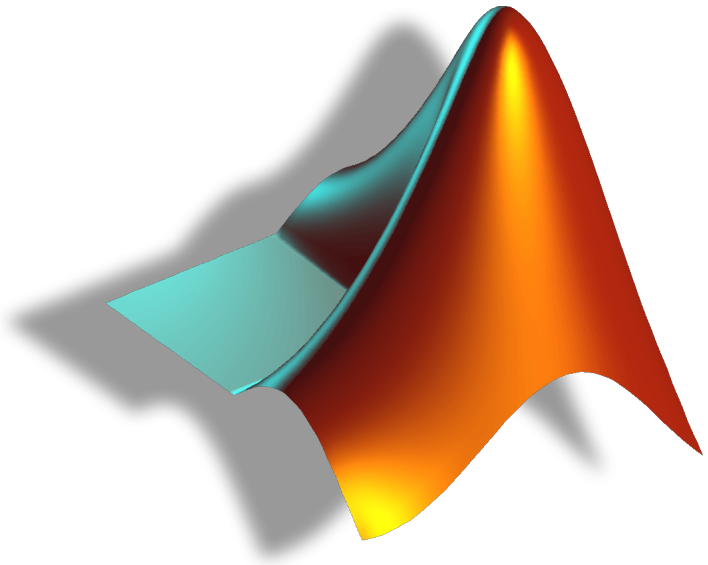


MATLAB de A a B

Dez fichas de trabalho

Adérito Araújo

Abril de 2020

1 2 9 0 UNIVERSIDADE D COIMBRA

O MATLAB (Matrix Laboratory) é um software interactivo para cálculo numérico e gráfico. Pelo nome, vemos que foi especialmente concebido para trabalhar com matrizes: sistemas lineares, cálculo de vectores e valores próprios, factorização de matrizes, etc. Além disso, possui série de potencialidades gráficas e pode ser usado como linguagem de programação. O MATLAB também pode efectuar álgebra simbólica mas, nesse caso, é preferível usar sistemas computacionais desenhados para esse efeito como o Mathematica ou o Maple.

Estas fichas pretendem ser apenas uma introdução muito breve ao MATLAB. Para uma abordagem mais aprofundada, recomendamos o curso on-line em https://matlabacademy.mathworks.com (em caso de erro de licenciamento consultar esta página) e o livro

Numerical Computing with MATLAB
Cleve Moler
SIAM Philadelphia, 2004, ISBN: 978-0-898716-60-3.

O sistema MATLAB é constituído pelas seguintes partes.

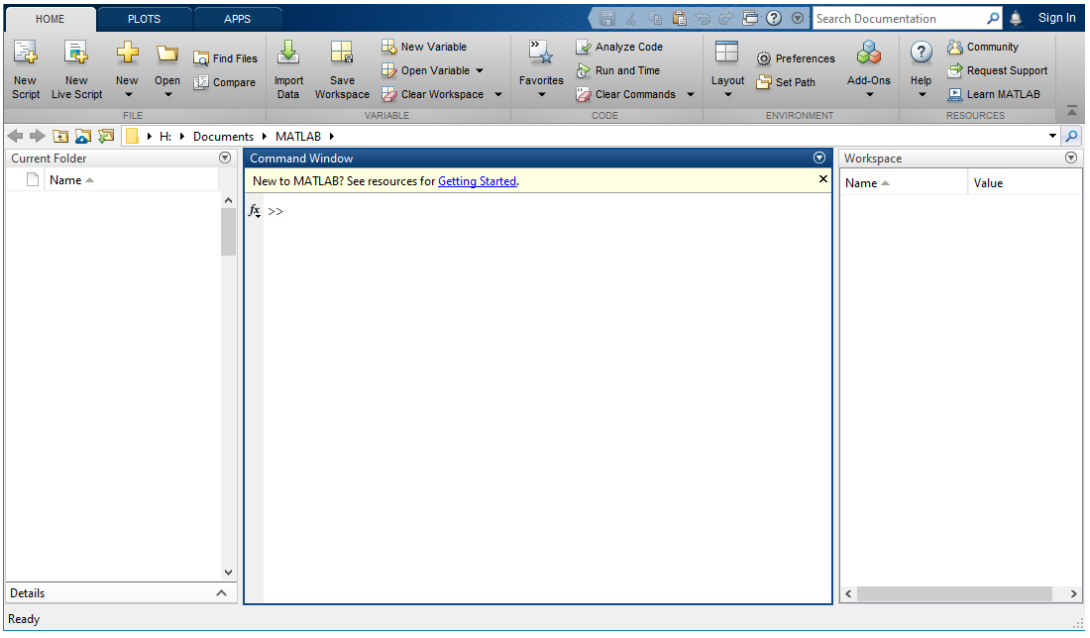
- A linguagem. Permite a manipulação e criação de matrizes de forma rápida e intuitiva. Diferentes soluções para um problema podem ser testadas numa fracção do tempo que levaria com outras linguagens (C ou Fortran, por exemplo). Possui um conjunto muito vasto de funções que permitem resolver problemas complexos de forma eficiente.

- O ambiente de trabalho. O MATLAB proporciona um ambiente de trabalho que permite a gestão e visualização das variáveis, ler e gravar variáveis em disco e gerar programas em linguagem MATLAB, possibilitando assim a automatização de cálculos complexos.

- Gráficos. As funções de criação, visualização e manipulação de gráficos são muito fáceis de usar e permitem a criação de gráficos 2D e 3D. O ajuste de escala é automático e o utilizador pode começar a utilizar as funções de geração de gráficos pouco tempo depois do primeiro contacto com o ambiente do MATLAB.

- *Toolboxes*. O MATLAB disponibiliza um conjunto de pacotes de funções para as mais variadas áreas de cálculo científico, sendo estes denominados *toolboxes*. Existem *toolboxes* para estatística, processamento de sinal, processamento de imagem, controlo, cálculo simbólico, etc.

Quando iniciamos o MATLAB, o ambiente de trabalho surge na sua configuração padrão.



O ambiente de trabalho inclui três janelas: janela da pasta actual (onde pode aceder aos seus ficheiros); a janela de comando (permite digitar os comandos na linha de comando, indicada pelo sinal (>>)); a janela da área de trabalho (onde pode explorar os dados que criou ou importou de ficheiros). Para além disso, ainda pode ser visível uma janela com o histórico dos comandos e um botão para aceder à ajuda do MATLAB  .

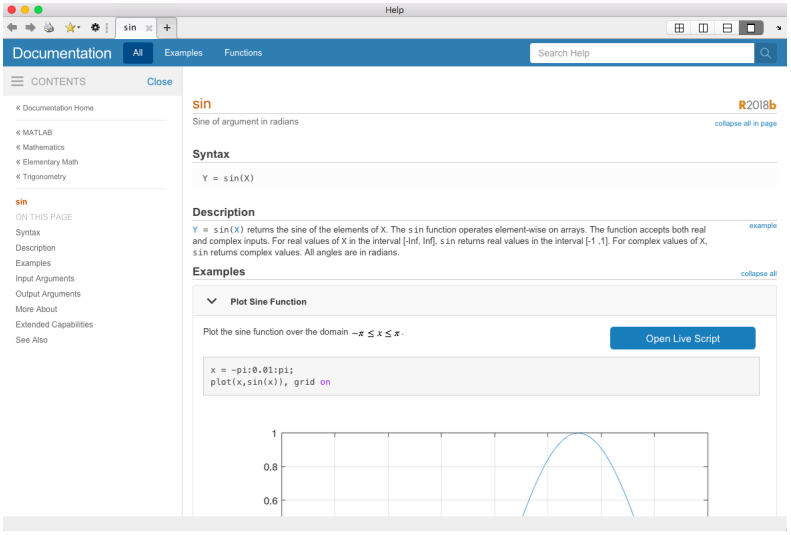
Todas as funções do MATLAB possuem documentação de suporte que inclui exemplos e descreve as entradas, saídas e sintaxe da função. Existem várias maneiras de aceder essas informações na linha de comando: usando o comando `help`, para uma versão abreviada da documentação da função na janela de comando; usando o comando `doc`, para uma documentação detalhada da função numa janela separada.

```
>> help sin
  sin    Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind, sinpi.

    Reference page for sin
    Other functions named sin

>> doc sin
```



A linha de comandos do MATLAB permite a criação de variáveis e a introdução de instruções a executar. Por exemplo, a instrução

```
>> N = 10
```

inicializa uma variável `N` com o valor `10`, que pode ser utilizada em instruções seguintes. O MATLAB adiciona a variável na à área de trabalho e exibe o resultado na janela de comando. Poderá recuperar o valor da variável digitando o seu nomes pressionando `<enter>`:

```
>> N
N =
        10
```

Os nomes das variáveis começam com uma letra, seguida de letras, números ou sublinhados (*underscore*). Note-se que o MATLAB faz distinção entre maiúsculas e minúsculas e que somente os 31 primeiros caracteres de um nome de variável são reconhecidos.

Existem três tipos de números no MATLAB: números inteiros, números reais e números complexos. Inteiros são inseridos sem o ponto decimal e os reais com ponto decimal. Números complexos são representado na sua forma cartesiana. A unidade imaginária $\sqrt{-1}$ é indicado por `i` ou `j`. (Cuidado: se já tinha definido `i` ou `j` anteriormente, o seu valor como unidade imaginária deixa de ser válido). Alguns exemplos de números permitidos são:

```
3               -99             0.0001
9.6397238       1.60210e-20     6.02252e23
1i              -3.14159j       3e5i
```

O MATLAB permite a construção de expressões matemáticas sem qualquer declaração do formato numérico ou dimensão das matrizes. Existem quatro constituintes básicos nas expressões do MATLAB: variáveis; números; operadores e funções.

Todas as variáveis do MATLAB são do tipo matriz (vemos isso na próxima ficha) e a sua criação é automática. Por exemplo, o comando

```
>> Custo_total = 1000
```

resulta na criação em memória de uma matriz de 1×1 com o valor 1000. O MATLAB distingue as letras maiúsculas das minúsculas nos nomes das variáveis e só toma em consideração os primeiros 31 caracteres. Para visualizar o valor de uma variável basta escrever o seu nome. Existe uma variável especial que é utilizada pelo MATLAB quando não se atribui o resultado de uma expressão a qualquer variável. Esta variável designa-se por `ans` (do termo *answer*, resposta) e pode ser utilizada numa sessão interactiva para continuação dos cálculos, tal como o exemplo seguinte demonstra.

```
>> 2*sin(2)
ans =
       0.017829
```

Quanto aos números, vimos que o MATLAB utiliza uma notação usual para a representação dos números, admitindo notação científica e números complexos. Nesta ficha vamos abordar dos operadores e das funções do MATLAB.

