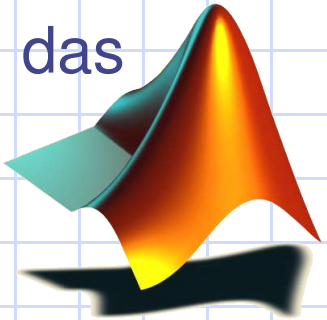
GRÁFICOS: 2D e 3D

Funções gráficas 2D elementares:

O MATLAB dispõe de 4 funções básicas para criar gráficos 2D. Estas diferenciam-se principalmente pelo *tipo de escala* que utilizam para os eixos. Estas quatro funções são as seguintes:

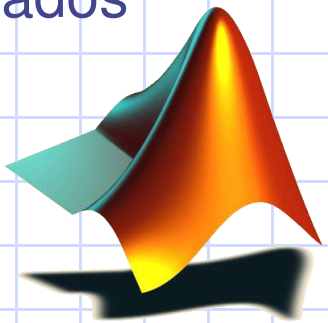
- plot() cria um gráfico a partir de vectores e/ou colunas de matrizes, com escalas lineares em ambos os eixos.
- loglog() idem com escala logarítmica.
- semilogx() idem com escala linear no eixo das ordenadas e logarítmica no eixo das abcissas
- semilogy() idem com escala linear no eixo das abcissas e logarítmica no eixo das ordenadas.



GRÁFICOS: 2D e 3D

Existem funções para atribuir títulos ao gráfico, aos eixos, desenhar uma quadrícula auxiliar, introduzir texto, etc.

- title('título') atribui um título ao gráfico
- xlabel('tal') atribui uma legenda ao eixo das abcissas (com ***xlabel off*** desaparece)
- ylabel('qual') idem para o eixo das ordenadas (com ***ylabel off*** desaparece)
- text(x,y,'texto') introduz 'texto' no lugar especificado pelas coordenadas **x** e **y** (se **x** e **y** são vectores, o texto repete-se por cada par de elementos)
- gtext('texto') introduz 'texto' com ajuda do rato: legend() define legendas para as distintas linhas ou eixos utilizados na figura
- grid activa a grelha (com ***grid off*** desaparece)



plot() é a função chave de todos os gráficos 2D em MATLAB. Já foi referido que o elemento básico dos gráficos bidimensionais é o **vector**. Utilizam-se também sequências de 1, 2 ou 3 caracteres para indicar *cores e tipos de linha*. A função ***plot()*** não faz mais do que desenhar vectores.

Exemplo1:

```
» x=[1 3 2 4 5 3]
x =
1 3 2 4 5 3
» plot(x)
```

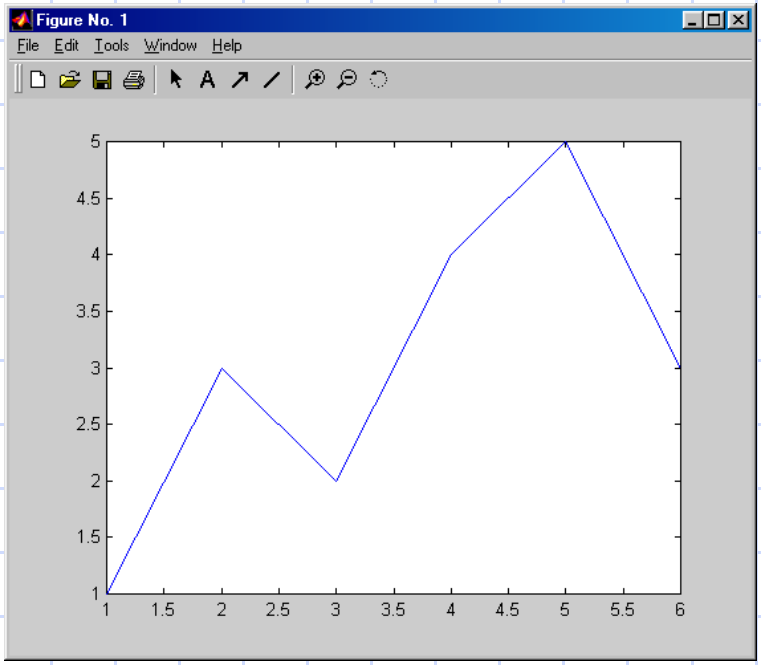
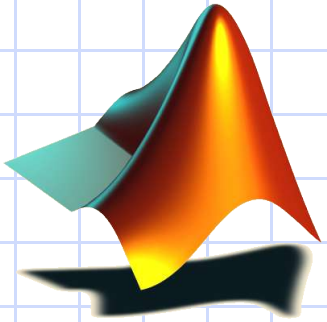


Figura 5.1. Gráfico do vector x=[1 3 2 4 5 3].

FUNÇÃO *plot*

Exemplo2

```
x=0:pi/90:2*pi;
y=sin(x).*cos(x);
plot(x,y)

grid on
grid off
xlabel('eixo x (em radianos)')
ylabel('eixo y')
title('y=sen(x)*cos(x)')
```

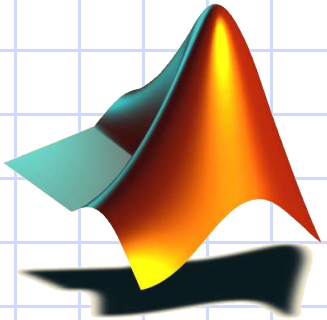


FUNÇÃO *plot*

É possível incluir no título ou na legenda dos eixos o valor de uma variável numérica. Já que o argumento dos comandos *title*, *xlabel* e *ylabel* é uma variável caracter, é preciso transformar as variáveis numéricas:

- *int2str(*n*)* converte o valor da variável inteira n em caracter

- *num2str(*x*)* converte o valor da variável real ou complexa x em caracter



Texto sobre o gráfico

gtext('texto')
text(x,y,'texto a representar')

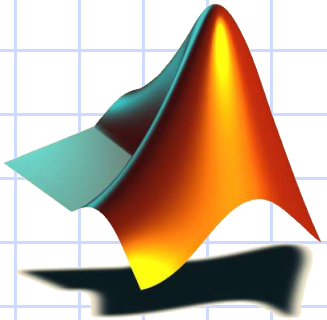
Calcular as coordenadas de pontos sobre a curva

ginput(n)
[x,y]=ginput(n)

Selecção do traço e cor da curva

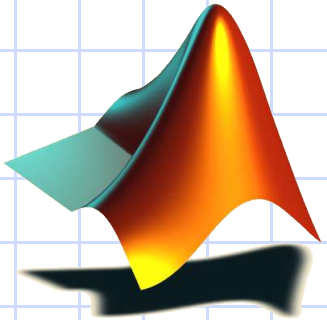
plot(x,y,'opcao')

hold on
hold off



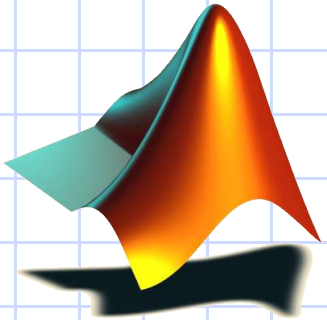
Opções de *plot*

y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		



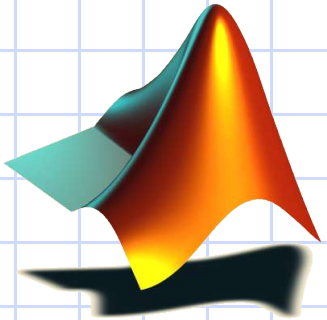
Exemplo3: Calcular graficamente as soluções da equação $\frac{2x-\cos(2x)}{2}=0.4$

```
teta=0:pi/360:pi/4;
f1=(2*teta-cos(2*teta))/2;
f2=0.4*ones(size(f1));
figure
plot(teta,f1,'g--',teta,f2,'r')
axis square
xlabel('Angulo (radianos)')
gtext('2x-cos(2x))/2')
text(0.2,0.43,'y=0.4')
[teta0,y0]=ginput(1)
title(['Raiz aproximada=',num2str(teta0)])
```



Selecção da escala dos eixos

- *axis*([x0 x1 y0 y1])
- *axis auto*: escala por defeito
- *axis off*: desactiva a legendagem dos eixos, desaparecendo dos eixos as suas legendas e a grid.
- *axis on*: activa de novo
- *axis xy*: sistema de coordenadas cartesianas com origem no canto inferior esquerdo, eixo ox da esq. para a dta. e oy de baixo para cima.
- *axis equal*: mesmos factores de escala para os dois eixos
- *axis square*: envolve com um quadrado a região delimitada pelos eixos actuais.



