Introdução ao MATLAB

Análise e Processamento Digital de Sinal (M3002)

Análise e Processamento de Imagem (M4031, M4094)

Processamento de Sinal e Imagem em Física Médica (F4012)

[Conteúdo]

click on it

André R. S. Marçal

Departamento de Matemática
Faculdade de Ciências, Univ. Porto (FCUP)
http://www.fc.up.pt/pessoas/andre.marcal

versão 1.3 - 23 Fevereiro 2021

Enquadramento

O MATLAB (MATrix LABoratory) é um sistema computacional de cálculo científico que permite o desenvolvimento expedito de algoritmos para processamento e análise de dados. É uma ferramenta de grande utilidade em diversas áreas de Matemática Aplicada, Análise de Dados, Engenharia, etc. Na Faculdade de Ciências da Universidade do Porto (FCUP) tem sido usado em várias disciplinas (ou UC - Unidades Curriculares), para além de Análise e Processamento Digital de Sinal (APDS), Análise e Processamento de Imagem (API) e Processamento de Sinal e Imagem em Física Médica (PSIFM).

O objectivo destes apontamentos é fazer uma introdução geral ao MATLAB, e também a alguns aspectos relacionados com o uso do MATLAB para Processamento de Sinal e Imagem. A necessidade deste elemento de trabalho está relacionada com a estruturação dos cursos da FCUP (e da U.Porto) no regime de Bolonha, que permite a frequência numa UC de estudantes com diferentes perfis. Por um lado há sempre alguns estudantes inscritos em APDS, API e PSIFM sem experiência prévia de MATLAB. Por outro lado, a escolaridade (tempo de contacto) tem vindo a diminuir, o que limita as possibilidades de acompanhamento da prática computacional. O documento não é uma apresentação exaustiva das funcionalidades e características do MATLAB, mas sim um guião que deve ser lido em paralelo com a experimentação prática em computador. Pretende-se que possa ser usado pelos estudantes de forma autónoma.

Convém referir que há alguma diversidade em termos de versões instaladas nos computadores da FCUP, pelo que poderá haver ligeiras diferenças entre o que é apresentado aqui e o que se observa no computador, em particular no que se refere ao interface gráfico, e posteriormente também em funções mais avançadas das *Toolboxes* específicas.

Estes apontamentos estão numa fase inicial de desenvolvimento (primeira utilização no 2º semestre de 2019/2020), pelo que são bem vindos contributos para melhorias (indicação de inconsistências, erros e sugestões de alterações) a incluir em versões futuras.

Conteúdo

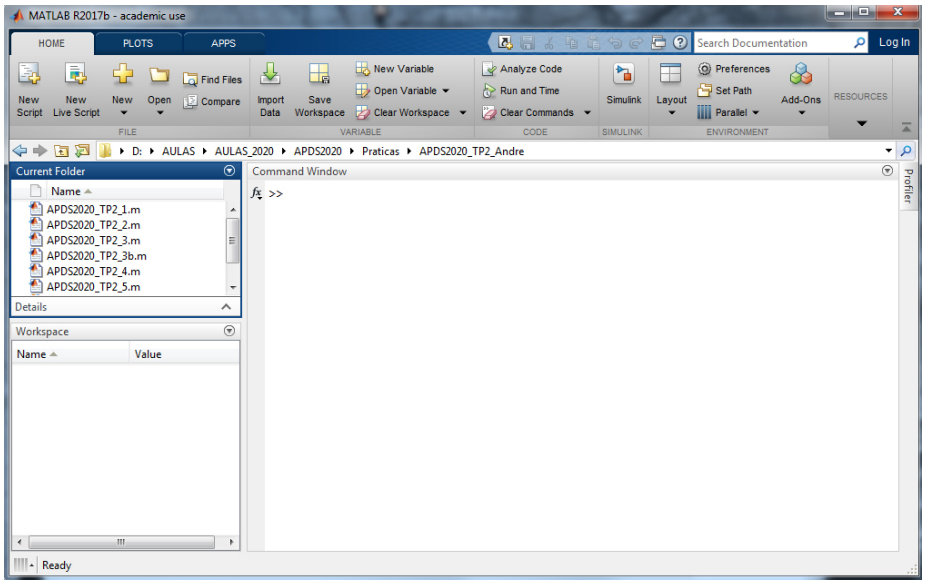


Figura 1: Consola do MATLAB no início de uma sessão.

1 Primeiros passos

1.1 Ambiente de trabalho MATLAB

Quando se inicia uma sessão MATLAB surge uma janela no ecrã, habitualmente dividida em várias sub-janelas. A figura 1.1 mostra um exemplo do inicio de uma sessão MATLAB (versão R2017b), onde surgem 3 secções (ou sub-janelas) na janela gráfica. A sub-janela de maior área, do lado direito, é a consola principal - *Command Window*, havendo neste caso 2 sub-janelas menores, do lado esquerdo: *Current Folder* em cima, e *Workspace* em baixo. Há outras configurações de sub-janelas e barras de ferramentas que podem surgir, dependendo da versão do MATLAB e configuração inicial. De qualquer forma é possível alterar as janelas activas e a forma de apresentação, através de 'Layout'.

A consola principal (*Command Window*) é usada para chamar instruções, podendo por exemplo fazer-se correr uma função ou um *script* (guião com uma sequências de instruções). Na consola apresentada na figura 1.1 aparece o simbolo >>, o que indica que a consola está pronta para receber uma nova instrução (ou comando).

A janela *Workspace* mostra as variáveis activas, ou seja as que estão em memória. No inicio de uma sessão MATLAB, como a apresentada na figura 1.1, não há qualquer variável activa, pelo que a janela se encontra vazia.

A janela *Current Folder* mostra o conteúdo da pasta associada à sessão de trabalho. No caso ilustrado, há já alguns ficheiros nesta pasta, com a extensão .m, usada para ficheiros MATLAB. O utilizador pode verificar qual a

pasta activa na barra de ferramentas (*toolbar*) que aparece acima das 3 subjanelas, e escolher uma outra pasta. É boa prática criar uma pasta para cada projeto, devendo ser evitado o uso de caracteres especiais (acentos, cedilhas, espaços, etc.). No exemplo apresentada na figura 1.1 a pasta chama-se "APDS2020_TP2_Andre", que é uma boa escolha.

1.2 Usar o MATLAB como calculadora

A consola principal do MATLAB (Command Window) pode ser usada como uma calculadora. Por exemplo, inserindo a instrução `1+2` na consola (indicado a laranja abaixo), resulta no *output* `ans = 3` .

```
>> 1+2
ans = 3
```

Poderá observar que surgiu uma entrada na janela *Workspace*, onde é indicada uma variável com nome ans (*name*) e valor 3 (*value*). Ou seja, foi criada uma variável com a resposta (*answer*) resultante da última instrução. Podemos usar esta variável em conjunto com outras que estejam definidas. Por exemplo,

```
>> a=2/7;
>> b=a+ans
b = 3.2857
```

o resultado é colocado numa variável a

o ; suprime o output

A primeira instrução é usada para calcular 2/7 e colocar o resultado na variável a. No final foi colocado um ponto e virgula (;), o que faz com que seja omitida a apresentação do resultado (*output*) na consola. Na segunda instrução é calculada a soma das 2 variáveis (a e ans), sendo o resultado colocado na variável b. Uma vez que não se colocou um ; no final desta instrução, o output é apresentado na consola. O valor é apresentado com 4 casas decimais, que é a forma normal (*default*) usada pelo MATLAB. No entanto o cálculo é feito com mais precisão (16 algarismos significativos). Podemos alterar a forma de apresentação do conteúdo das variáveis através da função `format`, que pode ter vários argumentos, tais como: `short` para 4 casas decimais (que é o *default*), `long` para 16 algarismos significativos, `shortE` e `longE` para notação científica, etc. Por exemplo,

```
>> format long
>> b
b = 3.285714285714286
>> format short
```

activa o formato 'long'

para mostrar o conteúdo de b

repõe o formato normal

A função `format` também pode ser usada para alterar o espaçamento entre linhas: `format loose` para o modo habitual com 1 linha de espaçamento (*default*) ou `format compact` para suprimir as linhas de espaçamento.

Alguns números notáveis estão pré-definidos, tais como o π (`pi`) e a unidade complexa (`i` ou `j`), mas o número de Euler (e) não está. Pode no entanto ser facilmente definido, usando a função exponencial (`exp`). Por exemplo,

```
>> e=exp(1)
e = 2.7183
```

A tabela em baixo apresenta outras funções matemáticas de uso comum. Para cada função é indicada uma breve descrição, um exemplo de utilização, e o resultado apresentado pelo MATLAB (formato *standard*, i.e `format short`).

Função	Descrição	Exemplo	Resultado
`sqrt`	raiz quadrada	`sqrt(2)`	`1.4142`
`exp`	exponencial (de base e)	`exp(2)`	`7.3891`
`log`	logaritmo (de base e)	`log(10)`	`2.3026`
`log10`	logaritmo (de base 10)	`log10(10)`	`1`
`sin`	seno (angulo em radianos)	`sin(pi/3)`	`0.8660`
`cos`	coseno (angulo em radianos)	`cos(pi/3)`	`0.5000`
`tan`	tangente (angulo em radianos)	`tan(pi/3)`	`1.7321`
`acos`	arco-coseno	`acos(1/2)`	`1.0472`
`asin`	arco-seno	`asin(1/2)`	`0.5236`
`atan`	arco-tangente	`atan(1)`	`0.7854`

As funções trignométricas são em geral para ângulos em radianos, havendo no entanto também versões para ângulos em graus: `cosd`, `sind`, `tand`, `acosd`, etc. Uma outra alternativa é usar as funções de conversão entre radianos e graus (`deg2rad`) e `rad2deg`) que estão listadas na tabela abaixo. A tabela inclui igualmente funções para operações com complexos (`abs`, `angle`, `real` e `imag`) e de arredondamento (`round`, `floor` e `ceil`). A função módulo, ou valor absoluto (`abs`), é frequentemente usada para reais. Outras funções incluidas na tabela são: resto da divisão de inteiros (`rem`) e sinal (`sign`).