As operações nas expressões do MATLAB seguem as regras habituais de precedência e podem ser aplicadas quer a matrizes quer a números.

Operação	Símbolo
adição	+
subtracção	-
multiplicação	*
divisão	/ ou \
exponenciação	^

O MATLAB tem dois operadores de divisão: `/`, a divisão à direita e `\`, a divisão à esquerda. Eles não conduzem aos mesmos resultados

```
>> dd = 5 / 1
dd =
        5
>> de = 5 \ 1
de =
        0.2000
```

Geralmente, não se deseja ver o resultado de cálculos intermédios. Para isso, finaliza-se a declaração ou expressão de atribuição com ponto e vírgula:

```
>> x = -13; y = 5*x, z = x^2+y
y =
        -65
z =
        104
```

Nos cálculos anteriores, o valor de `x` não é exibido. Observe também que podemos colocar várias instruções numa linha, separadas por vírgulas ou ponto e vírgula.

O MATLAB possui um conjunto muito grande de funções matemáticas que permitem resolver grande parte dos problemas de cálculo encontrados, tais como: `sin`, `cos`, `tan`, `sqrt`, `log`, etc. Estas aceitam

números complexos como argumento e podem também devolver resultados do tipo complexo. Por exemplo, ao calcular a raiz quadrada de um número negativo não ocorre um erro mas o correspondente número complexo é automaticamente devolvido. Quando as funções são aplicadas sobre matrizes a função é aplicada a cada um dos elementos. Eis uma lista de algumas das funções matemáticas mais comuns no MATLAB:

`abs`	Valor absoluto e módulo do complexo
`acos, acosh`	Arco cosseno e argumento cosseno hiperbólico
`angle`	Argumento de um complexo
`asin, asinh`	Arco seno e argumento seno hiperbólico
`atan, atanh`	Arco tangente e argumento tangente hiperbólica
`ceil`	Arredonda para cima
`conj`	Complexo conjugado
`cos, cosh`	Cosseno e cosseno hiperbólico
`exp`	Exponencial
`fix`	Arredonda para zero
`factorial`	Function factorial
`floor`	Arredonda para baixo
`gcd`	Máximo divisor comum
`imag`	Parte imaginária de um número complexo
`lcm`	Mínimo múltiplo comum
`log, log10`	Logaritmo natural e logaritmo na base 10
`max`	Máximo
`mod`	Módulo (resto da divisão com sinal)
`real`	Parte real de um número complexo
`round`	Arredonda para o inteiro mais próximo
`sign`	Função sinal
`sin, sinh`	Seno e seno hiperbólico
`sqrt`	Raiz quadrada
`tan, tanh`	Tangente e tangente hipérbólica

Existem algumas funções especiais que devolvem o valor das constantes matemáticas mais utilizadas.

Função	Valor
`pi`	3.141592653589793
`i`	$\sqrt{-1}$
`j`	$\sqrt{-1}$
`eps`	2.220446049250313e-16 (precisão do formato *double*)
`realmin`	2.225073858507201e-308 (o menor número real)
`realmax`	1.797693134862316e+308 (o maior número real)
`Inf`	Infinito
`NaN`	*Not-a-Number*

Exercícios

1. Defina uma variável x com o valor 2 e calcule: (a) $(x+6)!$; (b) $(x+1)^4$; (c) $\log(x-1)$; (d) $\exp(x^2)$; (e) $\sin(x)\cos(x)$; (f) $\sin^2(x)+\cos^2(x)$; (g) $\sin(\sin(\sin(x)))$.

2. Mostre que a adição em vírgula flutuante tem mais do que um elemento neutro. Para isso, veja que calcule `1 + eps/2` e explique o resultado obtido.

3. Explique o resultado obtido pela sequência de comandos: `x = 1.e-15; ((1+x)-1)/x`

4. A adição em vírgula flutuante não verifica a propriedade associativa. Para comprovar esse facto, calcule, usando o computador, a soma $x+y+z$ nas formas $x+(y+z)$ e $(x+y)+z$ para os casos em que: (a) $x=1.0$; $y=-5.0$; $z=6.0$; (b) $x=1\times10^{20}$; $y=-1\times10^{20}$; $z=1.0$; (c) $x=1\times10^{308}$; $y=1.1\times10^{308}$; $z=-1.001\times10^{308}$. Explique os resultados obtidos.

5. Em cada alínea, determine o valor da expressão no MATLAB e explique, com precisão, a ordem em que o cálculo foi realizado: (a) `-2^3+9`; (b) `2/3*3`; (c) `3*2/3`; (d) `3*4-5^2*2-3`; (e) `(2/3^2*5)*)3-4^3)^3`; (f) `3*(3*4-2*5^2-3)`.

Como no caso de variáveis escalares, as matrizes não precisam ser declaradas e o MATLAB reserva automaticamente o espaço de armazenamento. A seguir, descrevemos as técnicas básicas para criar e operar com matrizes unidimensionais (vectores) e matrizes bidimensionais (matrizes).

O seguinte comando permite criar um vector linha:

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

Os vectores coluna podem ser criados de forma similar. No entanto, um ponto e vírgula deve separar as componentes de um vector coluna:

```
>> b = [1;2;3]
b =
        1
        2
        3
```

De forma alternativa, as componentes de um vector coluna podem se introduzidas em linhas separadas (pressionando `<enter>` em vez de escrever o ponto e vírgula `;`).

A plica `'` é usada para obter a transposta conjugada de de um vector (matriz) enquanto que o operador ponto-plica `.'` permite obter o transposto do vector (matrix). O comando `length` devolve o número de componentes do vector:

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

O comando seguinte permite criar uma matriz 3×3:

```
>> A = [1 2 3;4 5 6;7 8 10]
A =
        1  2  3
        4  5  6
        7  8  10
```

Note-se que o operador ponto e vírgula `;` é usado para separar as linhas. O comando `size` devolve o número de linhas e colunas da matriz:

```
>> size(A)
ans =

     3     3
```

Para obter um elemento de uma matriz usam-se parêntesis curvos:

```
>> A(3,2)
ans =
        8
```

Os índices das matrizes são listas de números inteiros positivos que podem ser armazenadas em vectores declarados previamente. Se pretendermos por exemplo, extrair a segunda linha da matriz `A` podemos fazer

```
>> v = A(2,[1 2 3])
v =
         4  5  6
```

ou declarando primeiro um vector para os índices das colunas

```
>> k = [1 2 3];
>> v = A(2,k)
v =
         4  5  6
```

A criação de vectores elemento a elemento é bastante morosa e para matrizes de grandes dimensões quase irrealizável. O MATLAB permite gerar sequências de números de forma rápida se fizermos uso do operador dois pontos `:`. Se quisermos gerar o vector `a = [1, 2, 3, ... , 10]` podemos fazer

```
>> a = 1:10
a =
     1  2  3  4  5  6  7  8  9  10
```

Por outra palavras, o operador `:` permite obter uma gama completa de índices. Por ser usado, por exemplo, para obter uma linha ou uma coluna inteira de uma matriz:

```
>> A(2,:)
ans =
        4 5 6

>> A(:,3)
ans =
        3
        6
        10
```

A sub-matriz `B` composta pelas linhas 1 e 3 e pelas colunas 1 e 2 da matriz `A` é obtida da seguinte maneira:

```
>> B = A([1 3], [1 2])
B =
        1  2
        7  8
```

Para trocar as linhas 1 e 3 da matriz `A`, usa-se o vector de índices de linha juntamente com o operador dois pontos:

```
>> C = A([3 2 1],:)
C =
        7  8  10
        4  5  6
        1  2  3
```

Para apagar uma linha (coluna) usa-se o vector vazio `[]`:

```
>> A(:, 2) = []
A =
        1  3
        4  6
        7  10
```

A segunda coluna de `A` é apagada. Para inserir uma linha (coluna) usamos a técnica para criar matrizes e vectores:

```
>> A = [A(:,1) [2 5 8]' A(:,2)]
A =
        1  2  3
        4  5  6
        7  8  10
```

A matriz `A` voltou à sua forma original.

Concatenar matrizes consiste em formar matrizes a partir de outras mais pequenas. A notação é idêntica à utilizada para formar matrizes com números. Os seguintes exemplos ilustram a concatenação de matrizes.

```
>> a = [1 2; 3 4]
a =
        1  2
        3  4
>> A = [a a; a a]
A =
        1  2  1  2
        3  4  3  4
        1  2  1  2
        3  4  3  4
>> b = (1:4)'
b =
        1
        2
        3
        4
>> B = [b b [a;a]]
A =
        1  1  1  2
        2  2  3  4
        3  3  1  2
        4  4  3  4
```

Em MATLAB, o operador `:` é usado de várias maneiras. Como vimos, é especialmente útil para criar vectores de valores igualmente espaçados:

```
>> x = 1:5
x =
        1  2  3  4  5
```

De forma geral, `m:n` gera o vector com componentes `m`, `m+1`, ..., `n`, e outros incrementos podem ser especificados usando um terceiro parâmetro, como em:

```
>> x = 2:3:9
x =
        2  5  8
```

Esta formulação é frequentemente usada em ciclos `for`, como veremos nas folhas seguintes.

Exercícios

1. Gere uma sequência de números pares com início em 4 e a terminar no número 100.
2. Gere uma sequência numérica decrescente com início em 5 e a terminar em -5.
3. Gere uma sequência numérica com 100 elementos pertencentes ao intervalo $[0, 1]$.
4. Considere uma matriz A com 20 linhas e 30 colunas. Construa um comando que permita extrair para uma matriz B uma sub-matriz de A constituída pelas linhas de 10 a 15 e as colunas de 9 a 12.
5. Gere uma sequência de números múltiplos de 3 compreendidos entre 100 e 132, dispostos num vector por ordem decrescente.
6. Gere uma sequência a começar em π e a acabar em $-\pi$ com um passo de $-\pi/15$.
7. Crie uma matriz 3×3 em que todos os elementos são iguais a 3.
8. Crie um vector coluna com 100 elementos aleatórios com uma distribuição uniforme.
9. Crie uma matriz 4×4 em que todos os elementos são iguais a $1 + i2$.

