Funções em Matlab

- Para além das funções (comandos) pré-definidas no matlab, o utilizador pode também criar as suas próprias funções

- O comando **function** permite criá-las. Uma função deve ser definida da seguinte forma:

```
function y=nomefuncao(par1,par2,par3....)
  % o código da função é escrito aqui
 y=par1+sqrt(par2)+....%  o(s) valor(es) da função deve(m) ser associado(s)
                          % ao(s) parâmetro(s) de saída
```

- A função pode ter vários parâmetros de entrada e de saída (no exemplo anterior apenas há um parâmetro de saída. Caso houvesse mais do que um parâmetro de saída, em vez de **function y=** seria **function [y1,y2,y3,]=**)

- O código relativo à função deve ser gravado num ficheiro **.m** e a directoria onde o mesmo é gravado deve fazer parte do *path* do matlab

Funções em Matlab

- A função pode depois ser chamada a partir da janela de comando do matlab ou a partir do interior de um ficheiro **.m**, tal como se tratasse de uma função pré-definida do matlab

- **NOTAS IMPORTANTES:**
  - Caso o nome dado à função seja diferente do nome do ficheiro **.m** criado aquando da definição da mesma, é este último nome que deve ser usado para invocar a função!!!
  - As variáveis definidas no interior da função não serão acessíveis a partir do espaço de trabalho do matlab (workspace)
  - Tenha em mente que uma função não atribui nenhum valor a nenhuma variável
  - A excepção a esta regra acontece se definir variáveis globais no corpo da função (faça **help global** para mais informação)
  - A execução da função pode ser interrompida com o comando **return**

Funções em Matlab

- Eis um exemplo de uma função definida pelo utilizador, que recebe como parâmetro de entrada um vector e representa uma espiral a três dimensões:

```
function [x,y,z] = spir3(t)
x = cos(20*t).*exp(-t.^2);
y = sin(20*t).*exp(-t.^2);
z = exp(-t.^2);
plot3(x,y,z);
```

- A função vai ser gravada no ficheiro **spir3.m** e invocada a partir da janela de comando do matlab da seguinte forma

```
spir3(0:0.01:5);
```

- Caso se pretenda reter os valores de x, y e z, deve-se invocar a função da seguinte forma

```
[x,y,z]=spir3(0:0.01:5);
```

Polinómios em Matlab

- Embora o matlab não permita trabalhar directamente com polinómios, dispõe de um conjunto de comandos destinados à sua manipulação

- No matlab, os polinómios são representados por vectores linha cujos elementos são os coeficientes das sucessivas potências do polinómio, ordenadas por ordem decrescente. Por exemplo, o polinómio p(s)=4s^5-5s^3+10s^2+9s-3 representa-se por p=[4 0 -5 10 9 -3]

- Por outro lado, dada uma matriz B, quadrada, pode-se obter o polinómio característico associado a esta matriz com o comando **poly**. Um exemplo:

```
>> B=[1 3 5;2 4 6;1 2 3]
B =
     1     3     5
     2     4     6
     1     2     3
>> p=poly(B)          %    p(s)=det(sI-B)
p =
    1.0000   -8.0000   -4.0000         0
```

Polinómios em Matlab

- As raízes de um polinómio podem ser obtidas com o comando **roots**. Considerando o polinómio anterior virá:

```
>> r=roots(p)
r =
         0
    8.4721
   -0.4721
```

- As operações de soma e subtracção efectuam-se de forma idêntica à dos vectores mas o produto e divisão de polinómios é efectuado com os comandos **conv** e **deconv**. Por exemplo, considere dois polinómios

```
>> a=[2 4 7], b=[4 8 11]
a =
     2     4     7     3
b =
     4     8    11
```

Polinómios em Matlab

- O seu produto e divisão são iguais a

```
>> produto=conv(a,b)
produto =
     8    32    82   112   101    33
>> [quociente,resto]=deconv(a,b)
quociente =
    0.5000         0                          % q(s)=0.5s
resto =
         0         0    1.5000    3.0000    % r(s)=1.5s+3
```

- Por vezes é útil, dado um conjunto de pares ordenados (x,y), obter um polinómio que permita representar de forma aproximada a função y (pense por exemplo a utilidade que tem obter uma expressão analítica que permita conhecer o valor da indutância de uma bobina em função da corrente que a percorre, se a mesma tiver um núcleo saturável)

Polinómios em Matlab

- Nestas situações, o matlab com o comando **polyfit**, fornece-nos os coeficientes do polinómio que melhor se ajusta aos dados fornecidos

- A sintaxe é a seguinte: **polyfit(x,y,n)** , onde x e y são vectores que contêm as coordenadas dos pontos a interpolar e que servirão para gerar os coeficientes do polinómio, que terá grau n

- Um exemplo: vamos admitir que queremos arranjar um polinómio de grau 5 que permita obter valores idênticos àqueles fornecidos pela função raiz quadrada. Isto pode ser conseguido da seguinte forma:

```
x=0:0.01:5;
y=sqrt(x);
p=polyfit(x,y,5);
```

- Se agora fizer **plot(x,y,x,polyval(p,x))** poderá observar quanto boa é a interpolação produzida

Integração Numérica em Matlab

- Para efectuar uma integração numérica no matlab, usam-se habitualmente dois comandos (existem outros):
  - **quad**
  - **quadl**
- Como exemplo, pretende-se calcular o integral de uma dada função. Comecemos por definir a função m(x)=3x^2+5 da seguinte forma:

```
function y=m(x)
y=3*x.^2+5;          % definir a função num ficheiro .m
```