Função	Descrição	Exemplo	Resultado
`deg2rad`	Conversão de graus para radianos	`deg2rad(45)`	`0.7854`
`rad2deg`	Conversão de radianos para graus	`rad2deg(pi/3)`	`60.0000`
`abs`	Módulo (real ou complexo)	`abs(3-4i)`	`5`
`angle`	Fase (de um complexo)	`angle(3-4i)`	`-0.9273`
`real`	Parte real	`real(3-4i)`	`3`
`imag`	Parte imaginária	`imag(3-4i)`	`-4`
`floor`	arredondamento por defeito	`floor(4/3)`	`1`
`ceil`	arredondamento por excesso	`ceil(4/3)`	`2`
`round`	arredond. ao inteiro mais próximo	`round(4/3)`	`1`
`rem`	resto da divisão inteira	`rem(7/4)`	`3`
`sign`	sinal de um real (+1 ou -1)	`sign(-7/4)`	`-1`

.

1.3 Variáveis

Dependendo dos comandos que foram usados na sessão MATLAB, haverá um conjunto de variáveis definidas e valores atribuidos, o que poderá ser verificado na sub-janela *Workspace*. Uma outra forma de obter informação sobre as variáveis activas é através da função `whos`. Para eliminar (ou "limpar"a memória de) variáveis pode usar-se a função `clear`, que pode ser aplicada apenas a uma variável (ex. `clear a`) ou a todas (`clear` ou `clear all`). A seguinte sequência de instruções é útil para se ilustrar o processo.

```
>> clear all
>> a=1;
>> b=1+2i;
>> whos
Name      Size      Bytes      Class      Attributes
a         1x1       8          double
b         1x1       16         double      complex
```

limpar a memória

listar as variáveis em memória

O MATLAB usa em geral a estrutura de dados `double`, que é um formato de virgula-flutuante (*floating-point*) de 8 bytes (8x8=64 bits). A estrutura double permite representar valores entre cerca de 10^{-308} e 10^{+308}, com 16 algarismos significativos. A precisão é de 2^{-52}, ou seja aproximadamente 2×10^{-16} (pode ser verificado com a função `eps`). Apesar do conteúdo de `a` ser simplesmente 1, o MATLAB reservou 8 bytes para representar esta variável. No caso de `b`, são usados 16 bytes - um double para a parte real e outro para a parte imaginária.

Em certos casos é útil utilizar estruturas de dados diferentes, podendo as variáveis ser definidas inicialmente nesses formatos, ou convertidas posteriormente. Por exemplo, a função `int8` converte para inteiro de 8 bits (1 byte), ou seja um inteiro com valores entre -128 e 127. Por exemplo,

```
>> a8=int8(a);
>> whos a8
Name      Size      Bytes      Class      Attributes
a8        1x1       1          int8
```

A tabela abaixo apresenta alguns formatos de dados disponíveis no MATLAB para representar inteiros. No caso de inteiros sem sinal, há igualmente formatos de 16 (`uint16`), 32 (`uint32`) e 64 bits (`uint64`).

Tipo	Descrição	Nº bytes	Intervalo de valores
`int8`	inteiro de 8 bits	1	-128 a 127
`uint8`	inteiro de 8 bits sem sinal	1	0 a 255
`int16`	inteiro de 16 bits	2	-32768 a 32767
`int32`	inteiro de 32 bits	4	-2147483648 a 2147483647
`int64`	inteiro de 64 bits	8	-9.2 E18 a 9.2 E18]

Um ponto que é importante referir é que o MATLAB é *case-sensitive*, ou seja, o uso de maiusculas e minusculas não é indiferente (por exemplo `f` e `F`). Neste exemplo é definida a variável `f`, mas quando se chama a variável `F` (que não está definida), o MATLAB exibe uma mensagem de erro (a vermelho).

```
>> f=4;
>> F
Undefined function or variable 'F'.
```

mensagem de erro

É possivel gravar as variáveis que estão em memória (*Workspace*) num ficheiro, através da função `save`. Por exemplo, usando `save teste`, é criado um ficheiro 'teste.mat', o que pode ser verificado a partir da janela *Current Folder*. Em seguida, e fazendo primeiro `clear` para limpar a memória, podemos recuperar o *Workspace* anterior através de `load teste`.

1.4 Mais informação

O MATLAB tem uma extensa documentação de apoio, que pode ser consultada através da janela *Help*, normalmente acessível a partir de um *tab* (canto superior direito da figura 1.1). Pode igualmente ser obtida alguma informação directamente a partir da consola, usando a instrução `help`. Por exemplo, `help cos` mostra alguma informação relativa à função coseno (`cos`) na consola, embora com menos detalhe do que a informação obtida a partir da janela *Help*.

Em seguida são indicadas algumas informações relativas ao uso do MATLAB que poderão ser úteis:

- A função `clc` permite limpar a consola.
- A janela *Command History* mostra uma sequência das últimas instruções executadas, incluindo instruções de sessões anteriores.
- As setas para cima baixo podem ser usadas para chamar instruções anteriores. Poderá igualmente ser usado após a utilização de um ou mais caracteres na consola, chamando as entradas anteriores com inicio igual a essa sequência de caracteres. Por exemplo´, se se colocar `cl` na consola e se for carregando na seta para cima, vão aparecendo as instruções mais recentes começadas por cl.
- A versão do MATLAB pode ser verificada através das funções `version` e `ver` (com detalhes sobre as *Toolboxes*).

2 Vetores e Matrizes

Até ao momento, todas as variáveis utilizadas foram definidas com apenas um elemento, real ou complexo. No entanto, o grande potencial do MATLAB é precisamente a forma como opera com vetores e matrizes.

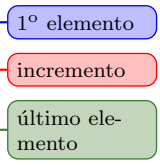
2.1 Criação de vetores

Um vetor linha (matriz com apenas 1 linha) pode ser definido colocando a lista de valores que se pretende atribuir dentro de parênteses retos, separados por espaço ou virgula. Por exemplo,

```
>> v=[1 5 6];
v = 1 5 6
>> u=[1, 3, 8];
u = 1 3 8
```

O operador `:` é particularmente útil para a criação de sequências de valores, como ilustrado nos seguintes exemplos. No primeiro caso são apenas indicados 2 valores (o primeiro e último elemento da sequência), pelo que o incremento toma o valor 1. No segundo caso são indicados 3 valores, separados por `:`, sendo o elemento do meio o valor do incremento.

```
>> v1=1:7
v1 = 1 2 3 4 5 6 7
>> v2=1:2:13
v1 = 1 3 5 7 9 11 13
```



Nestes casos não foi necessário usar `[]` para a definição dos vetores uma vez que foi possível usar apenas um "bloco"de valores. Para se criar um vetor coluna, pode usar-se o separador `;` ou a função transposta, através de `transpose` ou do operador `'`. O processo é ilustrado em seguida, onde são usadas 3 formas distintas para criar o mesmo vetor coluna.

```
>> Vcoluna1=[1;2;3;4;7];
>> Vlinha=[1:4 7];
>> Vcoluna2=transpose(Vlinha);
>> Vcoluna3=Vlinha'
Vcoluna3 =
     1
     2
     3
     4
     7
```

2.2 Operações com vetores

O modo normal de actuação do MATLAB é sobre vetores ou matrizes. Quando se chama uma função (por exemplo `sqrt`) com uma variável de entrada, esta pode ser uma variável simples (como em 1.2), ou um vetor (ou matriz). Por

exemplo, o código apresentado em seguida calcula a raiz quadrada de cada elemento do vetor `a`.

```
>> a=1:3:16
a = 1 4 7 10 13 16
>> b=sqrt(a)
b= 1.0000 2.0000 2.6458 3.1623 3.6056 4.0000
```

Há várias funções muito úteis para extrair informação sobre vetores. Por exemplo o comprimento (número de elementos) do vetor (`length`), soma dos elementos (`sum`), média (`mean`), desvio padrão (`std`), etc. O uso destas funções é direto, conforme ilustrado em seguida.

```
>> v=[1:4 7 0 2]
v = 1 2 3 4 7 0 2
>> NoElem = length(v)
NoElem = 7
>> Soma = sum(v)
Soma = 19
>> Media = mean(v)
Media = 2.7143
>> DesvPad = std(v)
DesvPad = 2.2887
```

As funções máximo (`max`) e mínimo (`min`) também podem ser usadas de forma semelhante aos exemplos anteriores. No entanto é possível obter não só o valor máximo (ou mínimo), como a posição em que esse valor ocorre pela primeira vez. Nesse caso é necessário considerar os 2 *outputs* da função, como no exemplo seguinte:

```
>> Valor = max(v)
Valor = 7
>> [Valor,Posicao] = max(v)
Valor = 7
Posicao = 5
```

Apenas 1 output

2 outputs

2.3 Criação de matrizes

Os vetores linha ou coluna, apresentados em 2.1, são casos particulares de matrizes. A criação de uma matriz (com mais de 1 linha / coluna) pode ser feita diretamente, colocando os valores entre parênteses retos, separados por espaço ou virgula, ao longo de uma linha, e separados por ponto e virgula para mudar de linha. Neste exemplo é criada uma matriz `M` de 3x4 elementos, sendo neste caso usado o espaço como separador entre elemento ao longo das linhas.

```
>> M=[1 2 3 1; 4 0 5 1; 0 0 2 0]
M =
     1     2     3     1
     4     0     5     1
     0     0     2     0
```

Para além desta forma básica, é possivel criar matrizes através de concatenação de outras matrizes. Neste exemplo é criada uma nova matriz (`C`), de 3x4 elementos, através da concatenação (ou agrupamento) de uma matriz linha de 1x4 com a matriz `M` de 3x4.

```
>> C=[6:9 ; M]
C =
     6     7     8     9
     1     2     3     1
     4     0     5     1
     0     0     2     0
```

Também é possível alterar valores de 1 ou de vários elementos de uma matriz. Por exemplo, para alterar o valor do elemento da linha 3 coluna 2 (3,2), de M

```
>> C(3,2)=6
     6     7     8     9
     1     2     3     1
     4     6     5     1
     0     0     2     0
```

