Matemática Computacional

com o MATLAB

Universidade do Minho
Escola de Ciências

Departamento de Matemática

maria irene falcão

2020/21

Conteúdo

Prefácio

De entre os diversos "ambientes de computação" atualmente disponíveis, um dos mais frequentemente utilizados, para fins educacionais, é o MATLAB®. Este sistema apresenta características que justificam a sua ampla divulgação e utilização em diversos cursos de ciências e engenharia:

- é extremamente fácil de utilizar, sendo os dados introduzidos e manipulados de uma forma simples, especialmente no caso de vetores ou matrizes;
- inclui uma linguagem de programação de alto nível e bastante acessível, especialmente adequada a jovens que vão tomar um primeiro contacto com uma linguagem de programação;
- tem suporte em muitos sistemas computacionais diferentes, proporcionando, em grande medida, uma independência de plataforma. Programas escritos em MATLAB podem migrar para novas plataformas quando as necessidades do utilizador se alteram.
- tem um grande número de funções matemáticas predefinidas que podem ser usados no âmbito de outras disciplinas da licenciatura em matemática, praticamente desde o início do curso;
- dispõe de boas capacidades gráficas, permitindo uma visualização, de forma simples, dos dados e resultados;
- é possível adquirir uma variedade de pacotes de programas (*toolboxes*) para fins mais específicos.

Em resumo, com o MATLAB os alunos podem realizar num ambiente agradável, tarefas de natureza diversa: desenvolvimento e análise de algoritmos, modelação, computação científica, visualização, etc.

Estes textos têm como principal objetivo servir de apoio às aulas da unidade curricular (UC) Matemática Computacional I do 1º ano da Licenciatura em Matemática da Universidade do Minho. Resultam de uma revisão dos textos de 2009 disponíveis em http://hdl.handle.net/1822/17080.

Depois de uma iniciação ao sistema MATLAB, é abordado sucintamente o tema algoritmos e feita uma introdução à programação em MATLAB. Os tópicos selecionados têm em consideração o programa da UC e os interesses do público-alvo.

Em cada capítulo são apresentados exercícios para serem resolvidos nas aulas teórico-práticas e laboratoriais e exercícios suplementares que, normalmente, foram selecionados de provas de avaliação. Para os exercícios assinalados com *, é apresentada uma sugestão de resolução.

Iniciação ao MATLAB

1. Introdução

⇛ O sistema MATLAB

O MATLAB é um sistema gráfico que integra a capacidade de realização de cálculos, programação e visualização gráfica num ambiente interativo bastante agradável. Este *software* é produzido pela companhia americana

MathWorks®

e o seu nome deriva do inglês Matrix Laboratory". Este sistema é particularmente popular em

1. Matemática e Computação
2. Desenvolvimento de algoritmos
3. Modelação simulação e prototipagem
4. Análise e visualização de dados
5. Desenvolvimento de aplicações

O MATLAB trabalha com matrizes quadradas ou retangulares, com elementos reais ou complexos, derivou dos projetos LINPACK e EISPACK que são considerados como a origem de algum do melhor *software* numérico disponível para computação com matrizes.
Para além da manipulação fácil de matrizes, o MATLAB permite o acesso a um número crescente de rotinas incorporadas, potencialidades gráficas a duas e três dimensões e a possibilidade de configurar o *software* às necessidades de cada utilizador.
O MATLAB possui ainda um conjunto de funções específicas - *toolboxes* - para uma dada aplicação e/ou tecnologia, como por exemplo, Estatística e Machine Learning, Otimização, Processamento de Sinal e de Imagem, Computação Paralela, etc.

O sistema MATLAB possui cinco partes principais:

1. As ferramentas para utilização e desenvolvimento;
2. A biblioteca de funções matemáticas;
3. A linguagem de programação (de alto nível);
4. As funções gráficas;
5. As bibliotecas de interfaces.

As versões atuais do MATLAB permitem realizar muitas tarefas recorrendo às suas *interfaces* gráficas - menus, botões, etc. Na maior parte dos casos o recurso a estas potencialidades pode ser feito de uma forma natural e intuitiva, pelo que os presentes textos darão mais ênfase aos comandos *escritos*.

⇛ Iniciar uma sessão em MATLAB

Quando se invoca o MATLAB[1], é criada uma janela com várias "subjanelas" e menus. Este ambiente de trabalho pode ser personalizado, de acordo com as preferências do utilizador.
Do lado direito do ecrã aparece, por defeito, a janela de comandos - *Command Window*, através da qual é possível comunicar com o interpretador do MATLAB. Quando aparece o símbolo >>, o MATLAB está pronto a receber instruções. A janela de comandos pode ser apagada através do comando **clc** ou usando a opção **Command Window** do botão **Clear Commands** da barra de ferramentas do menu **Home**.

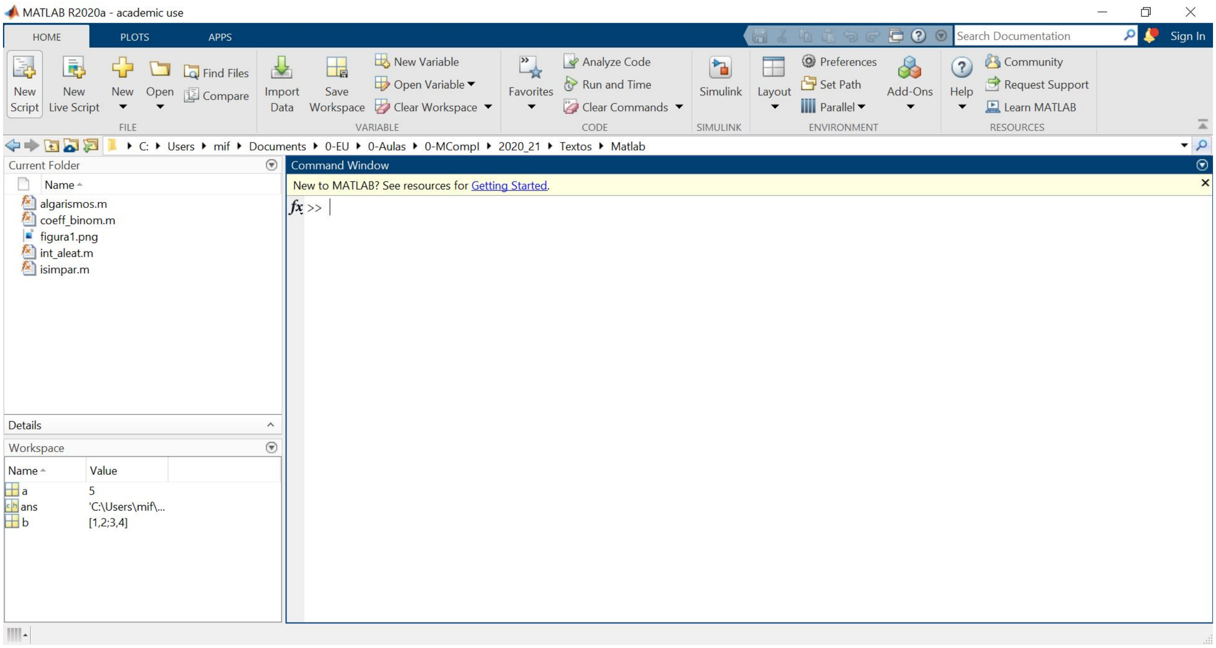


Figura 1.1: O ambiente de trabalho do MATLAB

No lado superior esquerdo pode ver-se a janela *Current Directory* onde pode ser obtida informação relativa aos ficheiros da pasta de trabalho do MATLAB. Em geral, o MATLAB só reconhece ficheiros que estejam nessa pasta ou no caminho de pesquisa do sistema -*path*. Os menus e botões desta janela podem ser usados para examinar os ficheiros, ou alternativamente, podem ser usados comandos equivalentes na janela de comandos: **pwd** - retorna o nome da pasta corrente; **cd** - altera a pasta de trabalho; **dir** - lista todos os ficheiros da pasta corrente; **what** - lista apenas os ficheiros MATLAB. Os comandos **delete** e **type** permitem apagar um ficheiro e mostrar na janela de comandos um ficheiro, respetivamente.

```
>> pwd

ans =
'C:\Users\mif\Documents\0-EU\0-Aulas\0-MCompI\2020_21\Textos\MATLAB'

>> what
```

[1] A versão utilizada nestes textos é a versão R2020b.

```
MATLAB Code files in the current folder
C:\Users\mif\Documents\0-EU\0-Aulas\0-MCompI\2020_21\Textos\MATLAB

algarismos    coeff_binom  int_aleat    isimpar
```

Todas as variáveis usadas na sessão de trabalho são gravadas no espaço de trabalho - *MATLAB Workspace*.
O conteúdo de *Workspace* pode também ser visto através do comando **whos** ou **who**.

```
>> whos
Name        Size              Bytes  Class     Attributes
a           1x1                   8  double
ans         1x66                132  char
b           2x2                  32  double

>> who
Your variables are:
a     ans   b
```

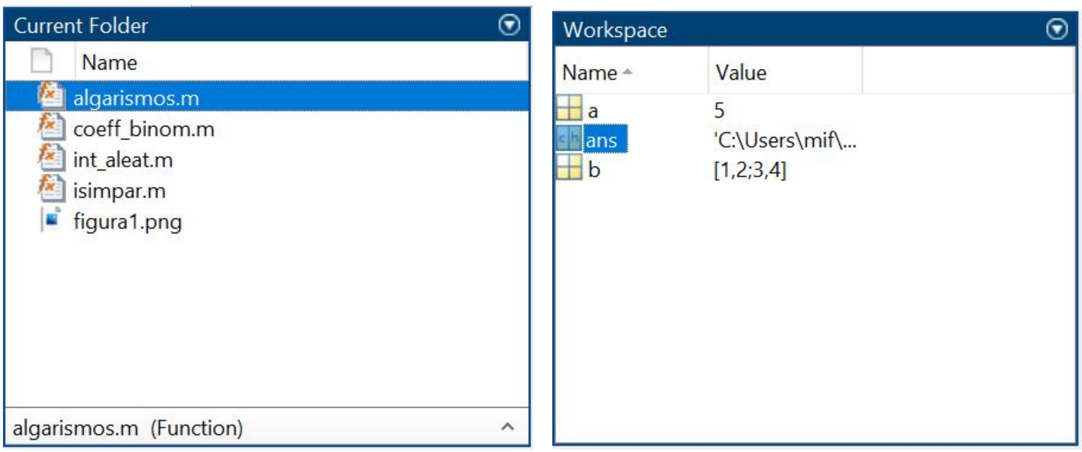


Figura 1.2: As janelas *Current Directory* e *Workspace*

O texto de cada sessão de trabalho pode ser guardado através do comando **diary**. Por exemplo, todo o texto relativo à sessão que se inicia imediatamente a seguir à instrução

```
>> diary sessao
```

e termina com o comando

```
>> diary off
```

é guardado num ficheiro de texto chamado *sessao*, o qual pode ser examinado com qualquer editor de texto.

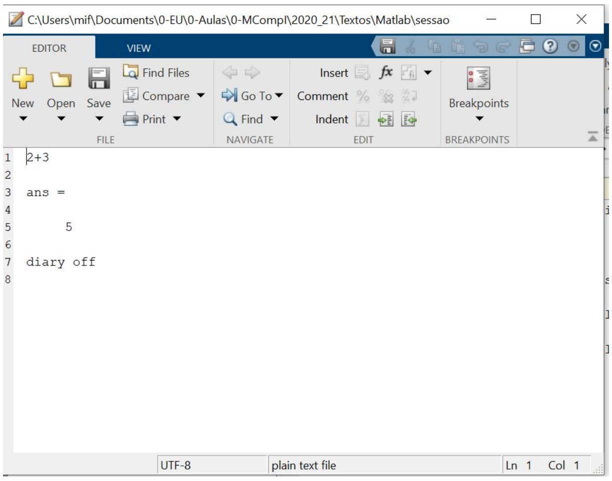


Figura 1.3: O ficheiro que contém uma sessão de trabalho

As variáveis definidas ou obtidas numa sessão de trabalho podem também ser guardadas num ficheiro para serem posteriormente usadas, recorrendo aos comandos **save** e **load**. Assim, o comando

```
>> save variaveis
```

guarda todas as variáveis do espaço de trabalho num ficheiro chamado *variaveis.mat*. Para recuperar estes valores basta fazer

```
>> load variaveis
```

é ainda possível guardar apenas algumas variáveis. Por exemplo, o comando

```
>> save apenas X Y
```

guarda as variáveis X e Y num ficheiro chamado *apenas.mat*.
Para apagar a variável X, pode usar-se o comando **clear** X. Todas as variáveis do espaço de trabalho serão apagadas, se for usado apenas **clear**. Alternativamente pode usar-se o botão **Clear Workspace** da barra de ferramentas do menu **Home** ou selecionar diretamente na janela *Workspace* as variáveis a apagar e usando os botões do rato.

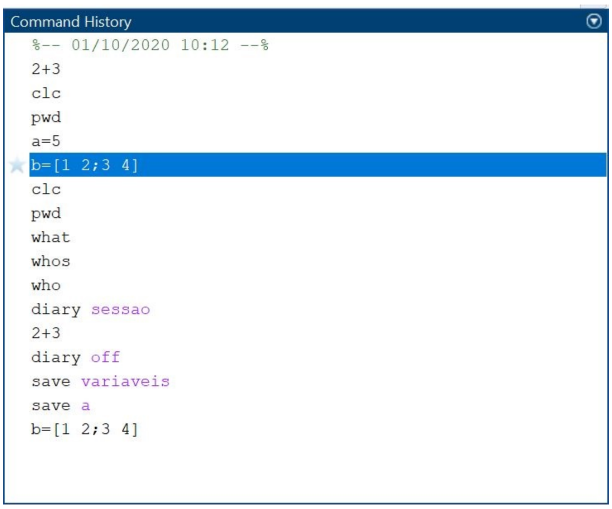


Figura 1.4. Command History do MATLAB

Finalmente, a opção **Layout** da barra de ferramentas do menu **Home** permite alterar a disposição das janelas e incluir ou retirar janelas. Em particular, na janela *Command History*, pode ver-se um historial de todas as sessões de trabalho realizadas, desde que se apagou pela última vez esta informação (opção **Command History** do botão **Clear Commands**). Verifique o que acontece se fizer um duplo clique numa das linhas do *Command History*.

⇛ declarações e variáveis

As declarações no MATLAB são frequentemente da forma

variavel=expressão

ou simplesmente

expressão.

No primeiro caso, o valor de *expressão* é atribuído à variável *variavel* para uso futuro. Quando o nome da variável é omitido (assim como o sinal =), o MATLAB cria automaticamente uma variável com o nome *ans* (de *answer*). Por exemplo, as declarações

```
>> a=2*3
```

e

```
>> 2*4
```

originam, respetivamente

```
a =
     6
ans =
     8
```

O nome de uma variável deve sempre começar por uma letra, seguido de um conjunto de letras ou números (ou ainda o sinal _), até ao máximo de 63 carateres. O MATLAB **distingue as letras minúsculas das maiúsculas**. Assim, a e A não representam a mesma variável. **Todas as funções do MATLAB têm nomes escritos em minúsculas** (ver Secção 1.3).

Sempre que se pretenda que o resultado de uma operação ou atribuição não apareça no ecrã (evitando assim o aparecimento de uma quantidade de informação pouco útil), deve usar-se o símbolo ;. A declaração

```
>> a=2*3;
```

atribui o valor 6 à variável a, mas não mostra no ecrã o resultado dessa atribuição.

Quando a expressão é muito complicada, necessitando de mais de uma linha para ser definida, deve usar-se o símbolo ... antes de mudar de linha, para indicar que a expressão continua.

```
>> 2*cos(3)+ 3*cos(2)-5*cos(1)+ 8*cos(5)-0.5*i*cos(4)+...
2*sin(3)+ 3*i*sin(2)-5*sin(1);
```

Em contrapartida, se as expressões forem muito simples, podem ser definidas simultaneamente na mesma linha de comandos, separando cada expressão através do símbolo ,.

```
>> a=2*3,b=2*4
a =
     6
b =
     8
```

⇛ Números e expressões aritméticas

O MATLAB usa a notação decimal convencional para representar números, sendo também possível incluir-se, na representação de um número, potências de base 10. Assim, as expressões 0.001 e 1e-3 representam o mesmo número. A notação usada para a unidade imaginária é o i ou j. As expressões 1+2i e 1+2j representam o número complexo cuja parte real é 1 e a parte imaginária é 2.
As expressões podem ser construídas usando os operadores aritméticos usuais e correspondentes precedências. Assim, tem-se

+ adição
- subtração
* multiplicação
/ divisão à direita[2]
\ divisão à esquerda
^ potenciação

O MATLAB tem incorporadas funções matemáticas elementares tais como **abs**, **sqrt**, **log**, etc. Para uma lista completa destas e outras funções, veja a Secção 1.3.
Por defeito, o MATLAB trabalha no sistema de numeração de norma IEEE em formato duplo, isto é, no sistema $F(2, 53, -1021, 1024)$[3]. Para controlar o formato de saída dos resultados pode usar-se o comando **format**. Por defeito, o MATLAB apresenta os resultados no **format short** que corresponde ao uso de 5 dígitos. é possível alterar este formato, usando formatos que permitem mais dígitos ou usam a notação científica, como se exemplifica na tabela abaixo.

Formato	Descrição
format	o mesmo que short
format short	notação de vírgula fixa, com 5 dígitos
format long	notação de vírgula fixa, com 15 dígitos
format short e	notação de vírgula flutuante, com 5 dígitos
format long e	notação de vírgula flutuante, com 15 dígitos
format short g	o melhor dos formatos de vírgula fixa ou flutuante, com 5 dígitos
format long g	o melhor dos formatos de vírgula fixa ou flutuante, com 15 dígitos

Por exemplo, a atribuição

```
>> x=sqrt(2), y=pi*1000
```

[2]As operações com matrizes tornam conveniente o uso de dois símbolos para a divisão; ver Secção 1.2.
[3]Relembre a matéria já lecionada sobre este tema

tem como resultado

```
x =
   1.4142
y =
   3.1416e+03
```

que corresponde ao formato por defeito, isto é, **short**:

```
>> format short,x,y
x =
   1.4142
y =
   3.1416e+03
```

Os exemplos seguintes ilustram outros formatos existentes.

Exemplo

```
>> format short e,x,y
x =
   1.4142e+00
y =
   3.1416e+03

>> format short g,x,y
x =
   1.4142
y =
   3141.6

>> format long,x,y
x =
   1.414213562373095
y =
   3.141592653589793e+03

>> format long e,x,y
x =
   1.414213562373095e+00
y =
   3.141592653589793e+03

>> format long g,x,y
x =
   1.4142135623731
y =
   3141.59265358979
```

⇛ A ajuda do MATLAB

O MATLAB disponibiliza várias formas de obter ajuda sobre uma função. A primeira delas é através do uso da função **help** cuja sintaxe é:

help nome

ou apenas

help

No primeiro caso é exibido um texto de ajuda para a funcionalidade especificada por nome, enquanto no segundo caso é mostrado conteúdo relevante para ações anteriores.
Suponhamos que pretendemos ajuda sobre a função **imag**. Para isso basta fazer

```
>> help imag
```

obtendo-se

```
imag   Complex imaginary part.
imag(X) is the imaginary part of X.
See I or J to enter complex numbers.

See also real, isreal, conj, angle, abs.
       ...
```

Se tiver sido feita a atribuição `a=imag(pi+2i)`, o uso de

```
>> help
```

resulta em

```
   i - Imaginary unit.
imag - Complex imaginary part.
pi - 3.1415926535897....
```

Uma outra forma de obter ajuda é usando a função **doc** para aceder a uma determinada página do navegador de ajuda. Ao executar a instrução:

```
>> doc imag
```

deverá visualizar a página referente à função **imag** (ver Figura 1.5)

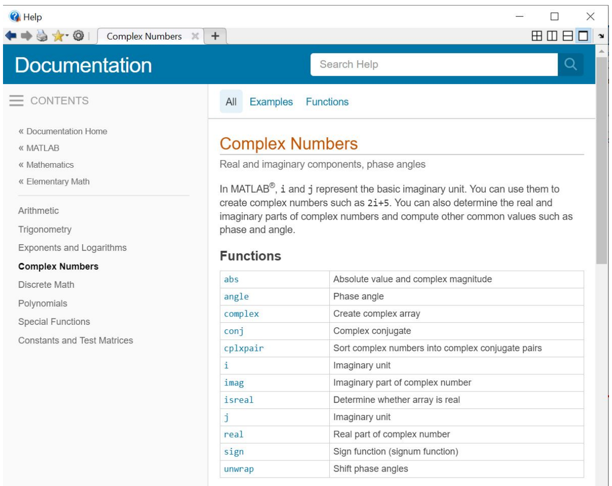


Figura 1.5. doc imag

Outra facilidade do MATLAB pode ser obtida recorrendo ao uso de **lookfor** *nome* que permite encontrar todas as funções que contêm a palavra *nome* na primeira linha de comentário do texto do *help*. Por exemplo,

```
>> lookfor imaginary
```

tem como resultado

```
imag                         - Complex imaginary part.
i                            - Imaginary unit.
j                            - Imaginary unit.
```

⇛ definição e manipulação de matrizes

O MATLAB trabalha essencialmente com um tipo de objeto, uma matriz numérica retangular com elementos eventualmente complexos. Para o MATLAB, uma matriz é um *array*[4] bidimensional; escalares são entendidos como matrizes 1×1; vetores são matrizes $1 \times n$ ou $n \times 1$. Assim, as operações e comandos em MATLAB devem ser entendidos duma forma natural, enquanto operações entre matrizes.
A maneira mais simples de definir uma matriz pequena é introduzir explicitamente os seus elementos da seguinte forma: cada elemento de uma linha da matriz é separado pelo símbolo , (ou espaço) e cada linha da matriz é separada por ; (ou mudança de linha). Os elementos da matriz devem estar rodeados pelos símbolos [e] .