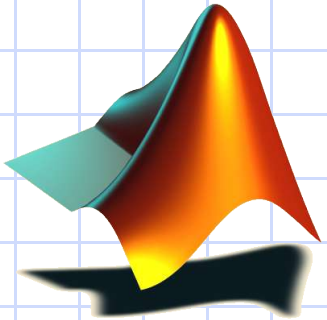
Impressão de gráficos

```
print  -dps     % PostScript for black and white printers
       -dpsc    % PostScript for color printers

       -deps    % Encapsulated PostScript
       -depsc   % Encapsulated Color PostScript
```

print -djpeg<nn> % imagem JPEG, nn = nível de qualidade

Exemplo:

print -djpeg90 figura1 (nn=75 por defeito)



GRÁFICOS: 2D e 3D

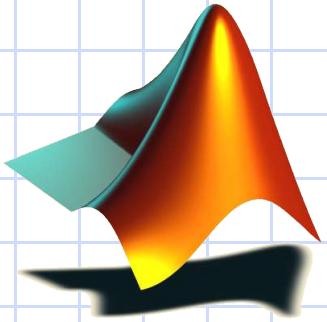
Exemplo4: plot(vector,Matriz)

```
x=0:pi/180:2*pi;
y=sin(x);
z=cos(x);
plot(x,y,x,z)
A=[y' z']
plot(x,A)
```

A função eval utiliza-se para funções definidas por um caracter.
y=eval(’caracter’)

Exemplo5:

```
f=‘sin(x)-2*cos(x)’;
x=0:pi/90:2*pi;
y=eval(f);
plot(x,y)
axis([0 6 0 2.4]);gtext(‘sen(x)-2cos(x)’)
```



GRÁFICOS: 2D e 3D

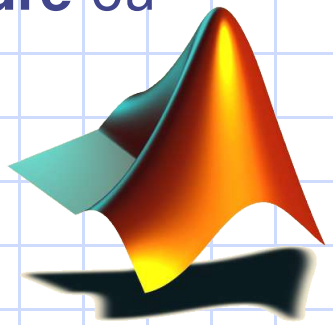
A função *fplot* utiliza-se com funções definidas com um caracter: **fplot=(f,[0 2*pi ymin ymax])**

Exemplo6:
f='sin(x)-2*cos(x)';
fplot(f,[0 2*pi],'g—')

Esta função pode utilizar-se também na forma:
[x,y]=fplot(f,[0 2*pi ymin ymax])
e neste caso são devolvidos os vectores **x** e **y**, mas não é desenhado nada.

Para chamar uma nova figura usamos o comando **figure** ou **figure(n)** para nos referirmos uma figura já feita.

Apagar a figura actual **clf**
close all apaga todas as figuras **close(figure(n))** a n

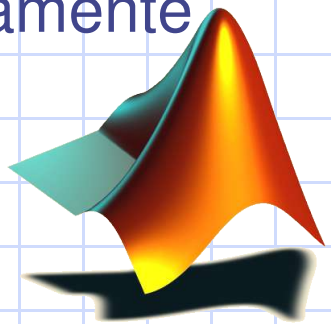


FUNÇÃO *subplot*

Uma janela gráfica pode ser dividida em **m** partições horizontais e **n** verticais, com o objectivo de nela representar múltiplos gráficos. Cada uma destas subjanelas tem os seus próprios eixos, apesar de outras propriedades serem comuns a toda a figura. A forma geral deste comando é:

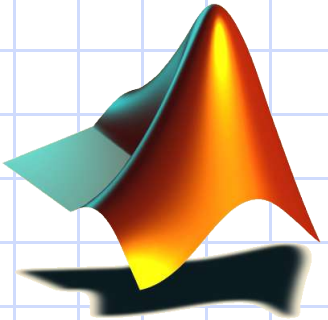
*subplot(***m**,**n**,**i***)*

onde **m e n** são o número de subdivisões em linhas e colunas, e **i** é a subdivisão que se converte em activa. As subdivisões são numeradas consecutivamente começando pelas da primeira linha.



Exemplo7:

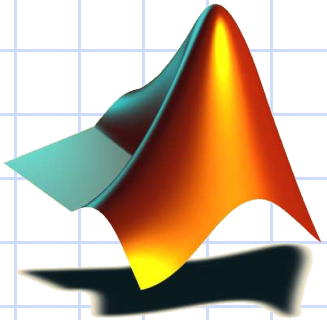
```
subplot(121)
f='sin(x)-2*cos(2*x)';
fplot(f,[0 2*pi])
legend('sen(x)-2cos(2x)')
subplot(122)
fplot('sin',[0 4*pi],'r')
legend('sen(x)')
```



GRÁFICOS: 2D e 3D

Outras funções gráficas 2D

- bar() cria diagramas de barras.
- barh() diagramas de barras horizontais.
- bar3() diagramas de barras com aspecto 3D.
- bar3h() diagramas de barras horizontais com aspecto 3D.
- pie() gráficos em forma de "tarte".
- pie3() gráficos em forma de "tarte" e aspecto 3D.
- area() similar a ***plot()***, mas para ordenadas de 0 a y.
- stairs() função análoga a ***bar()*** sem linhas internas.
- errorbar() representa sobre um gráfico –mediante barras– valores de erros.
- compass() desenha os elementos de um vector complexo como um conjunto de vectores partindo de uma origem comum.
- feather() desenha os elementos de um vector complexo como um conjunto de vectores partindo de origens uniformemente espaçadas sobre o eixo de abcissas
- hist() desenha histogramas de um vector.

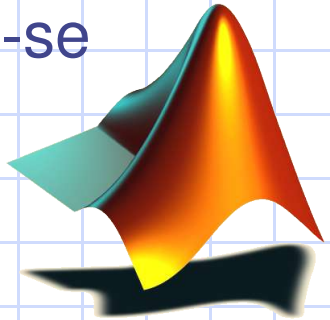


Representação de polígonos

Função especial para desenhar polígonos planos, preenchendo-os de uma determinada cor.
A forma geral é a seguinte:

» **fill(x,y,c)**

• Se **c é** um caracter de cor ('r','g','b','c','m','y','w','k'), ou um vector de valores [r g b], o polígono é preenchido de forma uniforme e com a cor especificada.
•Se **c é** um vector da mesma dimensão que **x** e **y**, os seus elementos transformam-se de acordo com um mapa de cores determinado, e o preenchimento do polígono –não uniforme neste caso – obtém-se interpolando entre as cores dos vértices.



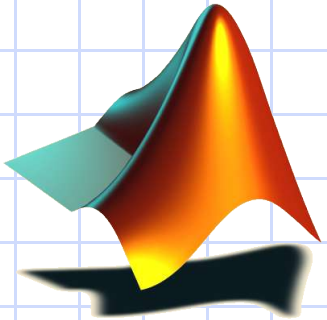
Este comando com matrizes aplica-se da seguinte forma:

» **fill(A,B,C)**

onde **A e B** são matrizes do mesmo tamanho. Neste caso desenha-se um polígono por cada par de colunas das matrizes. **C** pode ser um vector linha de cores uniformes para cada polígono, ou uma matriz do mesmo tamanho que as anteriores para obter cores de preenchimento por interpolação.

Exemplo8:

```
x=[1,2,1,0];
y=[0,1,2,1];
figure
fill(x,y,'r')
title('rombo')
```



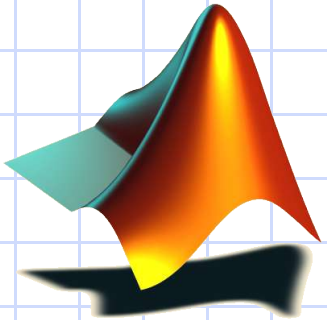
Funções gráficas 3D elementares:

A função **plot3 é** análoga à sua homóloga bidimensional **plot**. A sua forma mais simples é a seguinte:

» **plot3(x,y,z)**

Exemplo9:

```
teta=0:pi/80:8*pi;
x=1+2*cos(teta);
y=1+2*sin(teta);
z=4*teta;
plot3(x,y,z)
axis([-1 3 -1 3 0 120]);
xlabel('eixo x')
ylabel('eixo y')
zlabel('eixo z')
```



Representação gráfica de superfícies

mesh(x,y,Z),

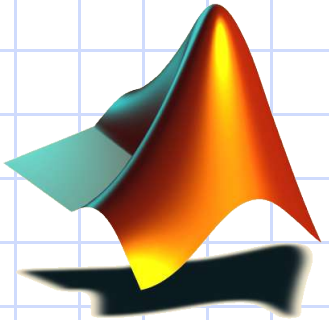
Criação de uma malha: **[X, Y]=meshgrid(x,y)**

Gráfico da malha construida sobre a superfície **z**:
mesh(X,Y,Z), **meshz(X,Y,Z)**

Ainda faz uma projecção sobre o plano **z**=0:
meshc(X,Y,Z), linhas de contorno no plano z=0

Exemplo10:

```
x=[0:2:200];y=[0:50];
% Obtemos a malha do domínio
[X Y]=meshgrid(x,y);
length(x),length(y)
size(X), size(Y)
Z=X.^2-Y.^2;
figure(1);mesh(X,Y,Z)
figure(2);meshz(X,Y,Z)
figure(3);meshc(X,Y,Z)
```



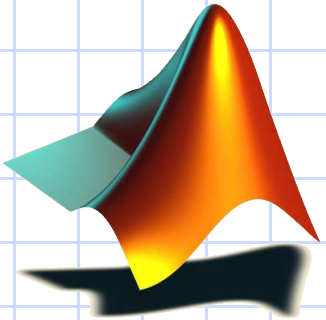
GRÁFICOS: 2D e 3D

O mesmo com **surf(X,Y,Z)**, **surfc(X,Y,Z)**, **surfl(X,Y,Z)**

Uma forma diferente de representar funções tridimensionais é por meio de isolinhas ou curvas de nível. Com **contour(x,y,Z)** e com **contour3 (X,Y,Z)** geramos as curvas de nível de uma superfície.
Existem legendas especiais: primeiro precisamos de saber os valores do contorno
cs=contour(Z) e logo pomos **clabel(cs)** ou **clabel(cs,v)**

pcolor(Z) desenha uma projecção com sombras de cor sobre o plano. A gama de cores está em consonância com as variações da matriz Z.