O MATLAB possui várias funções internas para cálculos com vectores e matrizes esparsos. O comando `sparse` é usado para criar a forma esparsa de um vector ou de uma matiz. Seja

```
>> A = [0 0 1 1; 0 1 0 0; 0 0 0 1];
```

Então, o comando `sparse` gera a forma esparsa indicada:

```
>> B = sparse(A)
B =
        (2,2) 1
        (1,3) 1
        (1,4) 1
        (3,4) 1
```

O comando `full` converte a forma esparsa na forma completa/cheia:

```
>> C = full(B)
C =
        0 0 1 1
        0 1 0 0
        0 0 0 1
```

O comando `sparse` tem a sintaxe seguinte: `sparse(k,l,s,m,n)` onde `k` e `l` são índices de linha e de coluna, respectivamente, `s` é uma matriz de números diferentes de zero cujos índices são especificados em `k` e `l`, e `m` e `n` indicam o número de linhas e de colunas, respectivamente:

```
>> S = sparse([1 3 5 2], [2 1 3 4], [1 2 3 4], 5, 5)
S =
        (1,2) 1
        (3,1) 2
        (5,3) 3
        (2,4) 4
```

A função `spy` cria um gráfico a partir de uma matriz. As entradas não nulas são exibidas como pontos.

```
>> spy(S)
```

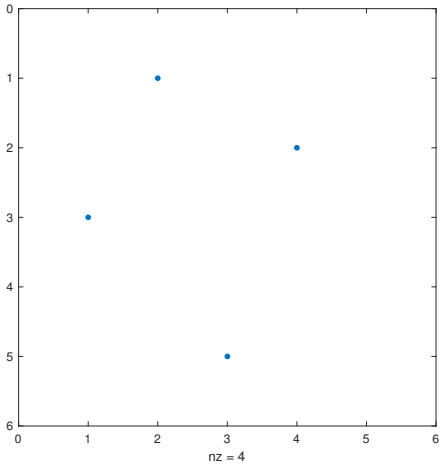


Figura 1: Saída de `spy(S)`.

Para obter uma matriz esparsa com várias diagonais paralelas à diagonal principal pode usar-se o comando `spdiags`. A sua sintaxe é: `spdiags(B,d,m,n)`. A matriz resultante é uma matriz esparsa $m \times n$. As suas diagonais são as colunas da matriz `B`. A localização das diagonais é descrita no vector `d`.

A função `diag` cria uma matriz diagonal com entradas diagonais retiradas de um dado vector:

```
>> d = [1 2 3];
>> D = diag(d)
D =
        1 0 0
        0 2 0
        0 0 3
```

Para extrair a diagonal principal da matriz `D`, usa-se, de novo, a função `diag`:

```
>> d = diag(D)
d =
        1
        2
        3
```

A função `inv` é usada para calcular a inversa de uma matriz. Seja, por exemplo, a matriz `A` definida da seguinte maneira:

```
>> A = [1 2 3;4 5 6;7 8 10]
A =
        1 2 3
        4 5 6
        7 8 10
```

Então,

```
>> B = inv(A)
B =
        -0.6667 -1.3333  1.0000
        -0.6667  3.6667 -2.0000
         1.0000 -2.0000  1.0000
```

O comando `find` pode ser usado para encontrar os índices das entradas não numas de um vector ou de uma matriz e é frequentemente usado com matrizes esparsas. A sua sintaxe é: `find(x)` devolve um vector contendo os índices dos elementos não nulos do vector `x`. `[i,j]=find(A)` devolve os índices de linha e de coluna dos elementos não nulos de uma matriz (esparsa) `A`. `[i,j,v]=find(A)` também devolve s índices de linha e de coluna dos elementos não nulos de uma matriz (esparsa) `A` e como terceiro parâmetro de saída devolve o vector contendo os valores dos elementos não nulos.
Por exemplo, podemos encontrar o valor absoluto do menor elemento não nulo de uma determinada matriz `A` da seguinte forma:

```
>> [i,j,v] = find(A);
>> minimum = min(abs(v));
```

Muitas vezes, é útil começar com uma matriz predefinida que forneça apenas a dimensão. Uma lista parcial dessas funções é:

`zeros`	Matriz preenchida com 0
`ones`	Matriz preenchida com 1
`eye`	Matriz identidade
`rand`	Matriz com números aleatórios distribuídos uniformemente
`linspace`	Vector com componentes espaçadas linearmente
`logspace`	Vector com componentes espaçadas logatimicamente
`pascal`	Matriz de Pascal
`hilb`	Matriz de Hilbert

Em MATLAB, as funções polinomiais de grau $\leq n$,

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

são representadas por vectores linha de comprimento $n+1$ contendo os coeficientes `p = [p(1),p(2),..., p(n+1)]`, onde `p(i)` $= a_i, i = 1, \ldots, n+1$. (Observe que `p(1)` é o coeficiente da potência de maior grau, e `p(n+1)` é o coeficiente constante.) Há várias funções do MATLAB que usam esta representação. `polyval` calcula o valor do polinómio num ponto (ou num conjunto de pontos):

```
>> p = [1 3 0]       % representa p(x) = x^2+3*x
p =
     1     3     0

>> polyval(p,2)      % x^2+3*x = 7 quando x = 2
ans =
     7
```

Outro exemplo, para calcular as suas raízes considera-se a função `roots` na forma

```
>> r = roots(p)
r =
     0
    -3
```

Para confirmar se o resultado é o certo, poder-se-ía fazer

```
>> polyval(p,r)
ans =
     0
     0
```

Algumas manipulações simbólicas básicas também estão disponíveis `polyder` (resp. `polyint`) diferencia (resp. integra) um polinómio. Por exemplo:

```
>> polyder(p)      % calcula a derivada de p(x)=x^2+3*x
ans =
     2     3       % a derivada é 2*x+3
```

O produto entre dois polinómios é obtido através da função `conv` (convolução). O exemplo seguinte ilustra o produto entre os polinómios:

```
>> p1 = [1 0 1]
>> p2 = [1 0 1 -1]
>> conv(p1,p2)
ans =
     1     0     2    -1     1    -1
```

A função `polyfit` é usada para ajustar uma curva polinomial, ou seja, calcula uma aproximação polinomial para um determinados conjunto de dados usando o critério dos mínimos quadrados. No exemplo seguinte, um ajuste de grau 3 é calculado para os pontos $(x_i, \exp(x_i)), i = 0(0{,}1)1$:

```
>> x = linspace(0,1,10);
>> y = exp(x);
>> n = 3;
>> p = polyfit(x,y,n)
p =
    0.2792    0.4231    1.0160    0.9996
```

No caso especial de que $n+1$ ser igual ao número de dados fornecidos, com abscissas diferentes duas as duas, a função `polyfit` calcula os coeficientes do (único) polinómio interpolador correspondente.

Exercícios

1. Com a função `eye` crie uma matriz diagonal $4\times$ 4 em que todos os elementos da diagonal são iguais a 3.
2. Defina um vector linha com os 1000 primeiros inteiros positivos de duas formas distintas.
3. Defina um vector colunas com os 1000 primeiros números pares de duas formas distintas.
4. Construa um vector com 128 elementos com a seguinte sequência:

$$[\ 0\ 1\ 0\ -1\ 0\ 1\ \cdots\ 0\ -1\ ].$$

5. Usando o comando `cond`, que permite calcular o número de condição de uma matiz, determine um vector com os números de condição das matrizes de Hilbert de ordens $n = 2, \ldots, 10$.
6. Considere o vector $x = [x_1, \ldots, x_n]^T$ de comprimento n dado. Escreva uma sequência de comandos MATLAB que permitam obter a chamada *matriz companheira* de ordem n.

$$\begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -x_n & -x_{n-1} & -x_{n-2} & \ldots & -x_1 \end{pmatrix}.$$

7. Calcule o produto dos polinómios $x^6 + 10$ e $x^2 - 2x + 3$.
8. Construa um polinómio com as raízes $x = 1$, $x = 4$ e $x = 7$. Determine as raízes da sua derivada
9. Calcule a parábola dos mínimos quadrados para a função f dada pela seguinte tabela

x_i	0	0.1	0.2	0.3	0.4	0.5	0.6
$f(x_i)$	2.9	2.8	2.7	2.3	2.1	2.1	1.7

.

10. Considere os dados da tabela

Tensão	0.00	0.06	0.14	0.25	0.31	0.47	0.60	0.70
Deformação	0.00	0.08	0.14	0.20	0.23	0.25	0.28	0.29

correspondentes aos valores da deformação para diferentes valores da tensão aplicada numa amostra de tecido biológico (um disco invertebral).

   (a) Calcule a equação da reta de regressão.
   (b) Estime o valor da deformação correspondente a uma tensão igual a 0.9.

11. Um pára-quedista efectuou 5 saltos de diferentes alturas, tendo medido a distância a um alvo constituído por uma circunferência de raio 5 metros traçada no solo. Supondo que as respetivas altura e distância de cada salto satisfazem a seguinte tabela

Altura do salto (m)	1500	1250	1000	750	500
Distância do alvo (m)	35	25	15	10	7

recorra à interpolação para estimar a distância do alvo a que o pára-quedista cairia se saltasse de uma altura de 850 m.

Vamos agora explicar como executar operações aritméticas com vectores e matrizes, a principal característica do MATLAB e o que o torna uma ferramenta muito relevante na Matemática Computacional. As operações aritméticas de matriz/vector são a adição (+), a subtracção (-) e a multiplicação (*). A adição e a subtracção são definidas apenas se as matrizes tiverem as mesmas dimensões. A multiplicação só está definida se as matrizes tiverem dimensões internas iguais: ou seja, se `A` for uma matriz $n \times m$ e `B` é uma matriz $p \times q$, então `A*B` está bem definido (e é calculado pelo MATLAB) se $m = p$. O MATLAB também permite calcular potências (`^`) de matrizes quadradas.

O operador ponto `.` desempenha um papel particular em MATLAB. É usado para a aplicação *componente a componente* (ou *termo a termo*) do operador que segue o operador ponto.

Operação	Resultado
`A+B`	adição
`A-B`	subtracção
`A*B`	multiplicação
`2*B`	multiplicação por um escalar
`A^2`	exponenciação
`A.*B`	multiplicação termo a termo
`A.^2`	exponenciação termo a termo

A transposta conjugada de uma matriz a pode ser obtida usando o operador plica `'` e matriz transposta pode ser obtido com o operador ponto-plica `.'`.

Alguns exemplos:

```
>> a = [1 2 3];
>> b = [1 2 3]';
>> (a+i*b')'
ans =
        1.0000 - 1.0000i
        2.0000 - 2.0000i
        3.0000 - 3.0000i
```

enquanto que

```
>> (a+i*b').'
ans =
        1.0000 + 1.0000i
        2.0000 + 2.0000i
        3.0000 + 3.0000i
```

O produto escalar (produto interno) e o produto externo dos vectores `a` e `b` são calculados da seguinte forma:

```
>> prodint = a*b
prodint =
        14
>> prodext = a'*b'
prodext =
        1 2 3
        2 4 6
        3 6 9
```

Aplicação componente a componente do operador de multiplicação:

```
>> a.*b'
ans =
        1 4 9
>> a.^2
ans =
        1 4 9
```

Continuando com as operações de matrizes:

```
>> A = [1 2 3;4 5 6];
>> A*A
??? Error using ==> * Inner matrix dimensions must agree.
>> A*A'
ans =
        14 32
        32 77
```

O operador \ resolve sistemas de equações lineares. Se desejarmos calcular a solução de `Ax = b`, o método mais simples a usar com o MATLAB para encontrar `x` é definir `x = A\b`. Se `A` for uma matriz $n \times m$ e `B` é uma matriz $p \times q$ então `A\b` é definido (e é calculado pelo MATLAB) se $m = p$. Para sistemas não quadrados e singulares, a operação `A\b` dá a solução no sentido dos mínimos quadrados.