Por exemplo as atribuições

```
>> A=[1,1,1;2,2,2;3,3,3];
>> B=[ 1 1 1
       2 2 2
       3 3 3];
>> C=[1 1 1;2 2 2;3 3 3];
```

produzem exatamente a mesma matriz:

```
     1     1     1
     2     2     2
     3     3     3
```

Em alguns casos, não é necessário definir explicitamente todos os elementos de uma matriz. O símbolo : permite definir vetores e/ou matrizes "especiais" de modo muito simples. Por exemplo,

```
>> x=-2:2
x =
    -2    -1     0     1     2
```

[4]um *array* é um conjunto de dados que podem ser acedidos por indexação. No MATLAB, o conjunto de índices é sempre uma sequência de inteiros que começa em 1

```
>> y=-2:2:6
y =
    -2     0     2     4     6

>> z=[0:3:9;2:2:8;5:5:21]
z =
     0     3     6     9
     2     4     6     8
     5    10    15    20
```

Para obter um determinado elemento de uma matriz, basta indicar a sua linha e coluna. Por exemplo, a instrução

```
>> z(3,2)
```

devolve o elemento da matriz z que está na linha 3 e coluna 2, isto é,

```
ans =
   10
```

Uma das vantagens do MATLAB é a de não ser necessário dimensionar as matrizes *a priori*. A cada nova instrução é feito o redimensionamente da matriz[5]. Assim, considerando a matriz A inicial, as declarações

```
>> A(5,5)=1,B(5,1)=1
```

produzem as novas matrizes

```
A =
     1     1     1     0     0
     2     2     2     0     0
     3     3     3     0     0
     0     0     0     0     0
     0     0     0     0     1

B =
     1     1     1
     2     2     2
     3     3     3
     0     0     0
     1     0     0
```

Outra forma simples de obter matrizes maiores à custa de matrizes menores já definidas é a seguinte: se pretendermos acrescentar à matriz A inicial a linha 4 4 4, basta fazer

```
>> linha=[4 4 4];A=[A;linha]
```

ou apenas

[5] este procedimento não é contudo aconselhável.

```
>> A=[A;4 4 4]
```

para se obter a nova matriz

```
A =
     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

Vetores com componentes inteiras podem também ser usados como índices. Por exemplo, considerando a matriz

```
>> A=[1 2 3;4 5 6;7 8 9;-1 -2 -3;1 1 1]
```

a declaração

```
>> B=A(1:2:5,[1,3])
```

resultará numa matriz que contém as linhas 1, 3 e 5 e as colunas 1 e 3 da matriz inicial,

```
B =
   1     3
   7     9
   1     1
```

enquanto a atribuição

```
>> C=A(1:3,:)
```

resultará numa matriz que tem as 3 primeiras linhas de A.

```
C =
     1     2     3
     4     5     6
     7     8     9
```

Pode ainda usar-se **end** para referir o índice mais elevado de uma dimensão. Por exemplo,

```
>> A(end-1,end)
```

resultará no elemento da matriz A que está na penúltima linha e última coluna, isto é,

```
ans =
   -3
```

Vetores como índices podem aparecer em ambos os lados de uma atribuição. Por exemplo, sendo E a matriz

```
>> E=[1 2 3; 2 4 2;3 0 3];
```

a atribuição

```
>> A([1 3],:)=E([2 1],:)
```

resultará na matriz

```
A =
     2     4     2
     4     5     6
     1     2     3
    -1    -2    -3
     1     1     1
```

Também é possível apagar linhas e colunas de uma matriz. Para retirar a segunda linha da matriz A basta fazer

```
>> A(2,:)=[]
    2     4     2
    1     2     3
   -1    -2    -3
    1     1     1
```

⇛ Tutoriais

www.mathworks.com/help/matlab/learn_matlab/desktop.html

www.mathworks.com/help/matlab/learn_matlab/workspace.html

www.mathworks.com/help/matlab/import_export/write-to-a-diary-file.html?s_tid=srchtitle

www.mathworks.com/help/matlab/learn_matlab/help.html

www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html

www.mathworks.com/help/matlab/learn_matlab/array-indexing.html

2. Operações com matrizes

⇛ Transposta de uma matriz

O símbolo ’ denota a transposta de uma matriz, no caso em que esta é real, ou a transconjugada de uma matriz, se esta for complexa.[6]
As declarações

```
>> A=[1 2 3;4 5 6;7 8 9]
>> B=A'
>> X=[-1+i 2-i 3]'
```

resultam nas matrizes

```
A =
     1     2     3
     4     5     6
     7     8     9
B =
     1     4     7
     2     5     8
     3     6     9
X =
  -1.0000 - 1.0000i
   2.0000 + 1.0000i
   3.0000
```

⇛ Soma de matrizes

Os símbolos $+$ e $-$ designam soma e subtração de matrizes e estas operações estão definidas de forma usual. Assim, a atribuição

```
>> C=A+B
```

resulta na matriz

```
C =
     2     6    10
     6    10    14
    10    14    18
```

No MATLAB é ainda possível definir a operação $A+B$ ou $A-B$ sempre que uma das matrizes tem ordem 1, i.e. é um escalar ou é um vetor (matriz $1\times n$ ou $n\times 1$). No primeiro caso, a matriz resultante é a que se obtém somando ou subtraindo o escalar a todos os elementos da outra matriz. As declarações

[6] A transposta de uma matriz complexa A pode obter-se fazendo `A.'` ou `conj(A')`.

```
>> D=A-1;
>> D=-ones(3)+A
```

originam exatamente a mesma matriz

```
D =
     0     1     2
     3     4     5
     6     7     8
```

Já no segundo caso, a operação corresponde à operação entre a matriz dada e uma matriz com todas as linhas/colunas iguais ao vetor linha/coluna e com a dimensão adequada à operação.

Exemplo

```
>> vlinha=A(1,:)
vlinha =
     1     2     3

>> repmat(linha,3,1)
ans =
     1     2     3
     1     2     3
     1     2     3

>> A+ans
ans =
     2     4     6
     5     7     9
     8    10    12

>> A+vlinha
ans =
     2     4     6
     5     7     9
     8    10    12

>> vcoluna=A(:,1)
vcoluna =
     1
     4
     7

>> repmat(vcoluna,1,3)
ans =
     1     1     1
     4     4     4
     7     7     7
```

Exemplo (cont.)

```
>> A+ans
ans =
     2     3     4
     8     9    10
    14    15    16

>> A+vcoluna
ans =
     2     3     4
     8     9    10
    14    15    16
```

⇛ Produto de matrizes

O símbolo * denota multiplicação de matrizes e esta operação está definida da forma usual. Por exemplo, se

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \qquad y = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix},$$

então as atribuições

```
>> M1=A*B
>> M2=4*A
>> M3=x'*y
```

originam as matrizes

```
M1 =
     22     28
     49     64
     76    100
M2 =
      4      8     12
     16     20     24
     28     32     36
M3 =
      4      5      6
      8     10     12
     12     15     18
```

e os comandos

```
>> M4=B*A;
>> M5=x*y
>> M6=A*x
```

produzem

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns
in the first matrix matches the number of rows in the second matrix...
```

uma vez que as operações indicadas não estão definidas.

⇛ Quociente de matrizes

Embora o quociente de matrizes não esteja definido, os símbolos $\backslash$ e $/$ representa a "divisão" de duas matrizes no sentido que se descreve de seguida.
Se A é uma matriz quadrada invertível, então $A\backslash B$ e B/A correspondem formalmente ao produto à esquerda e à direita, de B pela inversa de A, isto é,

$$A\backslash B = A^{-1} * B$$

$$B/A = B * A^{-1}$$

Como seria de esperar, o resultado destas operações não passa pelo cálculo explícito da inversa de A, mas sim pela resolução de sistemas[7]. Em geral

$$X = A\backslash B \text{ é solução da equação } A * X = B$$

$$Y = B/A \text{ é solução da equação } Y * A = B$$

Sejam

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad \text{e} \qquad B = \begin{pmatrix} 3 & -3 & 1 \\ 15 & 12 & 13 \\ 24 & 18 & 19 \end{pmatrix}$$

Então, se

```
>> X=A\B,Y=A/B
```

obtém-se

```
X =
   1.0000    1.0000   -1.0000
   1.0000   -2.0000    1.0000
   1.0000    3.0000    2.0000


Y =
  -0.4103   -1.2051    0.8462
  -0.0256    2.4872   -1.3846
  -0.1026    2.9487   -1.5385
```

Naturalmente que se as matrizes forem de ordem 1, i.e. escalares, as duas divisões correspondem à divisão usual. Assim, $2\backslash 8 = 8/2 = 4$ e $8\backslash 2 = 2/8 = 0.25$.

[7] Este problema será estudado com detalhe na unidade curricular álgebra Linear I.

⇛ Operações elemento a elemento

As operações elemento a elemento (ou ponto a ponto) diferem das operações usuais com matrizes, mas podem ter grande aplicação prática. Para indicar que uma dada operação é para ser feita elemento a elemento deve usar-se o ponto (.) imediatamente antes do operador.

Considerando as matrizes $A = (a_{ij})$, $B = (a_{ij})$, $C = (a_{ij})$, $X = (x_{ij})$ e $Y = (y_{ij})$, as atribuições

$$A = X.\hat{}\,Y, \;\; B = X. * Y \;\text{ e }\; C = X./Y,$$

têm, respetivamente, o seguinte significado:

$$a_{ij} = (x_{ij})^{y_{ij}}, \;\; b_{ij} = x_{ij} * y_{ij} \;\text{ e }\; c_{ij} = x_{ij}/y_{ij}.$$

Para que estas operações possam ser executadas, as matrizes (ou vetores) X e Y têm que ter (em geral) a mesma ordem. Note-se que $X. + Y$ e $X. - Y$ não estão definidas (Porquê?).

Consideremos as matrizes

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad \text{e} \qquad Y = \begin{pmatrix} 2 & 3 \\ 3 & 0 \end{pmatrix}$$

As seguintes operações elemento a elemento

```
>> E1=X.*Y,E2=Y./X,E3=X.^Y
```

produzem as matrizes

```
E1 =
     2     6
     9     0

E2 =
     2.0000    1.5000
     1.0000         0

E3 =
     1     8
    27     1
```

⇛ Tutoriais

www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html?s_tid=srchtitle

www.mathworks.com/help/matlab/matlab_prog/compatible-array-sizes-for-basic-operations.html

3. Funções predefinidas

O MATLAB contém um conjunto grande de funções já definidas. Algumas dessas funções são funções matriciais, como por exemplo **inv**(A), que calcula a inversa de uma matriz quadrada A (invertível), outras estão definidas elemento a elemento. Por exemplo, $C = \cos(A)$ é uma matriz cujos elementos c_{ij} são os valores do cosseno dos elementos de $A = (a_{ij})$, i.e. $c_{ij} = \cos(a_{ij})$.
Segue-se, a título de exemplo, uma lista das funções mais usadas, dividida em 6 categorias: funções matemáticas elementares, matrizes elementares e manipulação de matrizes, funções matriciais, polinómios, análise de dados e funções matemática específicas. Algumas destas funções são ainda *estranhas* aos alunos do 1º ano. Ao longo do ano, elas poderão vir a ser especialmente úteis e apreciadas nas várias unidades curriculares do plano de estudos do curso. Uma lista completa de todas as funções existentes em cada uma destas categorias pode ser obtida invocando o **help**. Por exemplo, `help elfun`, `help elmat`, `help matfun`, `help polyfun`, `help datafun`, `help specfun`.[8]

⇛ Funções matemáticas elementares

➟ Funções Trigonométricas

sin	- *Seno.*
sinh	- *Seno hiperbólico.*
asin	- *Arco-seno.*
asinh	- *Arco-seno hiperbólico.*
cos	- *Cosseno.*
cosh	- *Cosseno hiperbólico.*
acos	- *Arco-cosseno.*
acosh	- *Arco-cosseno hiperbólico.*
tan	- *Tangente.*
tanh	- *Tangente hiperbólica.*
atan	- *Arco-tangente.*
atanh	- *Arco-tangente hiperbólica.*
sec	- *Secante.*
sech	- *Secante hiperbólica.*
asec	- *Arco-secante.*
asech	- *Arco-secante hiperbólica.*
csc	- *Cossecante.*
csch	- *Cossecante hiperbólica.*
acsc	- *Arco-cossecante.*
acsch	- *Arco-cossecante hiperbólica.*
cot	- *Cotangente.*
coth	- *Cotangente hiperbólica.*
acot	- *Arco-cotangente.*
acoth	- *Arco-cotangente hiperbólica.*

[8]As diversas categorias de funções encontram-se em (pasta instalação)\MATLAB\R2020b\toolbox\MATLAB.

Exemplo

```
>> sin([0 pi/2 pi 3*pi/2 2*pi])
ans =
     0    1.0000    0.0000   -1.0000   -0.0000

>> atan([0 Inf])
ans =
     0    1.5708
```

➟ **Função Exponencial**

exp	- *Exponencial.*
log	- *Logaritmo neperiano.*
log10	- *Logaritmo na base 10.*
log2	- *Logaritmo na base 2.*
sqrt	- *Raiz quadrada.*
nthroot	- *Raíz índice n de um número real.*

Exemplo

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

>> log2(8)
ans =
   3

>> nthroot(16,4)
ans =
   2
```

➟ **Funções Complexas**

abs	- *Módulo.*
conj	- *Conjugado.*
imag	- *Parte imaginária.*
real	- *Parte real.*
isreal	- *Retorna verdadeiro para arrays reais.*

Exemplo

```
>> z=-3+4i;
>> abs(z)
ans =
   5

>> conj(z)
ans =
   -3.0000 - 4.0000i

>> real(z)
ans =
   -3
```

➟ **Arredondamento e resto da divisão**

fix	- *Arredondamento na direção de zero.*
floor	- *Arredondamento na direção de* $-\infty$.
ceil	- *Arredondamento na direção de* $+\infty$.
round	- *Arredondamento na direção do inteiro mais próximo.*
mod	- *Resto da divisão, com sinal.*
rem	- *Resto da divisão.*
sign	- *Sinal.*

Exemplo

```
>> x=-sqrt(7)
x =
   -2.6458

>> fix(x)
ans =
   -2

>> floor(x)
ans =
   -3

>> ceil(x)
ans =
   -2

>> round(x)
ans =
   -3

>> rem(-7,2),mod(-7,2)
ans =
   -1

ans =
   1

>> rem(-7,0), mod(-7,0)
ans =
   NaN

ans =
   -7

>> sign(x)
ans =
   -1
```

Note-se que **rem** (x, y), para $x \neq y$ e $y \neq 0$ (no MATLAB, **mod** $(x, 0) = x$), tem o mesmo sinal de x; **mod** (x, y) e **rem** (x, y) são iguais se x e y têm o mesmo sinal, mas diferem de y se x e y têm sinais diferentes.

⇛ Matrizes elementares e manipulação de matrizes

➟ Matrizes elementares

zeros	- *Matriz nula.*
ones	- *Matriz com todos os elementos 1.*
eye	- *Matriz identidade.*
repmat	- *Replicar e agrupar array.*
repelem	- *Replicar elementos de um array.*
linspace	- *Vetor de elementos igualmente espaçados.*

Exemplo

```
>> A1=eye(3)
A1 =
     1     0     0
     0     1     0
     0     0     1

>> A2=zeros(2,3)
A2 =
     0     0     0
     0     0     0

>> A3=2*ones(1,3)
A3 =
     2     2     2

>>A4=[1 2;3 4];

>> repmat(A4,2,3)
ans =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4

>> repelem(A4,2,3)
ans =
     1     1     1     2     2     2
     1     1     1     2     2     2
     3     3     3     4     4     4
     3     3     3     4     4     4

>> linspace(-1,1,5)
ans =
   -1.0000   -0.5000         0    0.5000    1.0000
```

➟ **Informação básica**

size	- *Tamanho do array.*
length	- *Tamanho do vetor.*
numel	- *Número de elementos.*
disp	- *Exibir matriz ou texto.*
height	- *Número de colunas.*
width	- *Número de linhas.*

Exemplo

```
>> a=repmat([1;2],1,4)
a =
   1     1     1     1
   2     2     2     2

>> b=a(:)'
b =
   1     2     1     2     1     2     1     2

>> size(a)
ans =
   2     4

>> size(b)
ans =
   1     8

>> length(a)
ans =
   4

>> length(b)
ans =
   8

>> [numel(a);numel(b)]
ans =
   8
   8

>> [height(a) height(b);width(a) width(b)]
ans =
   2     1
   4     8

>> disp(a)
   1     1     1     1
   2     2     2     2
```

➟ Variáveis especiais e constantes

ans	- *Resposta mais recente.*
eps	- *Epsilon da máquina.*
realmax	- *Maior número em vírgula flutuante.*
realmin	- *Menor positivo em vírgula flutuante.*
pi	- *3.1415926535897....*
i, j	- *Unidade imaginária.*
inf	- *Infinito.*
NaN	- *Not-a-Number.*

Exemplo

```
>> eps
ans =
   2.2204e-16

>> realmax
ans =
   1.7977e+308

>> realmin
ans =
   2.2251e-308

>> 0/0
ans =
   NaN

>> 1/0
ans =
   Inf

>> pi=5 % as funções/constantes não estão protegidas
pi =
5

>> clear pi, pi % para recuperar o valor original
ans =
3.1416
```

➟ Manipulação de matrizes

reshape	- *Redimensionar uma matriz.*
diag	- *Criar ou extrair diagonais.*
tril	- *Extrair parte triangular inferior.*
triu	- *Extrair parte triangular superior.*
fliplr	- *Rodar a matriz na direção esquerda/direita.*
flip	- *Trocar a ordem dos elementos.*
rot90	- *Rodar uma matriz 90 graus.*
find	- *Encontrar os índices dos elementos não nulos.*

Exemplo

```
>> a=reshape(1:9,3,3)
   1     4     7
   2     5     8
   3     6     9
>> diag(a)
ans =
   1
   5
   9

>> diag(ans)
ans =
   1     0     0
   0     5     0
   0     0     9

>> triu(a)
ans =
   1     4     7
   0     5     8
   0     0     9

>> [fliplr(a) flip(a)]
ans =
   7     4     1     3     6     9
   8     5     2     2     5     8
   9     6     3     1     4     7

>> rot90(a)
ans =
   7     8     9
   4     5     6
   1     2     3

>> rem(a,3)
ans =
   1     1     1
   2     2     2
   0     0     0

>> find(ans)'
ans =
1     2     4     5     7     8
```

⇛ Funções matriciais

Estas funções dizem respeito a conteúdos ainda não lecionados, mas a maior parte delas poderá ser útil num futuro próximo, nomeadamente nas unidades curriculares Álgebra Linear I e II.

➟ Análise Matricial

norm	- *Norma matricial ou vetorial.*
rank	- *Característica de uma matriz.*
det	- *Determinante de uma matriz.*
trace	- *Traço de uma matriz.*
null	- *Núcleo de uma matriz.*

Exemplo

```
>> a=reshape(1:9,3,3);
>> rank(a)
ans =
   2

>> det(a)
ans =
   0

>> trace(a)
ans =
   15
```

➟ Equações Lineares

\ e /	- *Solução de um sistema linear; use "help slash".*
linsolve	- *Solução de um sistema linear.*
inv	- *Inversa de uma matriz.*

Exemplo

```
>> a=[1 1 1;3 4 5;3 6 10];b=eye(3,1);
>> a\b
ans =
    10.0000
   -15.0000
     6.0000

>> linsolve(a,b)
ans =
    10.0000
   -15.0000
     6.0000

>> inv(a)
ans =
    10.0000    -4.0000     1.0000
   -15.0000     7.0000    -2.0000
     6.0000    -3.0000     1.0000
```

➟ Valores Próprios

eig	- *Valores e vetores próprios.*
poly	- *Polinómio característico.*
hess	- *Forma de Hessenberg.*

Exemplo

```
>> a=[-5 -4 0;9 5 -5;-3 0 6];
>> [vetores valores]=eig(a)
vetores =
    0.5263    0.4650    0.4082
   -0.7895   -0.8137   -0.8165
    0.3158    0.3487    0.4082

valores =
   1.0000         0         0
        0    2.0000         0
        0         0    3.0000

>> poly(a)
ans =
   1.0000   -6.0000   11.0000   -6.0000
```

⇛ Interpolação e Polinómios

As funções correspondentes à interpolação serão estudadas no 2º ano em Análise Numérica, pelo que não se listam aqui.

➟ Polinómios

roots	- *Raízes de um polinómio.*
poly	- *Construção de um polinómio, dadas as suas raízes.*
polyval	- *Avaliar um polinómio.*
polyder	- *Derivada de um polinómio.*
conv	- *Produto de polinómios.*
deconv	- *Divisão de polinómios.*

Exemplo

```
>> p=poly([1,2,3,4]) %  O polinómio p=(x-1)(x-2)(x-3)(x-4)
p =
   1   -10    35   -50    24

>> roots(p) % zeros do polinómio
ans =
   4.0000
   3.0000
   2.0000
   1.0000
```

Exemplo (cont.)