As funções **surf** e **pcolor** têm diversas possibilidades referentes à forma em que são representadas as facetas ou polígonos coloridos. As três possibilidades são as seguintes:
• shading flat: sombreado com cor constante para cada polígono. Este sombreado chama-se plano ou *fla*t.
• shading interp: estabelece que o sombreado se calcula por interpolação de cores entre os vértices de cada faceta. Chama-se também sombreado de Gouraud.
• shading faceted: consiste num sombreado constante com linhas pretas sobrepostas. Esta é a opção por defeito.

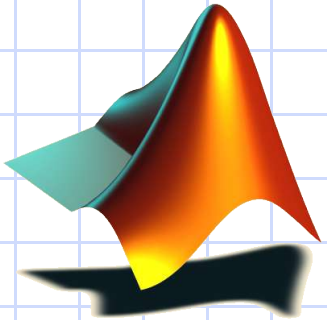


GRÁFICOS: 2D e 3D

Quando se pretende desenhar uma figura com um determinado mapa de cores estabelece-se uma correspondência (ou um *mapping*) entre os valores da função e as cores do mapa de cores.

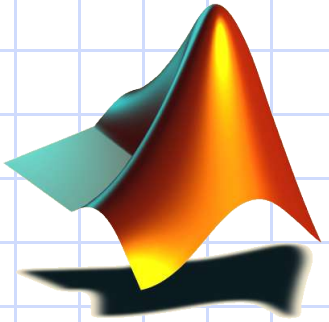
» caxis([cmin, cmax]) % escala o mapa de cores
» colormap(opcao) (distintas escalas de cores)

hsv	- Hue-saturation-value color map.
hot	- Black-red-yellow-white color map.
gray	- Linear gray-scale color map.
bone	- Gray-scale with tinge of blue color map.
copper	- Linear copper-tone color map.
pink	- Pastel shades of pink color map.
white	- All white color map.
flag	- Alternating red, white, blue, and black color map.
lines	- Color map with the line colors.
colorcube	- Enhanced color-cube color map.
vga	- Windows colormap for 16 colors.
jet	- Variant of HSV.
prism	- Prism color map.
cool	- Shades of cyan and magenta color map.
autumn	- Shades of red and yellow color map.
spring	- Shades of magenta and yellow color map.
winter	- Shades of blue and green color map.
summer	- Shades of green and yellow color map.



GRÁFICOS: 2D e 3D

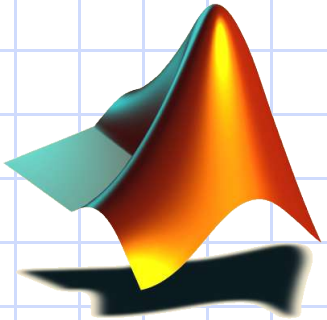
- colorbar('horiz'), colorbar('vertical') % barra com a escala de cores.
- brighten ajusta o brilho de cor do mapa de cores
- [X,Y,Z]=*sphere(*n*)* faz uma representação da esfera unitária com n pontos de discretização
- [X,Y,Z]=*cylinder(*rad,n*)* faz uma representação de um cilindro unitário com n pontos igualmente espaçados cuja secção é dada pela curva cujos raios se guardam no vector rad. (1,1) e 20 valores por defeito.
- fill3(x,y,z,c) % polígonos tridimensionais



Manipulação de gráficos

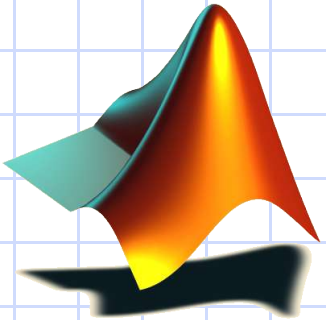
view: view(azimut, elev), view([xd,yd,zd]).
view(2)
view(3)
rotate(**h**,**d**,**a**) ou rotate(**h**,**d**,**a**,**o**)
h é o objecto, **d** é um vector que indica a direcção e **a** é um ângulo, ou a origem de rotação

Ao representar funções tridimensionais, às vezes também são úteis os *NaN*s. Quando uma parte dos elementos da matriz de valores **Z** são *NaN*s, essa parte da superfície não é representada, permitindo ver a restante superfície.



GRÁFICOS: 2D e 3D

hidden off desactiva a representação de linhas ocultas
hidden on activa a representação de linhas ocultas

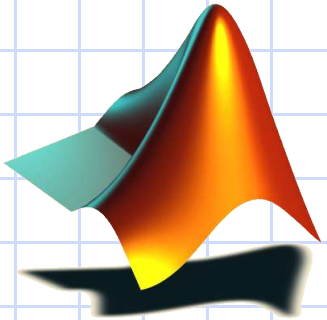


Transformação de coordenadas

```
[ang,rad]=cart2pol(x,y) %De cartesianas para polares
[ang,rad,z]=cart2pol(x,y,z) %De cartesianas para cilíndricas

[x,y]=pol2cart(ang,rad) %De polares para cartesianas
[x,y,z]=pol2cart(ang,rad,z) %De cilíndricas para cartesianas

[angx,angz,rad]=cart2sph(x,y,z) %De cartesianas para esféricas

[x,y,z]=aph2cart(angx,angz,rad) %De esféricas para cartesianas
```

Exemplo11:

```
% cilíndricas
[ang,rad,z]=cart2pol(sqrt(3),1,2)
% esféricas
[ang1,ang2,rad1]=cart2sph(sqrt(3),1,2)
```

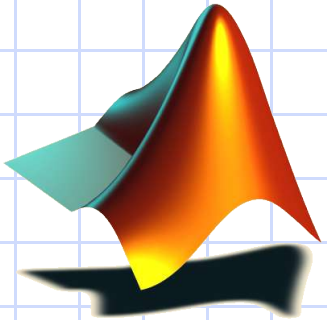


Criação de animações

Para preparar pequenas animações podem-se utilizar as funções **movie**, **moviein** e **getframe**. Uma animação é composta de várias imagens, denominadas *frame*s. A função **getframe** devolve um vector coluna com a informação necessária para reproduzir a imagem que se acaba de representar na figura ou janela gráfica activa, por exemplo com a função **plot**. O tamanho deste vector coluna depende do tamanho da janela, mas não da complexidade do gráfico. A função **moviein(n)** reserva memória para armazenar **n** frames. Uma vez criada a animação pode-se visualizá-la com o comando **movie**.

Exemplo12:

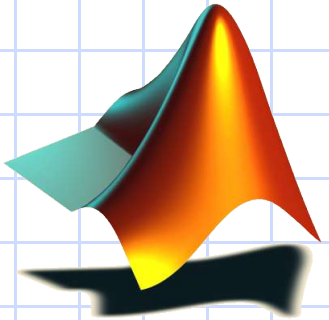
```
for j=1:10
x=0:0.01:2*pi;
plot(x,sin(j*x)/2)
M(j)=getframe;
end
movie(M,10)
```



Exercício5.1: Consideremos a equação de *Van der Waals*. Considera-se o benzeno para o qual *a*=18.78 atml^2/mol^2, *b*=0.1208 l/mol. Representar sobre um mesmo gráfico os dois subgráficos correspondentes a:
Isotérmicas de 100, 200, 300 e 400 ºC
Isobáricas de 25, 35, 45 e 55 atm
Cada curva deve representar-se com traço diferenciado, com o texto que indique a isolinha representada, assim como o título do gráfico e a legenda dos eixos.

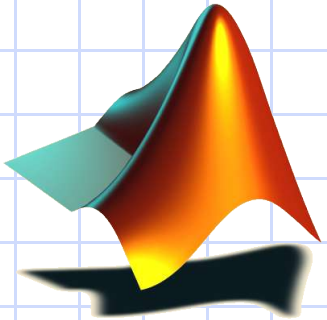
$$\left(P+\frac{a}{V^2}\right)(V-b)=RT$$

R=0.0821 e V=[2:100].