O operador `:` pode ser usado para modificar uma linha ou coluna completa, ou um bloco de elementos. Em seguida um exemplo de cada caso:

```
>> C(:,3)=0
     6     7     0     9
     1     2     0     1
     4     6     0     1
     0     0     0     0
>> C(2:3,1:3)=5
     6     7     0     9
     5     5     5     1
     5     5     5     1
     0     0     0     0
```

todos os elementos da coluna 3

linhas 2 a 3 e colunas 1 a 3

Uma outra possibilidade é a eliminação de linhas ou colunas, o que pode ser feito usando o operador `[]`. Por exemplo, para eliminar a linha 2 da matriz `C`,

```
>> C(2,:)=[]
     6     7     0     9
     5     5     5     1
     0     0     0     0
```

O MATLAB tem funções para criar matrizes com todos os elementos iguais a 1 (`ones`) ou 0 (`zeros`), que são particularmente úteis. Estas funções podem receber apenas um argumento N (inteiro), criando uma matriz de NxN elementos, dois argumentos (M,N) para criar uma matriz rectangular de M linhas por N colunas, ou mais argumentos (por ex. 3) para matrizes de dimensão 3 ou superior. Em seguida alguns exemplos,

```
>> A=ones(3)
A =
     1     1     1
     1     1     1
     1     1     1
```

```
>> B=zeros(2,4)
B =
     0     0     0     0
     0     0     0     0
```

As funções `rand` e `randn` podem ser usadas para criar matrizes com números aleatórios (ou pseudo-aleatórios), sendo a sintaxe semelhante à função `ones`. Estas funções cria um conjunto de números aleatórios com distribuição uniforme no intervalo]0, 1[(`rand`), ou distribuição normal (Gaussiana) com média 0 e variância 1 (`randn`). A função `randi` gera um conjunto de números aleatórios inteiros, havendo uma ligeira diferença na sintaxe - há um parâmetro adicional de input (o 1º) com o maior inteiro pretendido. Alguns exemplos em seguida,

```
>> Ru=rand(3)
Ru =
     0.9961    0.1067    0.7749
     0.0782    0.9619    0.8173
     0.4427    0.0046    0.8687
>> Rn=randn(2,4)
Rn =
    -1.0582   -0.2725   -0.2779   -2.0518
    -0.4686    1.0984    0.7015   -0.3538
>> Ri=randi(9,2,6)
Ri =
     5     7     7     4     8     9
     3     2     2     6     1     7
```

3x3 elementos com dist. uniforme

2x4 elementos com dist. normal

2x6 inteiros com dist. uniforme

Existem outras funções no MATLAB para a criação de matrizes, tais como: `eye` (matriz identidade), `pascal` (matriz de Pascal), `magic` (matriz com soma constante para todas as colunas e linhas), etc. Uma ferramenta particularmente útil para a manipulação de matrizes é a função `reshape`, que permite modificar a estrutura de uma matriz. A sintaxe é `Mo=reshape(Mi,nl,nc)`, onde `Mi/Mo` são as matrizes *intput/output*, e `nl/nc` são o número de linhas/colunas da nova matriz. Por exemplo,

```
>> Rout=reshape(Ri,3,4)
Rout =
     5     2     4     1
     3     7     6     9
     7     2     8     7
>> Rvl=reshape(Ri,1,[])
Rvl =
     5     3     7     2     7     2     4     6     8     1     9     7
>> Rvc=Ri(:);
```

matriz output de 3x4

o operador [] é usado como 'joker'

o operador : é todos. Rvc vetor coluna

Há várias fomas de se saber qual a estrutura de uma matriz, para além da observação da janela *Workspace*, da chamada da variável na consola, ou do uso da instrução `whos`. A função `numel` devolve o número de elementos de uma matriz, à semelhança de `length` para vetores (2.2). A função `size` permite obter o nº de linhas e colunas, e também o número de planos ou bandas, para matrizes tri-dimensionais (usadas em imagens coloridas, vídeos, etc.). Em seguida alguns exemplos de utilização,

```
>> No=numel(Ri)
No = 12
>> [NL,NC]=size(Ri)
NL = 2
NC = 6
```

2.4 Operações com matrizes

Como referido em 2.2, o modo normal de actuação do MATLAB é sobre matrizes. A aplicação de uma função 'comum' a uma matriz devolve como output uma matriz do mesmo tamanho, com o resultado da função aplicado a cada elemento. Por exemplo, para a função trignométrica seno,

```
>> M=rand(2,3)
M =
    0.7094    0.2760    0.6551
    0.7547    0.6797    0.1626
>> S=sin(M)
S =
    0.6514    0.2725    0.6092
    0.6851    0.6286    0.1619
```

sin(0.7094)
sin(0.2760)
sin(0.6551)

As operações aritméticas simples podem ser feitas entre variáveis elementares (matriz de 1x1) ou matrizes (os vetores são casos particulares de matrizes com apenas 1 linha ou coluna). Um aspecto importante é que no MATLAB o modo normal (*default*) de operar é para matrizes. Por exemplo, o operador de multiplicação (`*`) é usado para multiplicação de matrizes, enquanto que a multiplicação de matrizes elemento a elemento é feita através do operador `.*` .

```
>> A=[1 2 ; 3 4]
A =
    1    2
    3    4
>> B=[2 0 ; 1 1]
B =
    2    0
    1    1
>> MM=A*B
MM =
    4    2
   10    4
>> MEE=A.*B
MEE =
    2    0
    3    4
>> C=3*A
C =
    3    6
    9   12
```

multiplicação de matrizes

multiplicação elemento a elemento

multiplicação de constante por matriz

Operador	Descrição	Exemplos
`+`	Soma	`A+B` , `A+2`
`-`	Subtração	`A-B` , `A-1`
`*`	Multiplicação	`A*B` , `2*A`
`.*`	Multiplicação elemento a elemento	`A.*B`
`/`	Divisão (à direita)	`A/B` , `A/2`
`./`	Divisão elemento a elemento	`A./B`
`^`	Potenciação	`A^2`
`.^`	Potenciação elemento a elemento	`A.^2`

A tabela acima apresenta os principais operadores aritméticos do MATLAB, sendo o modo normal para operações entre matrizes, e a versão com ponto para operações elemento a elemento. O exemplo seguinte ilustra as diferenças entre os operadores com e sem ponto, para a potenciação,

```
>> A=[1 2 ; 3 4];
>> A^2
     7    10
    15    22
>> A.^2
     1     4
     9    16
```

semelhante a ter A*A

semelhante a ter A.*A

Há um grande número de funções MATLAB relacionadas diretamente com matrizes, estando algumas das mais usadas listadas na tabela em baixo. Para se obter a transposta de uma matriz pode usar-se a função MATLAB `transpose` ou, para matrizes reais, o operador `'` (transposta do complexo conjugado).

Função	Descrição	Exemplo
`transpose`	Transposta (ou `'` para matriz de reais)	`transpose(A)`
`det`	Determinante	`det(A)`
`inv`	Matriz inversa	`inv(A)`
`trace`	Traço (soma dos elementos diagonal)	`trace(A)`
`triu`	Triangular superior (upper)	`triu(A)`
`tril`	Triangular inferior (lower)	`tril(A)`
`diag`	Elementos da diagonal	`diag(diag(A),0)`

Em seguida apresentam-se dois exemplos que usam funções e operações simples com matrizes. No primeiro caso cria-se uma matriz Q com 3 linhas iguais, com valores impares entre 1 e 9, através da multiplicação de uma matriz coluna (com todos os elementos iguais a 1), e uma matriz linha (com os 5 impares).

```
>> vl=1:2:9;
>> vc=ones(4,1);
>> Q=vc*vl
Q =
     1     3     5     7     9
     1     3     5     7     9
     1     3     5     7     9
     1     3     5     7     9
```

vl é uma matriz linha

vc é uma matriz coluna

multiplicação de matrizes

No segundo caso cria-se uma matriz R, de 3x4, com números aleatórios, uniformemente distriguidos entre 1 e 10. A função `rand(3,4)` cria a matriz de 3x4 com distribuição uniforme entre 0 e 1, e os coeficientes multiplicativo (9) e aditivo (1) fazem a transformação do domínio de [0,1] para [1,10].

```
>> R=9*rand(3,4)+1
R =
    6.5444    8.4775    3.5726    7.7836
    5.2596    6.2674    9.2547    4.4240
    4.1649    5.9475    7.8148    6.1104
```

A aplicação da função `max` a uma matriz devolve como output um vetor (matriz linha) com os valores máximos de cada coluna da matriz de input. As funções `min`, `mean`, `std`, etc. tem um funcionamento semelhante, que é ilustrado em seguida para a matriz (`R`).

```
>> mean(R)
ans =
    5.3230    6.8974    6.8807    6.1060
>> Rmedio=mean(mean(R))
Rmedio = 6.3018
>> max(R)
ans =
    6.5444    8.4775    9.2547    7.7836
>> Rmax=max(max(R))
Rmax = 9.2547
```

Caso se pretenda também obter a posição (linha e coluna) do valor máximo, pode utilizar-se a seguinte sequência de instruções, onde `vvm` e `vpl` são vetores com os valores máximos (para cada coluna) e respectivas posições (linha). A posição correspondente ao valor máximo (`Rmax`) da matriz `R` tem coordenadas `PosL` (linha) e `PosC` (coluna).

```
>> [vvm, vpl]=max(R)
vvm =
    6.5444    8.4775    9.2547    7.7836
vpl =
    1    1    2    1
>> [Rmax, PosC]=max(vvm)
Rmax = 9.2547
PosC = 3
>> PosL=vpl(PosC)
PosL = 2
```

A função `max` (ou `min`) também pode ser usada com 2 variáveis de input, da forma `C = max(A,B)`. Neste caso `A` e `B` terão de ser do mesmo tamanho, sendo C também desse tamanho e com o valor máximo dos dois inputs elemento a elemento. A função `max` pode ainda ser usada para comparar uma matriz com um escalar (ex. `C = max(A,4)`).

À semelhança das funções `max` e `min`, há habitualmente várias formas de se usar uma função MATLAB. A documentação de apoio (em *help*) fornece informação detalhada sobre a sintaxe de cada função, normalmente com exemplos, e ligações a outras páginas relacionadas.