```
>> polyval(p,[1,6]) % p(1) e p(6)
ans =
   0   120

>> polyder(p) % derivada de p, i.e. -50+70x-30x^2+4x^3
ans =
   4   -30    70   -50

>> deconv(p,[1,-1]) % divisão de p por x-1, i.e. (x-2)(x-3)(x-4)
ans =
   1    -9    26   -24

>> poly([2,3,4])
ans =
   1    -9    26   -24

>> conv(ans,[1,-1]) % o produto de (x-2)(x-3)(x-4) por x-1, i.e. p
ans =
   1   -10    35   -50    24
```

⇛ Análise de dados

➟ Operações básicas

sum	- *Soma dos elementos.*
prod	- *Produto dos elementos.*
min	- *Mínimo.*
max	- *Máximo.*
mean	- *Média.*
median	- *Mediana.*
mode	- *Moda.*
std	- *Desvio padrão.*
var	- *Variância.*
sort	- *Ordenação.*

Exemplo

```
>> x=[2 -2 -1 8 -2 7 3 9 -2 2];
>> [sum(x);prod(x)]
ans =
      24
   48384

>> [min(x); max(x)]
ans =
   -2
    9
```

Exemplo (cont.)

```
>> [mean(x);median(x);mode(x)]
ans =
    2.4000
    2.0000
   -2.0000

>> [std(x) std(x)^2 var(x)]
ans =
   4.2999   18.4889   18.4889
>> sort(x)
ans =
   -2    -2    -2    -1     2     2     3     7     8     9

>> sort(x,'ascend')
ans =
   -2    -2    -2    -1     2     2     3     7     8     9

>> sort(x,'descend')
ans =
   9     8     7     3     2     2    -1    -2    -2    -2
```

⇛ Funções matemáticas específicas

➟ **Teoria de números**

factor	- *Decomposição em fatores primos..*
isprime	- *Verdadeiro, se o argumento é um número primo.*
primes	- *Gerar uma lista de números primos.*
gcd	- *Máximo divisor comum.*
lcm	- *Mínimo múltiplo comum.*
perms	- *Permutações.*
nchoosek	- *Combinações.*
factorial	- *Fatorial.*

Exemplo

```
>> factor(24) % fatorização em primos de 24=2^3*3
ans =
   2     2     2     3

>> isprime(24) % 24 não é primo
ans =
   logical
     0
```

Exemplo (cont.

```
>> primes(24) % lista de primos inferiores a 24
ans =
   2     3     5     7    11    13    17    19    23
>> gcd(24,16) % máximo divisor comum entre 24 e 16
ans =
   8

>> lcm(3,5) % mínimo múltiplo comum entre 3 e 5
ans =
   15

>> perms([1 2 3]) % todas as permutações dos elementos 1,2,3
ans =
   3     2     1
   3     1     2
   2     3     1
   2     1     3
   1     3     2
   1     2     3

>> nchoosek(6,4) % combinações dos 6, 4 a 4
ans =
   15

>> factorial(6)/(factorial(4)*factorial(2))
ans =
   15
```

$\Rrightarrow$ Tutoriais

www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html?s_tid=srchtitle

www.mathworks.com/help/matlab/matrices-and-arrays.html?s_tid=CRUX_lftnav

www.mathworks.com/help/matlab/trigonometry.html

www.mathworks.com/help/matlab/exponents-and-logarithms.html

www.mathworks.com/help/matlab/complex-numbers.html

www.mathworks.com/help/matlab/math/create-and-evaluate-polynomials.html

www.mathworks.com/help/matlab/math/roots-of-polynomials.html

www.mathworks.com/help/matlab/descriptive-statistics.html?s_tid=CRUX_lftnav

www.mathworks.com/help/matlab/discrete-math.html

4. Operadores

A linguagem MATLAB usa muitos dos operadores que normalmente são usados para realizar operações simples em arrays de qualquer tipo. Já foram descritos anteriormente os operadores aritméticos. Daremos agora destaque aos operadores relacionais, lógicos e funções associadas, referindo também as operações entre conjuntos.

⇛ Operadores relacionais

No MATLAB, os operadores relacionais são os seguintes:

Operadores relacionais	Descrição
$==$	igual
$<=$	menor ou igual
$>=$	maior ou igual
$\sim=$	diferente
$<$	menor
$>$	maior

O resultado da aplicação de um operador relacional é do tipo lógico - **logical** - e pode ser **1** (**true**) ou **0** (**false**). Assim,

```
>> 2>3
```

resulta em

```
   logical
     0
```

enquanto

```
>> 2==4-2
```

devolve

```
   logical
     1
```

⇛ Operadores lógicos

Adicionalmente podem ainda formar-se expressões lógicas mais complicadas combinando elementos lógicos e expressões de relação, com os seguintes operadores lógicos.

Operadores lógicos	Descrição
& (and)	e
$\mid$ (or)	ou
$\sim$ (not)	negação
xor	ou exclusivo
&&, $\Vert$	operadores de curto-circuito

Relembre as tabelas de verdade das seguintes operações:

p	$\sim$ p
V	F
F	V

p	q	p|q
V	V	V
V	F	V
F	V	V
F	F	F

p	q	xor(p,q)
V	V	F
V	F	V
F	V	V
F	F	F

p	q	p&q
V	V	V
V	F	F
F	V	F
F	F	F

Nas chamadas operações de curto-circuito, o segundo argumento é apenas executado ou avaliado se o primeiro argumento não for suficiente para determinar o valor da expressão: quando o primeiro argumento da função AND é avaliado como falso, o valor global é falso e quando o primeiro argumento da função OR for avaliado como verdadeiro, o valor global é verdadeiro.

Exemplo

```
>> ~[2>3,2==4-2]
ans =
   1×2 logical array
     1   0

>> (2<3)|(2>3)
ans =
   logical
     1

>> (2<3)&(2>3)
ans =
   logical
     0
```

O resultado da aplicação de um operador relacional a matrizes da mesma dimensão é uma matriz com 0's e 1's resultante da aplicação do operador elemento a elemento.

Exemplo

```
>> A=[1 2;3 4]; B=[2 2;1 5];
>> A>=2
ans =
   2×2 logical array
     0   1
     1   1

>> A>B
ans =
   2×2 logical array
    0   0
    1   0
```

Arrays lógicos podem também ser usados como índices. Obtém-se neste caso um array com os valores para os quais o índice é 1. Por exemplo,

```
>> a=[1 2 3 4];
```

```
>> a>2
ans =
   1×4 logical array
     0     0     1     1

>> a(ans)
ans =
     3     4
```

Note-se que as instruções

```
>> b=[0 0 1 1];
>> a(b)
```

resultam em

```
Array indices must be positive integers or logical values.
```

porque b é um *array* de reais. Teria que ser convertido num *array* lógico para que a operação fosse possível (veja secção seguinte).

```
>> a(logical(b))
ans =
     3     4
```

⇛ Funções para operações lógicas

Para além dos já referidos operadores lógicos, o MATLAB fornece um conjunto de outras funções para realizar operações lógicas. A tabela seguinte lista algumas dessas funções.

all	- *Determinar se todos os elementos de um array são não nulos ou true.*
any	- *Determinar se algum dos elementos de um array é não nulo ou true.*
false	- *Falso (logical 0).*
true	- *Verdadeiro (logical 1).*
find	- *Determinar índices e valores de elementos não nulos.*
islogical	- *Determinar se o argumento é do tipo logical.*
logical	- *Converter valores numéricos em lógicos.*

Exemplo

```
>> a=randi([-5,9],[3,4])
a =
   -4     6    -4     7
    9     7     0     1
   -5     8    -2     8

>> all(a>0)
ans =
   1×4 logical array
     0   1   0   1
```

Exemplo (cont.)

```
>> all(a>0,'all')
ans =
logical
0

>> any(a<0,2)
ans =
   3×1 logical array
     1
     0
     1

>> find(a>7)
ans =
    2
    6
    12

>> a(ans)
ans =
   9
   8
   8

>> a(a>7)
ans =
   9
   8
   8

>> [linha coluna]=find(a>7)
linha =
   2
   3
   3

coluna =
   1
   2
   4
```

Para além da função **islogical** e da função já mencionada **isprime**, existem ainda várias outras funções cujo nome começa por **is** e cujo resultado é true/false.
A título de exemplo referem-se as funções **isdiag**, **isempty**, **isletter**,**isnan**, **istril**, cuja designação é suficientemente sugestiva para antecipar o seu objetivo (confirme através do help e de exemplos).

⇛ Operações com conjuntos

As operações com conjunto comparam os elementos de dois conjuntos para encontrar semelhanças ou diferenças entre eles. As operações mais frequentes estão listadas de seguida.

intersect	- *Interseção.*
ismember	- *Determinar elementos que pertencem a um conjunto.*
setdiff	- *Diferença entre dois arrays.*
union	- *União.*
unique	- *Valores únicos de um array.*

Exemplo

```
>> a=[-3  -2  -2  -1  -3]; b=[-1 4 -1 -3];
>> intersect(a,b)
ans =
   -3    -1

>> union(a,b)
ans =
   -3    -2    -1     4

>> unique(a)
ans =
   -3    -2    -1

>> union(a,a)
ans =
   -3    -2    -1

>> setdiff(a,b)
ans =
   -2

>> setdiff(b,a)
ans =
   4

>> ismember(a,b)
ans =
   1×5 logical array
     1   0   0   1   1

>> ismember(b,a)
ans =
   1×4 logical array
     1   0   1   1
```

Exemplo (cont.)

```
>> a=reshape(1:9,3,3);
>> b=repelem(a(1:2,1:2),2,1);
>> intersect(a,b)'
ans =
   1     2     4     5

>> setdiff(a,b)'
ans =
   3     6     7     8     9

>> ismember(b,a)
ns =
   4×2 logical array
     1   1
     1   1
     1   1
     1   1
```

Para obter uma lista das funções predefinidas existentes nesta categoria, faça `help ops`.

$\Rrightarrow$ Tutoriais

www.mathworks.com/help/matlab/relational-operators.html?s_tid=CRUX_lftnav

www.mathworks.com/help/matlab/logical-operations.html?s_tid=CRUX_lftnav

www.mathworks.com/help/matlab/matlab_prog/find-array-elements-that-meet-a-condition.html

www.mathworks.com/help/matlab/matlab_prog/reduce-logical-arrays-to-single-value.html

www.mathworks.com/help/matlab/set-operations.html?s_tid=CRUX_lftnav

5. Carateres e strings

No MATLAB, para além de dados numéricos e do tipo lógico, podem também ser utéis dados do tipo `char` (carateres) ou `string` (cadeia de carateres).

⇛ Dados do tipo `char`

Um array de carateres é uma sequência de carateres, tal como um array numérico é uma sequência de números, podendo ser manipulado de forma análoga. É normalmente usado para armazenar pequenas partes de texto como vetores de carateres, devendo para o efeito usar-se o delimitador '.

Exemplo

```
>> a='Matemática Computacional I'
a =
   'Matemática Computacional I'

>> a(1)
ans =
   'M'

>> size(a)
ans =
   1    26

>> unique(a)
ans =
   ' CIMaceilmnoptuá'

>> isa(a,'char')
ans =
   logical
     1
```

⇛ Dados do tipo `string`

Uma string é uma cadeia de carateres, tendo associada um conjunto de funções que permite trabalhar texto como dados. A partir da versão R2017a, as strings podem ser criadas usando aspas ".

Exemplo

```
>> A="Matemática Computacional I"
A =
   "Matemática Computacional I"

>> size(A)
ans =
   1     1
```

⇛ Funções

Embora o foco deste curso esteja nas potencialidades numéricas do MATLAB, é ainda assim interessante conhecer algumas funções que permitem tratar e manipular texto. Destacamos aqui as funções **string**, **strings**, **join**, **plus**, **append**, **ischar**, **isstring**, **strlength**, **isletter**, **contains**, **count**, **strfind**, **replace**, **split**, **strjoin**, **erase**, cujos objetivos podem ser encontrados recorrendo ao help do MATLAB.

Exemplo

```
>> texto=["Matemática", "Computacional";"é uma UC","do 1º ano"]
texto =
   2×2 string array
     "Matemática"    "Computacional"
     "é uma UC"      "do 1º ano"

>> join(texto)
ans =
   2×1 string array
     "Matemática Computacional"
     "é uma UC do 1º ano"

>> strjoin(texto')
ans =
     "Matemática Computacional é uma UC do 1º ano"

>> matches(texto,"Computacional")
ans =
   2×2 logical array
     0    1
     0    0

>> texto(ans)
ans =
   "Computacional"

>> contains(texto,"C",'IgnoreCase',true)
ans =
   2×2 logical array
     1    1
     1    0

>> frase=strjoin(texto')
frase =
   "Matemática Computacional é uma UC do 1º ano"

>> replace(frase,["é","1º"],["não é","2º"])
ans =
"Matemática Computacional não é uma UC do 2º ano"
```

Exemplo (cont.)

```
>> erase(ans,"não")
ans =
   "Matemática Computacional  é uma UC do 2º ano"

>> append(frase," da licenciatura")
ans =
   "Matemática Computacional é uma UC do 1º ano da licenciatura"

>> strlength(frase)
ans =
   43
```

⇛ Tutoriais

www.mathworks.com/help/matlab/characters-and-strings.html?s_tid=CRUX_lftnav

www.mathworks.com/help/matlab/matlab_prog/represent-text-with-character-and-string-arrays.html

www.mathworks.com/help/matlab/matlab_prog/searching-and-replacing.html

6. Exercícios

⇛ Aulas laboratoriais

Exercício 1. Atribua o valor sen $\frac{\pi}{12}$ à variável a e e^2 à variável b. Calcule:

a) $12a^2 - a - 6$;

b) $\sqrt{a} + 3b$;

c) $\sqrt[3]{a} + \log b$.

Repita o exercício, usando `format long`.

Exercício 2. Sem usar o MATLAB, indique que valores se obtêm se efetuar os seguintes comandos; confirme a sua resposta.

```
>> i*(1-i)^2-2
>> 1/ans
>> 1/ans
>> 0/ans
```

Exercício 3. Defina, no MATLAB, a matriz

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & -1 \\ 1 & -5 & 4 \end{pmatrix}$$

e obtenha de uma forma simples, a partir de A, as seguintes matrizes.

$$A_1 = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & 1 \\ 1 & 5 & 4 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 2 & 1 \\ 0 & 0 & 4 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1 & -1 & 2 & 1 \\ 3 & 2 & -1 & 2 \\ 1 & -5 & 4 & 3 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & -1 \\ 1 & -5 & 4 \\ 1 & 2 & 3 \end{pmatrix} \quad A_5 = \begin{pmatrix} 2 & 1 & -1 & 2 \\ 0 & 1 & -1 & 2 \\ 0 & 3 & 2 & -1 \\ 0 & 1 & -5 & 4 \end{pmatrix} \quad A_6 = \begin{pmatrix} 1 & -1 & 2 & 0 & 0 \\ 3 & 2 & -1 & 0 & 0 \\ 1 & -5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$A_7 = \begin{pmatrix} 1 & -1 & 2 & 1 & -1 & 2 \\ 3 & 2 & 1 & 3 & 2 & 1 \\ 1 & 5 & 4 & 1 & 5 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A_8 = \begin{pmatrix} 1 & -1 \\ 3 & 2 \end{pmatrix}.$$

Exercício 4. Construa o vetor `z=[10,20,30,40,50,60,70,80]`. Sem usar o MATLAB, indique que vetores se obtêm se efetuarmos cada um dos seguintes comandos.

```
a. >> u=z(1:2:7)             b. >> v=z(7:-2:1)
c. >> w=z([3 4 8 1])         d. >> z(1:2:7)=zeros(1,4)
e. >> z(7:-2:1)=ones(1,4)    f. >> z(1:3)=[ ]
```

Verifique, no MATLAB, as suas respostas.

Exercício 5. Dada a matriz `A = [2 7 9 7;3 1 5 6;8 1 2 5]`, explique o resultado dos seguintes comandos:

```
a. >> A(1,[2 3])             b. >> A(:,[1 4])
c. >> A([2 3],[3 1])         d. >> reshape(A,2,6)
e. >> A(:)                   f. >> A(end,:)
g. >> A(1:3,:)               h. >> [A ;A(1:2,:)]
i. >> sum(A)                 j. >> sum(A')
k. >> sum(A,2)               l. >> [[A;sum(A)] [sum(A,2);sum(A(:))]]
```

Exercício 6. Defina (de uma forma simples) uma matriz A:

a) de ordem 5 com todos os elementos iguais a 3;

b) diagonal, de ordem 5, com todos os elementos da diagonal iguais a 8;

c) de ordem 4 em que cada coluna seja o vetor $v = (1, 2, 3, 4)^{\mathsf{T}}$;

d) com a seguinte estrutura

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 2 & 0 \\ 1 & 1 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Exercício 7. Defina as matrizes

```
>> x=[1 4 2], y=[-1+i -i 1+i]
>> A=[1 2 3;-1 3 -2], B=[1 -1 1;0 1 2; 1 1 1]
```

Verifique quais das seguintes operações estão bem definidas e, em caso afirmativo, comente o resultado obtido.