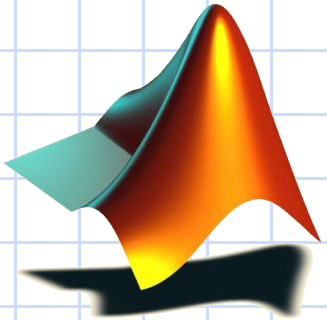
Exercício5.1:

```
a=18.78;
b=0.1208;
R=0.0821;
subplot(1,2,1)
T=[373:100:673];
V=[2:100];
fac1=R./(V-b);
fac2=a./V.^2;
P=zeros(4,length(V));
P(1,:)=T(1)*fac1-fac2;
P(2,:)=T(2)*fac1-fac2;
P(3,:)=T(3)*fac1-fac2;
P(4,:)=T(4)*fac1-fac2;
plot(V,P(1,:),'-',V,P(2,:),'--',V,P(3,:),':',V,P(4,:),'-.')
title('Equacao de Van der Waals: Isotermicas')
xlabel('Volume, ltr.')
ylabel('Pressao, atm.')
axis([0, 50,0,15])
legend('T=100ºC','T=200ºC','T=300ºC','T=400ºC')
```

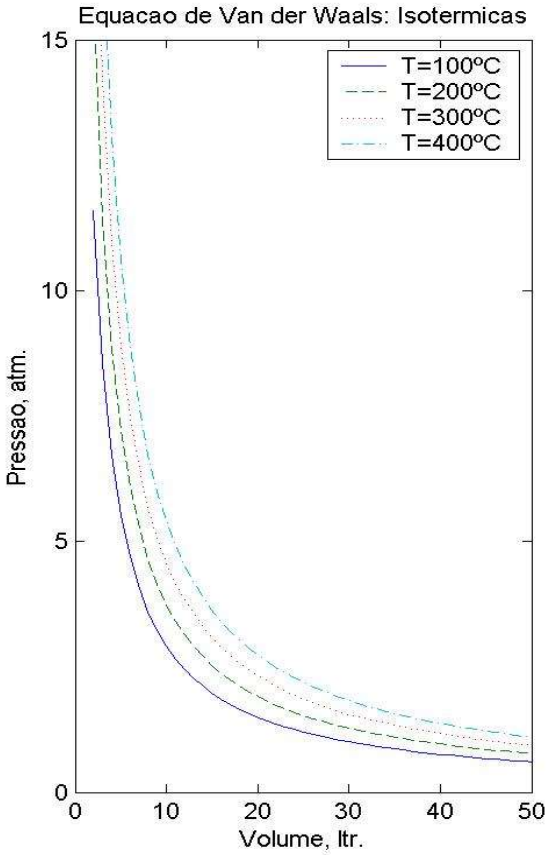


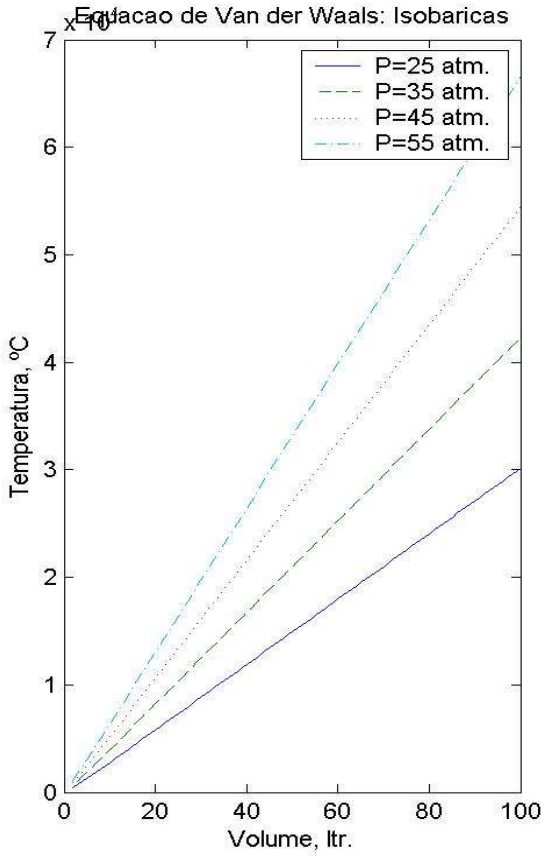
GRÁFICOS: 2D e 3D

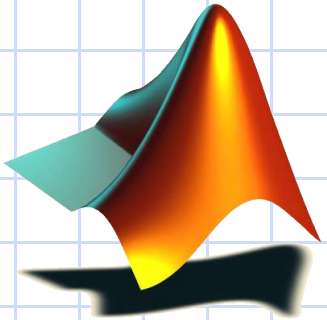
```
subplot(1,2,2)
P=[25:10:55];
T=zeros(4,length(V));
fac1=(V-b)/R;
T(1,:)=((P(1)+fac2).*fac1)-273.15;
T(2,:)=((P(2)+fac2).*fac1)-273.15;
T(3,:)=((P(3)+fac2).*fac1)-273.15;
T(4,:)=((P(4)+fac2).*fac1)-273.15;
plot(V,T(1,:),'-',V,T(2,:),'--',V,T(3,:),':',V,T(4,:),'-.')
title('Equacao de Van der Waals: Isobaricas')
xlabel('Volume, ltr.')
ylabel('Temperatura, ºC')
legend('P=25 atm.','P=35 atm.','P=45 atm.','P=55 atm.')
```



GRÁFICOS: 2D e 3D

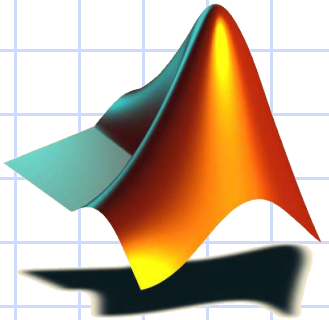






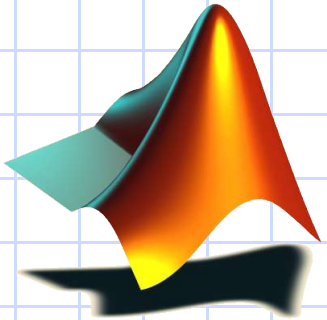
Exercício5.2

Dada a função f(x,y)=xy, obter numa janela gráfica as representações seguintes:
-A superfície definida pela função sobre o domínio [-10,10]*[-10,10].
-As linhas de contorno sobre a superfície
-A projecção das linhas de contorno sobre o domínio de definição
-A projecção das linhas de contorno sobre o plano xy, correspondentes aos valores –4,-1, 1 e 4.

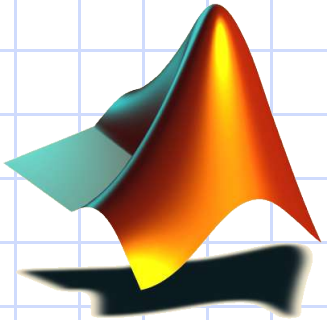


Exercício5.2

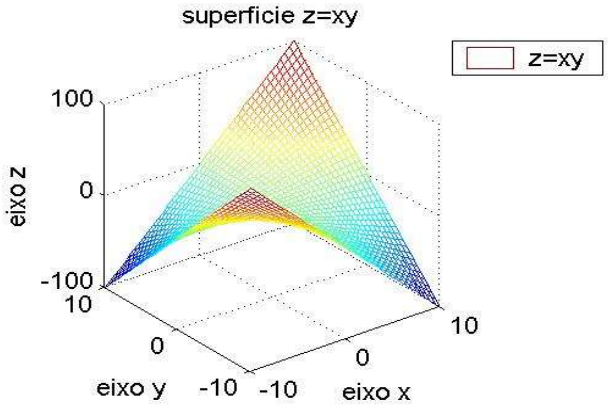
```
x=[-10:0.5:10]; y=x;
[X,Y]=meshgrid(x,y);
Z=X.*Y;
subplot(221)
mesh(X,Y,Z)
legend('z=xy')
xlabel('eixo x')
ylabel('eixo y')
zlabel('eixo z')
title('superficie z=xy')
subplot(222)
contour3(Z)
grid off
xlabel('eixo x')
ylabel('eixo y')
```

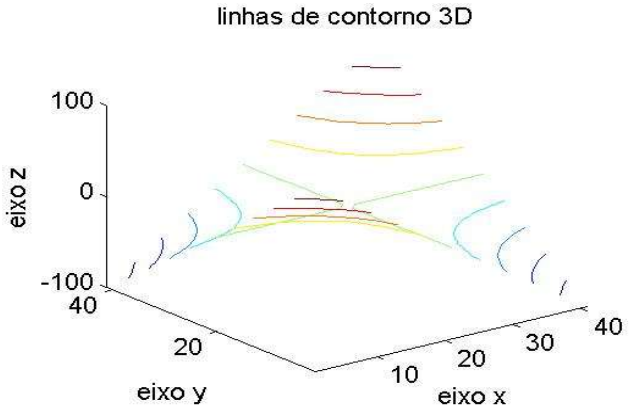


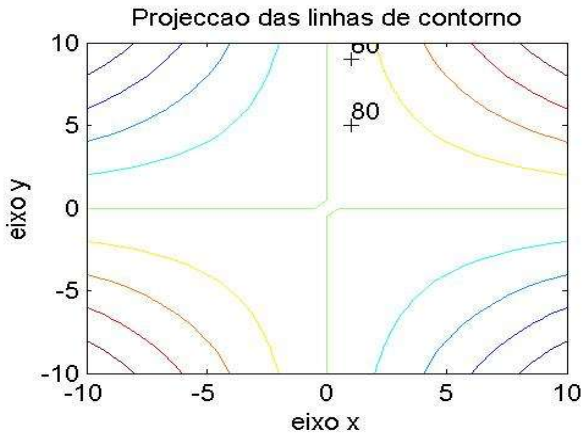
```
zlabel('eixo z')
title('linhas de contorno 3D')
subplot(223)
cs=contour(Z);
contour(x,y,Z)
grid off
clabel(cs)
xlabel('eixo x')
ylabel('eixo y')
title('Projeccao das linhas de contorno')
subplot(224)
contour(Z,[-4,-1,1,4])
grid off
xlabel('eixo x')
ylabel('eixo y')
title('Projeccao das linhas de contorno')
```