2.5 Operadores

O MATLAB usa operadores relacionais e lógicos de forma semelhante a outras linguagens de programação. A tabela abaixo apresenta uma lista de operadores ralacionais, que podem ser usados para testar variáveis simples, vetores ou matrizes. Nos exemplos apresentados na tabela, consideram-se as seguintes variáveis: `a=2`, `v=[1 2 0 3]` e `u=[1 2 4 3]`.

Operador	Descrição	Exemplo	Resultado
`<`	Menor	`a<2`	0
`<=`	Menor ou igual	`a<=2`	1
`>`	Maior	`v>2`	0 0 0 1
`>=`	Maior ou igual	`v>=2`	0 1 0 1
`==`	Igual (testa igualdade)	`v==u`	1 1 0 1
`~=`	Diferente	`v~=u`	0 0 1 0

Estes operadores podem por exemplo ser usados para criar um novo vetor (`vig`), apenas com os elementos que são iguais nos vetores `v` e `u`. Ou, em alternativa, um vetor (`udif`) com os elementos de `u` que são diferentes de `v`.

```
>> v=[1 2 0 3];
>> u=[1 2 4 3];
>> vig=v(v==u)
vig =
     1     2     3
>> udif=u(v~=u)
udif = 4
```

os elementos de v que satisfazem a condição

1 1 0 1

elementos de u que são diferentes de v

Os operadores lógicos são frequentemente usados em instruções condicionais (do tipo `if`), mas podem igualmente ser usados noutros contextos. A tabela abaixo apresenta uma lista de operadores lógicos. Estes operadores consideram os valores lógicos 1 e 0 (sim/não ou on/off), sendo valores não nulos ser interpretados como 1. Nos exemplos apresentados na tabela, consideram-se as seguintes variáveis: `a=2`, `b=3`, `va=[1 2 0 3]` e `vb=[1 0 0 2]`.

Operador	Descrição	Exemplo	Resultado
`&`	E (AND) elemento a elemento	`va & vb`	1 0 0 1
`|`	OU (OR) elemento a elemento	`va | vb`	1 1 0 1
`~`	NÃO (NOT) elemento a elemento	`~ va`	0 0 1 0
`&&`	E (AND), variáveis simples	`a>2 && b>1`	0
`||`	OU (OR), variáveis simples	`a>2 || b>1`	1
`xor`	OU EXCLUSIVO	`xor(va,vb)`	0 1 0 0
`all`	1 se todos elementos forem $\neq 0$	`all(va)`	0
`any`	1 se algum elemento for $\neq 0$	`any(va)`	1

No caso do operador `xor` (ou exclusivo), os elementos que são ambos 0 ou ambos diferentes de 0 tem output 0, tendo os elementos com um valor 0 e um valor não 0 output 1.

3 Scripts e Funções

A utilização da consola do MATLAB para a execução de instruções é útil, mas algo limitada. Uma forma alternativa, mais eficiente em muitos casos, é a criação de sequências de instruções (scripts) e de funções MATLAB. Isso pode ser feito em qualquer editor de texto, gravando o ficheiro texto com extensão `.m`, mas é normalmente feito através do próprio editor do MATLAB. O editor pode ser chamado de várias formas, dependendo da versão do MATLAB, por exemplo através de 'New' e escolhendo 'New Script'.

3.1 Criação de um script

Nesta secção será considerado um script para a criação de um conjunto de valores aleatórios, inteiros entre 1 e 6 com distribuição uniforme (simulando o lançamento de um dado), e uma breve análise dos dados. Inicialmente é colocada a seguinte lista de instruções no editor.

```
% Exemplo de Script - S1
N=100;
D=randi(6,1,N);
Dm=mean(D)
P6=100*sum(D==6)/N
```

o simbolo % é usado para colocar comentários

vetor com 1xN elementos

percentagem de lançamentos = 6

A primeira linha é simplesmente um comentário, não tendo qualquer impacto no funcionamento do script. De qualquer forma é aconselhável utilizar comentários para incluir informação relevante sobre o script ou função. Neste exemplo o número de lançamentos (N) foi colocado a 100. Não se usou o ; nas últimas 2 instruções, para aparecer na consola os valores das variáveis `Dm` (valor médio dos N lançamentos) e `P6` (percentagem de lançamentos 6).
O script deverá ser gravado, neste caso com o nome `S1.m`. Este ficheiro deverá surgir em *Current Folder*. A execução do script pode ser feita através do editor, por exemplo carregando no botão 'Run', ou chamando o script na consola, como se ilustra em seguida.

```
>> clear all
>> S1
Dm = 3.4300
P6 = 14
```

elimina todas as variáveis

corre o script

3.2 Criação de uma função

A primeira função considerada é uma variação do script `S1`, mas que recebe como parâmetro de entrada (input) um valor para N. Inicialmente deverá ser aberto um novo documento no editor, através de 'New Function', onde deverá ser colocado o código da função (`F1.m`).

```
% Exemplo de Função - F1
function []=F1(N)
    D=randi(6,1,N);
    Dm=mean(D)
    P6=100*sum(D==6)/N
end
```

não tem outputs

1 parametro de input, N

A função deverá ser gravada com o nome `F1.m`, que deverá aparecer em *Current Folder*. Uma vez que a função necessita de um parâmetro de entrada (input), pode ser chamada através da consola, por exemplo da seguinte forma,

```
>> clear all
>> F1(100)
Dm = 3.5600
P6 = 18
>> F1(200)
Dm = 3.5300
P6 = 19.5000
```

corre a função

atribui 100 a N

repetição com N=200

É interessante observar que não há qualquer variável em *Workspace*, uma vez que as variáveis usadas são locais (internas da função), sendo eliminadas quando a função termina. Caso se pretenda manter os valores calculados para uso posterior, podemos usar uma versão diferente da função (`F2.m`), que devolve 2 variáveis como output.

```
% Exemplo de Função com 1 input e 2 outputs - F2
function [Dm,P6]=F2(N)
    D=randi(6,1,N);
    Dm=mean(D));
    P6=100*sum(D==6)/N;
end
```

2 parametros de output

; para suprimir output

; para suprimir output

A função deverá ser chamada na consola, por exemplo da forma indicada em baixo. Neste caso como não se colocou `;` no final os valores atribuidos às variáveis são apresentados na consola. É de referir que os nomes das variáveis colocados para receber os 2 outputs da função não tem de ser iguais aos nomes na função.

```
>> [media,per6]=F2(1000)
media = 3.4290
per6 = 14.6000
```

Como exemplo final, apresenta-se um script (`S2`) que chama a função `F2` 5 vezes, com valores de N de 100 a 500, com incrementos de 100.

```
% Exemplo de script que chama função - S2
for f=1:5
    N=100*f;
    [media(f),per6(f)]=F2(N);
end
media
per6
```

ciclo for

suprime output

mostrar valores

mostrar valores

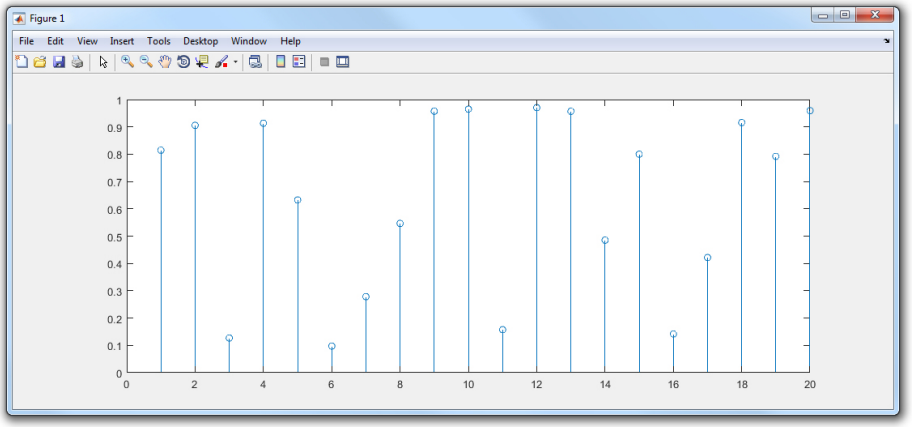


Figura 2: Janela com gráfico produzido através da função `stem`.

4 Gráficos

Nesta secção são consideradas algumas formas de apresentação de dados em formato gráfico, havendo no entanto muito mais opções disponíveis no MATLAB. Os vários tipos de gráficos são apresentados com exemplos de códigos, que deverão ser experimentados, e algumas figuras.

4.1 Gráficos simples

As funções mais utilizadas para gráficos simples (1 dimensão) são `plot` e `stem`, sendo este último mais indicado para vetores com poucos elementos. Em seguida um exemplo de utilização da função `stem`,

```
>> y=rand(1,20);
>> stem(y)
```

A execução da instrução `stem(y)` resulta na abertura de uma janela gráfica, como a que é apresentada na figura 2, que tem algumas funcionalidades disponívies (de edição, zoom, gravação, etc.). Caso se crie um novo gráfico, ele irá ser colocado nesta janela gráfica (activa), substituindo o anterior. Outra alternativa é abrir-se uma nova janela gráfica, usando a função `figure`, como se exemplifica em seguida.

```
>> x=-10:10;
>> y=randn(1,21);
>> figure
>> stem(x,y)
```

vetor para abcissa

vetor para ordenada

abre uma nova janela gráfica

2 vetores do mesmo tamanho

As funções `stem` e `plot` podem receber apenas um vetor de input, que é considerado para os valores de y (ordenada), sendo nesse caso usados valores inteiros para x (abcissa) a começar em 1 (exemplo da figura 2). Quando se usam 2 inputs, o primeiro é para os valores de x e o segundo para os valores de y.