```
a. >> x + y          b. >> x + A         c. >> x' + y'
d. >> A - [x ; y]    e. >> [x ; y']      f. >> [x ; y]
g. >> [A B]          h. >> [A ; B]       i. >> A - 3
j. >> A + B          k. >> A * B         l. >> B * A
m. >> A .* B         n. >> B * A'
```

Exercício 8. Construa, de uma forma simples, os seguintes vetores:

$$u = (1,1,1,1,1,1,1,1,1,1); \qquad v = (1,2,3,4,5,6,7,8,9,10);$$
$$x = (0,0.25,0.5,0.75,1); \qquad y = (2,4,6,8,10,12,14,16);$$
$$z = (10,8,6,4,2,0,-2,-4,-6); \qquad w = (0,0,0,0,0,1,1,1,1,1).$$

Exercício 9. Usando os vetores definidos na questão anterior, construa:

$$U = (3,3,3,3,3,3,3,3,3,3); \qquad V = (2,4,8,16,32,64,128,256,512,1024);$$
$$X = (5,5.25,5.5,5.75,6); \qquad Y = (4,16,36,64,100,144,196,256);$$
$$Z = (-6,-4,-2,0,2,4,6,8,10); \qquad W = (0,0,0,1,1,1).$$

Exercício 10. Defina, no MATLAB, o vetor linha x constituído:

a) pelos 10 primeiros números naturais;

b) pelos números pares com início em 4 e término em 100;

c) por uma sequência decrescente de números inteiros com início em 5 e término em -5;

d) pelos números múltiplos de 3 compreendidos entre 10 e 132, dispostos por ordem decrescente;

e) por 5 números aleatórios, usando a função **rand**;

f) por 6 números inteiros aleatórios entre 5 e 10, usando a função **randi**;

g) por 10 números igualmente espaçados do intervalo $[1,2]$, usando a função **linspace**.

h) por 128 elementos com a seguinte forma (`0 1 0 -1 0 1 ... 0 -1`)$^{\mathsf{T}}$.

i) pelos valores da função sen(πt), para $t = 0 : 0.25 : 2$.

Exercício 11. Defina a matriz `A=magic(4)` e indique os comandos para:

a) construir uma matriz `B` cujas colunas são as colunas *pares* da matriz `A`;

b) construir uma matriz `C` cujas linhas são as linhas *ímpares* da matriz `A`;

c) converter `A` numa matriz 2×8;

d) calcular o inverso de cada elemento da matriz `A`;

e) calcular a inversa da matriz `A`;

f) calcular o quadrado de cada elemento da matriz `A`;

g) obter uma matriz com todas as linhas e colunas repetidas (e seguidas).

Exercício 12. Construa um vetor `idades` com as idades dos alunos da turma. Calcule a média, mediana, moda e desvio padrão dessas idades.

Exercício 13.

a) Defina os polinómios $p(x) = 2x^3 - 3x^2 - 1$ e $q(x) = x^3 + 1$.

b) Determine:

- $p + q$ e $p * q$;
- $p^2 + 2q$;
- p' e p'';
- $p(5)$ e $[p(1), p(2), p(3)]$;
- as raízes de p e de q.

c) Construa, de duas formas distintas, um polinómio de grau 3 com zeros 1, i e $-i$.

Exercício 14. Dado `x = [1 5 2 8 9 0 1]` e `y = [5 2 2 6 0 0 2]` execute e explique o resultado dos seguintes comandos:

```
a. >> x > y                    b. >> y < x
c. >> x == y                   d. >> x <= y
e. >> y >= x                   f. >> x | y
g. >> x & y                    h. >> x & (~y)
i. >> (x > y) | (y < x)        j. >> (x > y) & (y < x)
```

Exercício 15. Dado `x = 1:10` e `y = [3 1 5 6 8 2 9 4 7 0]` execute e interprete o resultado dos seguintes comandos:

```
a. >> x(x > 5)                      b. >> y(x <= 4)
c. >> x( (x < 2) | (x >= 8) )       d. >> y( (x < 2) | (x >= 8) )
e. >> y((x > 2) & (x < 8))          f. >> x(y < 0)
```

Exercício 16. Defina o vetor `x=[3 15 9 12 -1 0 -12 9 6 1]` e indique o(s) comando(s) para:

a) atribuir o valor zero aos elementos de `x` que são positivos;

b) atribuir o valor 3 aos elementos de `x` que são múltiplos de 3 (use a função **rem**);

c) multiplicar os elementos pares de `x` por 5;

d) criar um vetor `y` extraindo os valores de `x` que são maiores que 10;

e) atribuir o valor zero aos elementos de `x` que são menores do que a média;

f) atribuir aos elementos de `x` superiores à média, a sua diferença em relação à média.

Exercício 17. Quantos primos com dois dígitos existem? Qual é a sua soma? Repita o exercício para três dígitos.

Exercício 18. Construa uma matriz aleatória 5×5 de números inteiros positivos inferiores a 10 e determine:

a) o maior elemento da matriz; o valor máximo em cada coluna e em cada linha;

b) o índice de linha e coluna de todos os elementos que são superiores a 8.

Exercício 19. Use a função **pascal** para definir uma matriz de Pascal A de ordem 6.

a) Verifique que a matriz A é simétrica.

b) Calcule a soma de todos os elementos da matriz A.

c) Qual é a mediana dos elementos de A?

d) Quantos números primos tem A? (Sugestão: use a função **find** e **isprime**.)

e) Qual é o índice de linha e coluna desses primos?

Exercício 20. Considere a seguinte frase: "Grão a grão enche a galinha o papo".

a) Quantas palavras tem esta frase? E quantos carateres?

b) Substitua na frase a palavra galinha por galo.

c) Verifique que a palavra grão aparece duas vezes no texto.

d) Construa um texto com o seguinte aspeto:

```
1 Grão a grão enche a galinha o papo
2 Filho de peixe sabe nadar
```

e) Quantas letras aparecem no texto?

f) Quantas vezes aparece no texto a vogal a?

Exercício 21. Um *palíndromo* é uma palavra ou um número cuja leitura é a mesma quando efetuada da esquerda para a direita ou da direita para a esquerda. Verifique que **RADAR** e 123321 são palíndromos, mas **RODAR** não.

⇛ Exercícios suplementares

Exercício 1. Obtenha, de uma forma simples, a matriz

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 3 \\ 2 & 1 & 4 & 0 & 6 \\ 3 & 1 & 9 & 0 & 9 \\ 4 & 1 & 16 & 0 & 12 \\ 5 & 1 & 25 & 0 & 15 \end{pmatrix}.$$

a) Construa, a partir de A:

(i) a matriz B, obtida eliminando a 4ª coluna de A;

(ii) a matriz C, obtida por substituição da 2ª coluna de A pela 1ª;

(iii) a matriz D, cujas linhas são a primeira e última linhas da matriz A;

(iv) a matriz E de ordem 5×6 com todas as colunas iguais à 1ª coluna de A;

(v) a matriz F, por substituição dos elementos pares de A por -1.

(vi) a matriz G, por substituição dos elementos pares de A pelo seu dobro.

b) Qual é a soma dos elementos da 3ª coluna de A?

c) Quantos elementos de A são superiores a 3?

d) Qual é a média dos elementos de A?

Exercício 2. Execute as seguintes instruções para construção das matrizes `A` e `B`.

```
x=1:6;A=x(ones(6,1),:),B=A+A'
```

a) Defina, a partir de `A` e `B`:

(i) a matriz `C`,obtida eliminando a 2ª e 4ª linhas de `B`;

(ii) a matriz `D`, obtida por substituição da 1ª linha de `B` pela 2ª coluna de `A`;

(iii) a matriz $\mathtt{E} = (e_{ij})$ tal que $e_{ij} = b_{ij}^2$, (b_{ij} designam os elementos de `B`);

(iv) a matriz `F`, por substituição dos elementos de `B` superiores a 8 por 0;

(v) a matriz `G`, por substituição dos elementos ímpares de `B` pelo seu dobro;

(vi) as matrizes

$$\mathtt{H} = \begin{bmatrix} 2 & 3 & 1 & 0 \\ 3 & 4 & 0 & 1 \\ 0 & 0 & 10 & 11 \\ 0 & 0 & 11 & 12 \end{bmatrix} \quad \mathtt{I} = \begin{bmatrix} 1 & 1 & 1 & 6 & 6 & 6 \end{bmatrix} \quad \mathtt{J} = \begin{bmatrix} 4 & 3 & 2 \\ 5 & 4 & 3 \\ 6 & 5 & 4 \end{bmatrix}$$

b) Sugira uma forma alternativa de construção da matriz `A`.

c) Complete a frase: os elementos b_{ij} da matriz `B` podem escrever-se como $b_{ij} =$ ________; $i, j = 1, \cdots, 6$.

d) Indique a instrução que permite obter:

(i) o maior elemento de `B`;

(ii) o produto dos elementos da diagonal de `B`;

(iii) um vetor linha com todos os elementos de `B`, ordenados por ordem decrescente e sem elementos repetidos, (use a função `unique`);

(iv) a matriz de ordem 6×6, $\mathtt{M} = (m_{ij})$ tal que $m_{ij} = \begin{cases} 1, & \text{se } 4 < i + j < 10 \\ 0, & \text{caso contrário} \end{cases}$

Exercício 3.* A função **magic** permite construir uma *matriz mágica* de ordem n, isto é, uma matriz quadrada de ordem n, em que cada inteiro $1, 2, \cdots, n^2$ aparece uma única vez e em que a soma de cada linha, de cada coluna e de ambas as diagonais é constante.

a) Construa, recorrendo à função **magic**, uma matriz mágica, `M`, de ordem 4.

b) Obtenha a soma das linhas e das colunas de `M`.

c) Construa um vetor que contenha a diagonal principal de `M`.

d) Construa um vetor que contenha a diagonal secundária de `M`.

(Pode recorrer à função **rot90**)

e) Construa a matriz `N`, obtida de `M`, por troca da 2ª com a 3ª coluna.

f) Verifique, usando os comandos adequados, se a matriz `N` é também uma matriz mágica.

Exercício 4. Explique o objetivo das seguintes instruções:

a) `>> reshape(setdiff(1:100,primes(100)),[15 5])`

b) `>> x=1:6;x=x(ones(6,1),:);M=abs(x-x')`

Exercício 5.* Construa uma matriz A cujos elementos a_{ij} satisfazem $a_{ij} = \begin{cases} 1, & \text{se } |i-j| \text{ é primo} \\ 0, & \text{caso contrário} \end{cases}$.

Exercício 6. Qual é o maior número primo com 4 dígitos? E o menor?

Exercício 7.* Um número natural n diz-se **triangular** se $n = 1 + 2 + \cdots + k$, para algum $k \in \mathbb{N}$.

		*	
	*	* *	
*	* *	* * *	
1	3	6	$\cdots$

a) Obtenha os 10 primeiros números triangulares.

Sugestão: Use a função **cumsum**.

b) Apresente uma solução alternativa para a questão anterior, usando a bem conhecida fórmula $1 + 2 + \cdots + k = \frac{k(k+1)}{2}$.

c) Escreva um conjunto de instruções que lhe permita concluir se o número $n = 153$ está entre os primeiros 20 números triangulares.

Exercício 8. Um número natural n diz-se **um número quadrado** ou um quadrado perfeito, se $n = k^2$, para algum $k \in \mathbb{N}$.

		* * *	
	* *	* * *	
*	* *	* * *	
1	4	9	$\cdots$

a) Obtenha os 10 primeiros números quadrados.

b) Como sabe, existem 90 números naturais inferiores a 100 que não são números quadrados. Apresente-os sob a forma de uma tabela com 9 linhas e 10 colunas.

Sugestão: Use a função **setdiff**.

c) Como pode concluir se o número $n = 2809$ é quadrado?

Exercício 9. Como sabe, dois números naturais dizem-se *números primos entre si* se o seu máximo divisor comum é 1.

a) Use a função `factor` para obter a decomposição em primos dos números 2023 e 14535.

b) Use a função `gcd` para obter o máximo divisor comum de 2023 e 14535.

c) Com base nas alíneas anteriores, defina de <u>duas formas distintas</u>, uma variável lógica cujo valor é 1 se 2023 e 14535 forem primos entre si e é 0 no caso contrário.

Exercício 10.* Dois números primos dizem-se *primos gémeos* se diferem 2 unidades, i.e., $(p, p+2)$ é um par de primos gémeos se p e $p+2$ são primos. Os primeiros pares de primos gémeos são $(3,5)$, $(5,7)$, $(11,13)$, $(17,19)$, $\ldots$.

a) Indique um processo de obter todos os pares de primos gémeos menores que 100, apresentando o resultado na forma

```
primosGemeos =
      3     5    11    17   ...
      5     7    13    19   ...
```

b) Verifique que 5 é o único primo menor que 100 que pertence a dois pares de primos gémeos distintos.

c) Um primo p diz-se *isolado* se $p-2$ e $p+2$ não são números primos. Indique duas formas diferentes de obter todos os primos isolados menores que 100.

Obs: Note que um primo isolado não pode pertencer a um par de primos gémeos. A função **setdiff** ou **ismember** do Matlab pode ser útil.

Exercício 11.* Um *número de Mersenne* é um número da forma $2^n - 1$, onde n é um número natural.

a) Determine os 15 primeiros números de Mersenne.

b) Quantos desses números são primos? Explicite-os.

c) Pode provar-se que, *se um número da forma* $2^n - 1$ *é primo, então* n *é primo.*

Ilustre este resultado para os primos de Mersenne encontrados anteriormente.

d) O recíproco deste resultado será válido? Justifique.

⇛ Solução dos exercícios selecionados

➟ Exercício 3 **back**

```
M=magic(4)
```

```
M = 4x4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
sl=sum(M)   % ou sum(M,1)
```

```
sl  = 1x4
    34    34    34    34
```

```
sc=sum(M') % ou sum(M,2)'
```

```
sc  = 1x4
  34    34    34    34
```

```
dp=diag(M)
```

```
dp = 4x1
    16
    11
     6
     1
```

```
ds=diag(rot90(M))
```

```
ds = 4x1
    13
    10
     7
     4
```

Note-se que a soma das diagonais também é 34:

```
sum(dp)
```

```
ans = 34
```

```
sum(ds)
```

```
ans = 34
```

```
N=M(:,[1 3 2 4])
```

```
N = 4x4
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

A soma das linhas e das colunas da matriz N continua igual à da matriz M. Vejamos que a soma das diagonais também é 34. A matriz N é uma matriz mágica.

```
sum(diag(N))
```

```
ans = 34
```

```
sum(diag(rot90(N)))
```

```
ans = 34
```

➟ Exercício 5 back

```
% Usando a matriz M do exercício anterior
isprime(M)
```

```
ans = 6x6 logical array
   0   0   1   1   0   1
   0   0   0   1   1   0
   1   0   0   0   1   1
   1   1   0   0   0   1
   0   1   1   0   0   0
   1   0   1   1   0   0
```

➟ Exercício 7 back

```
primeiros10triangulares=cumsum(1:10)
```

```
primeiros10triangulares = 1x10
     1     3     6    10    15    21    28    36    45    55
```

```
k=1:10;
soma=k.*(k+1)/2
```

```
soma = 1x10
     1     3     6    10    15    21    28    36    45    55
```

```
any(cumsum(1:20)==153)
```

```
ans =
   1
```

➟ Exercício 10 back

```
n=2:100;
p=n(isprime(n)&isprime(n+2));
primosGemeos=[p;p+2]
```

```
primosGemeos = 2x8
     3     5    11    17    29    41    59    71
     5     7    13    19    31    43    61    73
```

```
intersect(primosGemeos(1,:),primosGemeos(2,:))
```

```
ans = 5
```

```
% Primeiro processo
n(isprime(n)&~isprime(n+2)&~isprime(n-2))
```

```
ans = 1x10
     2    23    37    47    53    67    79    83    89    97
```

```
% Segundo processo
primos=primes(100);
setdiff(primos,primosGemeos(:))
```

```
ans = 1x10
     2    23    37    47    53    67    79    83    89    97
```

```
% Terceiro processo
primos(~ismember(primos,primosGemeos(:)))
```

```
ans = 1x10
     2    23    37    47    53    67    79    83    89    97
```

➟ **Exercício 11** back

```
n=1:15;
m=2.^n-1
```

```
m = 1x15
           1           3           7          15          31
          63         127         255         511        1023
        2047        4095        8191       16383       32767
```

```
sum(isprime(m))
```

```
ans = 5
```

```
pM=m(isprime(m))
```

```
pM = 1x5
           3           7          31         127        8191
```

```
find(isprime(m)) % Determina os valores de n tais que 2^n -1 é ...
    primo
```

```
ans = 1x5
     2     3     5     7    13
```

```
isprime(ans)   % Verifica que todos os valores de n obtidos ...
    são primos
```

```
ans = 1x5 logical array
   1   1   1   1   1
```

```
nprimo=n(isprime(n))  % Determina os primos n menores que 15
```

```
nprimo = 1x6
     2     3     5     7    11    13
```

```
isprime(2.^nprimo-1)  % Verifica que 2^n -1 não é ...
   necessariamente primo
```

```
ans = 1x6 logical array
   1   1   1   1   0   1
```

O recíproco não é válido. Para o primo 11, o correspondente número de Mersenne não é primo.

Algoritmos, fluxogramas e pseudo-código

1. Algoritmos

A palavra algoritmo deriva do nome do matemático persa Al-Khwarizmi (780-850). Este matemático e astrónomo introduz na sua obra "Aritmética" (em latim *Algoritmi de numero Indorum*) o sistema de numeração actual (indo-arábico). No seu texto, Al-Khwarizmi apresenta 9 símbolos indianos para representar os algarismos e um círculo para representar o zero. Depois explica como representar um número no sistema decimal posicional e descreve as operações de cálculo.
Ainda que os algoritmos datem de tempos babilónicos e os gregos tenham concebido algoritmos ainda hoje famosos (por exemplo, o algoritmo de Euclides para calcular o máximo divisor comum de dois números), foi Al-Khwarizmi o primeiro a conceber algoritmos tendo em conta a sua eficiência, para o caso concreto da determinação das raízes de equações. No seu tratado "álgebra" é apresentada uma introdução compacta ao cálculo, usando regras para completar e reduzir equações. A palavra álgebra deriva do título desse livro *Hisab al-jabr w'al-muqabala*- álgebra é a tradução latina de *al-jabr*.

Definição: Um algoritmo é uma sequência ordenada, finita e não ambígua de instruções bem definidas para realizar uma determinada tarefa.

Exemplos: Receita culinária; montagem de um kit; ordenação de um conjunto.

➠ Características de um algoritmo

Um algoritmo deve descrever exatamente como realizar determinada tarefa e deve ser:

1. completo;
2. preciso;
3. finito.

➠ Representação de algoritmos

1. Linguagem natural/narrativa

   Os algoritmos são escritos em linguagem natural, por exemplo, o português. é o caso de uma receita culinária.

   $\downarrow$ muito "palavroso";

   $\downarrow$ sensível ao contexto - depende da experiência do leitor.

2. Fluxogramas ou Diagramas de Fluxo

   Representação gráfica que usa formas geométricas padronizadas para indicar as diversas ações e decisões que devem ser executadas para resolver o problema dado.

   $\downarrow$ pode tornar-se muito complexo;

   $\downarrow$ difíceis de alterar/modificar;

   $\downarrow$ detalhes técnicos podem sobrepor-se ao essencial

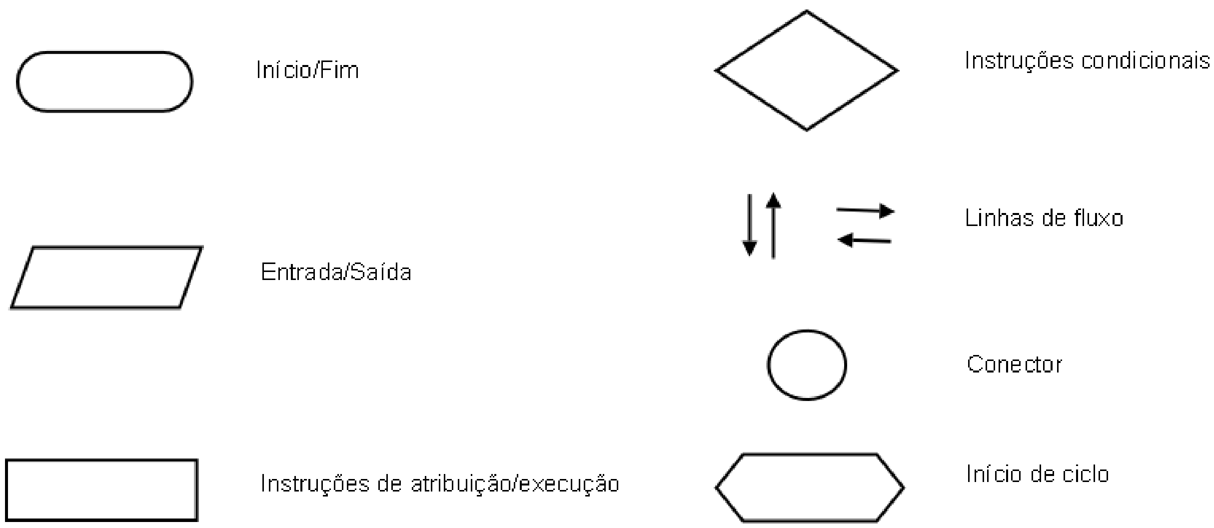
3. Pseudo-código

   Emprega uma linguagem intermédia entre a linguagem natural e uma linguagem de programação para descrever os algoritmos.

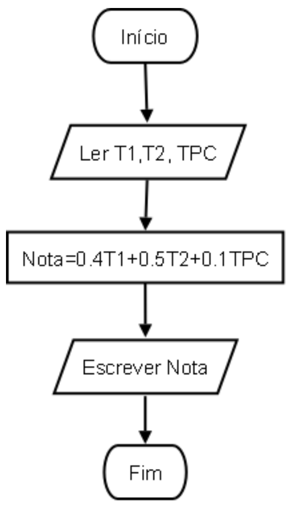
4. Linguagem de programação

2. Fluxogramas

Os fluxogramas descrevem o fluxo de um algoritmo através de um conjunto de símbolos standard. Os mais frequentes são:



Exemplo 1: Calcular a classificação final de uma disciplina com três componentes – T1, T2 e TPC – cujos pesos são: 40% T1, 50% T2 e 10% TPC.



3. Pseudo-código

Pseudo-código é uma lista ordenada/numerada de instruções para realizar uma tarefa, escrita numa linguagem informal, mas mais próxima da linguagem de programação.