Consideremos o seguinte exemplo. Seja

```
>> A = [1 2 3;4 5 6;7 8 10]
A =
        1 2 3
        4 5 6
        7 8 10
```

e

```
>> b = ones(3,1);
```

Então,

```
>> x = A\b
x =
        -1.0000
         1.0000
         0.0000
```

Para verificar se solução calculada está certa, calculemos o vector resíduo `r`:

```
>> r = b - A*x
r =
        1.0e-015 *
        0.1110
        0.6661
        0.2220
```

Teoricamente, as entradas do vector resíduo calculado `r` deveriam ser nulas. Este exemplo ilustra o efeito dos erros de arredondamento erros na solução calculada pelo MATLAB.

Se $m > n$, o sistema `Ax=b` é sobredeterminado e, na maioria casos, o sistema é indeterminado. Uma solução particular para o sistema `Ax=b`, obtida pelo operador \ , é a solução dos mínimos quadrados. Seja

```
>> A = [2  1; 1 10; 1 2];
```

e considere o vector do segundo o membro o mesmo do exemplo anterior. Então,

```
>> x = A\b
x =
        0.5849
        0.0491
```

O resíduo da solução calculada é:

```
>> r = b - A*x
r =
        -0.1208
        -0.0755
         0.3170
```

Na prática, operador \ analisa a estrutura da matriz `A` e escolhe o método (directo) que considera mais indicado para resolver o sistema. Alternativas:

```
>> [L,U,P] = lu(A);    % factorização LU
>> R = chol(A);        % factorização de Choleski
```

A tabela seguinte lista algumas funções matriciais relevantes em Álgebra Linear Numérica.

`cond`	Número de condição relativamente à inversão
`det`	Determinante de uma matriz
`mean`	Média aritmética das componentes de um vector
`norm`	Normas de vectores e matrizes
`null`	Dimensão do espaço nulo de uma matriz
`rank`	Característica de uma matriz
`trace`	Traço de uma matriz
`chol`	Factorização de Choleski
`inv`	Inversa de uma matriz
`lu`	Factorização
`eig`	Valores e vectores próprios
`sum`	Soma das componentes de um vector
`svd`	Decomposição em valores singulares

Geralmente, funções do MATLAB também podem receber argumentos matriciais. A regra geral é que a matriz resultante tem a mesma forma que o argumento de entrada. Por exemplo:

```
>> sin([1 2 3])
ans =
    0.8415    0.9093    0.1411
```

Com base nesse comportamento, a codificação explícita de ciclos pode ser evitada em muitas situações em que seria necessário um sistema convencional linguagem de programação.

Exercícios

1. Defina um vector `y` com os 100 primeiros inteiros positivos pares. Extraia para um vector auxiliar todos os números nesse vetor correspondentes às posições $1, 2, 4, 8, 16, \ldots, 2n$, tal que $n \leq 6$.

2. Considere `A=[1 2; 3 4]`. Calcule: (a) $\texttt{A}^4$ e $\texttt{A}.^4$ (compare os resultados); (b) os valores próprios de `A`; (c) $\texttt{A}^{-1}$ e `1./A` (compare os resultados).

3. Considere um barra horizontal fixa numa extremidade e livre no restante do seu comprimento. Um modelo discreto de forças na barra conduz ao sistema de equações lineares $Ax = b$, onde A é a matriz quadrada de ordem n, com estrutura de banda, dada por

$$\begin{bmatrix} 6 & -4 & 1 & & & & \\ -4 & 6 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 6 & -4 \\ & & & & 1 & -4 & 6 \end{bmatrix}.$$

   O vector b é dado pela carga que é imposta à barra (incluindo o seu próprio peso), e o vector x representa a deformação da barra que queremos determinar. Considerando a barra sujeita a uma carga uniforme dada por $b_i = 1$, para $i = 1, \ldots, n$ e $n = 100$, resolva o sistema linear indicando o vector deformação.

O MATLAB possui uma grande variedade de recursos para traçar gráficos em 2D e 3D. Comecemos com uma breve descrição dos comandos mais importantes para a criação de gráficos a partir dos valores armazenados em matrizes.

O caso mais simples é aquele em que se pretende traçar um gráfico a partir de dados (x, y). Por exemplo, para gerar o gráfico da função $y = \sin(x)$ no intervalo $[0, 10]$ podemos fazer:

```
>> x = 0:.1:10;
>> y = sin(x);
>> plot(x,y)
```

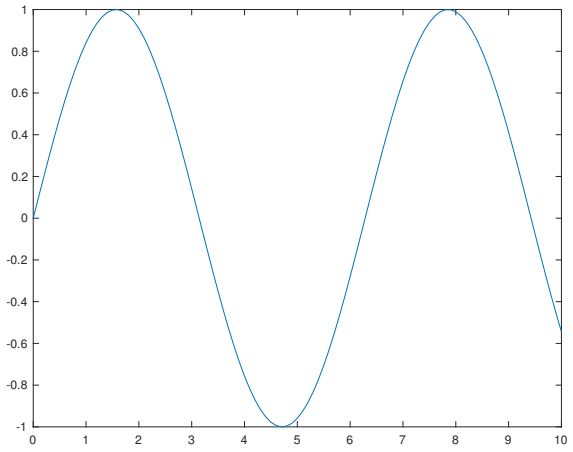


Figura 2: Saída de `plot(x,y)`.

A instrução `x = 0:.1:10;` define um vector com componentes a variar de 0 a 10 com espaçamentos de 0.1. Então, `y = sin(x)` define um vector cujas componentes são $\sin(0), \sin(0.1), \sin(0.2)$, etc. Finalmente, `plot(x,y)` usa os valores dos vectores `x` e `y` para construir o gráfico.

Vários tipos de linha, símbolos para os pontos e a cores podem ser definidas com `plot(X,Y,S)`, onde `S` é uma cadeia de caracteres (*string*) que consiste numa combinação das seguintes opções (existem outras):

cor		tipo de ponto		tipo de linha	
`r`	vemelho	`.`	ponto	`-`	sólida
`g`	verde	`o`	circunferência	`--`	tracejada
`b`	azul	`x`	cruz	`:`	pontilhada

Para colocar etiquetes num gráfico, usar os comandos `xlabel`, `ylabel` e `title`. Usando `axis` é possível controlar a escala e a aparência dos eixos. O exemplo a seguir demonstra o seu uso:

```
>> plot(x,y,'bo:'), title('sin(x)'), xlabel('x'), ylabel('y'), axis([0,2*pi,-1,1])
```

Para adicionar gráficos a uma figura existente, use o comando `hold on`. Até usar o comando `hold off` ou fechar a janela, todos os gráficos aparecerão na janela de figura actual. A função `plot` também admite a representação simultânea de várias curvas, acrescentando mais argumentos de entrada, devendo os vectores possuir o mesmo número de entradas.

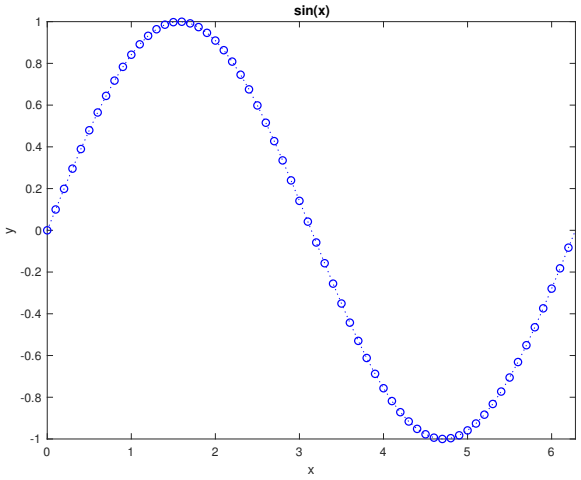


Figura 3: Saída de `plot(x,y,'bo:')`.

Os comandos

```
>> x = 0:.1:10;
>> y = sin(x); z = cos(x); w = y.*z;
>> plot(x,y,'-x',x,z,'-x',x,w,'-x')
>> legend(' y = sin(x) ',' y = cos(x) ',' y = sin(x)cos(x) ')
```

e

```
>> x = 0:.1:10;
>> y = sin(x); z = cos(x); w = y.*z;
>> plot(x,y,'-x')
>> hold on
>> plot(x,z,'-x'); plot(x,w,'-x')
>> legend(' y = sin(x) ',' y = cos(x) ',' y = sin(x)cos(x) ')
>> hold off
```

e

```
>> x = 0:.1:10;
>> M = [sin(x); cos(x); sin(x).*cos(x)];
>> plot(x,M,'-x')
>> legend(' y = sin(x) ',' y = cos(x) ',' y = sin(x)cos(x) ')
```

produzem todos a mesma figura.

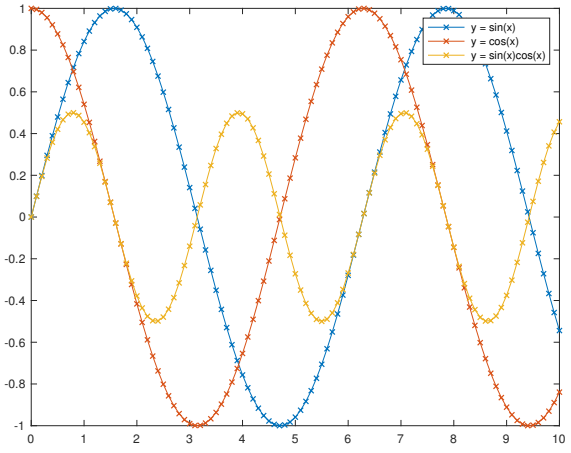


Figura 4: Saída de `plot(x,M,'-x')`.

Também é possível traçar curvas definidas parametricamente. O exemplo seguinte traça a circunferência unitária na sua representação paramétrica:

```
>> t = 0:.01:2*pi;
>> x = sin(t);
>> y = cos(t);
>> plot(x,y), axis square
```

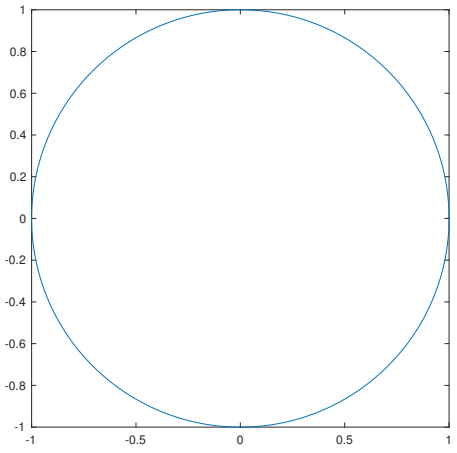


Figura 5: Saída de `plot(x,y)`.

