

**Instituto Superior de Engenharia de Lisboa**  
**Licenciatura em Engenharia Informática e de Computadores**  
**Teoria dos Sinais e dos Sistemas**

**Introdução ao MATLAB: Exploração de conceitos**

Versão 1.1

25 Junho 2001

Artur Ferreira {arturj@cc.isel.ipl.pt}

O presente guia contém um conjunto de indicações e exemplos de forma a dar ao aluno uma perspectiva global das funcionalidades do programa MATLAB. Ilustra-se a sua utilização como ferramenta de cálculo não só para operações de análise e processamento de sinal, mas para cálculos matemáticos de engenharia em geral.

O objectivo deste guia é apoiar o primeiro contacto do aluno com o MATLAB, a sua linguagem de programação e ambiente de execução, não sendo de forma alguma uma referência completa. A exploração de todas as funcionalidades do programa está fora do alcance deste documento. Para obter informação mais completa deverá consultar as referências no final do guia.

Destina-se aos alunos da licenciatura em Engenharia Informática e de Computadores e a todos que queiram utilizar o MATLAB como ferramenta de cálculo e visualização de dados. Aconselha-se a experimentação dos exemplos apresentados ao longo do guia de forma a facilitar a compreensão do funcionamento do programa.

### **Introdução ao MATLAB**

O MATLAB<sup>1</sup> (MATrix LABoratory) é um sistema interactivo para a execução de cálculos científicos e de engenharia, suportado por *software* sofisticado baseado em cálculo matricial. Para além de ter um ambiente de execução, define uma linguagem de programação. No entanto é possível efectuar cálculos complexos sem ser necessário escrever um programa.

As funções existentes no MATLAB manipulam dados na forma matricial. A visualização de dados em vários formatos é uma das suas grandes potencialidades. Existem três formas de efectuar cálculos:

1. pela execução de comandos na janela de comandos (consola), à semelhança de uma janela de DOS;
2. através de sequências de comandos, editados em ficheiro de texto constituindo um *script*;
3. definindo funções, à semelhança de uma linguagem de programação procedimental, tal como o C/C++.

### **1. Utilização da janela de comandos**

À semelhança de uma janela de DOS, o MATLAB disponibiliza uma janela de comandos (consola) onde o utilizador executa comandos em sequência. Todas as variáveis declaradas num comando estão disponíveis no comando seguinte, excepto se forem explicitamente removidas.

A figura 1 mostra o ambiente de execução e exemplifica a utilização da janela de comandos realizando o seguinte conjunto de acções:

- o primeiro comando define o vector  $x$  com 6 elementos (matriz 1x6);
- o segundo comando define o vector  $y$  à custa do vector  $x$  (também é uma matriz 1x6);
- o terceiro comando define o vector  $z = [-1 \ -0.5 \ 0 \ 0.5 \ 1]$ , (matriz 1x5);
- finalmente invoca-se o comando *whos* que mostra quais as variáveis em memória e a respectiva dimensão.

---

<sup>1</sup> O *site* da MathWorks, empresa que desenvolve e comercializa o MATLAB é [www.mathworks.com](http://www.mathworks.com)

```

MATLAB Command Window
File Edit Window Help
To get started, type one of these commands: helpwin, helpdesk, or demo
>> x = [3 1 2 1.5 -3 5.1 ]
x =
    3.0000    1.0000    2.0000    1.5000   -3.0000    5.1000
>> y = 2*x -1
y =
    5.0000    1.0000    3.0000    2.0000   -7.0000    9.2000
>> z = -1: 0.5 : 1
z =
   -1.0000   -0.5000         0    0.5000    1.0000
>> whos
Name      Size      Bytes  Class
x         1x6         48  double array
y         1x6         48  double array
z         1x5         40  double array
Grand total is 17 elements using 136 bytes
>> |

```

Figura 1 – Exemplos de cálculo sobre a janela de comandos do MATLAB.

Pelo resultado da execução do comando *whos* na figura 1 constata-se que o tipo de dados utilizado por omissão é o *double* (8 bytes). No entanto é possível definir variáveis com valores complexos. As operações aritméticas e em geral todas as funções de cálculo operam com números complexos. A figura 2 exemplifica a manipulação de números complexos:

- declaram-se os números complexos  $z_1 = 1 + j$   $z_2 = 2 + j3$ ;
- calculam-se o módulo e argumento de  $z_1$ :  $|z_1|$  (função *abs*) e  $\arg[z_1]$  (função *angle*);
- efectua-se a multiplicação de  $z_1$  com o seu conjugado, o que é equivalente a  $|z_1|^2$ ;
- efectua-se a soma  $z_1 + z_2$ .

```

MATLAB Command Window
File Edit Window Help
>> z1 = 1 + i;
>> z2 = 2 + 3i;
>> abs(z1)
ans =
    1.4142
>> angle(z1)*180/pi
ans =
    45
>> z1 * z1'
ans =
    2
>> z1 + z2
ans =
    3.0000 + 4.0000i
>> |

```

Figura 2 – Exemplo de cálculos com números complexos na janela de comandos.

