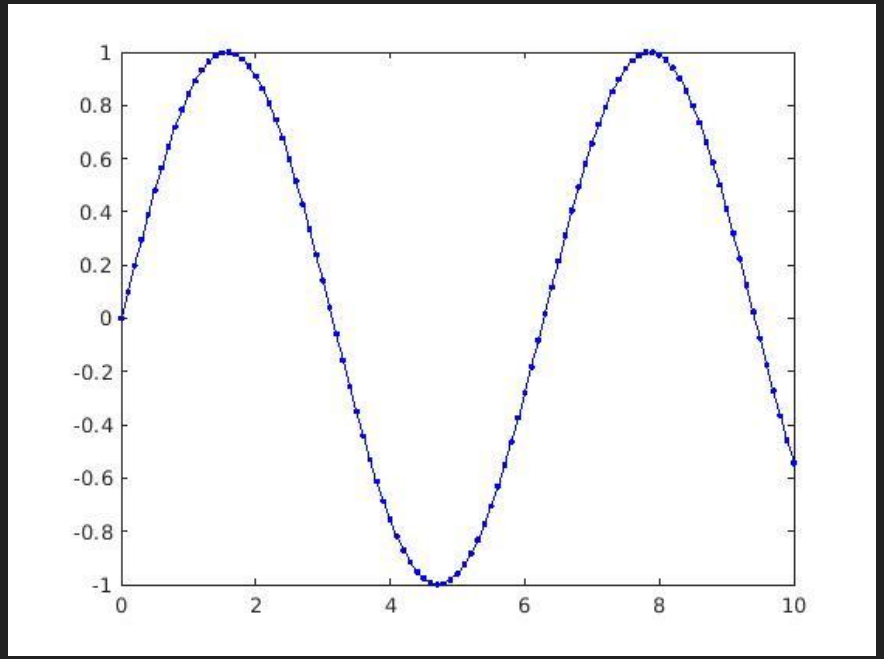
Plots - Personalização

De forma aos gráficos ficarem mais perceptíveis podemos dar-lhes uma legenda, título e nomear os eixos do nosso *plot*.

Considere o código:

```
x = 0:0.1:10;

y = sin(x);

plot(x, y, '.-b')
```

Plots - Personalização



Plots - Personalização

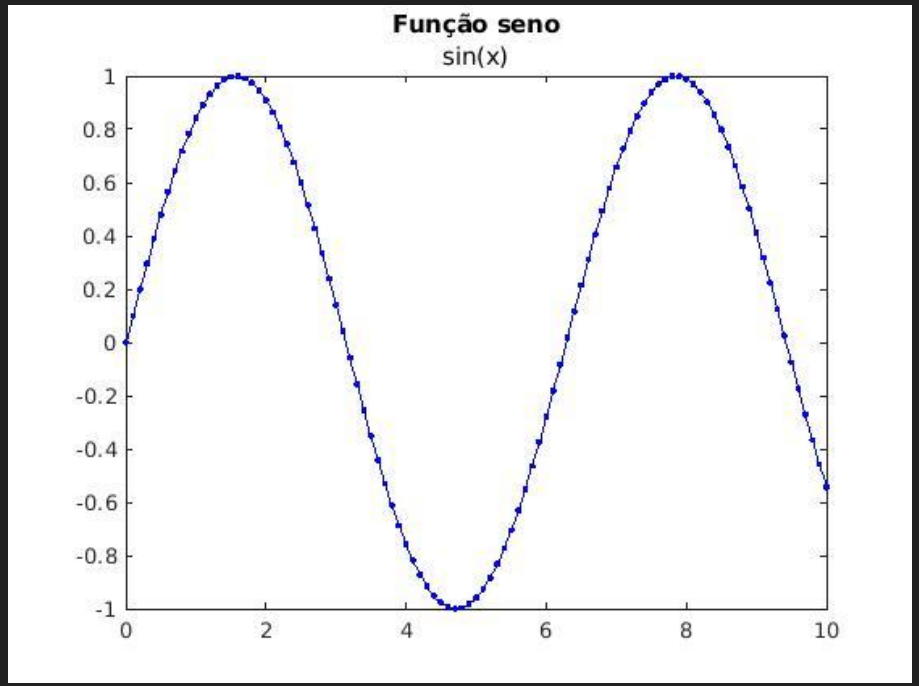
Para dar um título a este gráfico usamos a função `title` com os seguintes argumentos:

```
title(titleStr, subTitleStr)
```

Assim se no final do código anterior adicionar-mos

```
title(“Função Seno”, “sin(x)”)
```