Exemplo 1:

(1.) Ler T1,T2 e TPC

(2.) Calcular Nota através de 0.4T1+0.5T2+0.1TPC

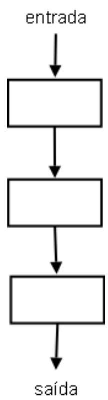
(3.) Escrever Nota

4. Estruturas lógicas de programação

A elaboração de um algoritmo pode envolver 3 estruturas lógicas fundamentais no controle do fluxo de dados e instruções. Estas 3 estruturas de controle são:

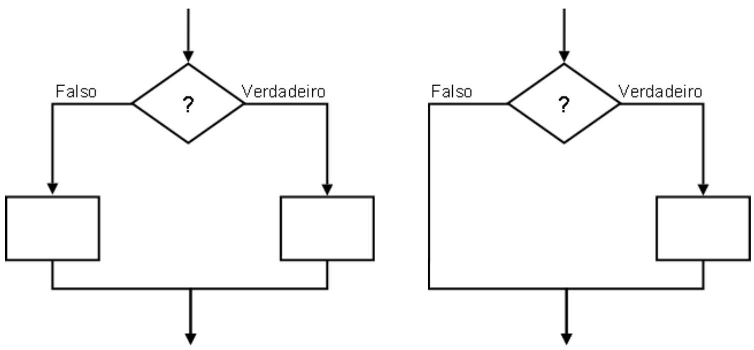
1. **Sequência**

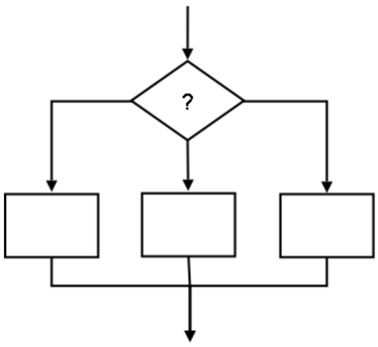
   As instruções são executadas uma após a outra, respeitando sempre a estrutura linear "de cima para baixo".

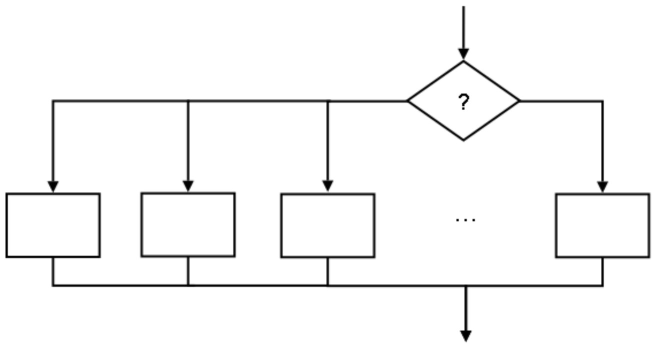


2. **Seleção**

   Esta estrutura exerce o controle sobre uma sequência de instruções a serem executadas, por meio de um teste ou verificação lógica.

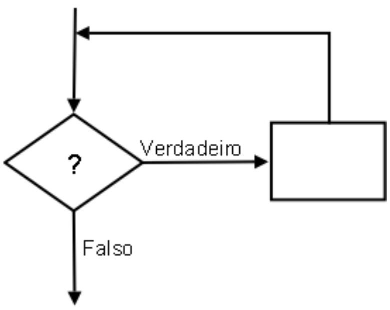




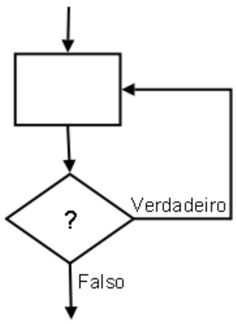


3. **Repetição**

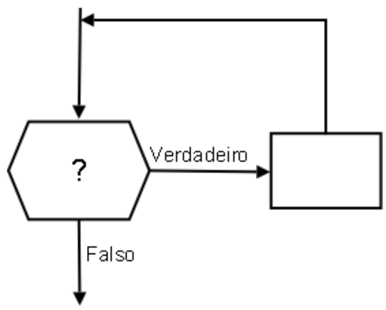
   Através de um teste ou verificação lógica, uma instrução ou uma conjunto de instruções é executado repetidamente.



A ação é executada repetidamente, enquanto o resultado for verdadeiro. O teste precede a ação.



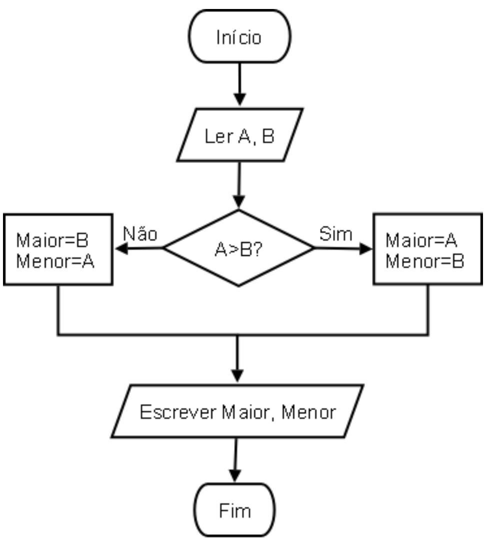
A ação é executada repetidamente, enquanto o resultado for verdadeiro. A ação precede o teste.



À partida é indicado o número de vezes que a ação é repetida. O teste precede a ação.

Exemplo 2: Ler dois números e escrevê-los por ordem decrescente.

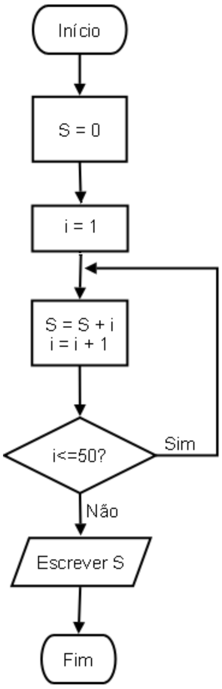
Fluxograma



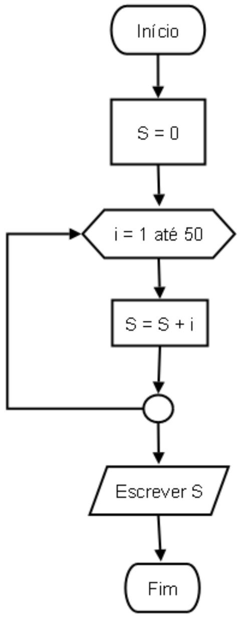
Pseudo-código

Ler A,B
Se A<B
    Maior=B
    Menor=A
Senão
    Maior=A
    Menor=B
Escrever Maior, Menor

Exemplo 3: Calcular $S = \sum_{i=1}^{50} i$



ou



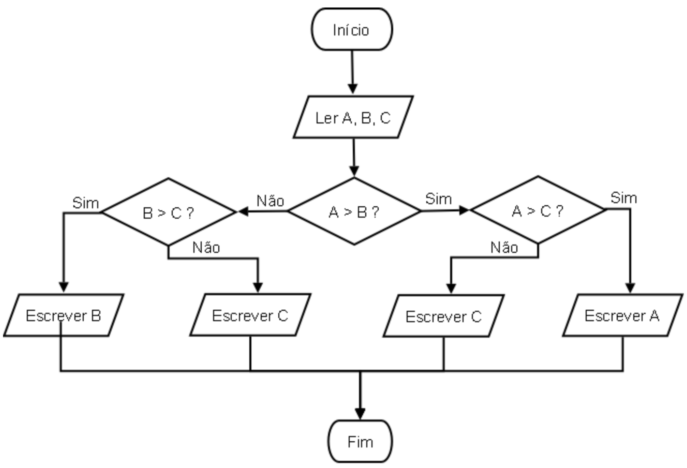
S=0
i=1
Repetir
    S=S+i
    i=i+1
enquanto i $\leq$ 50
Escrever S

ou

S=0
Para i=1 **até** 50
    S=S+i
Escrever S

Obs: Como se pode escrever o algoritmo com o teste a preceder a ação - *teste à cabeça*?

Exemplo 4: Dados três números A, B e C, determinar o maior.



```
Ler A,B, C
Se A>B
    Se A>C
        Escrever A
    Senão
        Escrever C
Senão, se B>C
    Escrever B
Senão
    Escrever C
```

5. Exercícios

⇛ Aulas laboratoriais

Exercício 1. Indique o resultado do seguinte algoritmo nos casos $a = 4$, $a = 3$ e $a = 0$.

Ler a
Se $a > 3$ ou $a < 0$
  $t = 0$
Senão se $a > 2$
  $t = 1$
Senão
  $t = 2$
Escrever t

Exercício 2. Indique o resultado do seguinte algoritmo.

$n = 2$
$p = 1$
Enquanto $n^2 < 25$
  $p = p(n - 1)$
  $n = n + 1$
Escrever p

Exercício 3. Diga qual o objetivo dos seguintes algoritmos:

Ler a, b
Se $a < b$
  $x = a$
  $a = b$
  $b = x$
Escrever a, b

$f = 1$
Para $n = 1$ **até** 12
  $f = f \times n$
Escrever f

$f = 1$
$n = 1$
Enquanto $f \leq 12$
  $n = n + 1$
  $f = f \times n$
Escrever f

Exercício 4. Considere o seguinte algoritmo em pseudo-código:

Ler um número natural N
i=1
Enquanto $i \times i < N$
  $i = i + 1$
Se $i \times i = N$
  Escrever "Sim"
Senão
  Escrever "Não"

a) Qual o resultado do algoritmo anterior para $N = 15,\ 16,\ 24,\ 25$?

b) Que propriedade de N está este algoritmo a testar?

Exercício 5. Escreva um algoritmo para, dado n, calcular:

a) $P = \prod_{i=1}^{n} i$ *b)* $P = \prod_{i=0}^{n} (2i + 1)$

Em ambos os casos, indique o significado de P.

Exercício 6. Qual o objetivo do seguinte algoritmo?

Ler um número natural N
$M = 0$
Enquanto $N \neq 0$
  $M = 10M + \text{mod}(N, 10)$
  $N = \text{int}(N/10)$
Escrever M

Exercício 7. Relembre que um número natural n é **triangular** se $n = 1 + 2 + \cdots + k$, para algum $k \in \mathbb{N}$ (ver Exercício suplementar 7 do Capítulo 1). O algoritmo seguinte pretendia verificar se um dado número n é triangular, mas tem um erro. Detete-o e corrija-o.

Ler um número natural N
$i = 1$
$S = 0$
Enquanto $S < N$
    $i = i + 1$
    $S = S + i$
Se $S = N$
    Escrever "O número é triangular"
Senão
    Escrever "O número não é triangular"

Exercício 8. Dados a, b e c, escreva um algoritmo para determinar as raízes reais da equação ax^2+bx+c=0.

Exercício 9. No calendário gregoriano, um ano é bissexto se é divisível por 400 ou se o ano é divisível por 4 mas não por 100. Escreva um algoritmo para determinar se um dado ano é bissexto.

Exercício 10. Escreva um algoritmo para determinar os divisores de um número.

Exercício 11. Escreva um algoritmo em pseudo-código para determinar os primeiros k números primos (pode usar a função **isprime** do MATLAB).

⇛ Exercícios suplementares

Exercício 1. Considere o seguinte algoritmo

Ler um número inteiro n
Se $0 < n < 1000$ ou $n \geq 10000$
    Escrever -n
Senão se $n > 0$
    $a =$ rem$(n, 100)$
    $b = (n - a)/100$
    Escrever b,a
Senão
    Escrever n

a) Qual o resultado deste algoritmo nos casos $n = 2345$, $n = 234$ e $n = -234$?

b) Qual é o objetivo deste algoritmo?

Exercício 2.* Escreva um algoritmo para obter um array com os dígitos que constituem um dado número natural.

Exercício 3.* Escreva um algoritmo para verificar se um dado número é primo (sem usar a função `isprime`).

Exercício 4.*

a) Escreva um algoritmo que permita obter o menor número natural n para o qual n^2-n+41 não é um número primo. (Pode usar a função `isprime`).

b) Use o Matlab para verificar que $n = 41$.

Exercício 5. O número 3025 pode ser escrito como $(30 + 25)^2$.

a) Escreva um algoritmo que permita obter todos os números de 4 algarismos que têm esta mesma característica.

*b)** Use funções predefinidas do Matlab para responder à questão da alínea anterior.

⇛ Solução dos exercícios selecionados

➟ Exercício 2

back

Ler um número natural N
digitos=[ ]
Enquanto $N \neq 0$
    digitos=[rem(N,10) digitos]
    $N =$ int$(N/10)$
Escrever digitos

➟ Exercício 3

back

Ler um número natural $N >= 2$
k=2;
primo=1;
Enquanto $k^2 \leq N$ e primo=1
    Se rem$(N, k) = 0$
        primo=0
    Senão
        k=k+1
Se primo=0
    Escrever "o número não é primo"
Senão
    Escrever "o número é primo"

➟ Exercício 4

back

$n = 1;$
Enquanto isprime($n^2 - n + 41$)=1
  $n = n + 1$
Escrever n

```
n=1:41;
find(~isprime(n.^2-n+41))
```

```
ans =
  41
```

➟ Exercício 5

back

```
n=1000:9999;
a=rem(n,100);
b=(n-a)/100;
n(n==(a+b).^2)
```

```
ans = 1x3
2025      3025      9801
```

Programar em Matlab

1. Ficheiros-M

As instruções em MATLAB são geralmente dadas e executadas linha a linha. É, no entanto, possível executar uma sequência de comandos que está guardada num ficheiro. Ficheiros que contêm código em linguagem MATLAB são chamados *ficheiros-M* (em inglês, *M-files*) e são do tipo **nome_ficheiro.m**.

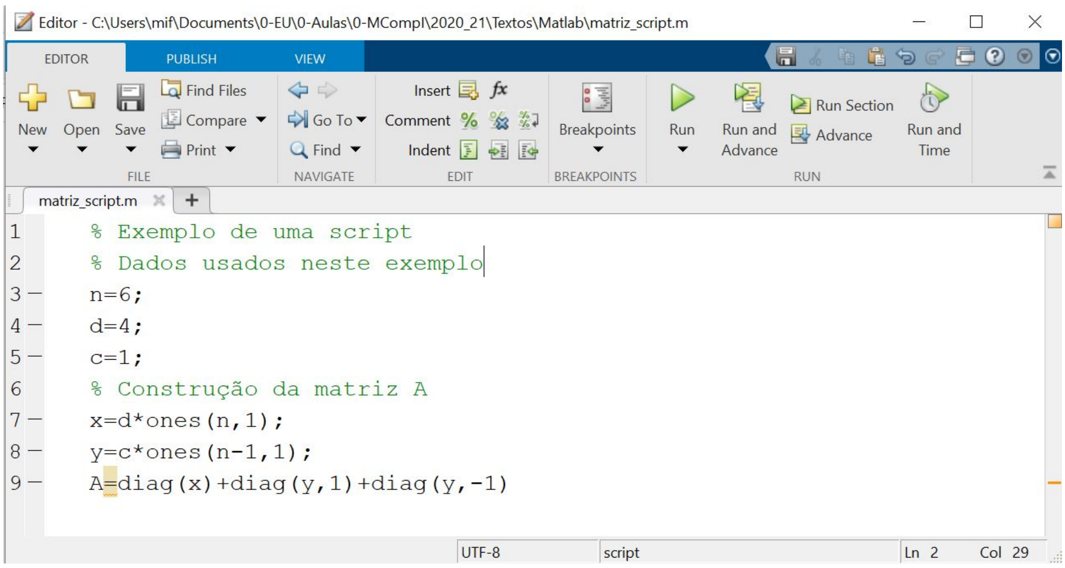
Um ficheiro-M consiste numa sequência de comandos MATLAB que pode inclusivamente fazer referência a outros ficheiros-M. Os ficheiros-M que vamos usar podem ser de dois tipos: *script* ou *function*.

⇛ *Scripts*

Para criar uma *script* basta selecionar, na barra de ferramentas, a opção **New** seguida de **Script** ou usar em alternativa a opção **New Script** da mesma barra de ferramentas; para editar uma *script* já existente deve usar-se a opção **Open**.

Quando uma *script* é invocada, o MATLAB executa os comandos encontrados no ficheiro e as variáveis que aparecem na *script* podem ser de novo usadas. Este tipo de ficheiro é muito útil quando há necessidade de executar um grande número de operações.

Suponhamos, por exemplo, que se pretende gerar uma matriz tridiagonal de ordem 6×6 que tem o valor 4 ao longo da diagonal principal e o valor 1 nas diagonais abaixo e acima da diagonal principal. Para o efeito, na barra de ferramentas do MATLAB, selecione a opção **New Script**, para criar um ficheiro de texto chamado, por exemplo, *matriz_script.m* com a seguinte sequência de instruções:



Para executar esta *script* basta selecionar o botão **Run** do menu *Editor* ou escrever, na janela do MATLAB

```
>> matriz_script
```

obtendo-se

```
A =
     4     1     0     0     0     0
     1     4     1     0     0     0
     0     1     4     1     0     0
     0     0     1     4     1     0
     0     0     0     1     4     1
     0     0     0     0     1     4
```

A matriz A e os vetores x e y estão disponíveis e podem ser de novo utilizados; por exemplo, se fizer

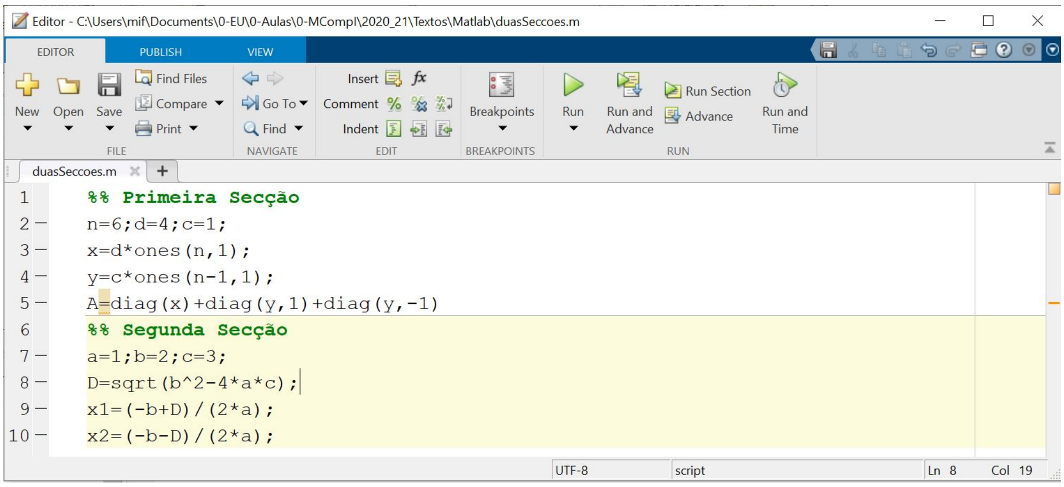
```
>> sum(x)-sum(y)
```

obterá

```
ans =
    19
```

Naturalmente que se quisermos alterar os dados deste problema, terá que ser editado o ficheiro *matriz_script.m* para alteração dos valores. Uma alternativa mais elegante e eficaz de alterar os valores dos parâmetros sem ter que alterar o código envolve as chamadas *functions* que a seguir se descrevem.

É ainda possível dividir uma *script* em várias secções que podem ser executadas separadamente. Para isso deve usar-se %% para separar as secções e o botão **Run Section** para executar cada secção. Qual o valor de `x1` e `x2` se executar a segunda secção da seguinte script?



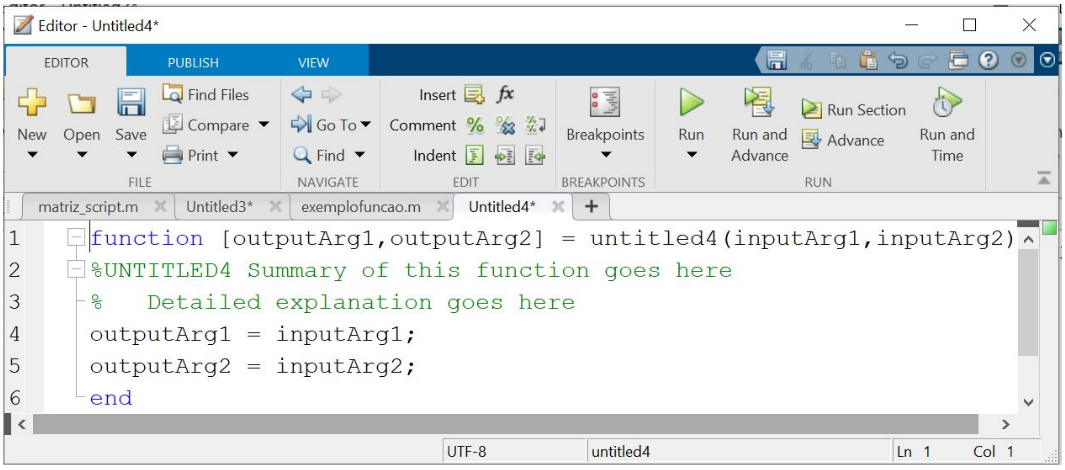
⇛ *Functions*

Um ficheiro-M que contém a palavra **function** no início da primeira linha é chamado uma função (*function*). As funções diferem das *scripts* na medida em que podem ser usados argumentos e as variáveis definidas e manipuladas dentro de uma função são locais, i.e. não operam globalmente no espaço de trabalho.