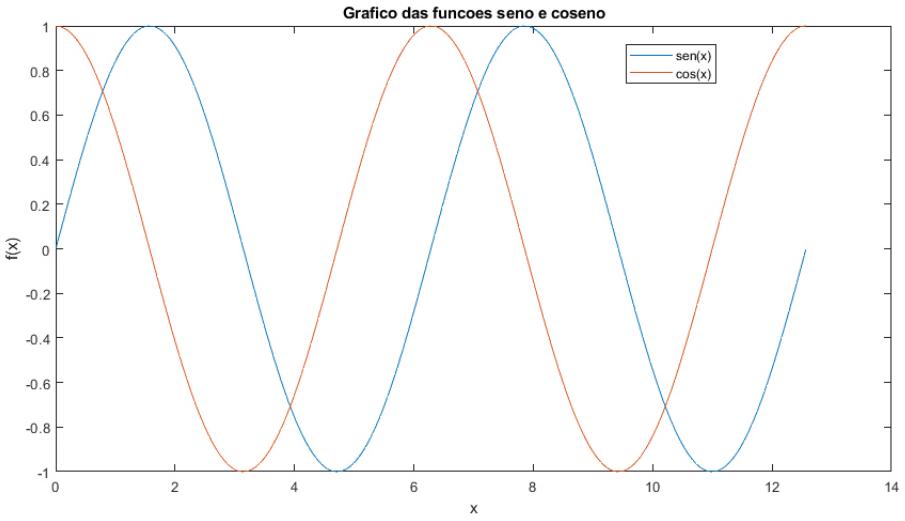


Figura 3: Janela com gráfico produzido através da função `stem`.

Há várias funções para colocar informação adicional em gráficos, tais como: `xlabel` e `ylabel` (etiquetas para os eixos xx / yy, abcissas / ordenadas), `title` (título), `legend` (legenda). A utilização destas funções é ilustrada através de um exemplo criado a partir do script S3. É no entanto conveniente primeiro limpar a memória e fechar as janelas, o que pode ser feito da forma indicada abaixo. Neste exemplo são colocadas 2 instruções na mesma linha, separador por uma virgula (`,`). É possível colocar várias intruções na mesma linha, separadas por `,` ou `;` (para suprimir o output), mas essa abordagem é evitada aqui para simplificar a explicação.

```
>> clear all, close all

% S3 - cria gráfico das funções seno e coseno
x=0:pi/100:4*pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,x,y2)
xlabel('x')
ylabel('f(x)')
title('Grafico das funcoes seno e coseno')
legend('sen(x)','cos(x)')
```

fecha todas as janelas

vetor com 401 elementos

um par de vetores para cada função

A execução deste script produz um gráfico como o apresentado na figura 3. É de referir que, neste caso, a posição da legenda foi alterada manualemte (carregando com o rato e arrastando), para evitar sobreposição com o gráfico da função coseno. Também é possível alterar o tamanho e tipo de letra, formato e cor das linhas, etc., havendo nas páginas help do MATLAB uma explicação da sintaxe e alguns exemplos. Uma vez que o número de pontos é razoavelmente

grande (401), foi neste caso utilizada a função `plot`, que une com segmentos de recta os pontos consecutivos. Para se usar `stem` com 2 vetores nas ordenadas, a sintaxe seria algo diferente: `stem(x',[y1',y2'])`. Ou seja um vetor coluna (`x'`) com os valores da abcissa, e uma matriz (`[y1',y2']`) com uma coluna para cada variável representada nas ordenadas.

Os limites dos intervalo de valores apresentados para x/y é determinado de forma automática pelo MATLAB, podendo no entanto ser alterado através da função `axis`. A sintaxe é `axis([xmin,xmax,ymin,ymax])`, sendo `xmin,xmax` o intervalo para x (abcissas) e `ymin,ymax` o intervalo para y (ordenadas). Neste caso poderia por exemplo usar-se `axis([0,4*pi,-1,1])` (ver figura 3).

4.2 Sub-janelas

Uma forma particularmente conveniente de representar dados gráficamente é através da utilização da função `subplot`, que permite dividir a janela gráfica em sub-janelas. A sintaxe é `subplot(nl,nc,nja)`, sendo `nl,nc` o número de linhas e de colunas em que se sub-divide a janela, e `nja` o número da sub-janela activa, sendo a contagem feita com a ordem de leitura habitual (da esquerda para a direita e depois de cima para baixo). O processo é ilustrado através do script S4, que dá origem à figura 4.

```
% S4 - cria gráfico das funções seno e coseno
x=0:pi/100:4*pi;
y1=sin(x);
y2=cos(x);
y3=sin(2*x);
y4=sin(x).*cos(x);
subplot(2,2,1)
plot(x/pi,y1)
xlabel('x/\pi'), ylabel('y'), title('sin(x)')
subplot(2,2,2)
plot(x/pi,y2)
xlabel('x/\pi'), ylabel('y'), title('cos(x)')
subplot(2,2,3)
plot(x/pi,y3)
xlabel('x/\pi'), ylabel('y'), title('sin(2x)')
subplot(2,2,4)
plot(x/pi,y4)
xlabel('x/\pi'), ylabel('y'), title('sin(x)cos(x)')
```

divide em 2x2 sub-janelas e activa a 1a

x/π na abcissa

\pi para simbolo π

três instruções numa linha

Neste exemplo a janela foi dividida em 2x2 sub-janelas, e foi usada a função `plot` para colocar o gráfico de uma função em cada uma delas. Os gráficos nas sub-janelas 3 e 4 são iguais, uma vez que sen(2x)=2sen(x)cos(x). Note-se o uso do operador `.*` (multiplicação elemento a elemento) para o cálculo do vetor `y4`. Para tornar o script um pouco mais compacto, colocaram-se 3 instruções numa só linha para as etiquetas e título de cada gráfico. Um outro ponto novo é o uso de caracteres especiais (π), o que é feito através de `\pi`, à semelhança do LaTeX (outros exemplos: `\alpha`, `\delta`, `\sigma`, `\omega`, etc).

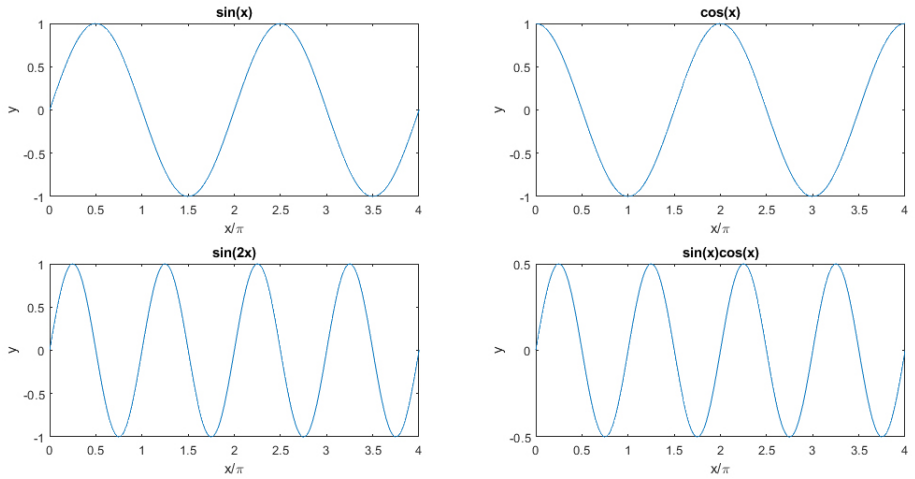


Figura 4: Apresentação de 4 gráficos numa janela através de `subplot`.

É também possível apresentar um gráfico a ocupar mais do que uma sub-janela. Por exemplo, através de `subplot(3,2,[1,2])` é activada uma região correspondendo às sub-janelas 1 e 2 da janela gráfica MATLAB (dividida neste caso em 3x2 secções). Desta forma é possível apresentar gráficos de diferentes tamanhos na mesma janela.

4.3 Histogramas, gráficos 3D, etc.

Há muitas outras funções interessantes para representação gráfica de dados, tais como `bar` (gráficos de barras), `histogram` (histogramas), `pie` (gráficos circulares), etc.

Para representação de gráficos 3D, para funções ou dados a 2 variáveis f(x,y), pode por exemplo usar-se: `plot3` , `scatter3` ou `contour3`.

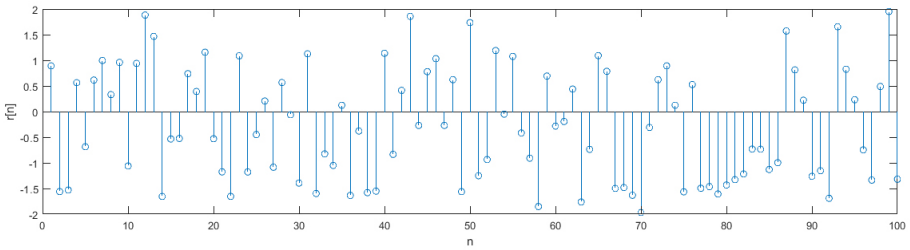


Figura 5: Exemplo de sinal sintético aleatório, com 100 elementos.

5 Processamento de Sinal - Intro.

Um sinal de variável discreta (uma sequência) pode ser facilmente criado no MATLAB usando um vetor. Por exemplo, um sinal $r[n]$ com 100 elementos, correspondendo a ruído aleatório uniformemente distribuido entre -2 e 2, pode ser criado da seguinte forma:

```
>> r=4*rand(1,100)-2;
>> stem(r);
>> xlabel('n') , ylabel('r[n]');
```

vetor linha com 100 elementos

stem com 1 argumento

A função `rand` gera números aleatórios com distribuição uniforme entre 0 e 1, tendo neste caso sido usado os valores `1,100` para se criar um vetor (linha) com 100 elementos (matriz de 1×100). Os factores multiplicativo (4) e aditivo (-2) são usados para ajustar o intervalo de valores de $[0, 1]$ para $[-2, 2]$. Neste caso a função `stem` é usada com apenas 1 argumento, pelo que os valores das abcissas (n) são apresentados como inteiros a começar em 1 (figura 5). Caso se pretenda ter outra correspondência entre os valores do vetor r e n no gráfico, isso pode ser feito por exemplo através do código abaixo, onde n começa em 0. A função `stem` recebe neste caso 2 vetores (do mesmo comprimento), com os valores de abcissa e ordenada a usar na representação gráfica.

```
>> n=0:99;
>> stem(n,r);
```