- Nota importante: repare no facto da função ter de aceitar como parâmetro de entrada um vector x e devolver também um vector como parâmetro de saída!! Daí a razão de se usar o símbolo '.' antes da potenciação
- O integral da função anterior, no intervalo [5,10], pode então ser calculado através de:

```
> quad('m',5,10) % método de cálculo: adaptação recursiva da regra de Simpson
ans =
  900.0000
```

Mínimos, Máximos e Zeros em Matlab

- Por vezes, dada uma função, é necessário achar os pontos onde ocorrem os seus mínimos e máximos locais, bem como os zeros
- Para calcular o valor da variável independente onde ocorre um mínimo da função (de uma variável apenas), num dado intervalo especificado, usa-se o comando **fminbnd.** A sintaxe é **fminbnd('fun',x1,x2)** onde [x1,x2] é o intervalo onde será procurado o mínimo local da função *fun*
- No caso da função ter mais do que uma variável usa-se o comando **fminsearch.** Neste caso a sintaxe é **fminsearch('fun',x0)**
- **Tenha algum cuidado ao usar estas funções, pois a função fun, no intervalo especificado, pode ter mais do que um mínimo local**
- Para encontrar os zeros de uma função usa-se o comando **fzero**. Sintaxe: **fzero('fun',x0)**

Equações Diferenciais em Matlab

- O matlab disponibiliza um conjunto de comandos destinados a obter a solução numérica de equações diferenciais

Comando	Precisão	Descrição
ode45	**Média**	O primeiro método a testar (Runge-Kutta (4, 5)). Resolve o problema na maioria dos casos
ode23	**Pequena**	Runge-Kutta (2, 3). Usar quando a precisão não é importante
ode113	**Pequena até elevada**	Usar em problemas onde o esforço computacional é grande
ode15s	**Pequena até média**	Usar se ode45 for lento, devido ao facto do problema ser duro (“stiff”)
ode23s	**Pequena**	Para resolver sistemas duros, onde a precisão não é importante
ode23t	**Pequena**	
ode23tb	**Pequena**	Para resolver problemas duros

Equações Diferenciais em Matlab

- A sintaxe mais simples para resolver uma equação diferencial (ou sistema de equações diferenciais) é **[t,y]=solver(funcao,intervalotempo,y0)**, onde **solver** representa uma das funções que constam da tabela anterior, **funcao** representa uma função (gravada num ficheiro **.m**) que descreve a(s) equação(ões) diferencial(is) a resolver, **intervalotempo** representa o intervalo ao longo do qual vai ser efectuada a integração numérica, e **y0** representa o vector com as condições iniciais do problema

- Se **intervalotempo** contiver mais do que dois elementos, o matlab devolve o valor de **y** apenas nesses pontos

- Os vectores de saída **t** e **y** contêm os valores da variável independente e os valores assumidos pela função **y**

Equações Diferenciais em Matlab

- Como exemplo, vamos procurar uma solução numérica para a equação diferencial y'=-2ty^2, nos pontos t=0, 0.25, 0.5, 0.75, 1, com a condição inicial y(0)=1. A solução analítica para esta equação é y(t)=1/(1+t^2) e servirá para aferir a precisão da solução numérica dada pelo matlab

- O primeiro passo para a resolução da equação anterior consiste na criação de um ficheiro **.m** com a descrição da mesma:

```
function dy = eq1(t,y)
% Ficheiro eq1.m com a descrição da equação y' = -2ty^2
dy = -2*t*y(1)^2;     % pode-se escrever y em vez de y(1)
```

Equações Diferenciais em Matlab

- Agora gera-se o vector **intervalotempo** contendo os pontos onde a solução é requerida bem como as condições iniciais do problema:
  intervalotempo=[0 .25 .5 .75 1]; y0 = 1;
- Finalmente pode-se chamar a função que irá resolver a equação diferencial:
  [t, y]=ode45('eq1', intervalotempo,y0);
- No vector **y** encontra-se a solução da equação diferencial nos instantes de tempo determinados pelos elementos de **t** (neste caso t=intervalotempo)
- Pode representar-se graficamente a solução com o comando **plot(t,y)** e verificar que a solução numérica é muito próxima da solução exacta

Equações Diferenciais em Matlab

- Vamos agora resolver um sistema de duas equações diferenciais (funções *f* e *g*): f '(t) = f(t) – 4g(t); g'(t) = -f(t) + g(t), com as condições iniciais f(0) = 1; g(0) = 0
- Cria-se o ficheiro **meusistema.m** com o seguinte código

```
function dydt=meusistema(t,y)
f=y(1);
g=y(2);
dfdt=f-4*g;             % primeira equação diferencial
dgdt=-f+g;              % segunda equação diferencial
dydt=[dfdt; dgdt];   % é forçoso que dydt seja um vector coluna
```

- Agora pode-se resolver o sistema de equações diferenciais da seguinte forma:

```
[t,funcoes]=ode45('meusistema',[-2 0.5],[1 0]);
```

- No vector **t** e nas duas colunas da matriz **funcoes** estão agora armazenados os instantes de tempo (no intervalo [-2,0.5]) e as soluções das funções **f** e **g**, respectivamente, que satisfazem o sistema de equações diferenciais dado