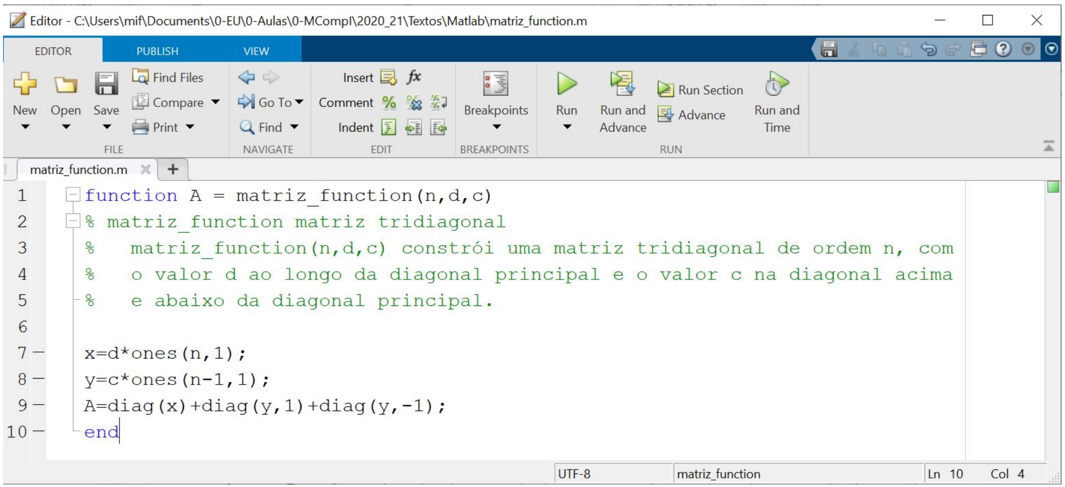
Uma *function* é definida do seguinte modo:

function *parâmetros_saida=nome_função(parâmetros_entrada)*

Quando se seleciona a opção **Function** do menu **New**, é criado automaticamente um ficheiro com o seguinte aspeto.



Para resolver o problema anteriormente considerado da construção de uma matriz tridiagonal, pode criar-se a função *matriz_function.m*, alterando convenientemente o ficheiro anterior (o ficheiro deve ser **sempre** gravado com o nome "nome_função" atrás referido.)



A função **matriz_function** passa a fazer parte das funções do MATLAB. As primeiras linhas de comentário que aparecem na função podem ser acedidas via *help*.

```
>> help matriz_function
  matriz_function matriz tridiagonal
    matriz_function(n,d,c) constrói uma matriz tridiagonal de ordem n, com
    o valor d ao longo da diagonal principal e o valor c na diagonal acima
    e abaixo da diagonal principal.
```

Para construir a matriz `A` definida anteriormente basta fazer

```
>> matriz_function(6,4,1)
ans =
     4     1     0     0     0     0
     1     4     1     0     0     0
     0     1     4     1     0     0
     0     0     1     4     1     0
     0     0     0     1     4     1
     0     0     0     0     1     4
```

Neste caso as variáveis x e y não estão definidas no espaço de trabalho.

```
>> x
Unrecognized function or variable 'x'.

>> y
Unrecognized function or variable 'x'.
```

O caso de funções com mais de um parâmetro de entrada ou saída é tratado de modo análogo. A função seguinte calcula as raízes de um polinómio do segundo grau.

```
function [x1,x2]=raizes(a,b,c)
% raizes Calcula as raizes  da equação ax^2+bx+c=0,
D=sqrt(b*b-4*a*c);
x1=(-b+D)/(2*a);
x2=(-b-D)/(2*a);
```

Listagem 1. Função com dois parâmetros

Para resolver, por exemplo, a equação $x^2 + x + 2 = 0$ basta fazer

```
>> [r1,r2]=raizes(1,1,2)
```

obtendo-se, então:

```
r1 =
  -0.5000 + 1.3229i

r2 =
  -0.5000 - 1.3229i
```

2. Instruções de *Input* e *Output*

É possível introduzir um texto ou um valor numérico através do teclado, de uma forma interativa. Para tal, pode usar-se a função **input** cuja forma é:

variavel_numerica=**input**(*'texto que vai aparecer'*)

ou

variavel_string=**input**(*'texto que vai aparecer','s'*).

No primeiro caso, aparece no ecrã o texto *texto que vai aparecer* e o utilizador deve introduzir um valor numérico que será atribuído à variável *variavel_numerica*. O segundo caso é usado quando se pretende introduzir um valor não numérico.

Como exemplo da utilização desta função, considerem-se as instruções

```
>> vn=input ('introduza o valor de vn')
```

e

```
>> vs=input ('Pretende continuar? (SIM/NÃO)','s')
```

Se for introduzido o valor 10 no primeiro caso e SIM no segundo, então as instruções anteriores originam `vn=10` e `vs= SIM`. Para criar uma *prompt* de várias linhas, use '\n' para indicar cada nova linha; por exemplo:

```
>>vs=input ('Pretende continuar?\n Responder SIM ou NÃO\n','s')
```

A função **disp** permite escrever um texto ou valor no ecrã. Assim, o comando

```
>> vs=disp(A)
```

escreve a matriz A no ecrã, enquanto que

```
>> disp('Estou a escrever esta frase' )
```

escreverá o texto *Estou a escrever esta frase*.

Para combinar texto e valores numéricos devem converter-se estes últimos a "texto" através da função **num2str**. O comando

```
>> disp(['Estou a escrever esta frase',' há ',num2str(n),' minutos'] )
```

escreverá no ecrã, no caso $n = 10$, a frase

```
Estou a escrever esta frase  há 10 minutos
```

Para outros formatos de entrada/saída, invoque o **help** do MATLAB para conhecer melhor o uso das instruções **fprintf**, **sprintf**, **fwrite**, **fscanf**, etc.

3. Instruções de controle

Como já foi referido, normalmente as instruções em MATLAB são executadas sequencialmente, isto é, depois de uma instrução ter sido executada, é executada a instrução imediatamente a seguir. As seguintes transferências de controle podem ser usadas para alterar a sequência de execução das instruções de uma *script* ou função: ciclos **for** e **while**, instruções **if** e **switch** e ainda **break**, **error** e **return**. Estas instruções são semelhantes às encontradas na maioria das linguagens de programação.

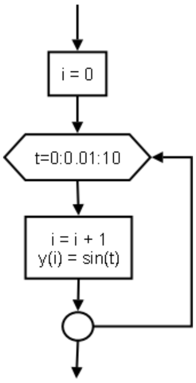
⇛ Ciclos for

A instrução **for** permite que um grupo de instruções seja executado repetidas vezes. Tem a forma

for *variavel=expressão*
    instruções
end

onde *expressão* é usualmente da forma **m:n** ou **m:i:n**, sendo **m** o valor inicial, **i** o valor incremental e **n** o valor final da *variavel*.

Suponhamos que pretendemos calcular um vetor com o valor da função seno em 1001 pontos igualmente espaçados do intervalo $[0, 10]$. A maneira mais óbvia de obter este vetor é:




```
i=0
for t=0:.01:10
    i=i+1;
    y(i)=sin(t);
end
```

Listagem 2. Ciclos for

Nota: O tempo de execução de programas em MATLAB pode ser substancialmente reduzido, se se tiver a preocupação de "vetorizar" os algoritmos. Vetorizar significa converter ciclos em operações com vetores ou matrizes. Por exemplo as instruções,

```
>> t=0:.1:10;
>> y=sin(t);
```

correspondem a uma versão vetorizada deste exemplo.
Note-se que `t=0:.01:10` é apenas um vetor linha. De facto, qualquer vetor linha pode ser usado num ciclo **for**. Por exemplo, o seguinte conjunto de instruções

```
x=1:100;soma=0;
for j=find(isprime(x))
    soma=soma+x(j);
end
```

Listagem 3. Ciclos for - outro exemplo

calcula a soma de todos os números primos inferiores a 100.

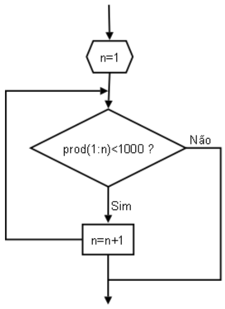
⇛ Ciclos while

A instrução **while** permite que um grupo de instruções seja executado um número indefinido de vezes, enquanto uma condição for satisfeita. Tem a forma

while *expressão*
    instruções
end

onde *expressão* é uma expressão de relação da forma *e1Re2*, sendo *e1* e *e2* expressões aritméticas e *R* um dos operadores de relação já definidos na Secção 1.4.

O exemplo seguinte ilustra o uso de um ciclo **while**, onde se pretende calcular o primeiro inteiro n para o qual $n!$ é um número com 4 dígitos.




```
n=1;
while prod(1:n)< 1000
  n=n+1;
end
n
```

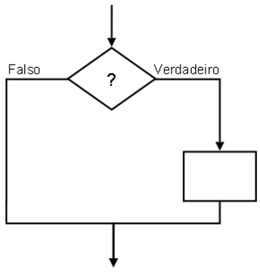
Listagem 4. Ciclos **while**

⇛ Instruções if – elseif – else

Os blocos **if** permitem alterar a ordem de execução de uma sequência de instruções, se determinada condição for (ou não) satisfeita.

➠ if · · · end

if *expressão_logica*
  instruções
  avaliadas, se expressão_logica verdadeira
end

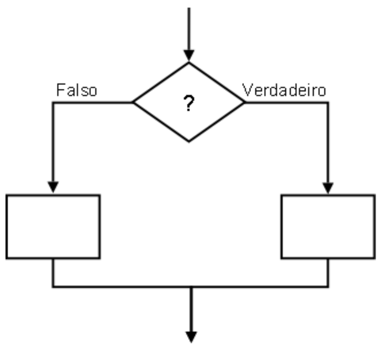



```
if a>0
   b=a;
   disp('a é positivo');
end
```

Listagem 5. Uso de **if – end**

➠ if · · · else · · · end

if *expressão_logica*
  instruções
  avaliadas, se expressão_logica verdadeira
else
  instruções
  avaliadas, se expressão_logica falsa
end



```
if T>37
   febre=1;
   disp('Com febre');
else
   febre=0;
   disp('Sem febre');
end
```

Listagem 6. Uso de **if** – **else** – **end**

➠ if ··· elseif ··· else ··· end

if *expressão_logica1*

instruções

avaliadas, se expressão_logica1 verdadeira

elseif *expressão_logica2*

instruções

avaliadas, se expressão_logica2 verdadeira

elseif *expressão_logica3*

instruções

avaliadas, se expressão_logica3 verdadeira

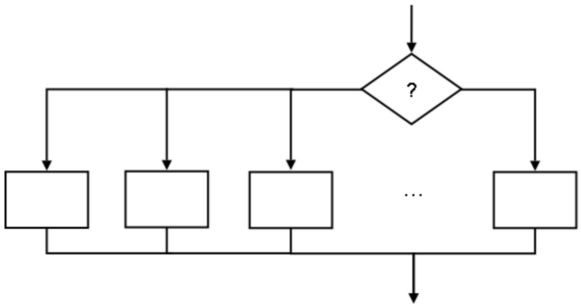
...

else

instruções

avaliadas, se nenhuma das expressões anteriores é verdadeira

end



Cada uma das expressões *expressao_logica1*, *expressao_logica2*, etc é uma expressão de relação da forma *e1Re2*, sendo *e1* e *e2* expressões aritméticas e *R* um dos operadores de relação definidos anteriormente. Podem também combinar-se estes operadores de relação com os operadores lógicos ou ainda usar-se funções lógicas.

```
if altura>190
   disp('Muito alto');
elseif altura>170
   disp('Alto');
elseif altura<150
   disp('Baixo');
else
   disp('Mediano');
end
```

Listagem 7. Uso de **if** – **elseif** – **else** – **end**

⇛ Instruções switch e case

A instrução **switch** executa um grupo de instruções, dependendo do valor de uma variável ou expressão. Tem a forma

switch *expressao*
  case *caso1*
    instruções
  case {*caso2a,caso2b, ...*}
    instruções
    ...
  otherwise
    instruções
end

Se $n = 23$, qual será o resultado da seguinte *script*?

```
switch rem(n,4)
case 0
  a=ones(n)
case {1,2}
  a=eye(n)
otherwise
  a=zeros(n)
end
```

Listagem 8. Uso de **switch** – **case** – **otherwise**

⇛ Instruções break, continue, error e return

Além das instruções de controle já definidas, o MATLAB dispõe de quatro comandos - **break**, **continue**, **error** e **return**, para controlar a execução de *scripts* e funções. Uma breve descrição destas instruções é feita de seguida.

A instrução **break** permite sair de um ciclo **for** ou **while**. A instrução **continue** pode ser usada em ciclos **for** ou **while**. Quando o MATLAB encontra a instrução **continue** dentro de um ciclo, passa imediatamente para a instrução final desse ciclo, ignorando todas as instruções entre **continue** e a declaração **end**.
Um exemplo trivial da utilização destas instruções encontra-se na *script* apresentada na Listagem 9. Esta *script* escreve no ecrã os números entre 5 e 10.

```
for i=1:20
   if i<5
      continue
   elseif i>10
      break
   end
   disp(i);
end
```

Listagem 9. Uso de **break** e **continue**

O comando

error ('*mensagem_de_erro*')

dentro de uma função ou *script*, aborta a execução, exibe a mensagem de erro *mensagem_de_erro* e faz regressar o controle ao teclado.

O comando **return** faz regressar o controle à função invocadora ou ao teclado. A Listagem 10 ilustra o uso das instruções **error** e **return**. Neste caso, pretende-se apenas obter os zeros reais de um polinómio do segundo grau, cujos coeficientes devem ser indicados num vetor.

```
function [x1,x2]=zeros_p(p)
if length(p) ~= 3
   error('p deve ser um polinómio de grau 2');
end
delta=p(2)*p(2)-4*p(1)*p(3);
if delta<0
   disp('Este polinómio não tem zeros reais');
   return
else
   x1=(-p(2)+sqrt(delta))/(2*p(1));
   x2=(-p(2)-sqrt(delta))/(2*p(1));
end
disp('Este polinómio tem zeros reais');
```

Listagem 10. Uso de **error** e **return**

4. Funções de novo!

⇛ Variáveis locais e globais

Quando uma variável é definida na janela de comandos do MATLAB, esta fica automaticamente gravada no espaço de trabalho *MATLAB Workspace*. O mesmo se passa se a variável for definida numa *script*, uma vez que uma *script* é apenas um conjunto de comandos. Tal não acontece com as funções. Na verdade, as funções não partilham o mesmo espaço de trabalho. Isto significa que as variáveis definidas numa função são variáveis **locais** ao espaço de trabalho da função. Cada função tem o seu próprio espaço de trabalho temporário, o qual é criado a cada chamada e apagado quando a função completa a execução.

Ocasionalmente, pode ser necessário definir variáveis que existam em mais que um espaço de trabalho, incluindo eventualmente o próprio *MATLAB Workspace*. Estas variáveis devem ser então declaradas como **globais**. Para isso, todos os "locais" que precisem de partilhar essa variável (funções, *scripts* ou janela de comandos) devem ter uma linha inicial que identifique as variáveis em causa como globais, usando o comando **global**. Por exemplo, o uso da instrução

```
>> global VARIAVEL_GLOBAL
```

permite a partilha da variável VARIAVEL_GLOBAL.
As variáveis globais podem ser apagadas fazendo

```
>> clear global
```

A função **isglobal** pode ser usada para verificar se uma dada variável é ou não global. Uma lista completa de todas as variáveis globais pode ser acedida via

```
>> who global
```

Na prática de programação, o uso de variáveis globais não é encorajado. Todavia se estas forem usadas, aconselha-se que o nome destas variáveis seja longo e escrito em letra maiúscula (para evitar possíveis conflitos com outras variáveis globais).

⇛ Subfunções

Um ficheiro-M pode conter código para definir uma ou mais funções. A primeira função que aparece e que tem o mesmo nome do ficheiro-M é chamada *função principal*. As restantes funções são chamadas *subfunções* e são visíveis apenas para a função principal e as outras subfunções do ficheiro-M.
As subfunções são definidas de modo análogo às funções e podem aparecer por qualquer ordem no ficheiro-M que as contém, desde que a função principal seja a primeira.
Geralmente, as subfunções realizam tarefas que precisam de ser executadas separadamente da função principal, mas que, em princípio, terão pouco interesse fora do âmbito em que foram criadas. A técnica de usar subfunções permite evitar a proliferação de ficheiros-M.

No exemplo da Listagem 11, foi criado um ficheiro-M chamado *exemplo_subfuncoes.m* que define três funções: a função principal, chamada **exemplo_subfuncoes** e as duas subfunções **mean** e **median**. Usamos propositadamente **mean** e **median** como nomes para as subfunções, os quais são os nomes de duas funções internas do MATLAB. Não há qualquer conflito neste caso, uma vez que quando estas funções são invocadas no ficheiro-M *exemplo_subfuncoes.m*, o MATLAB verifica primeiro se há alguma subfunção desta função com esses nomes e, em caso afirmativo, são estas as funções avaliadas. De facto, fazendo

```
>> x=[1 5 2 -3 8 9];
>> exemplo_subfuncoes(x)
```

obtém-se

```
Função invocada:subfunção mean da função exemplo_subfuncoes
Função invocada:subfunção median da função exemplo_subfuncoes

ans =
3.6667     3.5000
```

enquanto

```
>> [mean(x), median(x)]
```

resulta em

```
ans =
3.6667    3.5000
```

```
function estatisticas=exemplo_subfuncoes(x)
% exemplo_subfuncoes calcula a média e a mediana de uma amostra
%
%  Este exemplo permite ilustrar o uso de subfunções
n=length(x);
estatisticas=[mean(x,n),median(x,n)];

function y=mean(x,n)
% mean calcula a média de um conjunto de valores
%      Não há conflito com a função interna MEAN do MATLAB
%      porque esta função é executada primeiro.
disp('Função invocada:subfunção mean da função exemplo_subfuncoes')
y=sum(x)/n;

function y=median(x,n)
% median calcula a mediana de um conjunto de valores
%        Não há conflito com a função interna MEDIAN do MATLAB
disp('Função invocada:subfunção median da função exemplo_subfuncoes')
xordenado=sort(x);
if rem(n,2)==1
% Se n é impar, a mediana é o elemento na posição (n+1)/2
y=xordenado((n+1)/2);
else
% Se n é par, a mediana é a media entre os elementos
% nas posições n/2 e n/2+1
y=(xordenado(n/2)+xordenado(n/2+1))/2;
end
```

Listagem 11. Funções e subfunções

O comando **help** permite ter acesso à ajuda da função principal. Pode também aceder-se às ajudas das subfunções, indicando o nome da função principal e da subfunção, como a seguir se exemplifica.

```
>> help exemplo_subfuncoes
  exemplo_subfuncoes calcula a média e a mediana de uma amostra

   Este exemplo permite ilustrar o uso de subfunções

>> help exemplo_subfuncoes>mean
  mean calcula a média de um conjunto de valores
       Nao há conflito com a função interna mean do MATLAB
       porque esta função é executada primeiro.
```

⇛ Funções anónimas

É possível definir funções, de forma simples, através de uma linha de comando, sem recorrer à criação (e consequente armazenamento em disco) de um ficheiro-M. Esta alternativa passa pela utilização das chamadas *funções anónimas*.

A sintaxe para criar uma função anónima é a seguinte:

$$\textit{f\_anonima} = @(\textit{argumentos})\ \textit{expressão}$$

Um exemplo muito simples de uma função anónima é:

```
>>g=@(x) x^2+2*x+3
```

Para avaliar esta função basta fazer

```
>> g(2)
ans =
    11
```

Nota: É boa prática, definir sempre uma função anónima de forma que possa operar sobre escalares, vetores ou matrizes. Se

```
>> f=@(x) x.^2+2*x+3
```

qual será o valor de `g(1:5)` e `f(1:5)`?