stem com 2 argumentos

Um segundo exemplo é a criação de um sinal sinusoidal real $s[n]$. Este sinal é definido por $s[n] = Acos\big(2\pi\nu n + \phi\big)$, onde ν é a frequência, ϕ a fase, com n a tomar valores entre 0 e 80, podendo ser criado através do seguinte código:

```
>> n=0:80;
>> A=4; freq=0.05; fase=0;
>> s = A*cos(2*pi*freq*n - fase);
>> stem(n,x);
>> axis([ 0 80 -4 4]);
>> grid;
>> xlabel('n') , ylabel('s[n]');
```

vetor linha com 81 elementos

limites dos eixos

para incluir grelha

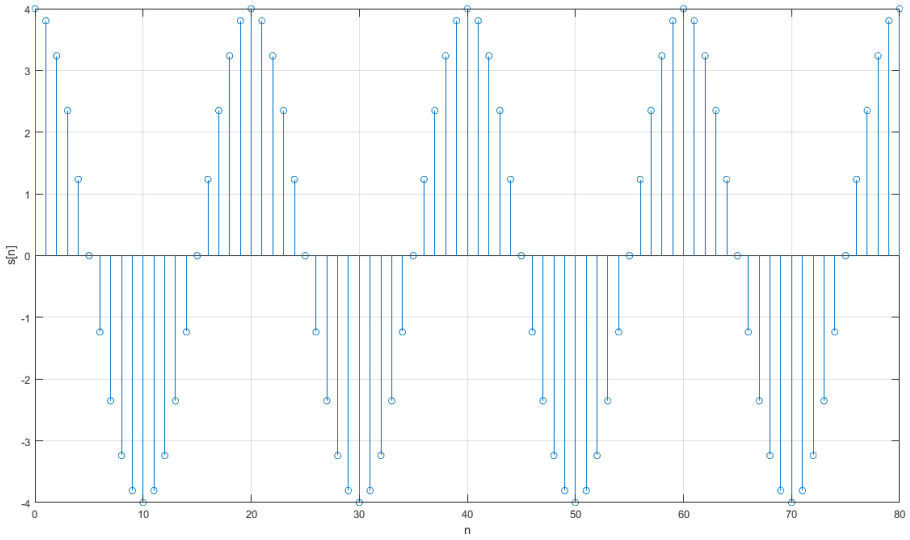


Figura 6: Exemplo de um sinal sinusoidal real.

A figura 6 mostra o sinal $s[n]$ onde se pode observar a grelha colocada através da função `grid`.

O terceiro exemplo considerado é a criação de um sinal sinusoidal complexo $x[n]$. Em baixo é apresentado o código usado para criar o sinal $x[n]$, e apresentar o gráfico da componente real (figura 7, em cima).

```
>> c=-0.1+0.2*pi*i;
>> A=1.9;
>> n=0:40;
>> x=A*exp(c*n);
>> subplot(4,1,1)
>> stem(n,real(x))
>> xlabel('n')
>> ylabel('Re \{x[n]\}')
>> title('Componente Real')
```

unidade imaginária

41 elementos

componente real de x

\{ para apresentar {

As restantes componentes do sinal, apresentadas na figura 7, foram calculadas usando as funções `imag(x)` (parte imaginária), `abs(x)` (módulo) e `angle(x)` (fase). Um detalhe a destacar é a inclusão de uma letra grega (θ) na etiqueta das ordenadas do gráfico da fase. A sintaxe para o uso destes caracteres especiais é semelhante ao LaTeX. Neste caso foi usada a seguinte instrução:

```
>> ylabel('\theta {x[n]}')
```

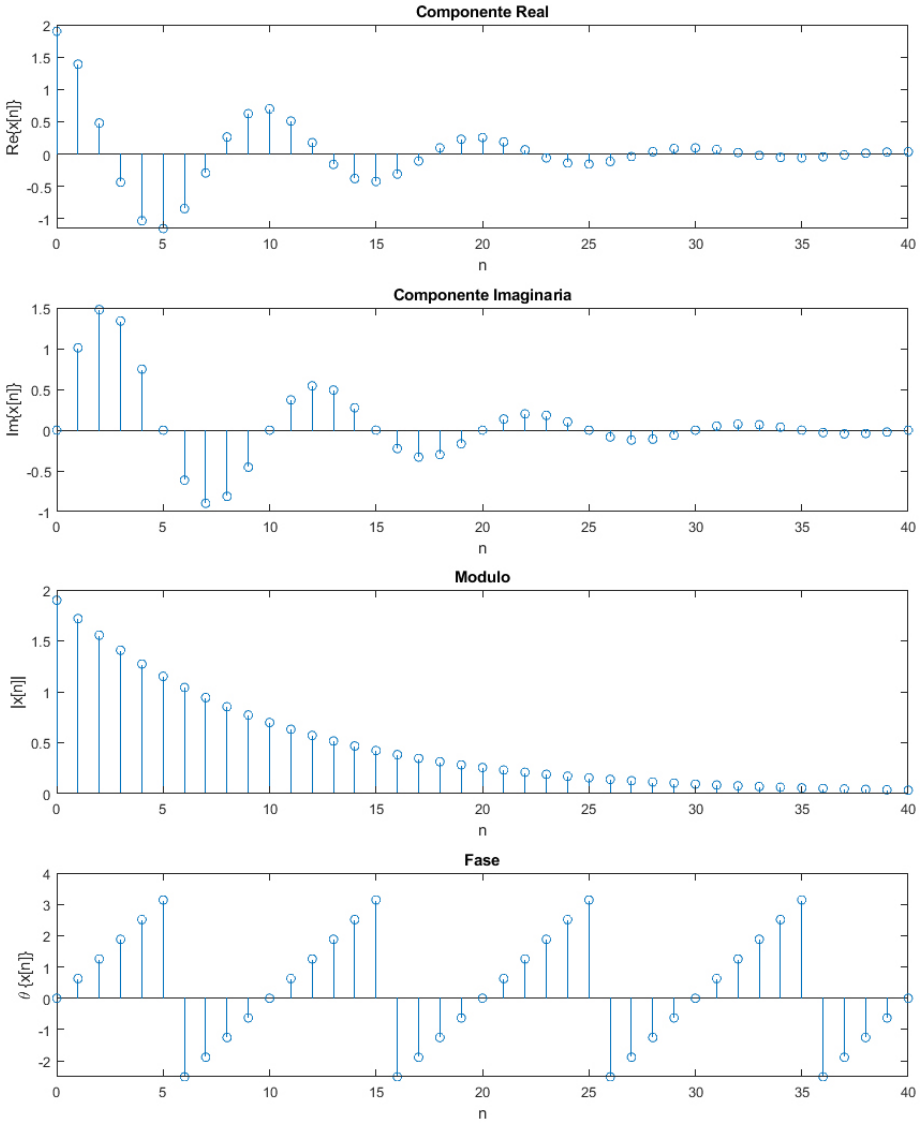


Figura 7: Exemplo de sinal sinusoidal complexo, com 41 elementos. De cima para baixo: componentes real, imaginária, módulo e fase.

6 Processamento de Imagem - Intro.

Esta secção apresenta alguns elementos relevantes para processamento digital de imagem com o MATLAB, alguns ligados à *Image Processing Toolbox* (IPT). A IPT considera 4 tipos de imagens: tons de cinzento (*Gray-scale*), binárias, cor indexada, e cor RGB.

6.1 Formatos de dados

Na secção 1.3 foram apresentados alguns formatos de representação de dados inteiros no MATLAB (ex. `int8`, `uint8`, `int16`, etc.), e a forma de conversão direta entre formatos. Também foi referido que o modo 'normal' usado pelo MATLAB é `double`, que é um formato de virgula-flutuante (*floating-point*) de 8 bytes (8x8=64 bits). No caso de processamentem digital de imagem, as imagens de tons de cinzento com variáveis do tipo `double` tem valores limitados ao intervalo [0,1], onde 0 corresponde ao preto, 1 ao branco, e valores entre 0 e 1 a tons de cinzento de diferentes intensidades.

Função	Descrição
`im2uint8`	conversão de imagem [0,1] para uint8
`im2uint16`	conversão de imagem [0,1] para uint16
`im2double`	conversão de imagem para double
`mat2gray`	conversão para double [0,1]
`im2bw`	conversão de imagem para binário

Há várias formas de conversão de formatos de dados utilizados para imagens, estando as mais importantes listadas na tabela acima, em que o formato imagem pode ser um dos aceites pela IPT. Em seguida alguns exemplos que ilustram o uso destas funções e da conversão direta entre formatos.

```
>> a = [-500 -0.5 0 0.5 50.1 500]
a = -500.0000 -0.5000 0 0.5000 50.1000 500.0000
>> aa = uint8(a)
aa = 0 0 0 1 50 255
>> ab = uint16(a)
ab = 0 0 0 1 50 500
>> ac = im2uint8(a)
ac = 0 0 0 128 255 255
>> ad = im2uint16(a)
ad = 0 0 0 32768 65535 65535
```

conversão direta para uint8

conversão direta para uint16

A conversão direta para formatos de inteiros arredonda ou trunca os valores originais que não pertencem ao novo dominio. Por exemplo, no 1º caso (`uint8`), os valores negativos passaram a 0, os valores 0.5 e 50.1 foram arredondados para o inteiro mais próximo, e os valores maiores do que o valor máximo possível (255) foram truncados para esse valor. No 2º caso (`uint16`) a única diferença é que como 500 é inferior ao valor máximo possível (65535), o valor original ficou inalterado. Usando as funções de conversão do MATLAB IPT, os resultados são muito diferentes. Uma vez que a variável de input é do tipo

`double`, é considerado que a gama de valores corresponde ao intervalo [0,1]. A função aplica uma transformação linear entre o intervalo [0,1] e o intervalo pretendido ([0,255] para `im2uint8` ou [0,65535] para `im2uint16`), arredondando os valores ao inteiro mais próximo. Os valores fora do intervalo [0,1] são truncados para 0 ou para o valor máximo do novo intervalo.