## 2. Execução de comandos com ficheiro *script* (\*.m)

A execução de comandos na consola tem a limitação de não permitir a sua aplicação sistemática (de forma repetitiva), o que dificulta a realização de testes sistemáticos, quando o número total de comandos a realizar em sequência é elevado. De forma a conseguir agrupar comandos e permitir a sua aplicação sequencial e sistemática, utilizam-se ficheiros em formato texto, obrigatoriamente com extensão .m, denominados de *script*, onde os comandos são escritos em sequência. Este procedimento é semelhante à utilização de ficheiros *batch* em DOS. O *script* “plot\_periodic.m”, apresentado na figura 3, efectua a seguinte sequência de acções:

- gera o vector  $t$  com 101 elementos, tendo como valor inicial 0, valor final 10, e espaçamento de 0.1 entre elementos consecutivos:  $t = [0 \ 0.1 \ 0.2 \ \dots \ 10]$ ; a numeração dos índices dos vectores começa em 1 e termina no índice cujo valor coincide com o comprimento do vector; para este caso  $t(1)=0$  e  $t(101)=10$ ;
- utiliza o vector  $t$  para gerar o vector  $x$  com a forma de onda sinusoidal  $x(t)$ , dada pela expressão seguinte:

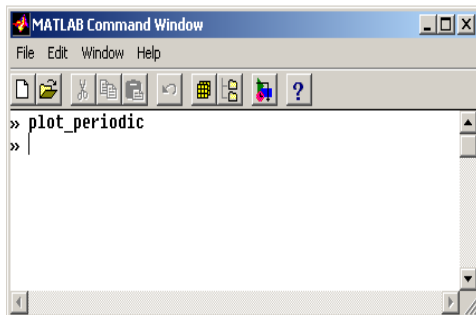
$$x(t) = A \cos\left(\frac{\pi}{2}t\right) \quad (1)$$

- o valor máximo da amplitude é definido pela variável  $A$  e corresponde neste exemplo a 2.5;
- em seguida mostra o gráfico do sinal  $x$ , tendo como abcissa o tempo  $t$  e como ordenada o valor da amplitude de  $x(t)$ .

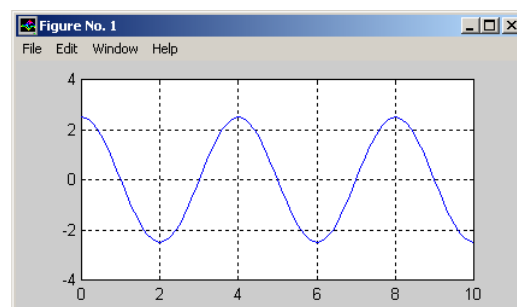
```
% Criar o vector de pontos.  
t = 0 : 0.1 : 10;  
% Definir a amplitude  
A = 2.5;  
% Aplicar a expressão do sinal.  
x = A * cos( (pi/2)*t );  
% Desenhar o sinal. Colocar uma grelha.  
plot( t, x ); grid on;
```

Figura 3 – Script “plot\_periodic.m”: sequência de comandos para desenho do sinal sinusoidal da expressão (1).

A invocação do *script* é feita na consola do MATLAB, através do seu nome, tal como exemplificado na figura 4 a). A figura 4 b) apresenta o resultado obtido, ou seja, o gráfico observado após a execução do *script*. Note que o comando *plot()* une os pontos do gráfico, obtendo uma função (aparentemente) contínua, cuja precisão depende do número de pontos no vector  $t$ . Pela observação do gráfico verifica-se que o sinal tem um período de 4 unidades de tempo.



a)



b)

Figura 4 – a) Invocação do script “plot\_periodic.m” na consola do MATLAB.  
b) Visualização da onda sinusoidal gerada pelo script “plot\_periodic.m”.