As funções anónimas podem também ter mais que um argumento. Por exemplo, a função $h(x,y) = (x^2+y^2, x+y)$ poderia ser definida e avaliada em x=2 e y=4 da seguinte forma

```
>> h=@(x,y) [x.^2+y.^2 x+y]
>> h(2,4)
```

Outra vantagem do uso de funções anónimas diz respeito à possibilidade de usar implicitamente variáveis definidas no espaço de trabalho, sem ter que as declarar como globais. Por exemplo:

```
>> peso1=0.4;peso2=0.5;peso3=0.1;
>> notafinal=@(T1,T2,TPC) peso1*T1+peso2*T2+peso3*TPC;
>> notafinal(8,11,14)
ans =
   10.1000
```

⇛ Quando uma função é um argumento

Muitas funções em MATLAB têm como argumentos outras funções. O MATLAB chama genericamente a estas funções **function functions** (faça `help funfun` para obter uma lista completa deste tipo de funções). Existem várias formas de passar uma função como argumento, dependendo da forma como esta foi escrita.
Uma forma é usar funções anónimas. Por exemplo, a função **fzero** permite encontrar um zero de uma função de uma variável. Podemos encontrar um zero de $f(x) = \sin(x)$ próximo de 3, fazendo

```
>> f=@(x) sin(x);
>> fzero(f,3)
ans =
    3.1416
```

é também possível passar a função `sin` como argumento, usando a sintaxe especial

```
>> fzero(@sin,3)
ans =
    3.1416
```

O nome `@sin` é chamado *function_handle* e é uma forma de referir abstratamente uma função, em vez de a invocar diretamente. *Function_handle* é um tipo de dado do MATLAB que contém toda a informação necessária para avaliar uma função. Uma função tipo *function_handle* pode também ser criada colocando o caracter **@** atrás do nome da função definida anteriormente através de um ficheiro-M.

Para obter informação sobre uma *function_handles* podem usar-se as funções **fun2str** ou **functions**.

```
>> func2str(f)
ans =
      '@(x)sin(x)'

>> functions(g)
ans =
     struct with fields:

           function: '@(x)x^2+2*x+3'
                 type: 'anonymous'
                 file: ''
            workspace: {[1x1 struct]}
     within_file_path: ''
```

Do ponto de vista da eficiência e versatilidade, o uso de funções do tipo *function_handles* deve ser encorajado, especialmente se estas forem passadas como argumento para outras funções.

⇛ Funções recursivas

As funções em MATLAB podem ser recursivas, isto é, podem chamar-se a si próprias. A recursividade é, de facto, uma ferramenta poderosa, mas nem sempre corresponde à melhor forma de programação.

Consideremos o problema de calcular o fatorial de um inteiro não negativo `n`, sem recorrer à função **factorial**. Este problema pode ser resolvido de várias formas.

Versão básica	Versão à MATLAB	Versão recursiva
`function f=fat1(n)`	`function f=fat2(n)`	`function f=fat3(n)`
`f=1;`	`f=prod(1:n);`	`if n==0`
`for i=2:n`	`end`	`    f=1;`
`   f=f*i;`		`else`
`end`		`    f=n*fat3(n-1);`
`end`		`end`
		`end`

5. Gráficos - 2D

O MATLAB dispõe de um grande número de facilidades gráficas; centraremos a nossa atenção em gráficos básicos 2-D. Todos os pormenores relativos às ferramentas de visualização incluídas nesta versão do MATLAB podem ser obtidos na página da Mathworks ou invocando diretamente o **help** para `graph2d`, `graph3d`, `specgraph` ou `graphics`.
O comando mais simples e, talvez o mais útil para produzir gráficos 2-D tem a forma

plot(*Abcissas,Ordenadas,'estilo'*)

onde *Abcissas* e *Ordenadas* são vetores (com a mesma dimensão) que contêm as abcissas e ordenadas de pontos do gráfico e *estilo* é um argumento opcional que especifica o estilo da linha ou ponto a desenhar. Na tabela seguinte estão indicados alguns dos estilos que é possível definir.

cor		ponto		linha	
y	amarela	.	ponto(.)	$-$	contínua
m	magenta	**o**	círculo (o)	:	ponteada
c	cião	**x**	cruz(x)	$-.$	'traço-ponto'
r	vermelha	$+$	mais($+$)	$--$	tracejada
g	verde	$*$	estrela(*)		
b	azul	**s**	quadrado		
w	branca	**d**	losango($\diamondsuit$)		
k	preta	**v**	triângulo (∇)		

A função **plot** também permite o uso de apenas um vetor como argumento. Se Pontos$=[x_1\ x_2 \cdots x_n]$, o comando

plot(*Pontos*)

desenha os pontos de coordenadas $(i, x_i),\ i = 1, \cdots, n.$

Títulos, designação dos eixos, legendas e outras características, podem ser acrescentadas a um dado gráfico, usando as funções **title**, **xlabel**, **ylabel**, **grid**, **text** **legend**,etc. Estas funções têm a forma seguinte.

Designação	Descrição
title('*título*')	produz um *título* na parte superior do gráfico
xlabel('*nome_x*')	o eixo dos xx é designado por *nome_x*
ylabel('*nome_y*')	o eixo dos yy é designado por *nome_y*
grid	coloca uma quadrícula no gráfico
text(x,y,'*texto_em_x_y*')	escreve o texto *texto_em_x_y* na posição (x, y)
gtext('*texto*')	permite colocar *texto* numa posição a indicar com o rato
legend('*texto1*','*texto2*')	produz uma legenda com *texto1* e *texto2*
legend off	retira legenda

É também possível controlar os limites dos eixos através do comando **axis**. A função **axis** tem várias opções que permitem personalizar os limites, a escala, a orientação, etc, de um gráfico. Escrevendo

axis([*xmin xmax ymin ymax*])

os limites passam a ser *xmin* e *xmax* para o eixo dos xx e *ymin* e *ymax* para o eixo dos yy. Este comando deve aparecer depois do comando **plot**.

A função **axis** também aceita palavras chave para controlar os eixos. Por exemplo, **axis square**, **axis equal**, **axis auto**, **axis on**, etc. (Faça **help axis**).

Numa mesma figura podem sobrepor-se vários gráficos, recorrendo ao comando **hold**. As instruções

hold on
  plot $(x_1, y_1,$ *'estilo 1'*)
  plot $(x_2, y_2,$ *'estilo 2'*)
  plot $(x_3, y_3,$ *'estilo 3'*)
hold off

originam a sobreposição de três gráficos. Este objetivo também pode ser atingido, usando

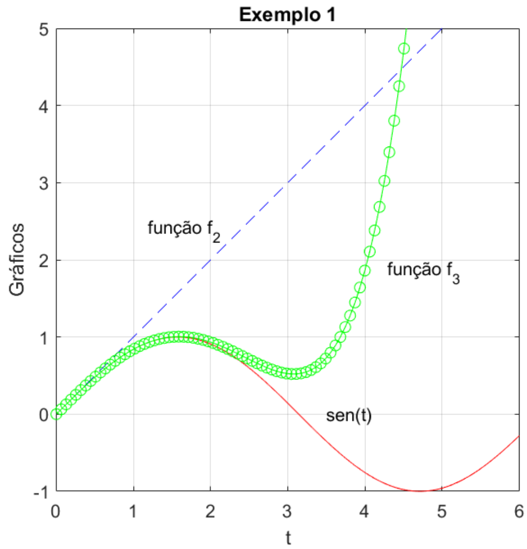
plot $(x_1, y_1,$ *'estilo 1'*$, x_2, y_2,$ *'estilo 2'*$, x_3, y_3,$ *'estilo 3'*)

O comando **hold** é especialmente útil quando o conjunto de pontos a desenhar não está disponível todo ao mesmo tempo.

Na figura seguinte estão representados simultaneamente os gráficos das funções

$$f_1(t) = \operatorname{sen} t, \quad f_2(t) = t, \quad f_3(t) = t - \frac{t^3}{3!} + \frac{t^5}{5!}.$$

Estes gráficos foram obtidos recorrendo à *script* apresentada na Listagem 13, onde foram usadas várias opções da instrução **plot**.




```
t=linspace(0,2*pi);
y1=sin(t);
y2=t;
y3=t-(t.^3)/6+(t.^5)/120;
plot(t,y1,'r',t,y2,'b--',t,y3,'go-')
axis equal
axis([0 6 -1 5])
grid
xlabel('t'), ylabel('Gráficos')
title('Exemplo 1')
text(3.5,0,'sen(t)')
gtext('função f_2'),gtext('função f_3')
```

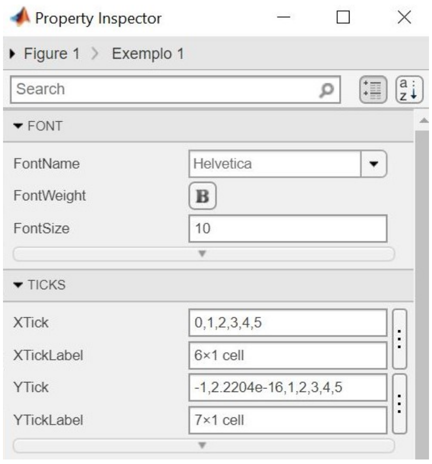
Listagem 13. A função **plot**

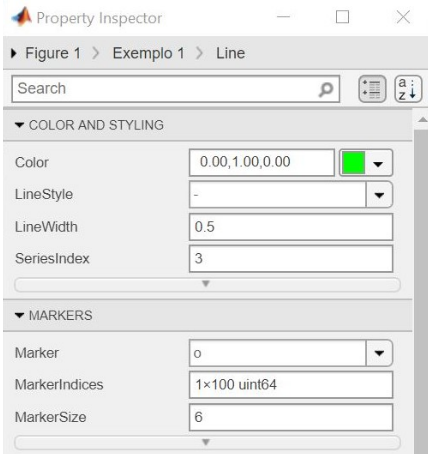
Alternativamente poder-se-ia obter o gráfico anterior, recorrendo à função **fplot**. Na listagem seguinte apresenta-se essa solução.

```
hold on
fplot(@(t)sin(t),[0,2*pi],'r')
fplot(@(t)t,[0,2*pi],'b--')
fplot(@(t) t-(t.^3)/6+(t.^5)/120,[0,2*pi],'go-')
.
.
.
hold off
```

Listagem 14. A função **fplot**

O MATLAB contém uma *interface* gráfica que permite criar e editar gráficos sem usar código. Para isso na janela da figura, selecione o botão *Open Property Inspector* ; irá aparecer uma janela com o aspeto da figura abaixo (lado esquerdo), onde poderá fazer as alterações. Selecionando um elemento do gráfico que se pretende alterar, aparecerá o menu correspondente. Por exemplo selecionando a linda correspondente à função f_3 irá ter acesso a um menu do tipo ilustrado no lado direita da figura.





6. Exercícios

⇛ Aulas laboratoriais

Exercício 1. Considere as seguintes funções MATLAB

```
function y=f1(x)
perimetro=2*pi*r;
area=pi*r^2;
end
```

```
function y=f2(r)
perimetro=2*pi*r
area=pi*r^2
end
```

```
function area=f3(r)
perimetro=2*pi*r;
area=pi*r^2;
end
```

```
function [area,perimetro]=f4(r)
perimetro=2*pi*r;
area=pi*r^2;
end
```

```
function [perimetro,area]=f5(r)
perimetro=2*pi*r;
area=pi*r^2;
end
```

```
function [area,perimetro]=f6(r)
perimetro=2*pi*r;
area=pi*r.^2;
end
```

a) se $r = 1$, qual o resultado das seguintes instruções (use o MATLAB apenas para verificar as suas respostas)?

```
>> f1(r)     >> area=f3(1)     >> [x,y]=f3(1)
>> f2(r)     >> area=f4(1)     >> [x,y]=f4(1)
>> f3(r)     >> area=f5(1)     >> [area,perimetro]=f5(1)
```

b) se $r = [1, 2]$, qual o resultado das seguintes instruções?

```
>> [area,perimetro]=f5(r)
>> [area,perimetro]=f6(r)
```

Exercício 2. Escreva uma *function* para avaliar as seguintes funções:

a) $f(x) = x^2 + 3x - 2$, em $x = 2$ e $x = [1\ 2]$

b) $f(x, y) = (x^2 + y, x + 2y)$, em $x = -2, y = 4$.

Exercício 3. Escreva uma *function* `int = int_aleat(a,b)` para gerar inteiros aleatórios entre a e b (inclusivé), sem recorrer à função **randi**.

Exercício 4. Escreva uma *function* `cb = coeff_binom(n,r)` para calcular os coeficientes binomiais ${}^nC_r$, sem recorrer à função **nchoosek**.

Exercício 5. Escreva uma *function* `[elems, mns] = nonzero(A)` que retorna em `elems` todos os elementos não nulos de uma dada matriz `A` e em `mns` a média de todas as colunas de `A`.

Exercício 6. Escreva uma função **isinteiro** que devolve 1 se n é um número inteiro e 0, nos outros casos. Teste a sua função e compare os resultados com os obtidos usando a função predefinida **isinteger**.

Exercício 7. Escreva uma função `m=reverso(n)` que, dado um número natural, calcule o número reverso, isto é, o número com os algarismos na ordem contrária à original.

Exercício 8. Use a função anterior para escrever uma função **iscapicua** tal que `iscapicua(n)` é 1 se n é uma capicua e 0, nos outros casos.

Exercício 9. Execute e comente as seguintes *scripts*.

a)
```
graus=input('Graus? ');
rads=graus/180*pi;
disp([num2str(graus),' graus, corresponde a ',num2str(rads), ' radianos'])
```

b)
```
x=input('x? ');
y(x>0)=1;
y(x<=0)=-1;
disp(y)
```

c)
```
x=1:5;
y=sqrt(x);
fprintf('%3i',x)
fprintf('%3i\n',x)
fprintf('%12.8f\n',x)
fprintf('%12.8f\n',y)
fprintf('%3i  %12.8f\n',[x;y]);
fprintf('  x       sqrt(x) \n'),fprintf(' %3i  %12.8f\n',[x;y]);
fprintf(' %3i  %6.2e\n',[x;10*y]);
```

Exercício 10. Escreva uma *script* que produza o seguinte resultado:

```
1  Hello world
2  Hello world
3  Hello world
4  Hello world

1  Hello world    3  Hello world
2  Hello world    4  Hello world

1  Hello world    2  Hello world
3  Hello world    4  Hello world
```

Exercício 11. Faça uma tabela de valores da função $f(x) = 10\,\text{sen}(\frac{\pi x}{2})$ para $x = -1, -0.5, 0, 0.5, 1$. Essa tabela deverá ter um formato análogo ao seguinte:

```
  x         f(x)
-------------------
-1.0    -10.000000
-0.5     -7.071068
 ...
```

Exercício 12. As respostas a cada uma das seguintes questões devem ser dadas recorrendo ao MATLAB **apenas** para confirmação da resposta.

a) Diga o que aparece na janela de comandos do MATLAB se executar as instruções:

```
for i=0:12
   x=pi*i/6;
   disp([x,cos(x)])
end
```

b) Indique o valor da variável s, após executar a seguinte script:

```
x =2:6;
s=0;
for i=1:length (x)
   s=s+( -1)^i*x(i);
end
```

c) Indique o valor da variável x, após executar a seguinte script:

```
y=2;
for i =0:2:4
   y=[y,y*i];
end
x=y;
```

d) Indique o valor da variável p, após executar a seguinte script:

```
x=2:6;
p=1;
for i=1:length (x)
   p=p*x(i)/i;
end
```

e) Indique o valor da variável p, após executar a seguinte script:

```
n=2;p=1;
while n^2 < 25
   p=p*(n -1);
   n=n+1;
end;
```

f) Diga o que aparece na janela de comandos do MATLAB se executar as instruções:

```
n=0;f=1;y=1;
disp('   n         f           y')
while (f < 10000 & y < 10000)
fprintf('%4i %12.6f %12.6f \n',n,f,y)
n=n+1;
f=f*n;
y=exp(n);
end
```

Exercício 13. Em cada uma das seguintes questões, avalie o fragmento de código MATLAB apresentado, para cada um dos casos indicados. Use, de seguida, o MATLAB para confirmar as suas respostas.

a) (i) $n = 7$; (ii) $n = 0$; (iii) $n = -10$

```
if n > 1
   m = n+1
else
   m = n-1
end
```

b) (i) $x = -1$; (ii) $x = 5$; (iii) $x = 20$

```
if 0 < x < 10
   y = 4*x
else
   y = 500
end
```

c) (i) $z = 1$; (ii) $z = 9$; (iii) $z = 60$; (iv) $z = 200$

```
if z < 5
   w = 2*z
elseif z < 10
   w = 9 - z
elseif z < 100
   w = sqrt(z)
else
   w = z
end
```

d)
```
clc
n=input('Número mecanografico? ');
last=rem(n,10);
disp(' Próximo TPC:')
switch last
   case {4,5,9}
      disp(' ------------> Exercícios 3 e 4');
   case {3,7,8}
      disp(' ------------> Exercícios 3 e 6');
   case {2,6}
      disp(' ------------> Exercícios 5 e 7');
   otherwise
      disp(' ------------> Exercícios 6 e 7');
end
pause
dia=randi([15,31]);
mes='Dezembro';
fprintf('Data de entrega:  %2i  de %8s de 2020',dia,mes )
```

Exercício 14. Escreva uma função **outraHilb** que, dado o valor de n, construa a matriz de *Hilbert* H, de ordem n, cujos elementos $h_{i,j}$ são da forma $h_{i,j} = 1/(i+j-1)$. Compare com a função predefinida `hilb`.

Exercício 15. Dada a matriz A de ordem 4×5, escreva uma *script* para obter a soma de cada coluna de A, sem recorrer à função predefinida **sum**.

Exercício 16. Analise, execute e comente a seguinte *script*.

```
clear;
n=1:100000;
tic;
for i=n
   b(i)=i^(1/3);
end
t=toc;
disp(['Tempo de execução ciclo for: ',num2str(t),' segundos']);
tic;
b=n.^(1/3);
t=toc;
disp(['Tempo de execução versão vetorizada: ',num2str(t),' segundos']);
```

Exercício 17. Escreva um programa que, conhecido o dia da semana, indique quais as UCs com tipologia T que constam no horário do 1º ano, nesse dia.

Exercício 18. Faça uma tabela de valores da função

$$f(x) = \begin{cases} x^3, & \text{se } 0 \le x \le 10 \\ \sqrt[3]{x-10}, & \text{se } 10 < x \le 15 \\ 0, & \text{nos outros casos} \end{cases}$$

para todos os números pares $x \in [1, 20]$. Essa tabela deverá ter um formato análogo ao seguinte:

```
 x        f(x)
-----------------
 2      8.00e+00
 4      6.40e+01
```

Exercício 19. Escreva uma *script* que, dado um valor fornecido pelo utilizador da temperatura T_F em graus Fahrenheit, calcule a temperatura equivalente T_C em graus Celsius. (Relembre que $T_F = 1.8T_C + 32$). Esta *script* só deve terminar, quando nenhum valor for introduzido pelo utilizador. (A função **isempty** pode ser útil).

Exercício 20. Escreva um programa que leia uma letra e escreva "Vogal" ou "Consoante" conforme o tipo da letra lida. Se o caracter lido não for uma letra válida, então o programa deve escrever uma mensagem de erro. (As funções **isletter** e **lower** podem ser úteis).

Exercício 21. Determine o maior valor de n para o qual $\sqrt{1^3} + \sqrt{2^3} + \cdots + \sqrt{n^3}$ é menor que 1000.

Exercício 22. Escreva uma função `teste = lados(a,b,c)` cujos parâmetros de entrada são 3 números reais positivos `a, b` e `c`. Se existir um triângulo cujas medidas dos lados sejam esses números, a função deverá retornar o valor 1 (caso contrário, `teste=0`). Modifique esta função para permitir classificar o triângulo em *equilátero*, *isósceles* ou *escaleno* (no caso `teste=1`).

Exercício 23. Chama-se número harmónico a todo o número h_n que possa ser escrito como

$$h_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}, \quad n \in \mathbb{N}.$$

Escreva uma *script* que lhe permita, dado um determinado natural n, encontrar o valor de h_n.

Nota: O inteiro n deve ser pedido interativamente ao utilizador, através do uso do comando **input**.

Exercício 24. *a)* Escreva uma função

`s=somaprogr(r,n)`

para calcular a soma de uma progressão geométrica $1 + r + r^2 + \ldots + r^n$, para r e n variáveis. Teste essa função com os valores de $r = 0.5$ e $n = 10, 20, 100$ e 1000.

b) Faça `help nargin` para obter informação sobre a função pré-definida **nargin**. Utilize **nargin** para poder invocar a sua função apenas com um argumento de entrada, tomando, por defeito, $n = 20$.

Exercício 25. Há apenas quatro números naturais diferentes de 1 que podem ser escritos como a soma dos cubos dos seus algarismos. Sabendo que estes números se encontram entre 100 e 999, escreva uma *script* para os obter.

Exercício 26. Defina, usando funções anónimas, as seguintes funções:

a) $f(x) = x^2 + 2x - 3$

b) $g(x, y) = x^2 + y^2$

c) $h(x, y) = (x + y, x - y, 4)$

d) $r(x) = \begin{cases} 2x - 4, \text{ se } 0 \leq x < 4 \\ x^2 - 1, \text{ se } x \geq 4 \\ 2, \text{ nos outros casos} \end{cases}$

Teste as suas funções, avaliando-as em vários pontos.