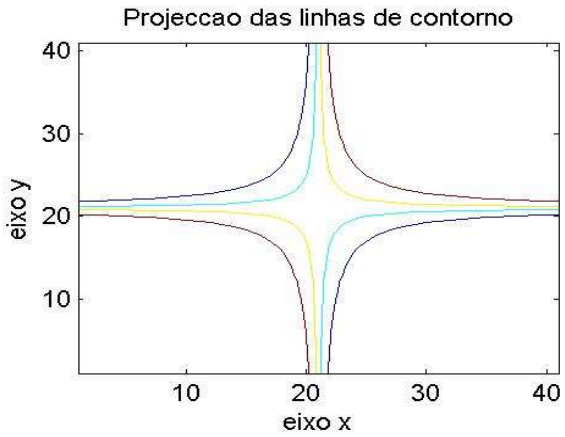


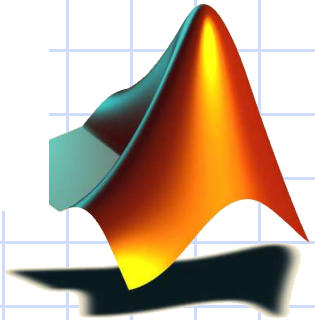
GRÁFICOS: 2D e 3D







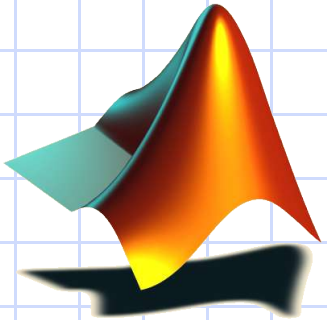




Exercício5.3

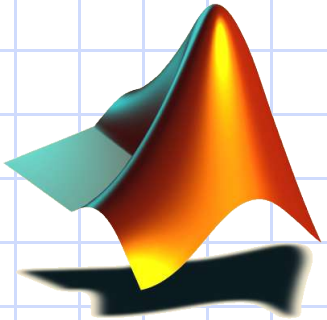
Representar a superfície de revolução obtida ao girar a curva $y=x^2+1$ em torno do eixo *ox*

x=[0:0.1:1]

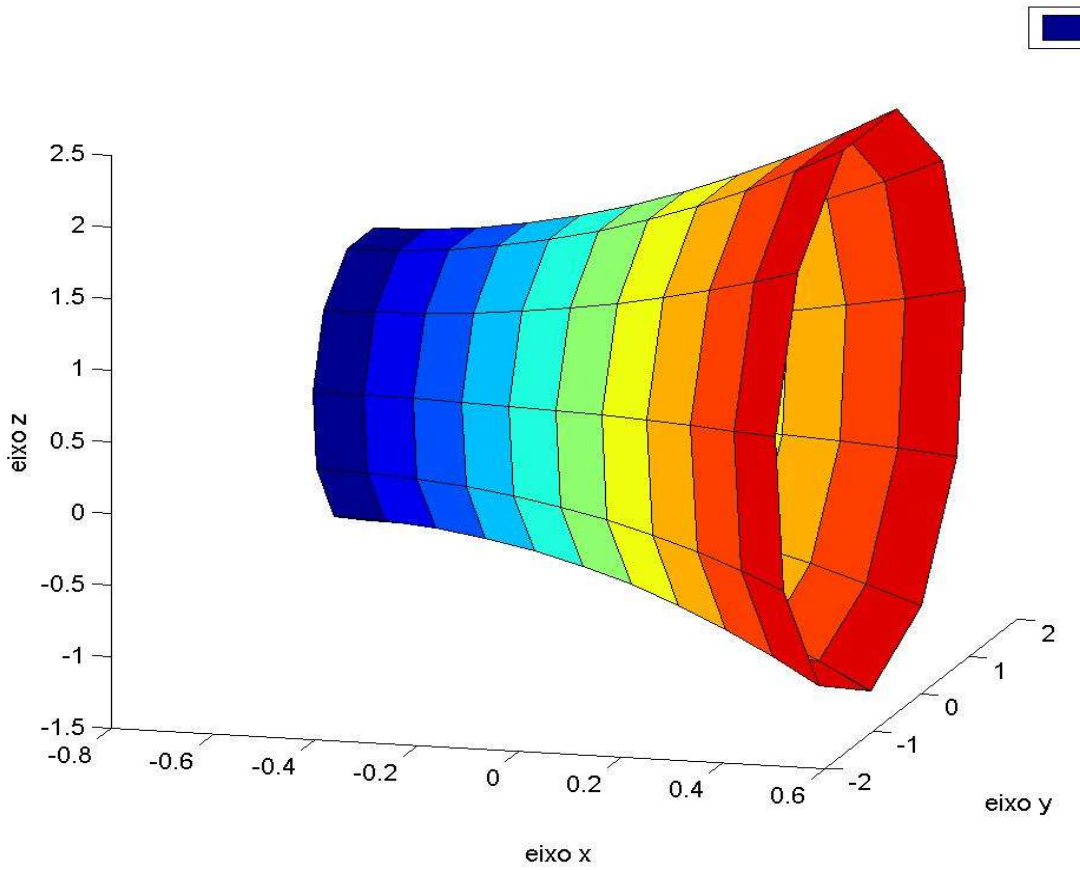


Exercício5.3

```
x=[0:0.1:1]; % pontos de discretizacao do eixo ox
rad=x.^2+1; % vector de raios
n=length(rad); % numero de raios
cylinder(rad,n) % representacao do cilindro
xlabel('eixo x')
ylabel('eixo y')
zlabel('eixo z')
[X,Y,Z]=cylinder(rad,n);
h=surf(X,Y,Z); % calculo do objecto
rotate(h,[0,1,0],90)
% Ao rodar desaparecem as legendas dos eixos
xlabel('eixo x')
ylabel('eixo y')
zlabel('eixo z')
view(15,15) % alteramos o ponto de observacao
grid off
legend('f(x)=x^2+1')
```



GRÁFICOS: 2D e 3D



PROGRAMAÇÃO

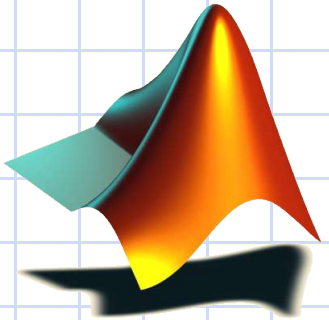
Todos os ficheiros de comandos Matlab devem ter a extensão *.m* e são de 2 tipos:

Ficheiros de função a primeira linha é executável e começa com a palavra function

Ficheiros de programa não são funções e constroem-se mediante uma sequência de comandos. Executam-se teclando o nome sem extensão

Exemplo1:

```
function temp_c=convert(temp_f);
%CRIAR UMA FUNCAO
%  PASSA DE GRAUS ºF A ºC
temp_c=5/9*temp_f-5/9*32;
```



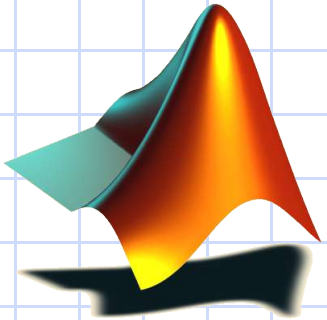
Comandos de entrada/saída

input: permite-nos introduzir dados
variavel=input('mensagem para o ecrã');

disp: mostra um texto no ecrã
disp('O algoritmo não convergiu')

menu: gera um menu que permite ao utilizador escolher entre diversas opções
opcao=menu('titulo da mensagem','opcao1',...'opcao_p')

error: informa da existência de um erro e pára a execução do programa, devolvendo o controlo ao utilizador



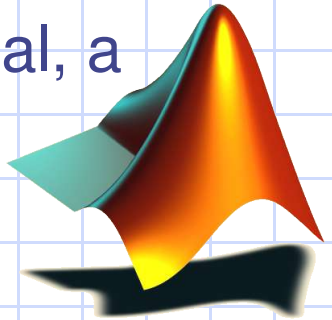
Programação de funções

A primeira linha é executável e começa pela palavra *function*:
function arg_saída=nome_funcao(arg_entrada)

Depois, quantos comandos sejam necessários, incluindo comentários como se se tratasse de um ficheiro programa. O ficheiro deve ser guardado como: *nome_funcao.m*

Para devolver o controlo ao programa a partir de qualquer ponto de uma função, basta escrever o comando return.

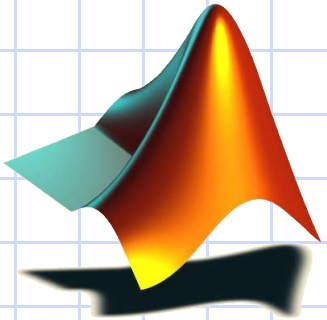
Exemplo2:
Construír uma função que defina, para um gás ideal, a temperatura em função da pressão e do volume.



```
function t=gases(p,v,n)
%t=gases(p,v,n)
%funcao que considera a lei dos gases ideais
%Argumentos de entrada:
%pressao p (atmosferas)
%volume v (litros)
%numero de moles n
%Argumentos de saida:
%temperatura t (Kelvin)
%R=0.0821 atm.litro/mol.grau
R=0.0821
t=p*v/(n*R);
```

Gravar como *gases.m*

```
temp=gases(20,10,10)
```



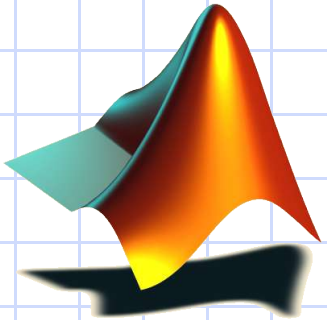
-O *m-file* só pode executar-se na *prompt* do Matlab ou a partir de uma linha de outro ficheiro.