A figura 5 mostra o código do *script* “plot\_periodic2.m” que tem uma funcionalidade semelhante ao *script* anterior, com a diferença que utiliza um conjunto de funções pré-definidas do MATLAB para obter informação sobre o sinal  $x(t)$ , nomeadamente os valores mínimo (função *min*), máximo (função *max*) e médio (função *mean*). Utilizam-se os comandos *who* e *whos* para obter informação sobre as variáveis e o espaço de memória ocupado pelas mesmas. São ainda utilizados comandos para colocar título e informação sobre os eixos no gráfico.

```

% Remover todas as variaveis de memoria.
clear;
% Criar o vector de pontos.
t = 0 : 0.1 : 10;
% Definir a amplitude
A = 2.5;
% Aplicar a expressão do sinal.
x = A * cos( (pi/2)*t + (pi/4) );
% Mostrar as variaveis e as suas dimensoes.
whos
fprintf( '\n O comando whos permite observar quais as variaveis em...
memória e a sua dimensão. ');
fprintf( '\n Prima uma tecla para continuar... \n' );
pause
% Limpar a figura.
clf;
% Desenhar o sinal. Colocar uma grelha sobre o desenho.
plot( t, x ); grid on;
% Definir as labels do grafico.
xlabel('Tempo'); ylabel('Amplitude');
% Definir o titulo do grafico.
title ( sprintf( 'x(t)=%.1fcos(\pi/2*t + \pi/4)',A ) );
% Obter e mostrar a mínima/máxima amplitude do sinal x[n].
min_x = min( x )
max_x = max( x )
% Mostrar mensagens formatadas.
fprintf( '\n O valor mínimo do sinal é %f ', min_x );
fprintf( '\n O valor máximo do sinal é %f ', max_x );
fprintf( '\n O valor médio do sinal é %f ', mean(x) );
% Mostrar as variaveis.
who
fprintf( '\n O comando who permite observar quais as variaveis em...
memória. ');
fprintf( '\n Após a execução do script as variaveis permanecem em...
memória. ');
fprintf( '\n Prima uma tecla para continuar... \n' );
pause

```

Figura 5 – Script “plot\_periodic2.m”: sequência de comandos para desenho de um sinal sinusoidal.

O gráfico observado após a execução do script “plot\_periodic2.m” apresenta-se na figura 6. Em relação ao gráfico apresentado na figura 4 b) note o desfasamento do sinal.

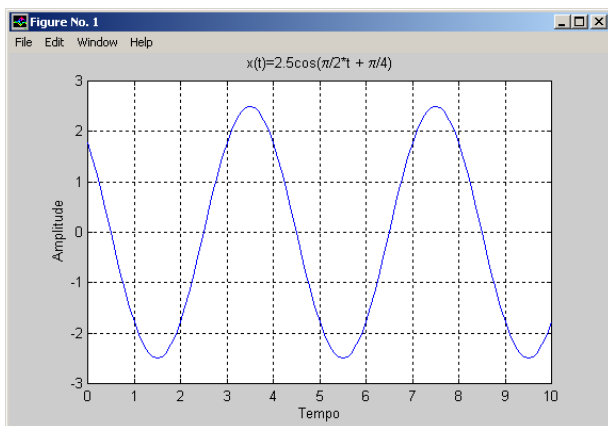


Figura 6 – Visualização da onda sinusoidal gerada pelo script “plot\_periodic2.m”. A expressão analítica é:

$$x(t) = 2.5 \cos\left(\frac{\pi}{2}t + \frac{\pi}{4}\right) \quad (2)$$

### 3. Funções definidas pelo utilizador

A utilização de ficheiros *script* é vantajosa em relação à execução de comandos na consola, pelos motivos referidos acima. No entanto, a alteração de parâmetros num ficheiro *script*, pode obrigar a que o ficheiro seja editado e alterado por cada execução. Outro problema na utilização de ficheiros *script* é que as variáveis (globais) permanecem em memória após a sua execução, o que pode causar conflitos de nomes e erros. Quando se pretende efectuar testes em que diferem os valores dos parâmetros faz mais sentido definir uma função, à semelhança de uma linguagem de programação, tal como por exemplo o C/C++.

Nesta secção mostra-se como o utilizador pode definir as suas funções. Cada função deve ser definida num ficheiro *<nomeficheiro>.m*, tal que *nomeficheiro* é igual ao nome da função. Na figura 7, apresenta-se a função “*decrease\_exponencial*” (definida no ficheiro *decrease\_exponencial.m*) que cria e desenha duas exponenciais decrescentes  $y_1(t)$  e  $y_2(t)$ , com factores de escala diferentes, tal como indicado pelas expressões seguintes:

$$y_1(t) = e^{-at}(u(t) - u(t - \alpha)) \quad y_2(t) = e^{-2at}(u(t) - u(t - \alpha)) \quad (3)$$

A função *decrease\_exponencial* tem dois parâmetros de entrada:

- *NSamples*, valor máximo no eixo dos tempos (o valor de ‘ $\alpha$ ’ na expressão (3));
- *ScaleFactor*, factor de escala a aplicar à exponencial (o valor de ‘ $a$ ’ na expressão (3));

Os valores retornados são:

- *y1*, vector com as amostras da exponencial com o factor de escala *ScaleFactor* ( $a$ );
- *y2*, vector com as amostras da exponencial com o factor de escala  $2*ScaleFactor$  ( $2a$ );

```
function [y1, y2] = decrease_exponencial ( NSamples, ScaleFactor )
% Construir o vector de amostras.
n = 0 : 0.1 : NSamples;
% Aplicar a exponencial decrescente.
y1 = exp( -n*ScaleFactor );
y2 = exp( -n*2*ScaleFactor );
% Limpar o gráfico.
clf
% Desenhar as duas funções no mesmo gráfico.
plot( n, y1 );
hold on;
plot( n, y2, 'r' );
grid on;
% Colocar título e legenda no gráfico.
title ( ' Exponencial decrescente' );
xlabel( ' Tempo ' );
ylabel( ' Amplitude ' );
```

Figura 7 – Função “*decrease\_exponencial.m*”: desenho de duas exponenciais decrescentes.