Plots - Personalização



Plots - Personalização

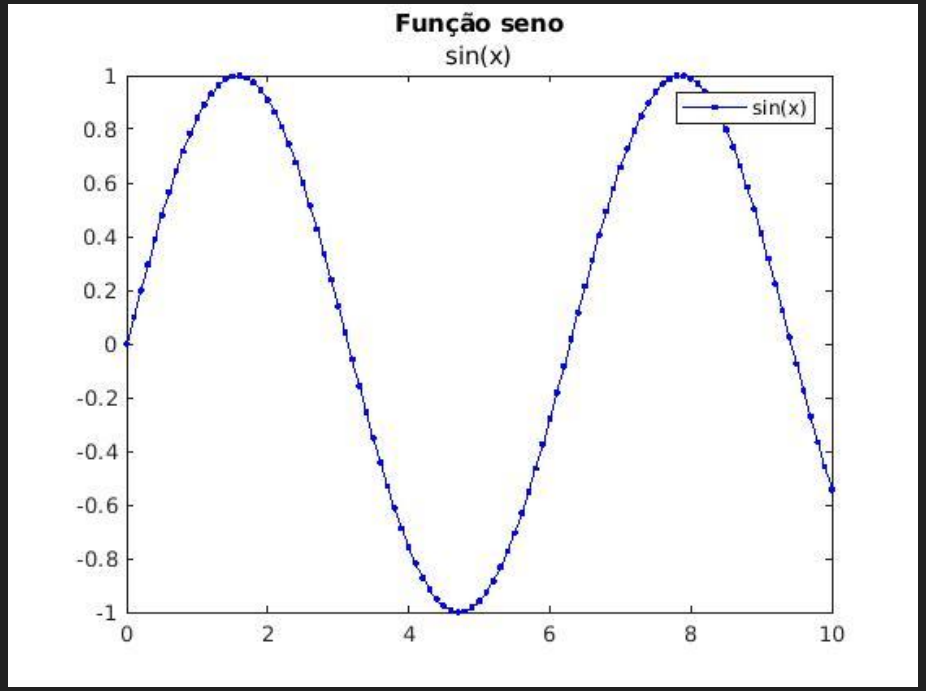
De forma a adicionar uma legenda podemos usar a função `legend`.

```
legend(label1, label2, ..., labelN)
```

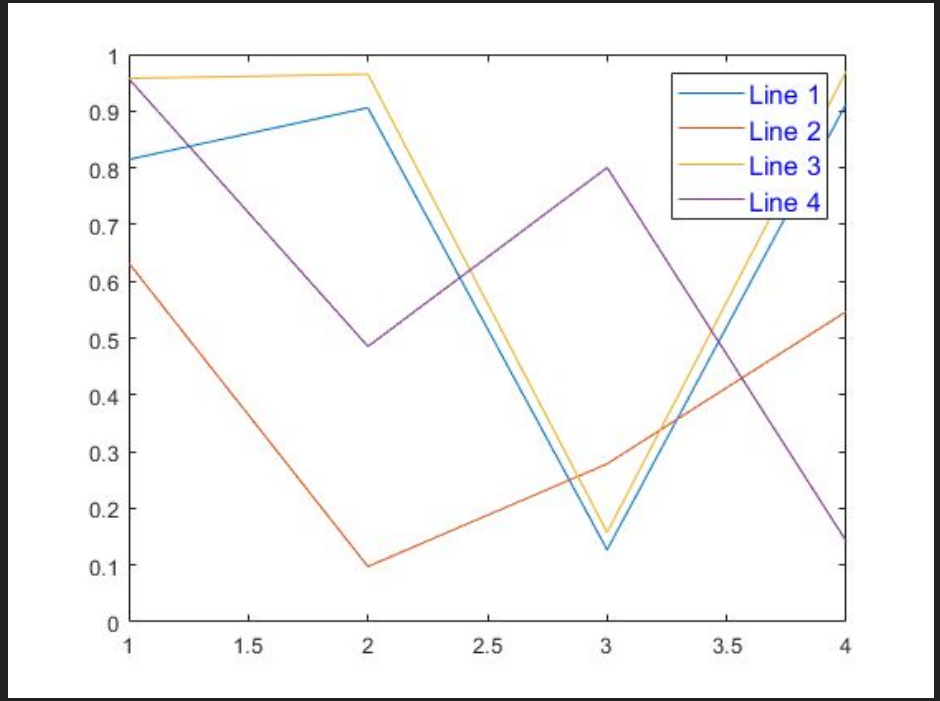
Se tivermos N *plots* na mesma janela podemos usar N argumentos para esta função tal que o argumento i vai ser a legenda da linha i. Assim se adicionarmos ao código original a seguinte linha,

```
legend("sin(x)")
```