Exercício 27. Os números de Fibonacci são calculados de acordo com a seguinte relação:

$$F_n = F_{n-1} + F_{n-2}, \text{ com } F_0 = F_1 = 1, \quad n = 2, 3, \ldots$$

a) Escreva uma função que, dado o valor de n, calcule o n-ésimo número de Fibonacci. Apresente uma solução recursiva e uma não recursiva.

b) Compare o tempo de execução das duas soluções, para obter os 40 primeiros números de Fibonacci.

c) Considere os rácios $r_n = \frac{F_n}{F_{n-1}}$. Calcule os primeiros 40 rácios.

d) Sabendo que $\lim r_n = \Phi$, onde $\Phi = \frac{1+\sqrt{5}}{2}$ é o famoso número de ouro, pode afirmar que os resultados obtidos ilustram esta propriedade?

Exercício 28. O *fatorial duplo* de um inteiro não negativo n é definido do seguinte modo:

$$n!! = \begin{cases} 1, & \text{se } n = 0 \\ n \times (n-2) \times \cdots \times 5 \times 3 \times 1, & \text{se } n > 0 \text{ ímpar} \\ n \times (n-2) \times \cdots \times 6 \times 4 \times 2, & \text{se } n > 0 \text{ par} \end{cases}$$

Escreva uma função recursiva para, dado o valor de n, calcular $n!!$.

Exercício 29. Considere a função $f(x) = \text{sen}(2\pi x)$.

a) Use a função **linspace** para obter uma tabela de valores da função f, em 100 pontos igualmente espaçados do intervalo $[0, 1]$. Use o comando **plot** para esboçar o gráfico da função.

b) Repita a alínea anterior, definindo uma função anónima e usando o comando **fplot**.

Exercício 30. A *script* seguinte desenha dois gráficos da função sen(x^3), no intervalo $[2,4]$, usando a função **plot** e **fplot**. Execute a *script* e interprete os resultados.

```
t=linspace(2,4,50);
y1=sin(t.^3);
subplot(2,1,1)
plot(t,y1,'b')
gtext('Uso da função plot');
subplot(2,1,2)
f=@(x) sin(x.^3);
fplot(f,[2,4],'g')
title('Uso da função fplot');
```

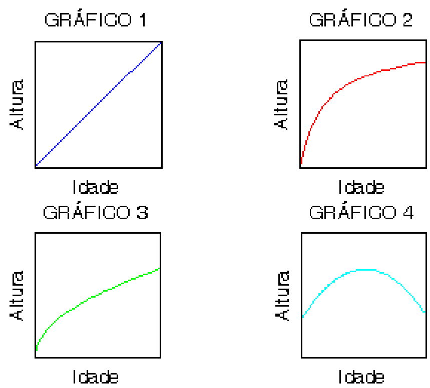
Exercício 31. Desenhe uma circunferência de centro no ponto (2,3) e raio 2, usando a função **plot**, a função **fplot** e a função **fimplicit**.

Exercício 32. Represente, num mesmo gráfico, as funções $f(x) = \sin(4x)$, $g(x) = x\cos(x)$ e $h(x) = (x+1)^{-1}\sqrt{x}$, no intervalo $[1,10]$, assinalando ainda o ponto $P = (4,5)$. Use cores e estilos diferentes para cada gráfico e escreva o texto 'ponto isolado' junto do ponto P. Altere depois os eixos, de forma a poder ter uma visão mais pormenorizada da função h.

Exercício 33. Complete a seguinte função

```
function meuplot(f,df,lim,ponto)
% meuplot desenha o gráfico da função f, no intervalo lim,
%         e da reta tangente ao gráfico de f no ponto de abcissa ponto
%
%         A equação da reta tangente é dada por
%         y-f(ponto)=df(ponto)*(x-ponto)
%
%         f e df podem ser especificadas como uma
%         função anónima ou através de uma function handle.
%
% Exemplo:
%          f=@(x) 1./(1+25*x.^2), df=@(x) -50*x./((1+25*x.^2)^2)
%          meuplot(f,df,[-1 1],0.5)
```

Exercício 34. Obtenha um gráfico análogo ao seguinte.



Exercício 35. Escreva uma função que, dados três pontos no plano p_1, p_2 e p_3, desenhe o triângulo com vértices nesses pontos. A função deverá enviar uma mensagem de erro, caso os pontos não definam um triângulo.

⇛ Exercícios suplementares

Exercício 1. Escreva uma *function* que permita avaliar a seguinte função.

$$f(x,y) = \begin{cases} \sqrt{x^2+1}, & \text{se } y > 0 \text{ e } x < 1 \\ \log xy, & \text{se } y > 0 \text{ e } x \geq 1 \\ x + 2y, & \text{nos restantes casos} \end{cases}$$

Exercício 2. Escreva uma função que receba como argumento um vetor de números inteiros `v` e retorne o par `[x y]`, onde `x` é o maior número par e `y` é o menor número ímpar do vetor.

Esta função deverá: (i) estar devidamente comentada; (ii) enviar uma mensagem de erro, caso o vetor `v` não seja de inteiros; (iii) estar preparada para lidar com a situação em que `v` não tem números pares ou ímpares.

Exercício 3. A área de um triângulo, conhecidas as medidas a, b e c dos seus lados, é dada pela expressão

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)},$$

onde $s = (a+b+c)/2$. Escreva uma função que aceite a, b e c como parâmetros de entrada e retorne o valor da área. Esta função deve estar preparada para enviar uma mensagem de erro, caso não exista um triângulo cujas medidas dos lados sejam esses valores. Teste essa função para: (i) $a = 1,\ b = 3,\ c = 5$; (ii) $a = 3,\ b = 4,\ c = 5$. Apresente o resultado destes testes.

Exercício 4. Escreva uma função para converter nós em quilómetros por hora, usando o fator de conversão 1 nó=1.852 km/h e produzir uma tabela análoga à seguinte:

```
  no      km/h
 --------------
  5.0      9.26
 15.0     27.78
 25.0     46.30
```

Esta função deverá:

- ter três parâmetros de entrada: o limite inferior da tabela em nós, o limite superior da tabela em nós e o espaçamento (no exemplo da tabela apresentada, o limite inferior é 5, o limite superior é 25 e o espaçamento é 10);
- estar devidamente comentada;
- enviar uma mensagem de erro, caso os dados de entrada não sejam válidos;
- retornar o vetor das velocidades calculadas em quilómetros por hora.

Exercício 5. Escreva uma função que, dados os números naturais a, d e n, calcule, sem recorrer à função `sum`, a soma dos primeiros n termos da sucessão

$$a, \frac{a}{1+d}, \frac{a}{1+2d}, \frac{a}{1+3d}, \ldots$$

Esta função deverá estar devidamente comentada e enviar uma mensagem de erro, no caso dos dados não serem adequados.

Exercício 6. O *fatorial generalizado* de um número a é definido do seguinte modo:

$$(a)_n := \begin{cases} 1, & \text{se } n = 0 \\ a(a+1)(a+2)\cdots(a+n-1), & \text{se } n \geq 1 \end{cases}.$$

Por exemplo, $(2)_4 = 2 \times 3 \times 4 \times 5 = 120$ e $\left(\frac{1}{2}\right)_3 = \frac{1}{2} \times \left(\frac{1}{2}+1\right) \times \left(\frac{1}{2}+2\right) = \frac{15}{8}$. Escreva um programa para, dado o valor de a e n, calcular $(a)_n$.

Exercício 7. Um primo de Mersenne é um número primo do tipo $2^n - 1$, onde n é um número natural. Por exemplo, 7 é um primo de Mersenne, porque $7 = 2^3 - 1$, mas 11 não é um primo de Mersenne. Faça uma lista dos primeiros 8 primos de Mersenne.

Exercício 8.* Escreva uma *script* que, dada um frase em português, indique quantas palavras constituem essa frase e apresente a frase com as palavras não acentuadas.

Por exemplo, se a frase for *Esta é uma questão opcional do teste de Matemática Computacional I*, o resultado deverá ser 11 e *Esta e uma questao opcional do teste de Matematica Computacional I*.

Exercício 9.* Chama-se **número perfeito** a um número natural n, quando a soma dos seus divisores próprios é igual a n. Quando esta soma é inferior a n, o número chama-se **deficiente** e quando é superior diz-se **abundante**. Por exemplo, 6 é um número perfeito $(1+2+3=6)$, 10 é um número deficiente $(1+2+5<10)$ e 12 é abundante $(1+2+3+4+6>12)$.

a) Escreva uma função que aceite como parâmetro de entrada um número natural e retorne a soma dos seus divisores próprios.

b) Escreva uma *script* que, dado um número natural n, apresente no ecrã, dependendo do valor de n, uma das seguintes mensagens: *n é um número perfeito*, ou *n é um número abundante*, ou *n é um número deficiente*.

c) Escreva uma *script* ou função que, dado um número natural n, indique quantos números inferiores a n são perfeitos, quantos são abundantes e quantos são deficientes.

d) Calcule o primeiro número abundante ímpar.

e) Liste os números perfeitos menores que 1000.

Exercício 10. *a)* Escreva uma *script*/função que, dado um número natural n, devolva o menor número primo p tal que $p > n$.

b) Altere o seu código de forma a permitir que n seja um vetor de números naturais. Por exemplo, se $n = [13, 26, 89]$, o resultado deverá ser $[17, 29, 97]$.

Exercício 11. Escreva um programa que dados dois inteiros positivos m e n (com $m < n$), determine todos os palíndromos (ou capicuas) entre m e n. Quantas capicuas há com 3 dígitos?

Exercício 12.* A conjetura de Brocard afirma que:

Há pelo menos quatro primos entre os quadrados de dois primos consecutivos maiores que 2.

Por exemplo, existem 5 primos entre 3^2 e 5^2 e 15 primos entre 7^2 e 11^2. Construa uma função/script que, dado um número natural k, verifique a conjetura para os primeiros k pares de primos consecutivos maiores que 2.

Exercício 13.* Um número *semiprimo* é um número natural que é o produto de dois números primos, não necessariamente distintos. Os primeiros semiprimos são $4, 6, 9, 10, 14, 15, \ldots$.

a) Defina uma função que aceite como argumento um número natural e retorne verdadeiro, se o número for um semiprimo e falso caso contrário (pode usar a função `factor`).

b) Escreva uma *script*/função que, dado um número natural, retorne um vetor com todos os semiprimos menores ou iguais a esse número.

c) Escreva uma *script* que liste os k primeiros semiprimos. O valor k deve ser pedido interativamente ao utilizador, através do uso de uma função apropriada.

Exercício 14. As regras do Totoloto (`https://www.jogossantacasa.pt/`) são bem conhecidas: a chave obtém-se mediante a extração de 5 bolas de uma esfera com 49 números e de 1 bola de outra esfera com 13 números (o chamado número da sorte).

a) Escreva um programa para simular uma extração do Totoloto. Use um formato de saída de resultados análogo ao seguinte.

```
Totoloto    Concurso: 01/2021   Data do sorteio: 06/01/21
*********************************************************
Chave:              2    9   11   34   40 +  4
Ordem de saída:     2   40    9   34   11 +  4
```

b) Escreva um programa que dada uma chave (6 números) indique se essa chave é premiada ou não, de acordo com a extração simulada. No caso de ser uma chave premiada, deve ter em conta que são 6 as categorias de prémios, em função do número de acertos:

1º Prémio - 5 números + nº sorte; **2º Prémio** - 5 números; **3º Prémio** - 4 números;

4º Prémio - 3 números; **5º Prémio** -3 números; **6º Prémio** - número da sorte.

Exercício 15.* A *função de Euler*, $\phi(n)$, é definida como o número de inteiros positivos menores que n que são primos com n. Por exemplo, há 8 inteiros positivos a inferiores a 24 tais que $\text{mdc}(24, a) = 1$, a saber, $1, 5, 7, 11, 13, 17, 19$ e 23. Logo, $\phi(24) = 8$.

a) Implemente a função de Euler, recorrendo à função do MATLAB **gcd**.

b) Modifique a função anterior, de forma a poder ser usada quando n é um vetor de inteiros positivos.

c) Naturalmente que, se p é um número primo, então $\phi(p) = p - 1$. Ilustre esta afirmação para os primos menores que 20.

Exercício 16.* Considere o seguinte algoritmo[1] que gera uma sequência de inteiros:

- *Comece com um inteiro positivo n.*

- *O próximo número da sequência será $n/2$, se n for par ou $3n + 1$ se n for ímpar.*

- *Repita este processo com o novo valor obtido, terminando quando atingir* 1.

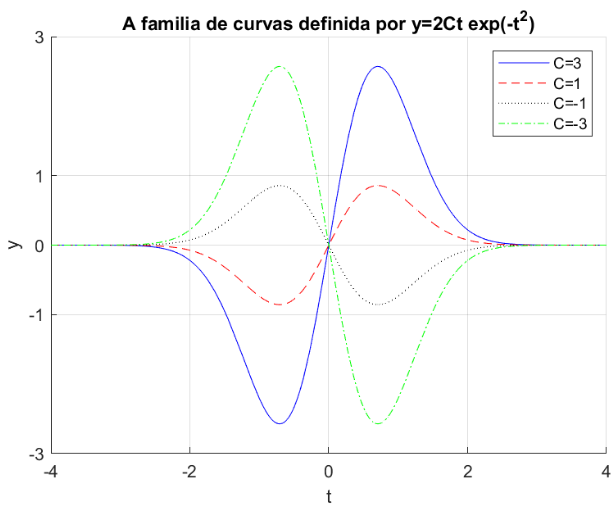
Por exemplo, a seguinte sequência de números será gerada para o valor inicial $n = 22$: 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1.

a) Implemente este algoritmo e teste-o para vários valores de n.

b) Para cada n, chama-se *vida de n* ao número de elementos da sequência obtida. Por exemplo, a vida de $n = 22$ é 16. Modifique o programa anterior, de forma a ser possível obter esta informação.

c) Represente graficamente os valores da vida de n, para valores de n entre 2 e 30.

Exercício 17. Esboce um gráfico **idêntico** ao da figura seguinte.



[1] Este problema é conhecido como Problema de Collatz ou sequência $3n+1$. É uma conjetura bem conhecida que este algoritmo termina para todo n positivo. (Veja, por exemplo, http://mathworld.wolfram.com/CollatzProblem.html.)

Exercício 18. Os *polinómios de Legendre* $\mathcal{P}_n(x)$ são definidos pela seguinte relação de recorrência

$$(n+1)\mathcal{P}_{n+1}(x) - (2n+1)x\mathcal{P}_n(x) + n\mathcal{P}_{n-1}(x) = 0,$$

com $\mathcal{P}_0(x) = 1$ e $\mathcal{P}_1(x) = x$.

a) Calcule os 4 polinómios de Legendre seguintes e represente graficamente os 6 polinómios, no intervalo $[-1, 1]$.

b) Sabendo que

$$(x^2 - 1)\mathcal{P}'_n(x) = nx\mathcal{P}_n(x) - n\mathcal{P}_{n-1}(x),$$

escreva um programa que represente, no mesmo gráfico, o polinómio de Legendre de grau n e da sua derivada, no intervalo $[-1, 1]$. Use cores e estilos diferentes para cada função e apresente o gráfico obtido para $n = 5$.

⇛ Solução dos exercícios selecionados

➟ Exercício 8

back

```
txt='Esta é uma questão opcional do teste de Matemática ...
   Computacional I';
for i=1:length(txt)
   switch txt(i)
      case {'á','à','ã'}
         txt(i)='a';
     case 'é'
        txt(i)='e';
     case 'í'
          txt(i)='i';
     case {'ó','õ'}
        txt(i)='o';
     case 'ú'
        txt(i)='u';
  end
end
contapalavras=sum(txt==' ')+1
txt
```

```
  contapalavras=11
  txt = 'Esta e uma questao opcional do teste de Matematica
   Computacional I'
```

➟ Exercício 9

back

Ver função somadivisores.m

a)

```
somadivisores(6)
```

```
ans = 6
```

```
for n=[6 10 12]
% cálculo da soma dos divisores de n,
% usando a função somadivisores da alínea anterior
soma=somadivisores(n);
if soma==n % o número é perfeito
   disp([num2str(n),' é perfeito'])
elseif soma<n % o número é deficiente
   disp([num2str(n),' é deficiente'])
else % o número é abundante
   disp([num2str(n),' é abundante'])
end
end
```

```
   6 é  perfeito
10 é  deficiente
12 é  abundante
```

b) A solução apresentada só permite obter o número total de nº perfeitos, abundantes e deficientes. Não está preparada para apresentar cada um deles. Quais as alterações a introduzir para obter uma lista de cada um dos números?

```
n=100;
totalperfeitos=0;totalabundantes=0;totaldeficientes=0;
for i=1:n-1
% vai ser calculada a soma dos  divisores
% de todos os valores inferiores a n
   soma=somadivisores(i);
   if soma==i % o número é perfeito
      totalperfeitos=totalperfeitos+1; % incrementa o ...
 contador de  números perfeitos
   elseif soma<i % o número é deficiente
      totaldeficientes=totaldeficientes+1; % incrementa ...
 o contador de números deficientes
   else % o número é abundante
      totalabundantes=totalabundantes+1; % incrementa o ...
 contador de números abundantes
   end
end
fprintf('Ha \n %6i numeros perfeitos,\n',totalperfeitos)
fprintf(' %6i numeros abundantes, \n',totalabundantes)
fprintf(' %6i numeros deficientes\n menores que ...
 %3i\n',totaldeficientes,n)
```

```
Ha
    3 numeros perfeitos,
   21 numeros abundantes,
   75 numeros deficientes
menores que 100
```

```
n=1;
while somadivisores(n)<=n % enquanto não for encontrado ...
 um número abundante,
   n=n+2; % vão sendo incrementados os números ímpares
end
fprintf('O primeiro numero impar abundante é o %3i\n',n);
```

```
O primeiro numero impar abundante é o 945
```

```
perfeitos=[];
for n=1:1000
   if somadivisores(n)==n
      perfeitos=[perfeitos n];
   end
end
perfeitos
```

```
perfeitos = 1x4
    1     6    28   496
```

➟ Exercício 12

back

Ver função contaPrimos

```
k=10;conta=0;n=3;p=[];% Esta script pode ser melhorada!
while conta<k
   if isprime(n)
      conta=conta+1;
      p=[p n];
   end
   n=n+1;
end
for i=1:length(p)-1
   teste(i)=contaPrimos(p(i)^2,p(i+1)^2);
end
teste(2:end)>=4
```

```
ans = 1x8 logical array
      1   1   1   1   1   1   1   1
```

```
primes(10)
```

```
ans = 1x4
       2     3     5     7
```

➟ Exercício 13 back

a) Ver função issemiprimo

```
issemiprimo(7)
```

```
ans =
     0
```

```
issemiprimo(4)
```

```
ans =
     1
```

b) Ver função semiprimos

```
semiprimos(38)
```

```
ans = 1x14
      4       6       9      10      14      15      21      22      25      26
     33      34      35      38
```

```
k=input('Introduza o valor de k ');
n=4;
lista=[];
while length(lista)<=k
    if issemiprimo(n)
        lista=[lista n];
    end
    n=n+1;
end
lista
```

```
lista = 1x7
        4       6       9      10      14      15      21
```

➟ Exercício 15 back

a) Ver função euler

```
euler(24)
```

```
ans = 8
```

b) Ver função eulerV

```
eulerV([24,32])
```

```
ans = 1x2
8     16
```

c)

```
p=primes(20)
```

```
p = 1x8
    2     3     5     7    11    13    17    19
```

```
eulerV(p)==p-1
```

```
ans = 1x8 logical array
    1   1   1   1   1   1   1   1
```

➟ Exercício 16

back

a)

```
n=22;
fprintf('%4i',n)
```

```
22
```

```
while n ~= 1
   if rem(n,2) == 1
      n = 3*n+1;
   else
      n = n/2;
   end
fprintf('%4i',n)
end
```

```
11  34  17  52  26  13  40  20  10   5  16   8   4   2   1
```

b)

```
n=22;
fprintf('%4i',n)
```

```
22
```

```
vida = 1;
while n ~= 1
   if rem(n,2) == 1
      n = 3*n+1;
   else
      n = n/2;
   end
vida = vida + 1;
fprintf('%4i',n)
end
```

```
11  34  17  52  26  13  40  20  10   5  16   8   4   2   1
```

```
vida
```

```
vida = 16
```

c)

```
figure
hold on
for j=2:30
    axis tight
    vida = 1;
    n=j;
    while n ~= 1
        if rem(n,2) == 1
            n = 3*n+1;
        else
            n = n/2;
        end
        vida = vida + 1;
    end
    plot([j;j],[0;vida],'r-')
end
hold off
xlim([-1.0 30.0])
ylim([-2 112])
```