A invocação da função *decrease\_exponencial* necessita obrigatoriamente dos dois parâmetros de entrada. Os parâmetros de saída são opcionais, ou seja, a função pode ser invocada de formas diferentes, com 0, 1 ou 2 parâmetros de saída, tal como ilustrado na figura 8.

```

MATLAB Command Window
File Edit Window Help
>> clear
>> decrease_exponencial( 2.5 , 2);
>> whos
Name      Size      Bytes  Class
ans       1x26      208    double array
Grand total is 26 elements using 208 bytes
>> y1 = decrease_exponencial( 2.5 , 2);
>> whos
Name      Size      Bytes  Class
ans       1x26      208    double array
y1        1x26      208    double array
Grand total is 52 elements using 416 bytes
>> [y1, y2] = decrease_exponencial( 2.5 , 2);
>> whos
Name      Size      Bytes  Class
ans       1x26      208    double array
y1        1x26      208    double array
y2        1x26      208    double array
Grand total is 78 elements using 624 bytes
>> |

```

Figura 8 – Três formas de invocação da função “decrease\_exponencial.m”: utilização de número variável de parâmetros de saída.

Sobre a figura 8, note-se que após a primeira invocação da função existe uma variável denominada de *ans*. Esta variável é criada automaticamente pelo MATLAB para guardar o resultado. Na primeira invocação *ans* toma o valor do primeiro parâmetro de saída e a partir daí o seu valor não sofre alteração, porque nas duas chamadas seguintes são explicitados parâmetros de saída. Em qualquer um dos três casos apresentados acima, a invocação da função *decrease\_exponencial* produz o resultado da figura 9.

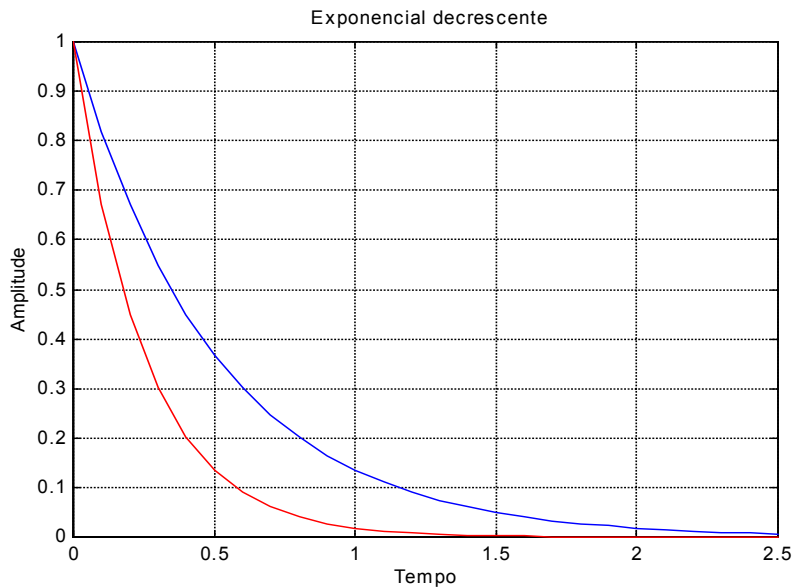


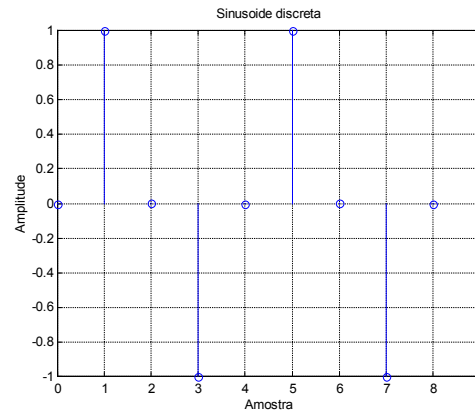
Figura 9 – Desenho de duas funções exponenciais decrescentes, segundo a expressão (3) com  $a=2$  e  $\alpha=2.5$ .