Plots - Personalização



Plots - Personalização



Plots - Personalização

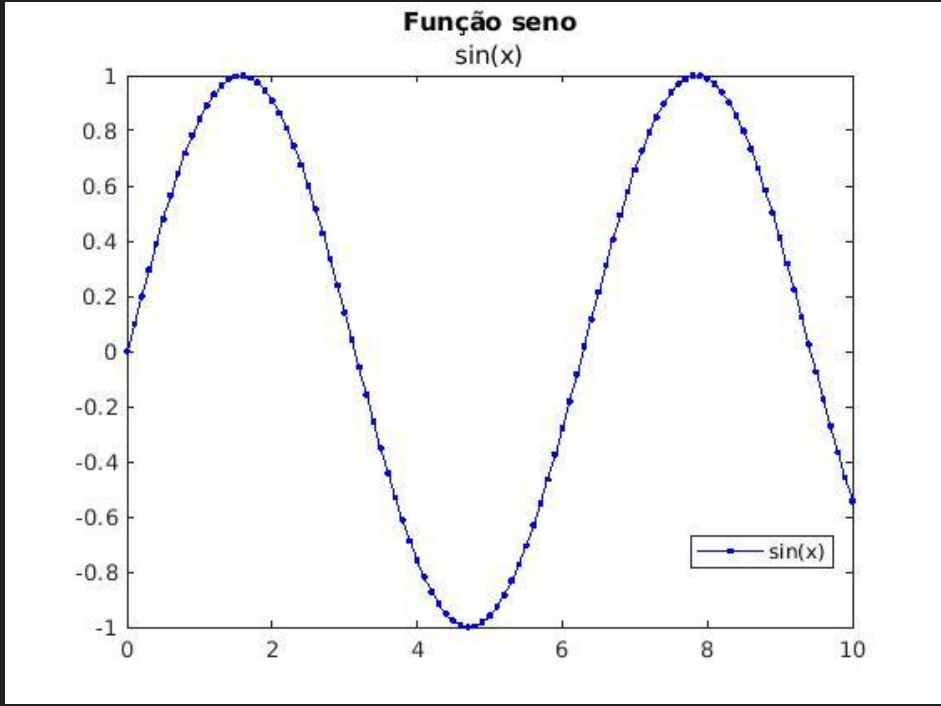
Nem sempre a legenda fica na melhor localização. Contudo há uma maneira de a mudar. Para tal usa-se:

```
legend(label1, ..., 'location', loc)
```

, onde `loc` pode assumir os valores dos pontos cardeais, como `"north"` ou `"southeast"`. Há também a opção `"best"`, onde o MATLAB escolhe a melhor localização para por a legenda. Vamos substituir a última linha do nosso código por

```
legend("sin(x)", 'location', "best")
```

Plots - Personalização



Plots - Personalização

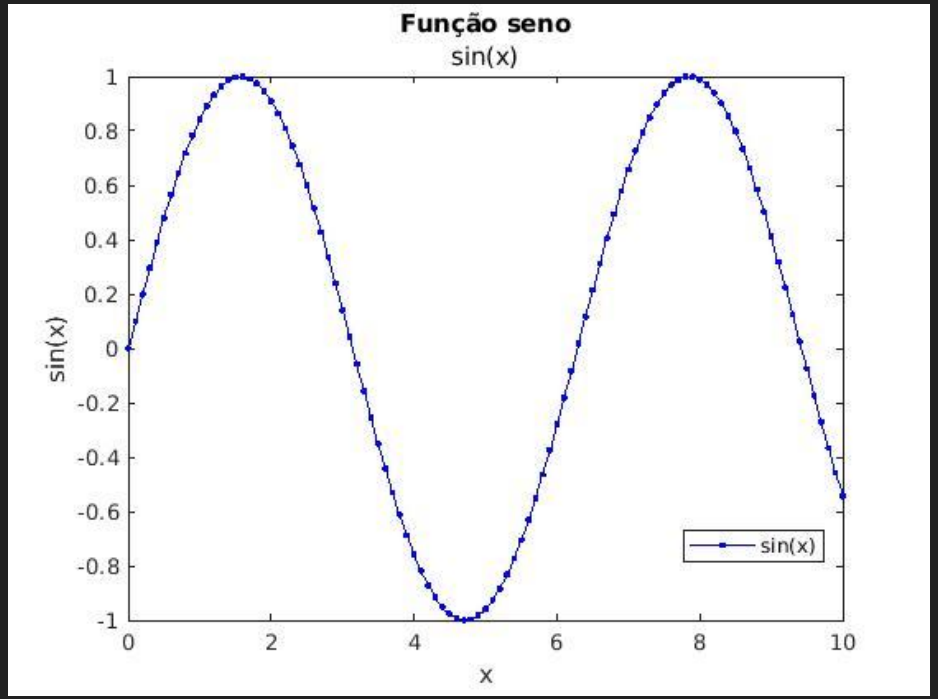
Por fim falta adicionar uma legenda a cada um dos eixos ordenados. Para tal usa-se as 3 funções seguintes

```
xlabel(label)

ylabel(label)

zlabel(label)
```

Ao adicionar as seguintes linhas ao código anterior

```
xlabel(“x”)

ylabel(“sin(x)”)
```

Plots - Personalização



Plots - Visualização de Informação

No MATLAB há a opção de delimitar os eixos dos xx, yy e zz. Podemos usar, respetivamente, as funções

```
xlim([min max])

ylim([min max])

zlim([min max])
```

Isto é útil uma vez que podemos querer olhar em detalhe para apenas uma secção do gráfico.