-O *m-file* não admite argumentos de entrada, simplesmente trabalha com dados existentes no espaço de trabalho.

-As variáveis de uma função são locais à função e as de um ficheiro-m são globais.

Se queremos que uma variável seja compartilhada por várias funções, tem de definir-se em todas elas como global:

- global variável
- echo escreve cada comando do ficheiro no ecrã
- pause suspende a execução até se premir uma tecla
- keyboard idem e além disso permite intercalar comandos. A execução é retomada com return



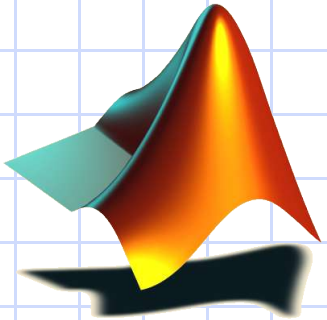
Testar o número de argumentos

- nargin devolve o número de argumentos de entrada com os quais a função foi chamada.

- nargout devolve o número de argumentos de saída com os quais a função foi chamada.

- nargchk verifica se o número de argumentos de entrada calculados com nargin está entre o valor máximo r mínimo previsto, se não dá erro.

nargchk(mínimo,máximo,número_entrada)



Ciclos

```
for k=n1:incre:n2
...
end
```

```
for k=vector_coluna
...
end
```

Com *break* é interrompida a execução

Estruturas de controlo

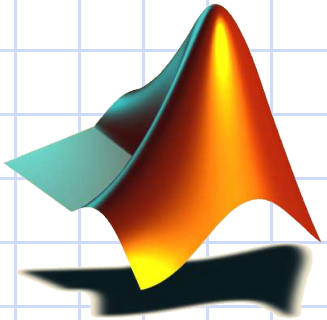
```
if
end
```

```
if
else
end
```

```
if
elseif
end
```

```
if
elseif
else
end
```

```
while
end
```



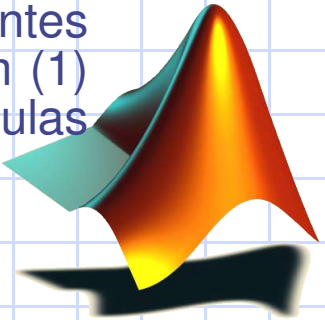
Operadores relacionais e lógicos

Menor: < Menor ou igual: <= Maior: > Maior ou igual: >=
Igual: == Diferente: ~= Ou: | E: & Não: ~

O resultado de realizar operações relacionais ou lógicas será um (1) se verdadeiro ou zero (0) se falso

Outras funções relacionais ou lógicas:

- xor(x,y): ("or" exclusivo) devolve 0 se x ou y são não nulos e 1 caso contrário.
- any(x): se x é um vector devolve um (1) se alguma componente de x é não nula. Se for uma matriz devolve um vector linha com um (1) para cada coluna da matriz x que tenha alguma das suas linhas não nulas e zero (0) caso contrário.
- all(x): Se x é um vector devolve um (1) se todas as suas componentes são não nulas. Se for uma matriz devolve um vector linha com um (1) para cada coluna da matriz x que tenha todas as suas linhas não nulas e zero (0) caso contrário.

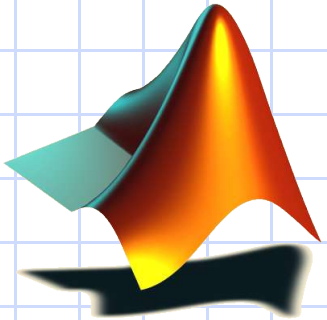


- isempty(x): devolve um (1) se a matriz x é vazia e zero (0) caso contrário.
- isequal(x1,x2,…,xn): Devolve um (1) se todas as matrizes são idênticas e zero (0) caso contrário.

Estruturas de controlo predefinidas:

Exemplo3:

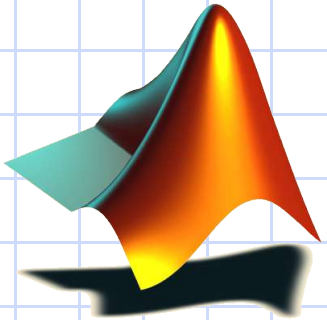
```
q=[];v=[1,-1,2,-1];
for i=1:length(v)
if(v(i)>=0)
q=q[q,1];
else
q=[q,0];
end
end
q=v>=0
```



```
r=v==q
s=(v>=0)&(v<2)
y=v(abs(v)>=2)
A=[2,4,6;1,3,5];
[i,j]=find(A<3)
```

Desde o Matlab 5 que foi incorporada a estrutura *switch* que permite realizar bifurcações no programa atendendo ao resultado de uma expressão:

```
switch expressao
        case expressao_0
                comandos_0 que devem ser executados
        case expressao_1
                comandos_1 que devem ser executados
        case expressao_2
                comandos_2 que devem ser executados
        otherwise
                comandos que devem ser executados
end
```

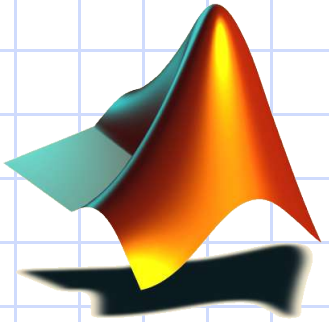


Leitura e escrita em ficheiros externos

A leitura e escrita de informação em ficheiros externos é efectuada essencialmente com os comandos *fread* e *fscanf* para leitura e *fprintf* e *fwrite* para escrita.

O procedimento geral em todos os casos é:

-Abrir o ficheiro a partir do qual se deseja ler ou no qual desejamos escrever.
-Colocar o apontador de leitura ou escrita na posição desejada.
-Ler ou escrever as variáveis.
-Fechar o ficheiro.



-Para Abrir ficheiros o comando é *fopen*

ident=fopen('nome de ficheiro')

Em *ident* é guardado um número de identificação (se for (-1) o ficheiro não pôde ser aberto.

-Para Fechar ficheiros o comando é *fclose*

fclose(ident)

fclose('all')

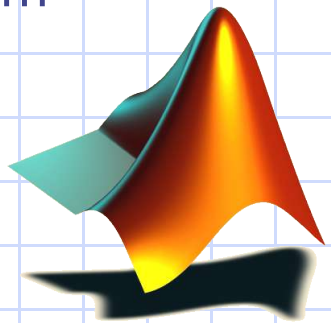
verif=fclose('all')

-Posicionamiento do apontador.

1.-Posicionar o apontador no início do ficheiro com número de identificação *ident*: *frewind(ident)*

2.-Posicionar o apontador dentro de um ficheiro:

test=fseek(ident,posi,'origem')



Este comando coloca o apontador do ficheiro *ident* na posição indicada em *posi*:

.-Se *posi*>0 move *posi* bytes para a frente.
.-Se *posi*<0 move *posi* bytes para trás.
.-Se *posi*=0 o apontador não se move.

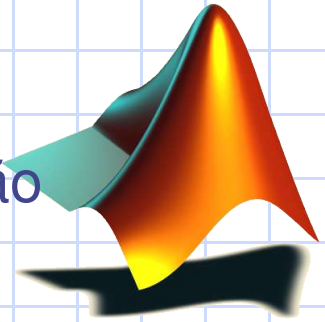
A variável caracter *origem* indica de onde se começa a mover o apontador.

.-'*bof*': principio do ficheiro.
.-'*cof*': posição actual no ficheiro.
.-'*eof*': final do ficheiro.

A execução do comando *fseek* devolve a variável *test* que valerá (-1) se algo falhar.
Para conhecer a posição do apontador: *posi=ftell(ident)*

Leitura de dados

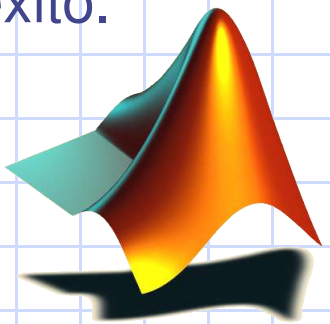
Devemos distinguir entre ficheiros formatados e não formatados.



Leitura de dados formatados:

[dados,contador]=fscanf(ident,'formato',quantos)

1. Lê *dados* do ficheiro *ident*.
2. Os dados lidos guardam-se em *dados*.
3. *quantos*, indica quantos dados vamos ler.
   .-um escalar k
   .-um vector[n,m], neste caso ler-se-ão os dados necessários para preencher uma matriz de ordem *n x m,* coluna a coluna.
   .-*inf* todos os dados do ficheiro.
4. A variável contador indica quantos foram lidos com êxito.
5. A variável *formato* indica o formato de leitura:
   %d: decimais; %e: notação exponencial



%f: notação de ponto fixo;

%g: não considera os zeros não significativos.