### 3.1 Desenho de sinais discretos

Todos os exemplos apresentados até ao momento são referentes a sinais “contínuos”<sup>2</sup>. A manipulação de sinais discretos é feita da mesma forma, com a excepção de que no desenho deve-se utilizar a função *stem()* em vez da função *plot()*, tendo o cuidado de colocar apenas valores inteiros no eixo das abcissas (índices das amostras). A figura 10 mostra a função *discrete\_sine(K, A, N)* que gera o sinal dado pela expressão seguinte:

$$x[n] = A \cdot \sin\left(\frac{2\pi}{N}n\right)(u[n] - u[n - K]) \quad (4)$$

```
function discrete_sine ( K, A, N )
% Construir o vector de amostras.
n = 0 : 1 : K-1;
% Aplicar a expressao do seno.
y = A * sin ( ((2*pi)/N) * n );
% Limpar o gráfico.
clf
% Desenhar.
stem( n, y ); grid on;
% Colocar título e legenda.
title ( ' Sinusoide discreta ' );
xlabel( ' Amostra ' );
ylabel( ' Amplitude ' );
```



a)

b)

Figura 10 – a) Código da função *discrete\_sine*. b) Resultado da execução com  $K=10$ ,  $A=1$ ,  $N=4$ .

### 3.2 Operações sobre sinais discretos

Atente-se agora numa função que efectua quatro operações sobre um sinal discreto de energia: calcula o valor médio, valor mínimo, valor máximo e a energia. O código da função *signal\_operation* apresenta-se na figura 11 a), onde se pode verificar que recebe um parâmetro de entrada e retorna quatro parâmetros. Note que as funções pré-definidas MATLAB *mean*, *min* e *max* manipulam valores complexos, como se pode constatar pelos resultados obtidos na invocação da função com o sinal  $x = [j \ j \ 2j \ 2j]$ , na figura 11 b).

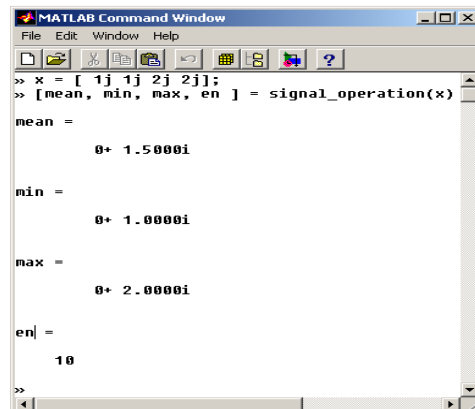
```
function [ meanValue, minValue,
maxValue, energy ] =
signal_operation ( Signal )

% Calculo do valor medio.
meanValue = mean( Signal );

% Calculo do valor minimo.
minValue = min( Signal );

% Calculo do valor maximo.
maxValue = max( Signal );

% Calculo da energia.
energy = Signal * Signal';
```



a)

b)

Figura 11 – a) Código da função *signal\_operation*. b) Resultado da execução com o sinal  $x = [j \ j \ 2j \ 2j]$ .

<sup>2</sup> Na realidade os sinais são sempre discretos. No entanto, devido à proximidade dos pontos no eixo das abcissas e a interpolação dos pontos efectuada pela função *plot* consegue-se visualizar um sinal contínuo.

## 4 Funções utilitárias

Nesta secção apresentam-se mais algumas funções que o MATLAB disponibiliza. Devido à sua frequente utilização são aqui apresentadas como exemplo.

Considere o sinal discreto periódico complexo dado pela expressão seguinte, sobre o qual se representa um período completo (N=8 amostras).

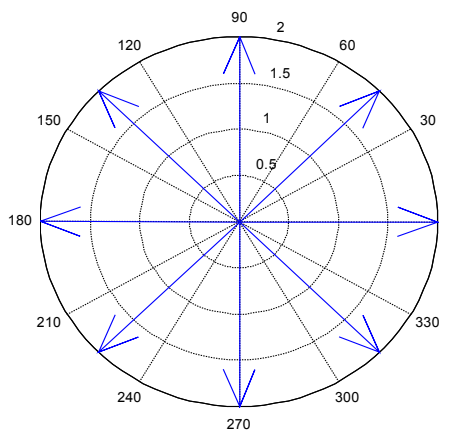
$$x[n] = 2e^{j\frac{\pi}{4}n} (u[n] - u[n-8]) \quad (5)$$

A visualização de sinais complexos, tal como  $x[n]$ , é muitas vezes necessária em operações de processamento de sinal. Para tal existem funções pré-definidas que facilitam esta visualização. A função *compass* permite a observação das amostras do sinal na forma de vectores. A figura 12 a) mostra a criação do sinal  $x[n]$  e a invocação da função *compass* para efectuar o desenho. A figura 13 mostra o mesmo sinal na forma polar (ou trigonométrica), separando módulo e fase, com recurso às funções *subplot*, *stem*, *abs* e *angle*.