Plots - Visualização de Informação

Alternativamente e de uma forma mais compacta podemos usar a função `axis` que permite delimitar todos os eixos em uma só linha.

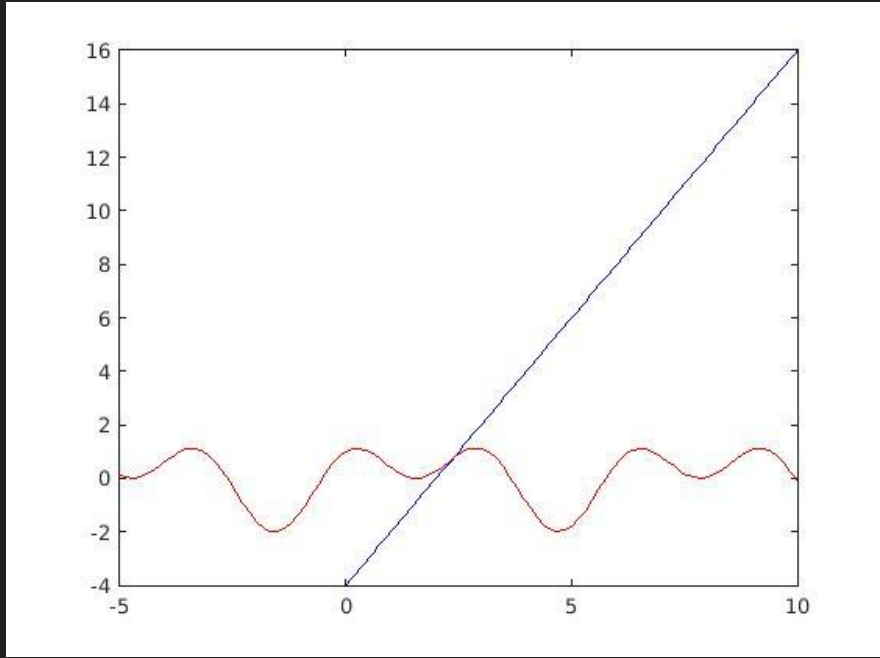
```
axis([xmin xmax ymin ymax zmin zmax])
```

Não é necessário usar todos os argumentos, contudo têm de estar nesta ordem específica, e ao definir um mínimo temos de automáticamente definir o máximo.

Plots - Visualização de Informação

Exemplo: consideremos este código,

```
x1 = 0:0.1:10;
x2 = linspace(-5, 10, 100);
y1 = 2 * x1 - 4;
y2 = cos(2 * x2) + sin(x2);
plot(x1, y1, '-b')
hold on
plot(x2, y2, '-r')
```



Plots - Visualização de Informação

Como mudar o código anterior de forma a poder focar o gráfico na interseção das duas linhas para obter uma estimação do ponto onde elas se cruzam?

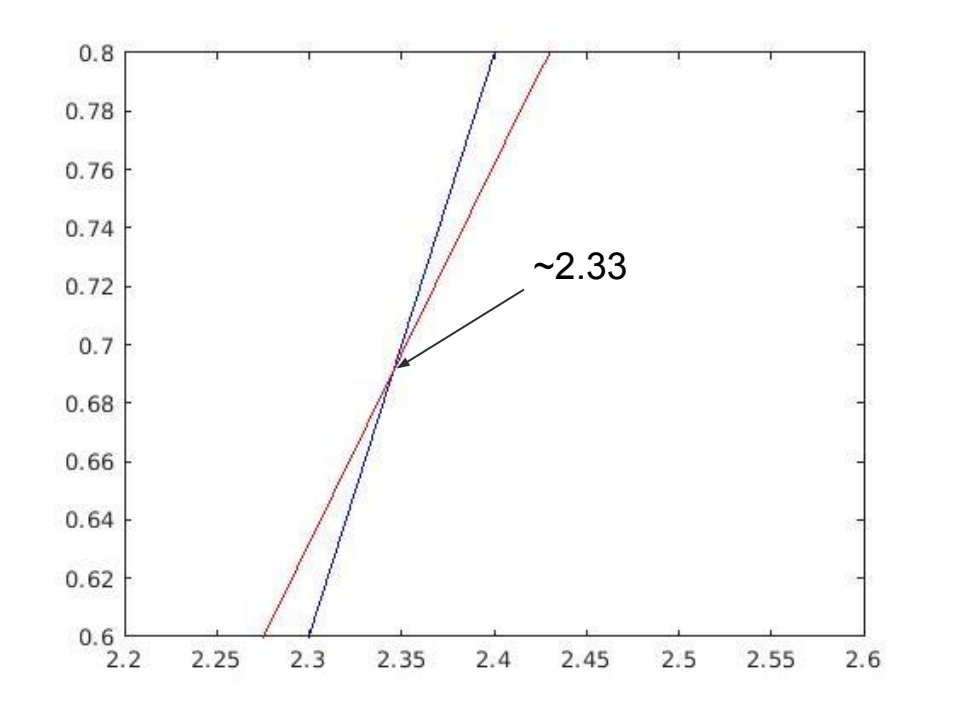
Usar as funções `xlim` e `ylim`, ou, alternativamente, usar a função `axis`. Deste modo basta adicionar ao código anterior

```
xlim([2.2 2.6])

ylim([0.6 0.8])
```