%s: variável caracter

Leitura de dados não formatados:

dados=fread(ident,quantos,'precisao')

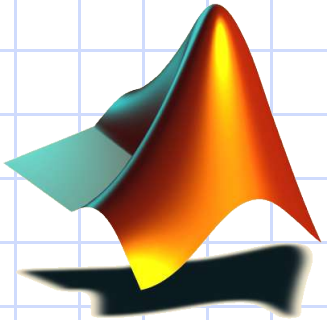
Escrita em ficheiro

Escrita de dados formatados:

contador=fprintf(ident,'formato',dados,controlos)

\n linha nova

\t avança até à posição seguinte de tabulação

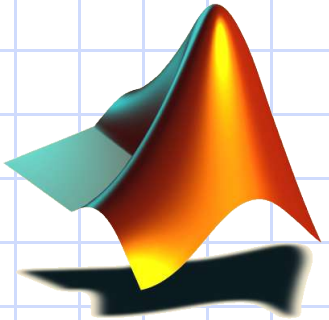


Escrita de dados não formatados:

contador=fwrite(ident,dados,'precisao')

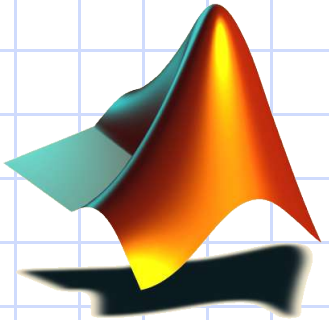
Exercicio6.1:
Utilizando estruturas de controlo construir uma função que calcule as raizes de uma equação de segundo grau

$$ax^2+bx+c=0$$



Exercicio6.1:

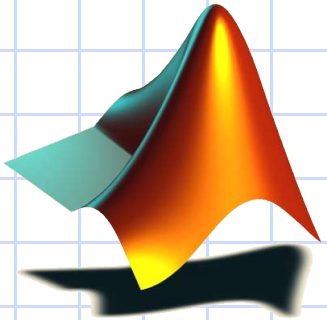
```
function raiz=sole2(a,b,c)
%raiz=sole2(a,b,c)
%solucao da equacao de segundo grau
%ax^2+bx+c=0, a~=0
%
if (nargin ~=3)
        error('O numero de argumentos de entrada deve ser 3')
end
discri=b^2-4*a*c;
raiz=[];
if(discri==0)
        raiz=-b/(2*a);
        disp(['Raiz dupla=',num2str(raiz)])
        return
elseif(discri>0)
```



```
        raiz(1)=(-b+sqrt(discri))/(2*a);
        raiz(2)=(-b-sqrt(discri))/(2*a);
        disp(['Raizes reais simples=',num2str(raiz)])
        return
else
        raiz(1)=(-b+sqrt(-discri)*i)/(2*a);
        raiz(2)=(-b-sqrt(-discri)*i)/(2*a);
        disp(['Raizes complexas=',num2str(raiz)])
        return
end
```

Esta função executa-se do seguinte modo:

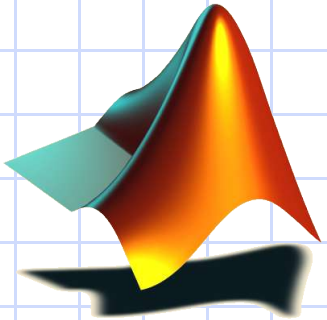
```
raiz=sole2(2,3,1);
raiz=sole2(1,0,1);
```



Exercicio6.2:

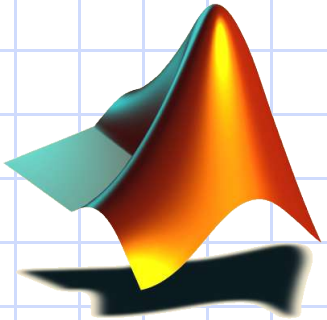
Programa para leitura de dados

Nome	Análise	Álgebra	Física	Estatística
Fernando Gomes Pereira	6	12	10	14
Susana Rodrigues Peres	14	8	6	2
Carlos Leal Alvarinho	16	16	14	17
Artur Gomes Alvarinho	10	8	10	17
Sílvia Taís Álvares	17	16	18	17
Andreia Galego Nunes	6	6	4	8
Alice Cavaleiro Leal	12	16	16	10
António Fraga Gomes	10	14	12	10
Beatriz Machado Gomes	8	6	10	8
Laura Tobias Macieira	14	16	10	18
João Rico Fraga	8	14	10	10
André Pena Gomes	12	16	10	12
Luís Branco Vila	16	12	12	8
Sandra Pontes Galego	16	17	14	18
Isolda Preto Gomes	10	10	12	12
Teresa Cieiro Gonçavles	8	4	10	6
Ricardo Lopes Amigo	16	12	4	18

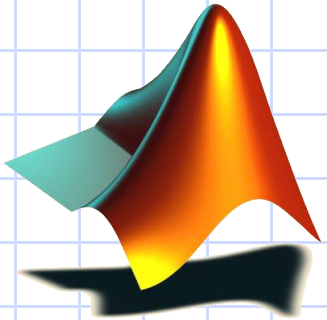


Exercicio6.2:

```
ident=fopen('dados.m');
frewind(ident)
variavais=[];
for i=1:5
   variavel=fscanf(ident,'%s',1);
   long_v(i)=length(variaveis)+length(variavel);
   variaveis=[variaveis variavel];
   if(i==1)
      fprintf(1,'\t%s\t\t',variavel)
   else
      fprintf(1,'\t%s\t',variavel)
   end
end
nomes=[];
notas=[];
```



```
for i=1:17
    nome=fscanf(ident,'%s,%c',1);
    apelido1=fscanf(ident,'%s,%c',1);
    apelido2=fscanf(ident,'%s,%c',1);
    l=3*(i-1)
    long_n(l+1)=length(nomes)+length(nome);
    long_n(l+2)=long_n(l+1)+length(apelido1);
    long_n(l+3)=long_n(l+2)+length(apelido2);
    nomes=[nomes nome apelido1 apelido2];
    notas=[notas; fscanf(ident,'%i',4)'];
    fprintf(1,'\n%9s %s %9s',nome, apelido1, apelido2)
    fprintf(1,'\t%i\t\t%i\t\t%i\t\t%i',notas(i,:))
end
fclose(ident)
```



ANÁLISE NUMÉRICA

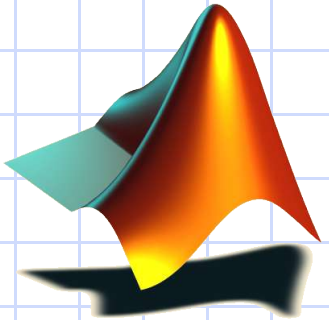
Neste capítulo introduziremos a análise numérica de problemas básicos relacionados com o estudo de funções de uma e várias variáveis, a aproximação de extremos locais ou o cálculo de zeros de uma função.

O cálculo dos mínimos locais de uma função de uma variável faz-se utilizando o seguinte comando:

min=fminbnd('funcao',a,b,opcoes)

Exemplo1:
Calcular um mínimo local da função $f(x)=3x^4-4x^3$ no intervalo [-1 2]

fminbnd('3*x^4-4*x^3',-1,2)



Calcular um máximo de f no intervalo [a,b] é o mesmo que calcular um mínimo de –f

*fminbnd('-(3*x^4-4*x^3)',-1,2)*

*fminbnd('-(3*x^4-4*x^3)',0,2)*

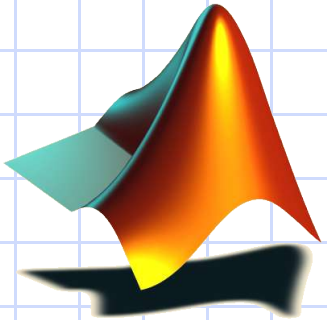
O cálculo de mínimos locais para funções de várias variáveis pode levar-se a cabo executando:

min=fminsearch('funcao',x0)

A diferença é que em vez de dar um intervalo damos um vector x0 que indica o ponto em torno do qual desejamos minimizar a função.

Exemplo2:
Minimizar a função f(x)=sen(xy) em torno de [0,0]

*fminsearch('sin(x(1)*x(2))',[0,0])*



Para calcular os zeros de uma função temos o comando *fzero*

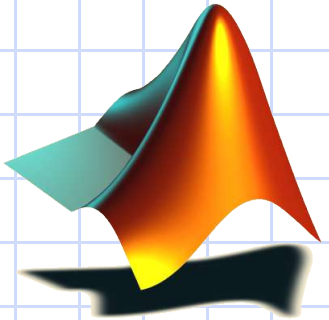
raiz=fzero('funcao',x0)

Calcula a raíz da função f(x)=0 a partir do ponto inicial x0

Exemplo3:

Calcular uma solução da equação $sen(x)-2cos(2x)+x^2=\pi^2-2$.
Esta é $f(x)=sen(x)-2cos(2x)+x^2-\pi^2+2=0$.
A função é contínua e para a escolha do ponto inicial procuramos um intervalo onde a função mude de sinal, [0,10]

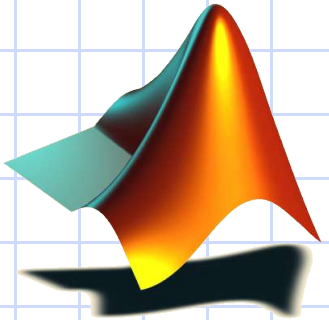
```
x=0;eval('sin(x)-2*cos(2*x)+x^2-pi^2+2')
x=10;eval('sin(x)-2*cos(2*x)+x^2-pi^2+2')
```



No intervalo [0,10] há uma mudança de sinal e isto implica que existe pelo menos um zero. Escolhemos x0=5:

```
x=fzero('sin(x)-2*cos(2*x)+x^2-pi^2+2',5)
```

```
eval('sin(x)-2*cos(2*x)+x^2-pi^2+2')
```



Métodos de integração numérica:

Podemos calcular a área sob curvas F(x) por integração numérica.