O funcionamento da função `mat2gray` é algo diferente. Converte os dados de input para a escala [0,1] (como `double`), usando uma função linear ajustada ao minimo e máximo dos dados (que passam a 0 e 1). Para o exemplo considerado (vetor `a`), o resultado é apresentado em baixo, sendo neste caso o valor mínimo -500 (que é convertido para 0), e o valor máximo 500 (que passa a ser 1).

```
>> ae = mat2gray(a)
ae = 0 0.4995 0.5000 0.5005 0.5501 1.0000
```

A função `im2double` não tem qualquer efeito no vetor `a`, sendo útil para converter de formatos de inteiros para double. A escala da dados de input é convertida para [0,1]. Por exemplo,

```
>> v8=uint8([0 10 51 153 255])
v8 = 0 10 20 51 255
>> vD=im2double(v8)
vd = 0 0.0392 0.2000 0.6000 1.0000
```

A função `im2bw` permite converter formatos de dados imagem para binário. O critério de decisão usa um limiar (ou valor de corte) numa escala [0,1], que por omissão (*default*) é 0.5, mas pode ser alterado. Os valores menores ou iguais a este limiar passam a 0 e os valores maiores passam a 1. Por exemplo,

```
>> vb1=im2bw(vD)
vb1 = 0 0 0 1 1
>> vb2=im2bw(vD,0.1)
vb2 = 0 0 1 1 1
>> vb3=im2bw(v8,0.1)
vb3 = 0 0 1 1 1
```

limiar de corte 0.5 (default)

limiar de corte 0.1

Convém notar que no último caso (`vb3`), apesar da variável de input (v8) ser do tipo `uint8`, o limiar é um valor no intervalo [0,1], e não um inteiro entre 0 e 255. A variável de output da função `im2bw` é do tipo `logical`, ou seja 1bit (1/0, ON/OFF, SIM/NÃO), que no caso de processamento de imagem é normalmente usado para dados auxiliares (bitmaps).

6.2 Imagens - observação e leitura / gravação

Há uma grande diversidade de formatos para colocação de uma imagem num ficheiro, tendo o MATLAB ferramentas para leitura e gravação dos formatos mais correntes (.jpg, .tif, .bmp, etc.). A importação (ou leitura) de dados de um ficheiro imagem pode ser feita através da função `imread` e a gravação usando `imwrite`. Por exemplo, a seguinte instrução importa a imagem contida no ficheiro `SapoC.bmp`, colocando-a na variável `IM`.

```
>> IM=imread('SapoC.bmp');
```

ficheiro imagem entre plicas simples '

suprimir output

Figura 8: Janela gráfica aberta pela função `imshow`.

Após esta instrução a imagem está contida na variável `IM`, o que pode ser verificado na janela *Workspace* ou através da função `whos`,

```
>> whos IM
Name      Size          Bytes      Class     Attributes
IM      371x416        154336      uint8
```

A variável `IM` é uma matriz com 371 linhas e 416 colunas, do tipo `uint8`, ou seja inteiros de 8-bits sem sinal (valores entre 0 a 255). Isto corresponde a uma imagem de tons de cinzento com 371 x 416 pixels, com 1 byte (8 bits) por pixel, o que resulta num total de 154336 bytes. A imagem pode ser observada através da função `imshow`, que apresenta a imagem na janela gráfica activa (existente ou nova), como ilustrado na figura 8. A sintaxe é a seguinte:

```
>> imshow(IM)
>> imshow(IM,'InitialMagnification',25)
```

abre a janela da figura 8

alternativa, para escolher o tamanho inicial

A janela gráfica associada à função `imshow` tem algumas funcionalidades disponíveis (de edição, zoom, gravação, etc.) embora algo limitadas. Uma alternativa interessante em certos casos é a função `imtool`, que tem melhores ferramentas de edição e exploração interactiva de uma imagem.

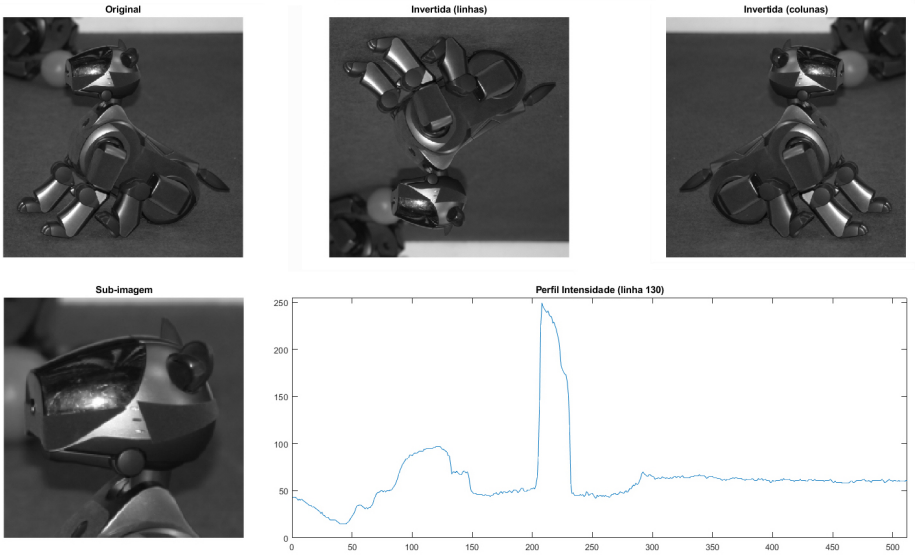


Figura 9: Exemplo de operações básicas numa imagens de tons de cinzento.

Um aspecto que é importante destacar é o facto da sequência de cordenadas associadas a um pixel não ser apresentada de forma consistente pelo MATLAB. Internamente (na matriz) o primeiro elemento é a linha e o segundo a coluna. No entanto, na representação gráfica (via `imshow` ou `imtool`), a sequencia apresentada é coluna,linha. Por exemplo, a instução `IM(360,10)` devolve o valor 171, correspondente à intensidade do pixel na linha 360 e coluna 10 (próximo do canto inferior esquerdo da imagem). Utilizando a ferramenta *Data Cursor*, deslocando o cursor e identificando o mesmo pixel através da janela gráfica (figura 8), aparece a indicação `[X,Y]: [10 360] Index:  171`.

A modificação e extração de informação de uma imagem pode ser feita diretamente na matriz que contém a imagem (valores numéricos), usando as funções e operadores elementares do MATLAB. Por exemplo, a inversão (ou reflexão) da imagem ao longo das linhas ou colunas pode ser feita da seguinte forma:

```
>> IM=imread('RobotC.tif');
>> IMinvL=IM(end:-1:1,:);
>> IMinvC=IM(:,end:-1:1);
```

Importação da imagem

da ultima linha até à 1ª, com incrementos de -1

todas as colunas

O resultado é apresentado na figura 9, na linha de cima (da esquerda para a direita): imagem original, inversão em linhas, inversão em colunas. Uma outra operação ilustrada na figura 9 é a extração de uma sub-imagem (em baixo à esquerda). A imagem original tem 512x512 pixels, e a sub-imagem apresentada tem 201x201 pixels (linhas 30 a 230 e colunas 115 a 315), tendo sido utilizado seguinte código para preparar esta 2 componentes da figura:

```
>> IMsub=IM(30:230,115:315);
>> subplot(2,3,4), imshow(IMsub), title('Sub-imagem')
```

Pedaço da imagem

Figura 10: Imagem RGB original (esquerda), e gravada em formato jpg com níveis de compressão crescentes (da esquerda para a direita). 25, 15, 5, 0.

O gráfico apresentado na figura 9 corresponde aos valores de intensidade (nível de cinzento) ao longo de uma linha. Foi criado com o seguinte código:

```
>> subplot(2,3,[5,6]), plot(IM(130,:))
>> title('Perfil Intensidade (linha 130)')
>> axis([0 512 0 255])
```

Linha 130, todas as colunas

Intervalos para os eixos x e y

Cada uma das novas versões da imagem original pode ser gravada num ficheiro em formato 'standard' usando a função `imwrite`. Por exemplo, para gravar a sub-imagem com a cabeça do robot (`IMsub`), para formato tif:

```
>> imwrite(IMsub,'CabecaRobot.tif');
```

Nome do ficheiro

A função `imwrite` permite uma utilização simplificada, como do exemplo anterior, bastando a colocação da extensão no nome do ficheiro (neste caso .tif) para se definir o formato pretendido. No entanto, em certos casos poderá ser necessário colocar parâmetros adicionais. Por exemplo, o formato jpg usa compressão com perdas, pelo que poderá ser importante definitir o valor do parâmetro que controla o nível de compressão. Quanto maior o factor de compressão maior as perdas, embora a relação não seja linear e dependa muito das características da imagem. Como exemplo, a figura 10 mostra uma imagem original (esquerda) e gravada em formato jpg com níveis de compressão crescentes. O parâmetro usado pelo MATLAB para controlar o nível de compressão é `'Quality'`, tomando valores entre variando entre 0 (maior compressão / pior qualidade) e 100 (sem perdas). A sintaxe pode ser vista na linha de código em baixo, usado para criar uma das imagens apresentadas na figura 10.

```
>> imwrite(IMmacaco,'MacacoC1.jpg', 'Quality',25);
```

Nome do ficheiro

Neste exemplo a imagem original tem 397x285 pixels, com 3 bandas (RGB), o que corresponde a 342 Kbytes (Kb). Apresentam-se em seguida os valores usados para o parâmetro `'Quality'` e o tamanho dos ficheiros resultantes, começando na 2a imagem (da esquerda para a direita): 25 (7 Kb), 15 (6 Kb), 5 (4 Kb) e 0 (3 Kb). Visualmente a imagem original e a versão comprimida com `'Quality'` a 25 são quase indistinguíveis. No entanto esta última ocupa cerca de 50 vezes menos volume em disco. Ao contrário, os níveis de compressão mais extremos (`'Quality'` a 5 e 0) alteram de forma considerável a imagem, não reduzindo muito mais o volume de dados.