ou

```
axis([2.2 2.6 0.6 0.8])
```

Plots - Visualização de Informação



Plots - Várias Janelas

Por vezes é mais útil criar outra janela para por outro gráfico, ao invés de simplesmente dividir a janela atual. Para tal podemos usar o comando `figure`.

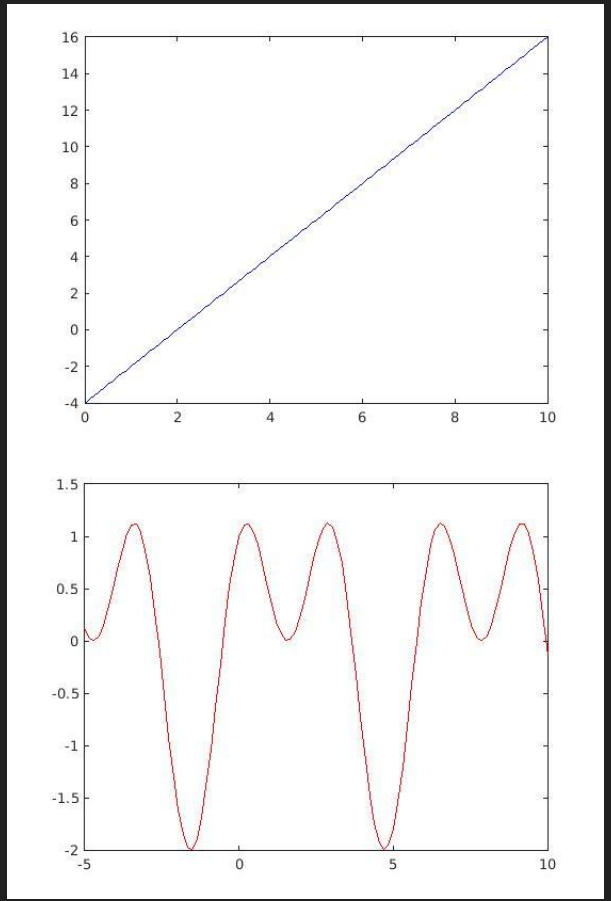
```
figure(n_janela)
```

Este comando cria uma janela nova. O argumento `n_janela`, denomina o número da janela. Todos os gráficos feitos depois da chamada deste comando ficam nessa respetiva janela.