.-Integração de funções unidimensionais

quad

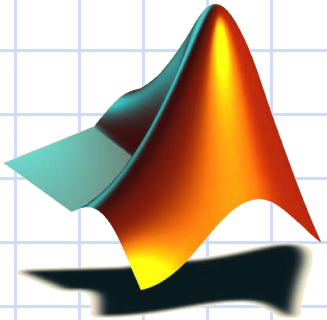
quadl

.-Integração dupla

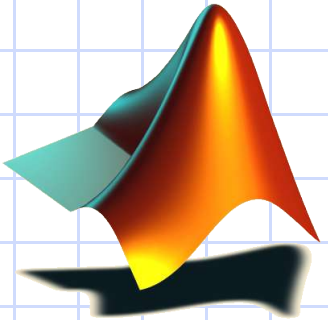
dblquad

Exemplo4:

```
ia=quad('sin(x)+1',0,2*pi); %(6.2832)
xmin=pi;xmax=2*pi;
ymin=0;ymax=pi;
result=dblquad('y*sin(x)+x*cos(y)', ...
xmin, xmax, ymin, ymax)
%-9.8698
```



```
int('sin(x)^2*cos(x)^2',0,4*pi) %1/2*pi=1.5708
ia=quad('sin(x).^2.*cos(x).^2',0,4*pi) % 1.0051e-030
ial=quadl('sin(x).^2.*cos(x).^2',0,4*pi) %1.5708
```



Métodos de resolução de equações diferenciais

Resolução de problemas de valores iniciais para equações diferenciais ordinárias (ODEs)

[T,Y]=solver('F',tspan,y0)

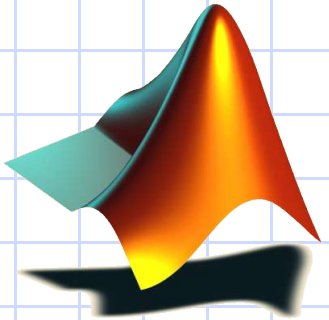
.- *solver*: algoritmo de resolução de ODEs:

ode45, ode23, ode113, ode15s, ode23s

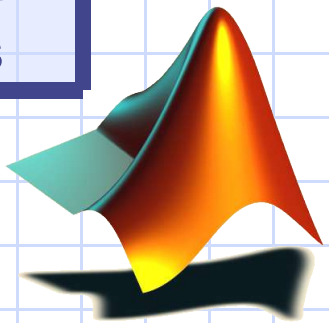
.-*F*: string contendo o nome do ficheiro ODE.

.-*tspan* vector de limites [t0 tfinal] de integração.

.-y0 vector coluna de condições iniciais em t0



Solvers	ode45 ode23 ode113 ode15s ode23s	Eq. Dif. Não rígidas. Ordem média Eq. Dif. Não rígidas. Ordem baixa Eq. Dif. Não rígidas. Ordem variável Eq. Dif. rígidas. Ordem variável Eq. Dif. rígidas. Ordem baixa
Opçoes	odeset odeget	Criar/modificar opções Obter opções
Saídas	odeplot odephas2 odephas3 odeprint	Representar séries temporais Representar fases bidimensionais Representar fases tridimensionais Saída para a janela de comandos



Exemplo5:

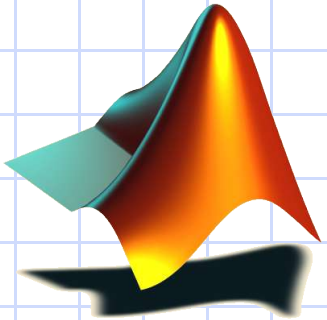
A equação de Van der Pol

$$y_1'' - \mu\left(1 - y_1^2\right) y_1' + y_1 = 0$$
$$\mu > 0$$

Reescrevemos o sistema

$$y_1'' = y_2$$
$$y_2' = \mu\left(1 - y_1^2\right) y_2 - y_1$$

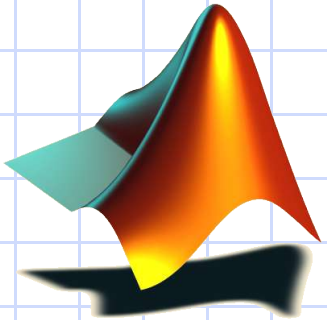
Escrevemos o ficheiro ODE

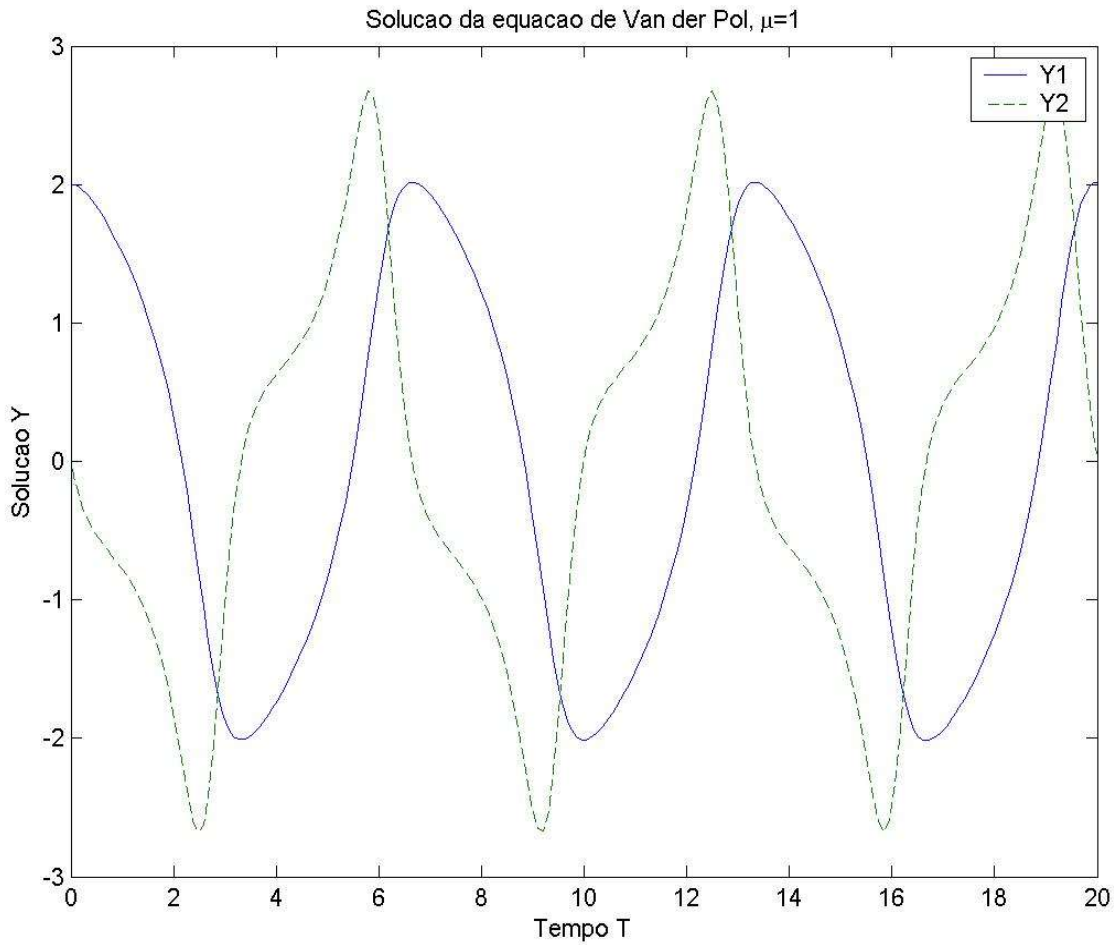


```
function dy=vdp1(t,y)
dy=[y(2); (1-y(1)^2)*y(2)-y(1)];
```

Chamamos o solver

```
[T,Y]=ode45('vdp1',[0 20],[2;0]);
plot(T,Y(:,1),'-',T,Y(:,2),'--')
title('Solucao da equacao de Van der Pol, \mu=1')
xlabel('Tempo T')
ylabel('Solucao Y')
legend('Y1','Y2')
```




Solucao da equacao de Van der Pol, μ=1
Y1
Y2
Solucao Y
Tempo T
3
2
1
0
-1
-2
-3
0
2
4
6
8
10
12
14
16
18
20

AJUDA

A ajuda é o mais importante no Matlab,
help nome_comando
help nome_toolbox

Podemos ver exemplos de aplicações de Matlab escrevendo:

demo

Para saber mais: http://www.mathworks.com/

http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/compiler.shtml
Esta última faz referência aos compiladores de C do Matlab, os mex files