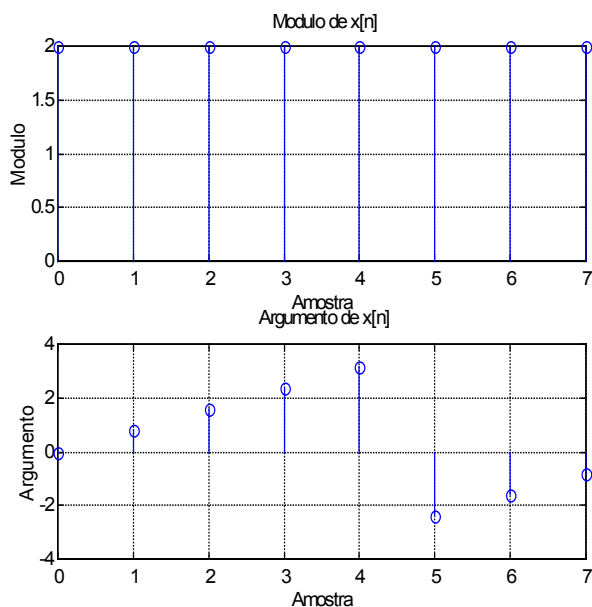
```

» n = 0:1:7;
» x = 2*exp( sqrt(-1)*(pi/4)*n );
» compass(x);
    
```

a)



b)



a)

```

» subplot(2,1,1);
» stem(n, abs(x));
» grid on;
» xlabel(' Amostra ');
» ylabel(' Modulo ');
» title (' Modulo de x[n]');
» subplot(2,1,2);
» stem(n, angle(x));
» grid on;
» xlabel(' Amostra ');
» ylabel(' Argumento ');
» title (' Argumento de x[n]');
    
```

b)

Figura 13 – a) Desenho na forma polar com a separação de módulo e fase. b) Código correspondente.

## 5. Indicações para exploração das potencialidades do MATLAB

Nesta secção apresenta-se um conjunto de tópicos a explorar na utilização do MATLAB, que poderão ou não estar no contexto desta disciplina. Este conjunto está longe de reflectir todas as potencialidades do MATLAB. Procura-se apenas identificar diversas variantes de cálculo e visualização que tornam o MATLAB uma ferramenta poderosa.

Os tópicos a considerar são os seguintes:

- os comandos *save* e *load* servem para escrever e ler, respectivamente, variáveis em ficheiro;
- a função *roots* calcula as raízes de um polinómio de grau  $n$ ;
- o comando  $A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9]$  declara a matriz quadrada  $3 \times 3$ :

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

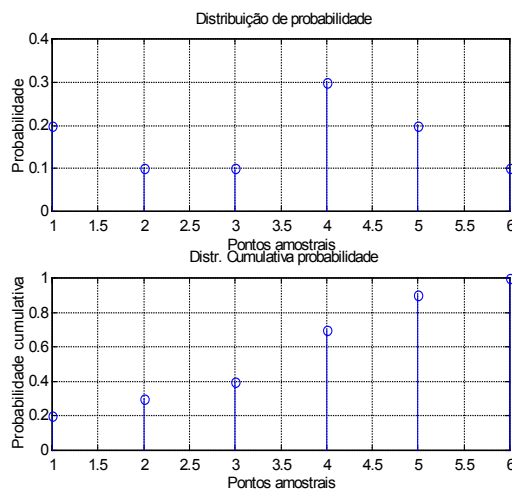
- a função *eig* calcula os valores e vectores próprios de uma matriz quadrada;
- a função *inv* calcula a matriz inversa; a função *det* calcula o determinante;
- as funções *quad* e *int* efectuam cálculo integral;
- a manipulação de ficheiros é feita de forma semelhante à linguagem C; existe um vasto conjunto de funções: *fopen*, *fclose*, *fread*, *fwrite*, *frewind*, *ftell*, *fprintf*,...
- as funções *fft* e *fft2*, calculam a transformada de Fourier de sinais discretos (DFT-Discrete Fourier Transform) 1D e 2D respectivamente; este assunto será objecto de estudo nas disciplinas seguintes: PDS1 e PDS2;
- o desenho 3D é conseguido através das funções *surf*, *mesh*;
- para desenhar curvas de nível tem disponível a função *contour*;
- pode sempre recorrer ao comando *help* para procurar uma função ou informação sobre determinada função;
- qualquer função possui sempre dois parâmetros extra: *nargin* e *nargout*, que tomam o valor correspondente ao número de parâmetros de entrada e de saída, respectivamente;
- a execução do comando *intro* permite-lhe ter um primeiro contacto com a linguagem e ambiente MATLAB;
- para conhecer a fundo todas as potencialidades do MATLAB execute o comando *demo*.