Plots - Várias Janelas

Exemplo:

```
figure(1)
plot(x1, y1, '-b')
figure(2)
plot(x2, y2, '-r')
```



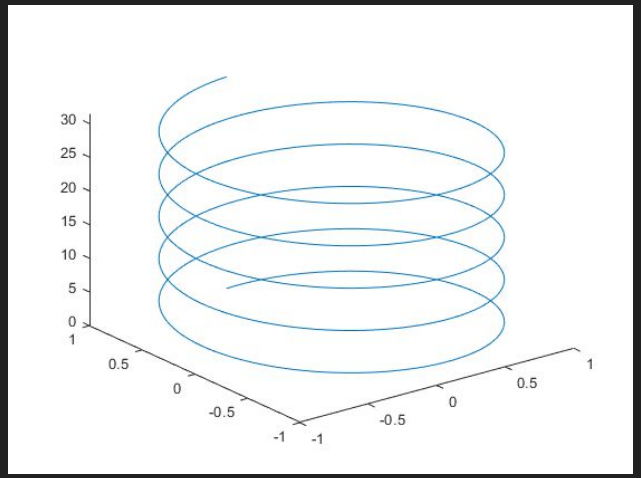
Plots - `plot3`

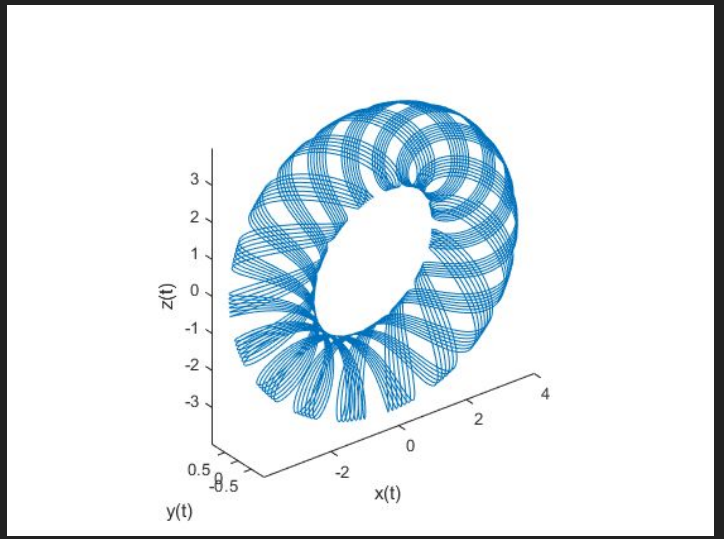
O MATLAB também permite fazer gráficos a 3 dimensões. O *plot* convencional pode ser feito usando a função

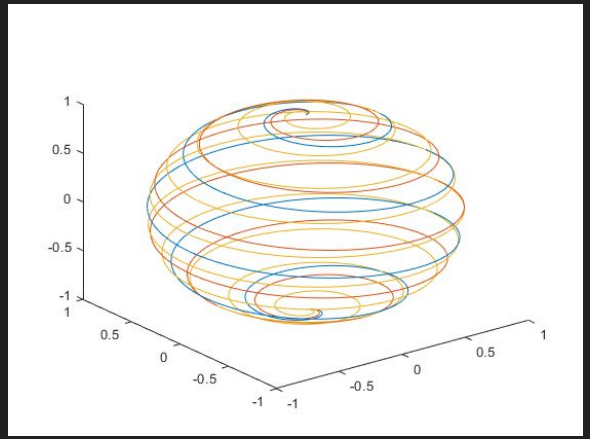
```
plot3(x1, y1, z1, lineSpec1, x2, y2, z2, lineSpec2, ...)
```

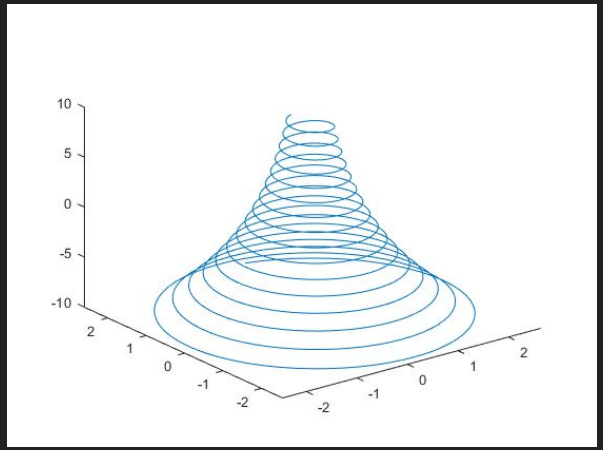
Todas as coisas já expostas sobre os *plots* a duas dimensões é válida para estes tipos de gráficos.

Plots - `plot3`


30
25
20
15
10
5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


z(t)
3
2
1
0
-1
-2
-3
0.5
0
-0.5
y(t)
-2
0
2
4
x(t)


1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


10
5
0
-5
-10
2
1
0
-1
-2
-2
-1
0
1
2

Plots - `surf` e `mesh`

No MATLAB também podemos criar gráficos a 3 dimensões que contêm superfícies de funções de duas variáveis (`z = f(x,y)`). Duas funções capazes de o fazer são