Para além da função `plot`, o MATLAB tem ainda várias funções de desenho de gráficos, que se utilizam de forma semelhante mas cujo resultado é diferente. Na tabela em baixo podemos ver alguns exemplos:

`area`	Preenche a área debaixo de um gráfico xy
`bar`	Gráfico de barras verticais
`bar`	Gráfico de barras horizontais
`hist`	Histograma
`loglog`	Gráfico xy com a escala dos dois eixos logarítimicas
`polar`	Gráfico em coordenadas polares
`semilogx`	Gráfico xy com a escala do eixo horizontal logarítmica
`semilogy`	Gráfico xy com a escala do eixo vertical logarítmica

Para gerar os vectores para os gráficos com escalas logarítmicas convém gerar vectores cujo espaçamento entre pontos seja igualmente logarítmico. A função `logspace` facilita esta tarefa e tem a seguinte sintaxe: os dois primeiros argumentos são a potência de base 10 dos pontos inicial e final respectivamente, o terceiro argumento é o número de pontos. Se se pretender gerar, por exemplo, 15 pontos entre 10 e 100 com espaçamento logarítmico, pode-se utilizar o comando:

```
>> x = logspace(1,2,15);
```

Outra possibilidade mais conveniente de gerar o gráfico de uma dada função é usar o comando `fplot`. Para traçar o gráfico de $\sin(x)$ no intervalo $[0, 10]$ basta fazer:

```
>> fplot(@sin,[0,10]);
```

Aqui, o nome da função a ser traçada surge como o primeiro parâmetro em `fplot` precedida do símbolo `@`. Essa maneira de passar funções como argumento para outros comandos ou funções será explicada com mais detalhes na próxima ficha. Para traçar funções mais complicadas com o `fplot` pode usar-se a seguinte notação (mais detalhes na próxima ficha):

```
>> fplot(@(x) x.*sin(x),[0,10]);
```

Este comando traça o gráfico da função $y = x\sin(x)$ no intervalo $[0, 10]$.

O traçado de curvas em três dimensões pode ser feito com `plot3`, que funciona de forma análoga a `plot` em duas dimensões. Por exemplo

```
>> x = 0:.01:30*pi;
>> y = x.^2.*sin(x);
>> z = x.^2.*cos(x);
>> plot3(x,y,z);
```

traça uma espiral em três dimensões.

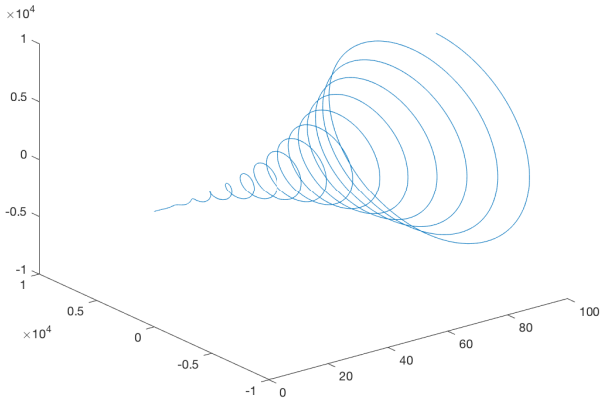


Figura 6: Saída de `plot3(x,y,z)`.

No exemplo seguinte, uma função de duas variáveis $z = \sin(x^2+y^2)$ é interpolada no quadrado $-1 \leq x \leq 1$, $-1 \leq y \leq 1$ usando um interpolador bilinear. Para isso, usa-se a opção `'linear'` na função `interp2` que permite obter o interpolador linear bidimensional.

```
>> [x, y] = meshgrid(-1:.25:1);
>> z = sin(x.^2 + y.^2);
>> [xi, yi] = meshgrid(-1:.05:1);
>> zi = interp2(x, y, z, xi, yi, 'linear');
>> surf(xi, yi, zi), title('Interpolador bilinear para sin(x^2 + y^2)')
```

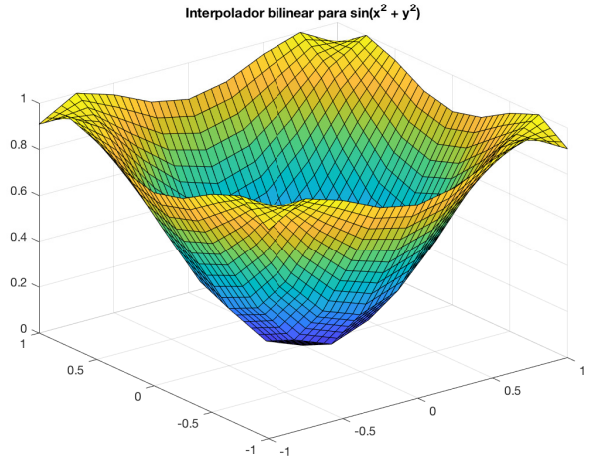


Figura 7: Saída de `surf(xi, yi, zi)`.

O comando `meshgrid(x,y)` é um comando muito útil que transforma o domínio especificado pelos vectores `x` e `y` em matrizes `X` e `Y` que pode ser usado para o cálculo de funções de duas variáveis e para o traçado gráficos de superfície 3D.

Usando a opção `shading` podemos controlar a aparência de uma superfície (fazer aparecer ou apagar as linhas da malha e controlar o sobreado).

```
>> [x,y,z] = peaks; surf(x,y,z), shading interp, hold on, contour3(x,y,z,20,'k'), hold off
```

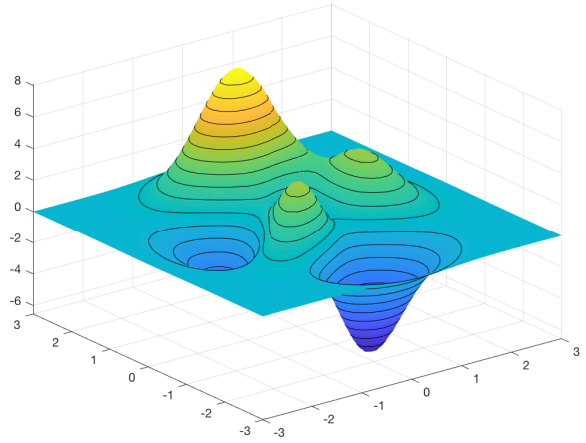


Figura 8: Saída de `peaks`.

Podemos exibir vários gráficos em diferentes sub-regiões da mesma janela usando a função `subplot`. As duas primeiras entradas da função indicam o número de gráficos em cada linha e cada coluna. A terceira entrada especifica qual o gráfico que está activo. O seguinte exemplo cria quatro gráficos numa matriz 2 por 2 numa figura.

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(4*cos(t));
>> subplot(2,2,1), mesh(X), title('X')
>> subplot(2,2,2), mesh(Y), title('Y')
>> subplot(2,2,3), mesh(Z), title('Z')
>> subplot(2,2,4), mesh(X,Y,Z), title('X,Y,Z')
```

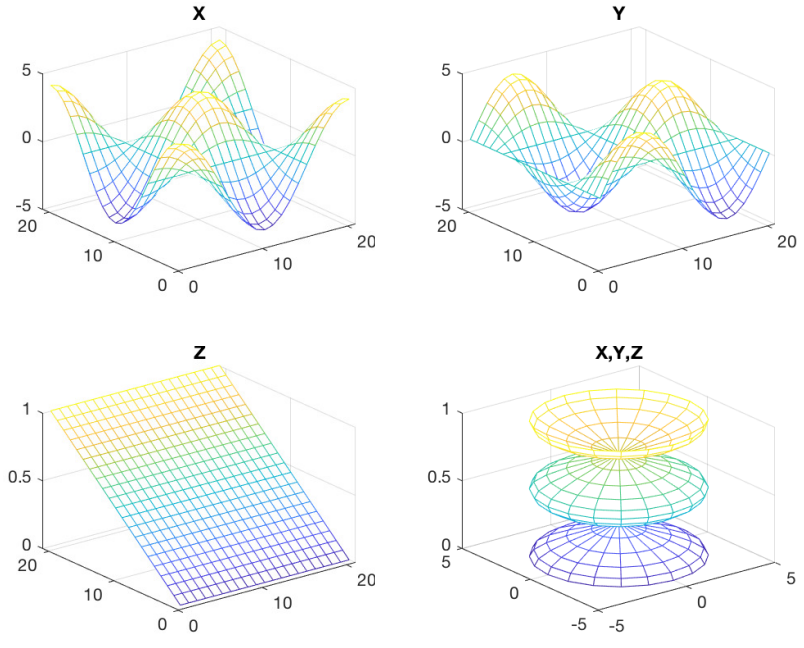


Figura 9: Saída de `subplot`.

Quando o argumento da função `plot` é complexo, na abcissa é colocada a parte real do vector e na ordenada a parte imaginária. Pode-se pensar neste processo como uma forma compacta de representação. O comando que se segue:

```
>> plot(z)
```

é equivalente a fazer

```
>> plot(real(z),imag(z))
```

Vejamos um exemplo em que se utiliza a função `axis square`, para que a figura obtida seja uma circunferência.