O conjunto de comandos da figura 14 declara o vector  $p$  contendo uma distribuição de probabilidades discreta com 6 pontos. Recorre-se às funções *subplot* e *stem* para efectuar o desenho da distribuição de probabilidade. Em seguida utiliza-se a função *cumsum* para desenhar a função de distribuição cumulativa de probabilidade.

```

» p = [0.2 0.1 0.1 0.3 0.2 0.1];
» subplot(2,1,1);
» stem( 1:length(p), p);
» grid on; xlabel( 'Pontos amostrais' );
» ylabel(' Probabilidade ');
» title(' Distribuição de probabilidade' );
» subplot(2,1,2);
» stem( 1:length(p), cumsum(p));
» grid on; xlabel( 'Pontos amostrais' );
» ylabel(' Probabilidade cumulativa');
» title(' Distr. Cumulativa probabilidade' );

```



a)

b)

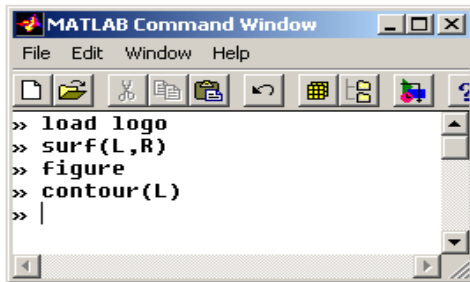
Figura 14 – a) Sequência de comandos para utilização da funcionalidade de soma cumulativa, aplicada a uma função de distribuição de probabilidade. b) Função distribuição de probabilidade e função distribuição cumulativa de probabilidade.

A função *replace\_z* apresentada na figura 15, recebe uma matriz e dois valores escalares. A função percorre toda a matriz (*Matrix*) e substitui todas as ocorrências de um elemento (*CurrentValue*) por outro (*NewValue*). Utiliza-se a função *size* para obter as dimensões da matriz.

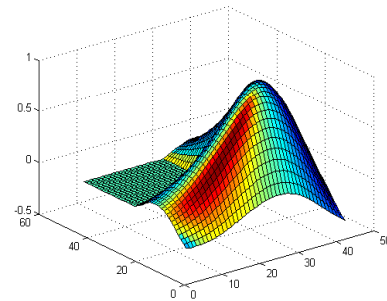
```
function Matrix = replace_z( Matrix, CurrentValue, NewValue )
% Obter o numero de linhas e colunas da matriz.
[NLines, NColumns] = size( Matrix );
% Percorrer a matriz elemento a elemento.
for i=1 : NLines
    for j=1 : NColumns
        % Verificar se encontra o valor pretendido.
        if Matrix(i,j) == CurrentValue
            % Encontrou. Efectuar a substituição.
            Matrix(i,j) = NewValue;
        end
    end
end
end
```

Figura 15 – Função  $Matrix = \text{replace\_z}(Matrix, CurrentValue, NewValue)$ .

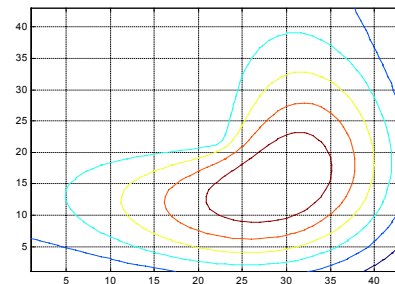
A figura 16 mostra o desenho de uma matriz em dois formatos: superfície 3D e curvas de nível. Esta matriz é a imagem que constitui o logotipo do MATLAB, sendo lida para memória pela execução do comando *load logo*. A função *surf* produz a superfície 3D e a função *contour* obtém as curvas de nível.



a)



b)



c)

Figura 16 – Exemplo das capacidades de desenho.

- a) Invocação na consola.
- b) Superfície 3D.
- c) Curvas de nível.

A manipulação e desenho de funções complexas de variável complexa é ilustrada na figura 17 com o desenho 3D da função  $f(z)=z$ ;

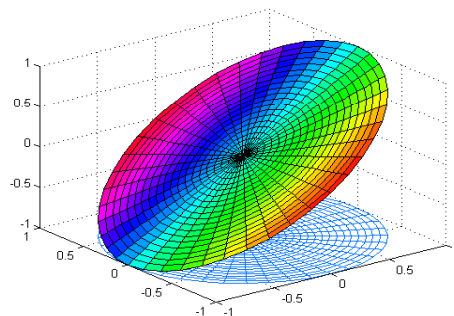
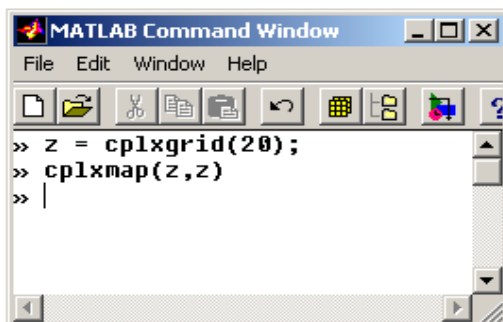


Figura 17 – Desenho 3D da função  $f(z)=z$ .