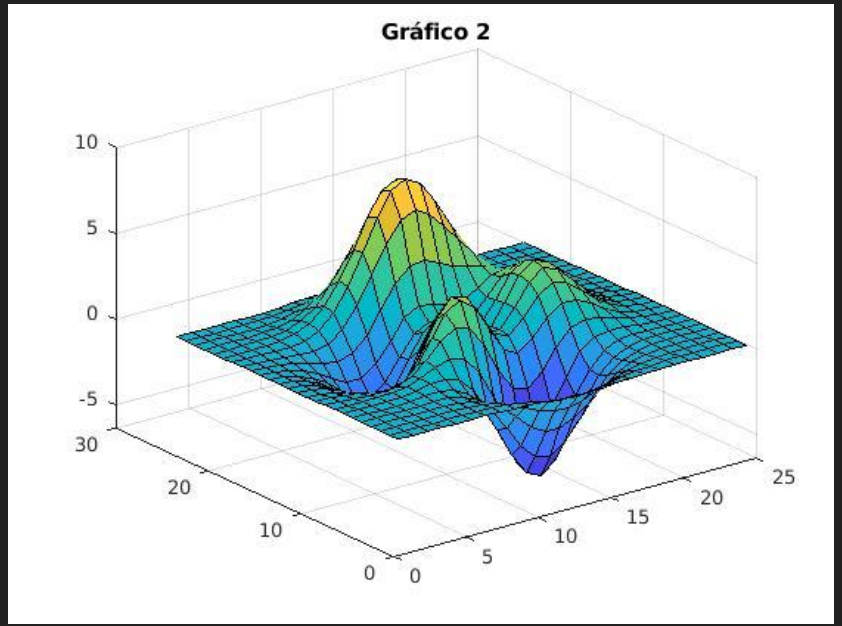
```
surf(X, Y, Z)
```

```
mesh(X, Y, Z)
```

O vetor `Z` define a altura de cada um dos pontos do plano, definido pelos vetores `X` e `Y`. Caso os dois primeiros parâmetros não sejam passados o MATLAB usa o seu plano *default*.

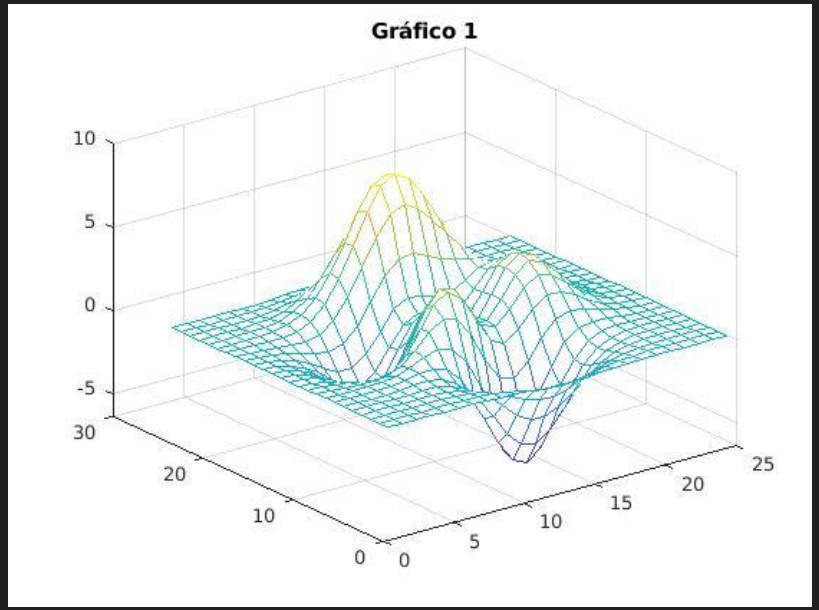
Plots - `surf` e `mesh`

A função `surf` cria uma superfície preenchida a 3 dimensões.



Plots - `surf` e `mesh`

A função `mesh` cria uma superfície de malha a 3 dimensões.