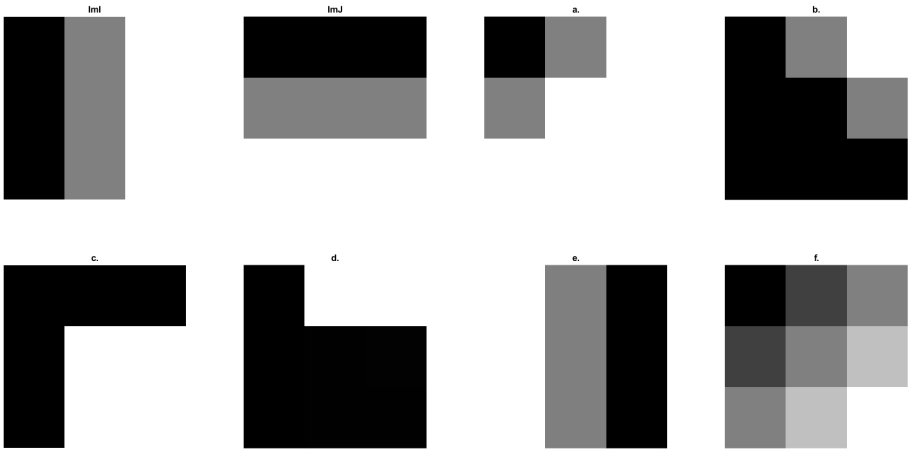


Figura 11: Imagens sintética `ImI` (canto superior esquerdo) e variações: (pela ordem de leitura) ImJ, a, b, c, d, e, f (explicação no texto).

6.3 Criação de imagens sintéticas

Para além da importação de imagens a partir de ficheiros, é também possível criar imagens sintéticas no MATLAB, recorrendo apenas às operações básicas introduzidas nas secções anteriores. No exemplo que se apresenta em seguida, é inicialmente criada uma imagem de 300x300 pixels, a 8 bits (0-255), com 3 faixas verticais de intensidades 0 (preto), 128 (cinzento de intensidade média), e 255 (branco). Esta imagem (`ImI`), incluida na figura 11 (canto superior esquerdo) foi criada através do seguinte código

```
>> f1=zeros(300,100);
>> f2=128*ones(300,100);
>> f3=255*ones(300,100);
>> fh=[f1, f2, f3];
>> ImI=uint8(fh);
```

Faixa preta

Faixa cinzenta

Faixa branca

conversão para uint8

Uma outra imagem, obtida a partir de `ImI`, é a imagem transposta `ImJ`, que contém 3 faixas horizontais de 100x300 pixels cada (2a imagem da figura 11, pela ordem de leitura). Estas duas imagens foram usadas para calcular a soma, subtração, multiplicação e divisão, através das funções `imadd`, `imsubtract`, `immultiply` e `imdivide`, de acordo com os códigos listados abaixo.

```
>> ImJ=ImI';
>> a=imadd(ImI,ImJ);
>> b=imubtract(ImI,ImJ);
>> c=immultiply(ImI,ImJ);
>> d=imdivide(ImI,ImJ);
```

O resultado teria sido o mesmo recorrendo aos operadores `+`, `-`, `.*`, e `./`. No entanto, se as imagens tivessem sido defindas como double (numa escala de valores [0,1]) os resultados da multiplicação e divisão seriam diferentes. As últimas duas imagens apresentadas na figura 11 são o inversão de intensidades

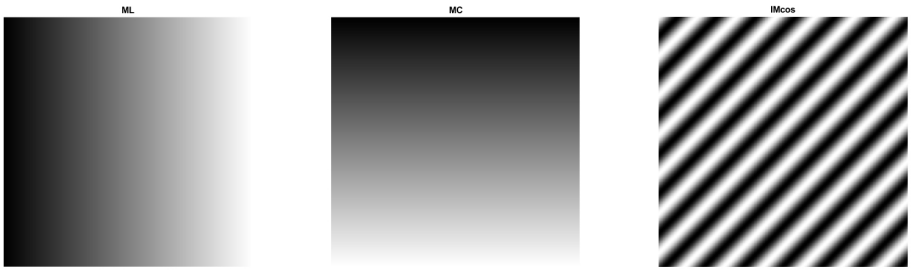


Figura 12: Imagens sintéticas com o nº da coluna (esquerda), linha (centro) e função de linha,coluna (direita).

de `ImI` (`e`) e a combinação linear de `ImI` e `ImJ` (`f`), neste caso correspondendo à média (factores 0.5 e 0.5), de acordo com os códigos seguintes.

```
>> e=imcomplement(ImI);
>> f=imlincomb(0.5,ImI,0.5,ImJ);
```

inversão de intensidades

combinação linear

O outro exemplo de criação de imagens sintéticas apresentado faz uso da posição de cada pixel (linha,coluna) para calcular a intensidade (nível de cinzento). Numa 1a fase são criadas 2 matrizes, ou imagens auxiliares (`ML` e `MC`), uma com o nº da linha e outra com o nº da coluna para cada pixel (figura 12, esquerda e centro). O código abaixo foi usado para criar estas imagem, de 500x500 pixels (parametro N), podendo em alternativa ser usada a função `meshgrid`.

```
>> N=500;
>> vL=1:N;
>> vC=ones(N,1);
>> ML=vC*vL;
```

vetor linha

vetor coluna

multiplicação de matrizes

Convém realçar que, uma vez que `ML` e `MC` são matrizes do tipo double, e tomam valores entre 1 e 500, a aplicação direta da função `imshow` não dá o resultado da figura 12 (mas sim tudo a branco). Para a representação da figura, foi usada a função `mat2gray` para converter a gama de valores da imagem ([1,500]) para a escala [0,1], através de `imshow(mat2gray(ML))`.
A 2a fase do processo é a aplicação de uma função, neste caso usando o coseno, com os valores de posição de cada pixel, obtidos de `ML` e `MC`. O código usado é listado abaixo e o resultado apresentado na figura 11 (direita).

```
>> IMcos=0.5+0.5*cos(ML/(4*pi)+MC/(4*pi));
```

Uma última nota é relativa à possibilidade de se criar uma imagem sintética como `IMcos` recorrendo a ciclos `for` (um para linhas e outro para colunas). Embora esta abordagem seja fácil de codificar, à semelhança de outras linguagens de programção, não é recomendável em MATLAB por questões de eficiencia. O tempo de execução de uma sequência de instruções pode ser controlado através das funções `tic` (inicio do cronómetro) e `toc` (fim, registo de tempo). Para o exemplo apresentado (N=500) o tempo de execução é desprezável, em qualquer das abordagens. No entanto por exemplo para N=5000, a abordagem via ciclos for mais lenta por um factor de 4.1 (1.6784 s em vez de 0.4066 s).

7 Exercicios propostos

Nesta secção são propostos alguns exercícios que podem ser usados para consolidar os conceitos apresentados. Alguns exercícios são de carácter geral (G), e outros mais orientados para Processamento de Sinal (PS) ou Processamento de Imagem (PI).

I - Simulação de lançamento de dados (G)

Pretende-se simular o lançamento de um par de dados, repetido N vezes. Numa 1ª fase, poderá escrever um script para simular o lançamento dos dados e calcular os seguintes itens:

1. O valor médio para o par de dados.
2. Número de vezes que ocorre cada valor (2 a 12)
3. A percentagem de 'doubles' (valor dos dois dados igual).
4. Número de vezes que sai um par de 6 duas vezes seguidas.
5. Maior número de lançamentos sem se obter 12 (2x6).

Numa 2ª fase, o código poderá ser transformado numa função para calcular estes parâmetros e devolve-los como output. Depois poderá ser escrito um script que repita o teste várias vezes, e apresente os resultados de forma compacta, por exemplo com gráficos. Outra vertente poderá ser a avaliação dos resultados simulados por comparação com a distribuição esperada (teórica).

II - Imagem simulada de tabuleiro de xadrez (PI)

Pretende-se ter uma função para criar uma imagem de um tabuleiro de xadrez, com iluminação gradual. Poderá ser conveniente fazer o exercício em 3 fases, de dificuldade crescente: primeiro um script, depois uma versão simplificada da função, e finalmente a versão final da função.

1. Script para criar imagens de 512×512 pixels, uma para o tabuleiro de xadrez simples (quadrados pretos a 0 e brancos a 255), e outra com os quadrados brancos a 20% da intensidade máxima do lado direito e 90% do lado esquerdo, e que os quadrados pretos mantém o valor 0.
2. Função que recebe os seguintes inputs: N (tamanho imagem $N \times N$ pixels), Imin e Imax (factores de iluminação minimo e máximo).
3. A versão final deverá receber 2 parametros de *input* adicionais: ILAng e graf. ILAng é o ângulo de iluminação (fonte de luz) em relação estrutura da imagem (ex. horizontal). O parâmetro graf é usado para controla a apresentação de imagens (ex. graf=1 para mostrar figuras). A função deverá devolver como *output* a imagem do tabuleiro de xadrez criada.

7.1 APPENDIX - Exercices (in English)

This Appendix proposes some exercises that can be used to assist in the introduction to MATLAB. Some exercises are of a general nature (G), and others are more oriented to Signal Processing (SP) or Image Processing (IP).

I - Dice throw simulation (G)

The goal is to simulate the throw of a pair of dice, repeated N times. In a 1st stage, you can write a script to simulate the dice throw and calculate the following items:

1. The average value for the pair of dice.
2. Number of times each value occurs (2 to 12)
3. The percentage of 'doubles' (equal value of the two dice).
4. Number of times a pair of 6 comes out twice in a row.
5. Highest number of releases without obtaining 12 (2x6).

In a 2nd stage, the code can be transformed into a function to calculate these parameters and return them as an output. Then a script can be written that repeats the test several times, and presents the results in a compact form, for example using graphics. Another aspect may be the evaluation of simulated results by comparison with the expected (theoretical) distribution.

II - Chess board simulated image (IP)

The goal is to have a function to create an image of a chess board, with gradual illumination. It may be convenient to do the exercise in 3 stages, of increasing difficulty: first a script, then a simplified version of the function, and finally the final version of the function.

1. Script to create two 512×512 pixel images, one for a simple chessboard (black squares at 0 and white squares at 255), and one with white squares at 20 % of the maximum intensity on the right side and 90 % on the left side, with the black squares keeping the value 0.
2. Function that receives the following inputs: N (image size $N \times N$ pixels), Imin and Imax (minimum and maximum lighting factors).
3. The final version should receive 2 additional input parameters: ILAng and graf. ILAng is the illumination angle (light source) in relation to the image structure (e.g. horizontal). The parameter graf is used to control the display of images (e.g. graf = 1 to show figures). The function should return as output the image chess board created.