A facilidade de manipulação e processamento de imagem é outra das características do MATLAB. Imagens são representadas por matrizes. A figura 18 a) mostra um troço de *script* cuja funcionalidade consiste em ler uma imagem a partir de ficheiro *bitmap*, através da função *imread*. Esta função devolve a matriz *I* com a imagem e o mapa de cores na variável *map*. Em seguida com recurso à função *image* apresenta-se a imagem. A figura 18 b) apresenta a imagem obtida.

Na figura 19 apresenta-se um *script* para determinar e mostrar os contornos da imagem, recorrendo às funções *edge*<sup>3</sup> e *imshow*.

Nas figuras 20 e 21 mostra-se o processo de filtragem através da aplicação do filtro de mediana. No caso da figura 20 contamina-se a imagem com ruído impulsivo, utilizando a função *imnoise* e verifica-se o resultado obtido. Na figura 21 aplica-se um filtro de mediana com janela 3x3, através da função *medfilt2* de forma a remover esse ruído e verifica-se o funcionamento.

```
% Ler a imagem a partir de um
% ficheiro bitmap.
% I - imagem; map - colormap.
[ I, map ] = imread( 'im.bmp', ...
'bmp' );
% Definir o novo colormap.
colormap( map );
% Desenhar a imagem.
image( I );
% Retirar os eixos
axis off;
```

a)

Figura 18 – Leitura e apresentação de imagem.

a) Script para efectuar a leitura.

b) Imagem obtida.



b)

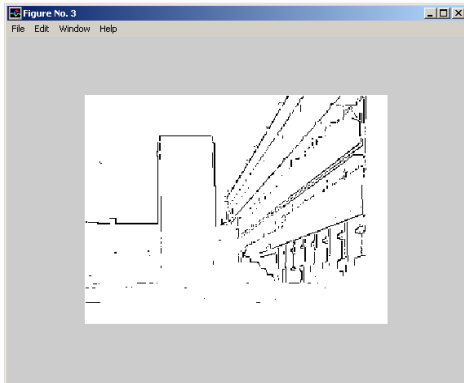
```
% Detectar os contornos.
I2 = edge( I, 'roberts' );
% Inverter os contornos
% para ter fundo branco.
I2 = ~I2;
% Criar nova figura
figure;
% Desenhar os contornos.
imshow( I2 );
```

b)

Figura 19 – Detecção de contornos numa imagem.

a) Imagem com os contornos.

b) Script com os comandos.



a)

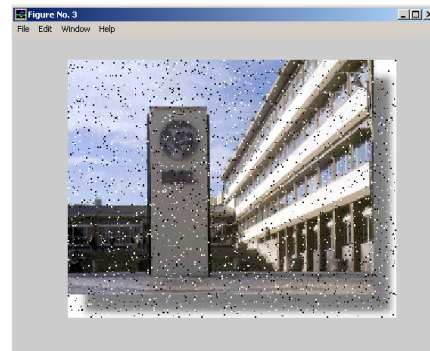
```
% Criar uma nova figura.
figure
% Adicionar ruído impulsivo.
I2 = imnoise( I, 'salt & pepper' );
% Definir o novo colormap.
colormap( map );
% Mostrar a imagem com ruído.
image( I2 );
% Retirar os eixos
axis off;
```

a)

Figura 20 – Adição de ruído impulsivo a uma imagem.

a) Script com os comandos.

b) Imagem obtida.



b)

<sup>3</sup> A funções *edge*, *imnoise* e *medfilt2* são específicas da *toolbox* de processamento de imagem.

```

% Criar uma nova figura.
figure
% Aplicar um filtro de mediana 3x3.
I3 = medfilt2( I2 );
% Definir o novo colormap.
colormap( map );
% Mostrar a imagem após filtragem.
image( I3 );
% Retirar os eixos
axis off;

```

a)

Figura 21 – Aplicação de filtro de mediana para remoção de ruído impulsivo.

a) Script com os comandos.

b) Imagem obtida após filtragem.



b)

Note as diferenças entre as imagens da figura 18 (original) e a imagem recuperada após filtragem da figura 21.

## **6. Referências**

- [1] [http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/brief\\_introduction\\_to\\_matlab.pdf](http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/brief_introduction_to_matlab.pdf), guia introdutório resumido.
- [2] [http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/matlab\\_primer\\_3rd.pdf](http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/matlab_primer_3rd.pdf), guia introdutório resumido.
- [3] [http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/getstart\\_matlab5.pdf](http://www.deec.isel.ipl.pt/analisedesainai/tss/Bibliografia/getstart_matlab5.pdf), manual de referência completa para a versão 5 do MATLAB.