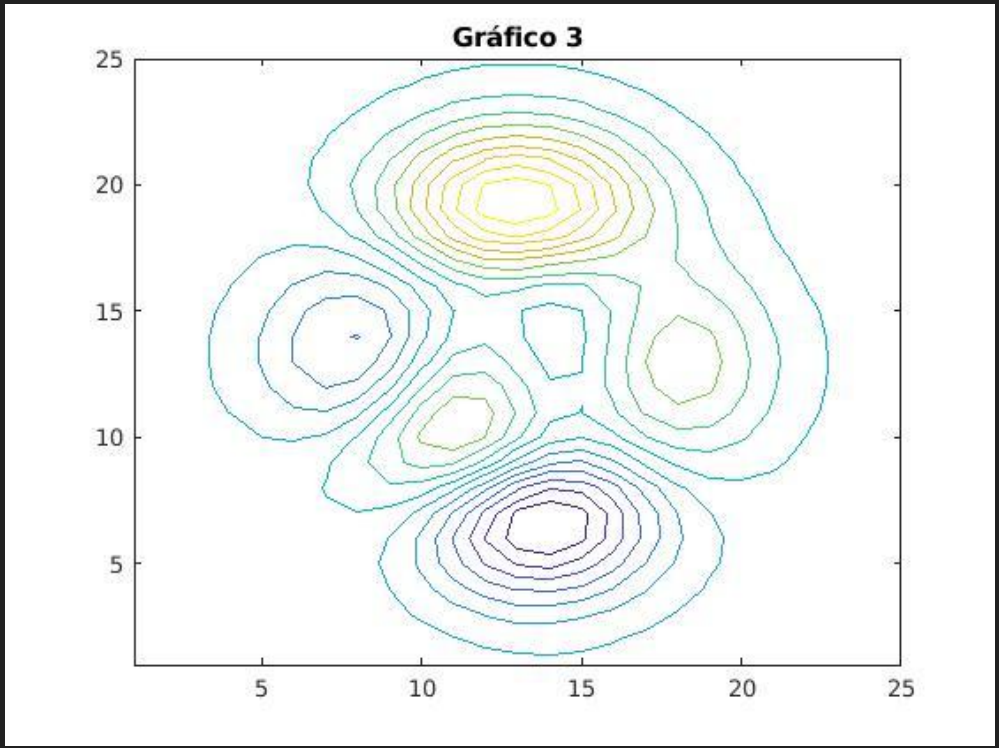
Plots - `contour`

Caso pretendamos fazer o contorno de uma superfície 3D num gráfico a duas dimensões, podemos usar a função

```
contour(X, Y, Z, levels)
```

Tal como nas duas últimas funções, o vetor `Z` define a altura de cada um dos pontos do plano, definido pelos vetores `X` e `Y`, e estes dois últimos não são obrigatórios. O argumento `levels` define quantas linhas serão representadas entre o nível mais baixo e o mais alto.

Plots - `contour`



Loops

Loops - `while`

Como todas as linguagens o MATLAB também dispõe de ciclos `for` e `while`. Comparando com python, a sintaxe dos ciclos `while` são idênticas.

python

```
while condição:
    do_stuff()
```

MATLAB

```
while condição
    do_stuff()
end
```

Loops

No MATLAB é obrigatório escrever a *keyword* `end` quando se conclui um ciclo. Serve o mesmo propósito que a indentação em python, é para o MATLAB saber qual o código executado pelo ciclo `while` ou `for`.

Loops - `for`

No caso do ciclo `for`, a sintaxe já é um bocado diferente. Vejamos a comparação,

python

```
for i in range(n):
    do_stuff()
```

MATLAB

```
for i = 1:n
    do_stuff()
end
```

Ambos os ciclos executam a função `do_stuff()` `n-1` vezes. É costume começar os ciclos no MATLAB em 1, visto que os índices dos *arrays* começam em 1 também.

Controlo de Fluxo

Controlo de Fluxo - `if`

Como em todas as linguagens, no MATLAB também é possível controlar o modo de como o nosso programa avança com condições lógicas, como condições `if -> else` ou `if -> elseif -> else`. Tal como nos ciclos, estes blocos têm de acabar com um `end`. A indentação também não é obrigatória, mas deixa o código mais legível.

Controlo de Fluxo - `if`

Têm a seguinte sintaxe:

```
if condição
    case1()
else
    case2()
end
```

```
if condição
    case1()
elseif condição
    case2()
else
    case3()
end
```

Funções

Funções

No MATLAB as funções têm uma sintaxe muito diferente de todas as outras linguagens. Para criar uma função em python faz-se

```
def my_func(arg1):
    return arg1 + 1
```

Funções

A mesma função no MATLAB tem a seguinte sintaxe,

```
function output = my_func(arg1)
    output = arg1 + 1;
end
```

Para um caso geral temos

```
function [y1,...,yN] = my_func(x1,...,xM)
```

Funções

```
function [y1,...,yN] = my_func(x1,...,xM)
```

Começando pela *keyword* `function`, para o MATLAB saber que o que vem a seguir se trata de uma função. Depois escrevemos o nome das variáveis que a nossa função vai retornar, pode ser apenas uma ou várias. No caso de ser várias, temos de as devolver como um *array*, como está em cima.

De seguida escrevemos o nome da nossa função e entre parênteses declaramos os argumentos da mesma.

Funções

Exemplo:

```
function y = f1(x)
    y = 2 * x;
end
```

```
function [a, b] = f2(x)
    a = 2 * x + 1;
    b = 0.5 * sin(x);
end
```

Funções

A maneira de chamar funções também é diferente no MATLAB. Para chamar as funções do slide anterior faz-se

```
x = 0:100;

y = f1(x);

[y1, y2] = f2(x);
```

Como o *output* da função `f2` é um *array*, quando a chama-mos podemos dividir o seu *output* por duas variáveis.

Funções - `return`

O MATLAB também tem a *keyword* `return`. A sua função é poder sair a qualquer ponto de uma função, não para devolver os valores de *output*, como no python. Ao invocar esta *keyword*, o MATLAB sai imediatamente da função e retorna os valores já calculados.

Funções - `return`

Exemplo:

```
function i = f(n_max)
    i = 0;
    while i < n_max
        i += 1;
        if i > 10
            return
        end
    end
end
```

Animações

FIM

João Inácio - inacio.joao16@ua.pt