```
>> w = 0:pi/10:2*pi;
>> plot(exp(i*w),'-o')
>> axis square
```

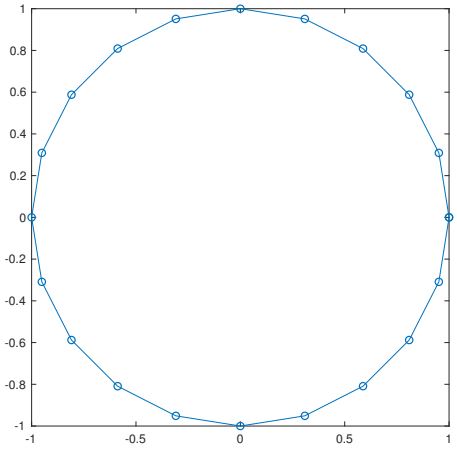


Figura 10: Saída de `plot3(x,y,z)`.

Exercícios

1. Faça o gráfico da função seno para dois períodos.
2. Compare o gráfico da função seno com o da função coseno para ângulos entre 0 e 2π e um passo de $\pi/150$. O seno deve ser representado por círculos azuis e o coseno por uma linha verde a cheio.
3. Desenhe o gráfico da função $\sin(\theta)/\theta$ com θ a variar de -2π a 2π e passos de $\pi/10$.
4. Visualize o gráfico da função e^x para 100 valores de $x \in [0, 5]$.
5. Visualize o gráfico da função e^{-x} para 100 valores de $x \in [0, 5]$.
6. Visualize o gráfico da função $\ln x$ para valores de $x \in [1, 5]$.
7. Construa uma matriz `X` em que a primeira coluna é constituída com 100 valores da função e^x e a segunda coluna por 100 valores da função $\ln x$ para o intervalo $x \in [1, 2]$. Execute o comando `plot(X)`.
8. Determine graficamente uma solução aproximada da equação $\ln x = e^{-x}$. Utilize para a análise o intervalo $[1/2, 2]$.
9. Desenhe no mesmo gráfico a função $e^{\alpha x}$ em que a variável α toma os seguintes valores $\alpha \in [1, 1.1, 1.2, 1.3, 1.4, 1.5]$ e a variável $x \in [0.1, 2]$.
10. Visualize o gráfico da função $\log x$ utilizando uma escala logarítmica para o eixo horizontal. Utilize 100 pontos em $[10^{-5}, 10^5]$.
11. Visualize em gráfico a parte real e a parte imaginária da função e^{jw}, para valores de $\omega \in [0, 2\pi]$ e passo de $\pi/10$.

O MATLAB inclui uma linguagem de programação moderna e interpretada. Os ficheiros que contêm código-fonte MATLAB são chamados de ficheiros-m (extensão `.m`). Existem dois tipos de ficheiros-m: ficheiros *script* (iremos usar a designação inglesa) e ficheiros função :

- Os ficheiros-m do tipo *script* não processam nenhum argumento. Ao digitar o nome do ficheiro (sem a extensão `.m`), os comandos contidos no ficheiro são executados como se tivessem sido digitado no teclado.
- Os ficheiros-m do tipo função contém uma linha com uma definição da `function`. Estes podem ter parâmetros/argumentos de entrada e parâmetros/argumentos de saída. Estas `function` podem ser chamados da mesma maneira do que qualquer outra função pré-definida do MATLAB.

Para criar um ficheiros-m pode usar o editor incorporado ao MATLAB (comando `edit`), e gravá-lo posteriormente numa directoria pessoal (recomenda-se que crie primeiro a directoria e inicie o `matlab` nessa directoria). Para visualizar qual a directoria corrente pode-se utilizar o comando `pwd`. Para mudar de directiroa pode-se utilizar o comando `cd`. Quando se escreve um comando na janela do MATLAB, esse nome será pesquisado em primeiro lugar na directoria corrente. Se não for encontrado, o MATLAB pesquisa nas directoriass que constarem duma lista interna (comando `path`).

Segue-se um exemplo de um pequeno *script*, que iremos guardar com o nome `programa1.m`:

```
% Este script corresponde a programa1.m
x = pi/100:pi/100:10*pi;
y = sin(x)./x;
plot(x,y), grid
```

Vamos analisar o conteúdo deste ficheiro. A primeira linha começa com o sinal de percentagem `%`. Esta é uma linha de comentário. Todos os comentários são ignorado pelo MATLAB; eles são adicionados para melhorar a legibilidade do código. Nas duas linhas seguintes, são criadas as matrizes `x` e `y`. Observe o ponto e vírgula no final dos dois comandos, usado para suprimir a exibição do conteúdo de ambos os vectores no ecrã. A matriz `x` contém 1000 números igualmente espaçados no intervalo $[\frac{\pi}{100}, 10\pi]$, enquanto a matriz `y` mantém os valores da função $y = \frac{\sin(x)}{x}$ nesses pontos. Lembre-se que o operador de ponto `.` antes do operador de divisão à direita `/` especifica a divisão componente a componente das matrizes `sin(x)` e `x`. O comando `plot` cria o gráfico da função `sin` usando os pontos gerados anteriormente. Por fim, o comando `grid` é executado. Este comando adiciona uma grelha ao gráfico.

Para executar o programa, basta digitar o seu nome na janela de comando e, de seguida, pressionar a tecla `<enter>`.

```
>> programa1
```

O programa também pode ser executado clicando na tecla .

A linha de comentário pode ser acedida através do comando `help` da seguinte forma:

```
>> help programa1
   Este script corresponde a programa1.m
```

Segue-se um exemplo de um ficheiro-m do tipo função:

```
function f_result = meufactorial(n)
% MEUFACTORIAL(n)  calcula n!
%                  MEUFACTORIAL(n) calcula n!=1*2*...*n
%                  de forma recursiva.
if n == 0
    f_result = 1;
else
    f_result = n*meufactorialt(n-1);
end
```

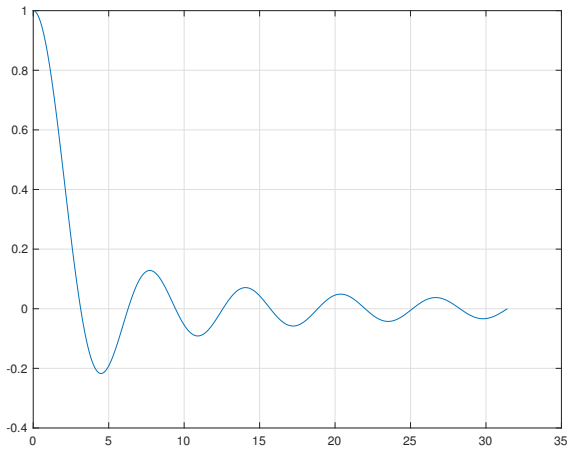


Figura 11: Saída de `programa1`.

Observe-se que a diferença entre o nome da função `meufactorial` e a variável que nos dá o resultado, ou parâmetro de de saída, `f_result`. Este exemplo também mostra que uma função MATLAB pode ser recursiva.

O nome da função deve coincidir com o nome do ficheiro-m na qual a função está armazenada. A função pode ser chamada da mesma maneira que uma função pré-definida, por exemplo.

```
>> x = meufactorial(5)
x =
        120
```

No exemplo anterior, a função considera um argumento/parâmetro de entrada e devolve o valor da função (ou argumento/parâmetro de saída). No caso geral, podemos considerar vários argumentos de entrada e também vários argumentos de saída.

Para além disso, é muitas vezes útil definir uma função que possa ser chamada com um número variável de argumentos de entrada ou de saída. As funções do MATLAB `nargin` e `nargout` devolvem o número de argumentos realmente considerados. O exemplo a seguir ilustra a definição de uma função com um número variável de argumentos (usa as funções predefinidas `sum` e `cumsum` e o controlador de fluxo `if`, que será explicado com mais detalhe na próxima ficha):

```
function [s,sa] = vecsum(x,absval)
% VECSUM  soma flexivel de elementos de um vector.
%         VECSUM(X) devolve a soma dos elementos de X.
%         [S, SA] = VECSUM(X) devolve a soma e a
%         soma acumulada de dos elementos de X.
%         Se VECSUM for chamada com dois argumentos de entrada
%         (X, ABSVAL) e o valor de ABSVAL for diferente de 0,
%         usa o vector ABS (X) em vez de X.
if (nargin > 1 && absval ~= 0)
   s = sum(abs(x));
   if nargout > 1, sa = cumsum(abs(x)); end
else
   s = sum(x);
   if nargout > 1, sa = cumsum(x); end
end
```

The função `vecsum` pode, por exemplo, ser chamada nas seguintes formas:

```
>> x = [1 -2 3];
>> vecsum(x,1)
ans =
     6
```

ou

```
>> [soma, soma_acumulada] = vecsum(x)
soma =
     2
soma_acumulada =
     1    -1     2
```

Uma técnica muito importante para escrever funções em MATLAB é a chamada *vectorização*. Da mesma forma que nas funções pré-definidas, as funções definidas pelo utilizador podem ser escritas de tal maneira que aceitam automaticamente argumentos do tipo matriz e devolvem matrizes como parâmetros de saída. Considere a seguinte função:

```
function f = quot(x)
f = 1/(1+x);
```

A função `quot` pode apenas ser chamada com o argumento de entrada `x` escalar. O cálculo desta função para vários elementos de uma matriz é apenas possível usando um ciclo `for`. Uma alternativa mais eficiente consiste em considerar a versão vectorizada `quotv`:

```
function f = quotv(x)
f = 1./(1.+x);
```

Esta função pode ser chamada tanto com argumentos escalares como matriciais:

```
>> x = linspace(1,5,5)
x =
     1     2     3     4     5

>> quotv(x)
ans =
    0.5000    0.3333    0.2500    0.2000    0.1667
```

Uma questão essencial em programação é o tratamento de erros. Em MATLAB, esse tratamento pode ser realizado de forma conveniente por meio da função `error`. Por exemplo, a função `error` é frequentemente usada para verificar a validade dos argumentos transmitidos para uma função, como no exemplo seguinte:

```
function f = factorial(n)
if n < 0 error('erro: n tem que ser maior ou igual a zero') end
...
```

Uma chamada da função `error` emite a mensagem de erro especificada e finaliza o procedimento.

Uma função anónima (do inglês *anonymous function*) é uma forma simples de definir uma função numa única instrução. Consiste numa única expressão MATLAB e pode ter um qualquer número de argumentos de entrada e saída (embora, geralmente, o número de parâmetros de saída seja apenas um). Pode definir-se uma função anónima diretamente na linha de comando ou dentro de uma função ou *script*. Isso fornece uma forma rápida de criar funções simples sem precisar criar um ficheiro-m. A sintaxe para criar uma função anónima a partir de uma expressão é `expressão f = @(argumentos)`. O exemplo seguinte cria uma função anónima que devolve o quadrado de um número. Quando se chama essa função, o MATLAB atribui o valor do parâmetro de entrada à variável `x` e, em seguida, usa `x` na expressão `x.^ 2`:

```
>> sqr = @(x) x.^2;
>> a = sqr(5)
a =
    25
```

Considere, agora,

```
>> f = @(x,y) sqrt(x.^2+y.^2)
f =
  function_handle with value:
    @(x,y)sqrt(x.^2+y.^2)
```

Podemos calcular o valor desta função da forma habitual:

```
>> f(3,4)
ans =
        5
```

Esta função também pode ser usada com argumentos matriciais:

```
>> A = [1 2;3 4];
>> B = ones(2);
>> C = f(A, B)
C =
    1.4142    2.2361
    3.1623    4.1231
```

Algumas funções podem usar outras funções como parâmetro, como o caso da função `fplot` que vimos na ficha anterior. Como vimos, o comando `fplot(@sin,[0,10])` permite traçar o gráfico da função seno no intervalo $[0, 10]$. Se quisermos traçar o gráfico de uma função anónima, podemos proceder da seguinte forma:

```
>> f = @(x) x.^2;
>> fplot(f,[0,5]);
```

As funções definidas num ficheiro-m também aceitam outras funções como parâmetro, como é exemplificado na função seguinte:

```
function [m1, m2] = fminmax(f,p)
% calcula o minimo e o maximo de uma funcao (f)
% sobre uma determinada matriz de pontos (p)
% usando as funcoes do MATLAB min e max
if (nargin == 2 && nargout == 2)
  t = feval(f,p);
  m1 = min(t);
  m2 = max(t);
else
  error('fminmax requer dois argumentos de entrada e dois de saida');
end
```

Note-se o uso da função `error` no exemplo anterior.

Exemplos de como chamar a função acima durante uma sessão MATLAB:

```
>> [m1,m2] = fminmax(@sin,0:1:10);
```

ou

```
>> f = @(x) abs(x.^2-2.*x+1);
>> [m1,m2] = fminmax(f,-5:1:5);
```

Exercícios

1. Considere a função $f(x) = \sin(x) + x^2 + 3$. Calcule uma aproximação do integral $\int_a^b f(x)dx$, com $a = 0$ e $b = 100$, usando a regra dos trapézios

$$\int_a^b f(x)dx \approx \frac{h}{2}(f(a) + 2f(a+h) + 2f(a+2h) + \cdots + 2f(b-2h) + 2f(b-h) + f(b)),$$

com $h = (b-a)/n$ e $n = 100$. Usando o comando `area` visualize a área debaixo da curva.

2. Considere a função

$$f(x) = 1/(1+x^2)$$

definida no intervalo $[-5, 5]$.

   (a) Para $n = 4, 8, 12, 16$ e 20, faça o gráfico do polinómio que interpola f em $n+1$ pontos equidistantes no intervalo $[-5, 5]$. Observe o que acontece quando n aumenta.

   (b) Repita o procedimento mas usando, em vez de pontos equidistantes, os pontos

$$x_k = 5\cos\left(\frac{(2k-1)\pi}{2(n+1)}\right),\ k = 1, 2, \ldots, n+1,$$

   que são designados por nós de Chebyshev. Observe também o que acontece quando n aumenta.

3. Considere a matriz $H_n = (h_{ij})_{i,j=1}^n$ tal que

$$h_{ij} = \frac{1}{i+j-1}, \quad i, j = 1, \ldots, n$$

chamada matriz de Hilbert de ordem n (em MATLAB a matriz H_n é dada por `hilb(n)`).

   (a) Fazendo n variar entre 2 e 10, determine o número de condição (relativamente à norma $||\cdot||_2$) da matriz H_n e represente os resultados graficamente.

   (b) Supondo que pretende resolver o sistema $H_n x = b$, onde b é determinado tal que $x = (1, 1, \ldots, 1)$ é a solução do sistema, determine como evolui o erro relativo da solução numérica calculada $\bar{x}$ em função de n (faça n variar entre 2 e 10).

4. Execute a sequência de comandos:

```
f = @(x)(x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-21*x.^2+7*x-1); fplot(f,[1-2e-4, 1+2e-4])
```

   De que forma o resultado contraria o teorema fundamental da álgebra? Execute, agora, a sequência de comandos

```
f = @(x)(x-1).^7; fplot(f,[1-2e-4, 1+2e-4])
```

   O que pode concluir?

Para controlar o fluxo de comandos, podemos usar as seguintes estruturas: ciclos `for`, ciclos `while`, `if-else-end` e `switch-case`. Elas podem ser usados no modo interactivo, mas o seu principal objetivo é a programação usando ficheiros-m.

A sintaxe para os ciclos `for` é a seguinte:

```
for k = matriz
    comandos
end
```

Os comandos entre `for` e `end` são executados para todos valores armazenados em `matriz`. Note-se que a instrução `end` é sempre necessária, mesmo quando o corpo do ciclo contém apenas um comando.

Como exemplo, suponha que se pretende obter os valores da função `sin` nos 11 pontos igualmente distanciados $\pi\frac{n}{10}$, for $n = 0, 1, \ldots, 10$. Para isso, podemos fazer:

```
for n = 0:10
    x(n+1) = sin(pi*n/10);
end

x =
   Columns 1 through 8
         0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511    0.8090
   Columns 9 through 11
    0.5878    0.3090    0.0000
```

Os ciclos podem ser encaixados:

```
H = zeros(5);
  for k = 1:5
    for l = 1:5
        H(k,l) = 1/(k+l-1);
    end
  end

H =
        1.0000   0.5000   0.3333   0.2500   0.2000
        0.5000   0.3333   0.2500   0.2000   0.1667
        0.3333   0.2500   0.2000   0.1667   0.1429
        0.2500   0.2000   0.1667   0.1429   0.1250
        0.2000   0.1667   0.1429   0.1250   0.1111
```

A matriz `H` obtida é chamada de matriz de Hilbert. A primeira instrução atribui um espaço na memória para a matriz ser gerada. Essa instrução é adicionada para reduzir a sobrecarga que é requerida pelos ciclos em MATLAB. Esta pré-reserva de memória é importante para melhorar a eficiência do código.

Uma regra geral é que os ciclos `for` devem ser usados apenas quando os métodos *vectorizados* não podem ser aplicados. Se pretendermos gerar uma matriz `A` de ordem 10 de entradas $a_{k\ell} = \sin(k)\cos(l)$, podemos usar ciclos `for` encaixados usando o seguinte código:

```
A = zeros(10);
for k = 1:10
    for l = 1:10
        A(k,l) = sin(k)*cos(l);
    end
end
```

No entanto, a versão sem ciclos

```
k = 1:10; A = sin(k)'*cos(k);
```

revela-se muito mais eficaz (ver última ficha). Esta técnica deve ser usada sempre que possível.

A sintaxe dos ciclos `while` é a seguinte:

```
while expressão
    comandos
end
```

Este ciclo é usado quando o número de repetições não é determinado *a priori*. Apresenta-se um pequeno problema que requer este tipo de ciclo: o código seguinte executa várias iterações do método de Newton para encontrar o valor $\sqrt[3]{2}$, ou seja, a solução da equação $x^3 - 2 = 0$. O número de iterações necessárias para satisfazer o requisito de tolerância $|x^3 - 2| <$ `tol` não é conhecido *a priori*.

```
format long;
x = input('Valor inicial = ');
tol = input('Tolerancia para o erro = ');
disp(x);
while (abs(x^3-2) > tol)
   x = (2+2*x^3)/(3*x^2);
   disp(x);
end
```

```
>> iter
Valor inicial = 1
Tolerancia para o erro = 1e-12
   1.00000000000000
   1.33333333333333
   1.26388888888889
   1.25993349344998
   1.25992105001777
   1.25992104989487
```

A sintaxe da forma mais simples da estrutura `if-else-end` é a seguinte:

```
if expressão
    comandos
end
```

Esta forma é usada se houver apenas uma alternativa. Duas alternativas requerem a seguinte construção:

```
if expressão
    comandos (executados se expressão é verdadeira)
else
    comandos (executados se expressão é falsa)
end
```

Se existem várias alternativas, escrevemos:

```
if expressão1
    comandos (executados se expressão1 é verdadeira)
elseif expressão2
   comandos (executados se expressão2 é verdadeira)
elseif ...
  .
  .
  .
else
    comandos (executado se todas as expressões anteriores forem falsas)
end
```

As comparações são realizadas com o auxílio dos seguintes operadores:

Operador	Descrição
`<`	menor do que
`<=`	menor ou igual a
`>`	maior do que
`>=`	maior ou igual a
`==`	igual a
`~=`	diferente de

Ao lidar com expressões lógicas, observe que 0 representa falso e 1 representa verdadeiro:

```
[3 == 4,5 == 5]
ans =
     0     1
```

Os operadores lógicos são:

Operador	Descrição
`&`	operador lógico e
`&&`	curto circuito do operador lógico e
`|`	operador lógico ou
`||`	curto circuito do operador lógico ou
`~`	operador lógico de negação
`xor`	operador lógico ou exclusivo
`all(x)`	1 (verdade) se todos os elementos do vector `x` são nulos
`any(x)`	1 (verdade) se algum elemento do vector `x` for nulo

Os operadores `&&` e `||` dizem-se um curto circuito dos operadores `&` e `|`, respectivamente, pois quando usados na forma `A && B` ou `A || B` apenas calculam `B` caso o valor lógico da expressão não seja imediatamente determinado por `A`.

Considere-se o seguinte exemplo. Os polinómios de Chebyshev de primeira espécie $T_n(x)$, $n = 0, 1, \ldots$, são de grande importância em Análise Numérica. Eles são definidos recursivamente na seguinte forma:

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x),\ n = 2, 3, \ldots, \quad T_0(x) = 1, \quad T_1(x) = x.$$

A função `ChebT` seguinte calcula os coeficientes dos polinómios de Chebyshev de primeira espécie e armazena-os em vectores linha de acordo com a forma como o MATLAB representa polinómios (ver Ficha 4).

```
function T = ChebT(n)
% Coeficientes T do n-esimo polinomio de Chebyshev de  especie.
% Eles sao armazenados por ordem decrescente da potencia correspondente.
t0 = 1; t1 = [1 0];
if n == 0
    T = t0;
elseif n == 1;
    T = t1;
else
    for k = 2:n
        T = [2*t1 0] - [0 0 t0];
        t0 = t1;
        t1 = T;
    end
end
```

Os coeficientes do polinómio cúbico de Chebyshev de primeira espécie são:

```
>> coef = ChebT(3)
coef =
        4    0    -3    0
```

Então, $T_3(x) = 4x^3 - 3x$.

A sintaxe da estrutura `switch-case` é a seguinte:

```
switch expressão (escalar ou cadeira de caracteres)
    case value1 (executa se a expressão é avaliada como valor1)
        comandos
    case value2 (executa se a expressão é avaliada como valor2)
        comandos
      . . .
    otherwise
        comandos
end
```

A forma `switch` compara a expressão de entrada com o valor de cada caso. Uma vez encontrado, ele executa os comandos associados. No exemplo seguinte, é gerado um número inteiro aleatório x do conjunto $\{1, 2, \ldots, 10\}$. Se $x = 1$ ou $x = 2$, a mensagem 'Probabilidade = 20%' é exibida no ecrã. Se $x = 3$ ou 4 ou 5, a mensagem 'Probabilidade = 30%' é exibida, caso contrário é exibida a mensagem 'Probabilidade = 50%' . O ficheiro *script* `fswitch` utiliza uma opção como ferramenta para lidar com todos casos mencionados acima:

```
% Ficheiro script fswitch.
x = ceil(10*rand); % Gera um inteiro aleatorio em {1,2,...,10}
switch x
    case {1,2}
        disp('Probabilidade = 20%');
    case {3,4,5}
        disp('Probabilidade  = 30%');
    otherwise
        disp('Probabilidade  = 50%');
end
```

Observe o uso de chavetas depois da palavra `case`. Isso cria um matriz de células em vez de uma matriz unidimensional (ver a próxima ficha).

Exercícios

1. Faça um *script* para comparar uma versão que use ciclos com a forma vectorizada para os seguintes casos.

   (a) Gere uma sequência com 10000 valores aleatórios com distribuição uniforme e guarde num vector apenas os que possuem amplitude superior a 0.9. Coloque noutro vector os índices das amostras anteriores ciclos

   (b) Obtenha o gráfico da função $\sin(x)/x$ variando o argumento entre -100 e 100 com passos de 0.1.

2. Os polinómios de Legendre $P_n(x)$, $n = 0, 1, \ldots$, são definidos recursivamente por

$$P_0(x) = 1, P_1(x) = x$$
$$nP_n(x) = (2n-1)xP_{n-1} - (n-1)P_{n-2}(x), \qquad n = 2, 3, \ldots.$$

   Escreva uma função `P = LegendP(n)` no MATLAB que tenha como parâmetro de entrada um inteiro n – o grau de $P_n(x)$ – e devolva os coeficientes do polinómio de Legendre correspondente por ordem decrescente das suas potências.

3. Escreva e teste uma função `newton` no MATLAB que resolva uma equação não linear $f(x) = 0$ usando o *método de Newton*: dada uma aproximação inicial x_0, obtenha as aproximações:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0, 1, \ldots.$$

Pare quando $|x_{i+1} - x_i|$ for inferior a uma dada tolerância `tol` ou quando um número máximo de iterações `maxit` for atingido.

As especificações de `newton` dever ser as seguintes:

```
function [x,y] = newton(fun, fun_d, x0 ,tol, maxit)
%NEWTON  Determina o zero mais próximo de x0 usando o método de Newton
%
%   fun             especifica a função f
%   fun_d           especifica a derivada de f
%   x0              aproximação inicial
%   tol             tolerância para o erro
%   maxit           número máximo de iterações
%
%   x               vector (linha) das aproximações ao zero de f
%   y               vector (linha) dos valores da função fun(x)
```

4. A sucessão

$$x^{(2)} = 2, \qquad x^{(k+1)} = 2^{k-1/2}\sqrt{1 - \sqrt{1 - 4^{1-k}(x^{(k)})^2}}, \qquad k = 2, 3, \ldots,$$

converge para π quando n tende para infinito. Mostre que o erro relativo da aproximação $x^{(k)} \approx \pi$ diminui nas primeiras 16 iterações e cresce nas seguintes. Para isso, execute a sequência de comandos:

```
x = 2; erro(1) = abs(pi-x)/pi;
for k = 2:30
    x = 2^(k-1/2)*sqrt(1-sqrt(1-4^(1-k)*x^2));
    erro(k) = abs(pi-x)/pi;
end;
semilogy(erro)
```

5. Escreva um programa para calcular a sucessão

$$I^{(0)} = \frac{1}{e}(e - 1), \qquad I^{(n+1)} = 1 - (n + 1)I^{(n)}, \qquad n = 0, 1, \ldots.$$

Compare o resultado obtido com o valor exacto do limite $\lim_{n\to\infty} I^{(n)} = 0$.

6. Qual das aproximações de π

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\right),$$

$$\pi = 6\left(0{,}5 + \frac{(0{,}5)^3}{2 \cdot 3} + \frac{3(0{,}5)^5}{2 \cdot 4 \cdot 5} + \frac{3 \cdot 5(0{,}5)^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{3 \cdot 5 \cdot 7(0{,}5)^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} + \cdots\right),$$

é mais sensível à propagação de erros de arredondamento? Compare, usando o MATLAB, os resultados em função do número de termos que considera em cada soma.

7. Considere o seguinte algoritmo para calcular o valor de π. Gere n pares de números aleatórios $\{(x_k, y_k)\}$ no intervalo $[0, 1]$ e calcule o número m desses pares que se encontram no interior do primeiro quadrante do círculo unitário. Obviamente, o valor de π será dado pelo limite da sucessão $x^{(n)} = 4m/n$. Escreva um programa MATLAB para calcular essa sucessão e o erro relativo que se comete na aproximação $x^{(n)} \approx \pi$ à medida que o valor de n aumenta.

8. Escreva um programa para calcular o coeficiente binomial $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, sendo n e k dois números naturais com $k \leq n$.

9. Dado um número inteiro positivo n, apliquemos-lhe o seguinte procedimento:

   - se n é par, dividimo-lo por dois;
   - se n é impar, multiplicamo-lo por 3 e somamos um.

   Repetimos indefinidamente parando apenas se chegarmos ao número um. O número de passos que demoramos a atingir um é denotado por `collatz(n)`. A *conjectura de Collatz*, proposta por Lothar Collatz em 1937, sugere que `collatz(n)` é sempre finito, ou seja, que o procedimento acaba sempre por parar. A conjectura foi verificada experimentalmente até 5×2^{60}, mas o problema continua em aberto.

   (a) Programe a função `collatz(n)`, que recebe como input um inteiro n e devolve o número de passos que a sequência demora a atingir 1.

   (b) Programe a função `collatz_graf(n)`, que recebe como input um inteiro n e desenha o gráfico valores sucessivos da sequência que começa em n até atingir 1.

   (c) Faça um gráfico dos valores de `collatz(n)` com n a variar de 1 a 10000.

10. Dada uma matriz A quadrada n por n, uma forma de reduzir o problema de calcular o seu determinante ao de calcular determinantes de matrizes $(n-1) \times (n-1)$ é dada pela expansão de Laplace aplicada à primeira linha, que nos diz que

$$\det(A) = \sum_{i=1}^{n} (-1)^{1+i} A_{1,i} \det(B_{1,i})$$

    onde $B_{1,i}$ é a matriz obtida a partir de A apagando a primeira linha e a coluna i.

    (a) Implemente uma função recursiva `determinante(A)` que calcule recursivamente o determinante de uma matriz A usando a fórmula indicada acima.

    (b) Gere uma matriz aleatória 6×6 e compara o tempo que a função `determinante(A)` demora a calcular o seu determinante com o tempo que demora a função $\det(A)$ pré-definida no MATLAB. O que acontece quando o tamanho da matriz aumenta?

11. Comente a eficácia, no que diz respeito à propagação de erros de arredondamento, dos seguintes códigos MATLAB para o cálculo de $f(x) = (e^x - 1)/x$ para $|x| \ll 1$.

```
% Algoritmo 1
if x == 0
f = 1;
else
f = (exp(x) - 1) / x;
end
```

```
% Algoritmo 2
y = exp (x);
if y == 1
f = 1;
else
f = (y - 1) / log (y);
end
```

12. Considere a função $\sin : [-\pi, \pi] \longrightarrow \mathbb{R}$.

    (a) Escreva uma função em MATLAB para calcular o polinómio de Taylor de grau n da função sin, em torno do ponto $x_0 = 0$.

    (b) Considerando $n = 0, 1, 2, 3, \ldots$ na função anterior, calcule aproximações para $x = \frac{\pi}{4}$.

    (c) Calcule estimativas para os erros absoluto e relativo respeitantes às aproximações obtidas na alínea anterior.

    (d) Represente graficamente os erros calculados em função do grau polinomial considerado.

O MATLAB permite o desenvolvimento rápido de algoritmos de processamento, mas quando os dados a tratar possuem uma dimensão elevada é necessário ter alguns cuidados para que a execução do código não se torne demasiado lenta. Como vimos anteriormente, existem duas formas principais de optimizar a execução do código: a reserva de memória para as variáveis e a *vectorização* de algoritmos.

A reserva de espaço em memória para as variáveis faz-se simplesmente preenchendo a variável a ser utilizada com zeros, de forma a que a sua dimensão não volte a ser alterada. No código que se segue o vector `x` vê a sua dimensão ser alterada, o que leva a um novo pedido de reserva de memória para armazenar o vector

```
>> x = 1:5;  x = [x x];
```

o que não acontece nesta versão que apesar de mais eficiente é menos legível

```
>> x = zeros(1,10);  x(1:5)= 1:5;  x(6:10)= x(1:5);
```

Mas vejamos que diferença pode ocorrer nos tempos de processamento com um pequeno programa, em que se testam duas versões de um algoritmo muito simples, em que na primeira não se reserva memória para a variável.

```
N = 10000;
x = rand(1,N);
% Versao sem reserva de memoria
t = cputime;
y = 0;
for n = 1:N,
    y = [y x(n)];    % Instrucao critica
end
tsem = cputime-t
clear y
%
% Versao com reserva de memoria
t = cputime;
y = zeros(1,N+1);
for n = 1:N,
    y(n+1) = x(n);
end
tcom = cputime-t
% Velocidade relativa
% Num processador 1,6 GHz Intel Core i5 a versao com reserva de memoria
% foi cerca de 10 vezes mais rapida
rel = tsem/tcom
```

Eis mais alguns conselhos úteis sobre a gestão de memória no MATLAB.

- Evitar criar variáveis intermédias dos dados quando não são necessárias utilizando antes variáveis da mesma dimensão já existentes.
- Caso tal não seja possível utilizar a função `clear var1 var2 ....`
- Deve-se ter em conta que o MATLAB utiliza 8 bytes para armazenar cada elemento de uma matriz. Como no caso das variáveis complexas esse número duplica, deve-se evitar ter este tipo de variáveis sempre que a parte real ou a imaginária sejam nulas. Nestes casos deve utilizar-se as funções `real` e `imag`.
- No caso de ser necessário processar dados com muitas amostras deve-se realizar o processamento por blocos com as seguintes etapas: (i) ler bloco do ficheiro de entrada; (ii) processar bloco; (iii) escrever o bloco com o resultado no ficheiro de saída.

Esta técnica apesar de permitir um aumento da rapidez de execução, resulta normalmente num algoritmo mais complexo. Outra técnica consiste na *vectorização* dos algoritmos.

Todos os operadores e funções do MATLAB podem ser aplicados sobre vectores e matrizes. Se, por exemplo, quisermos calcular o seno de cada um dos elementos do vector `x = 1:100` basta fazer `v = sin(x)` para se obter o resultado pretendido. O MATLAB permite assim utilizar como argumento da função `sin` não apenas um número mas um vector, aplicando a função a cada um dos seus elementos. No entanto, existe uma outra forma de realizar os cálculos que consiste em recorrer ao ciclo for aplicando a função `sin` a um elemento de `x` de cada vez:

```
>> for n = 1:100
>>    v(n) = sin(x(n));
>> end
```

Contudo, o MATLAB utiliza uma linguagem interpretada, o que significa que antes de ser executada cada linha de código é lida e descodificada. Este processo é repetido para cada linha e pode em certos casos, ser da ordem de grandeza do próprio processamento ou mesmo superior. Um caso ilustrativo é o do cálculo do produto interno entre dois vectores linha a e b, definido pela expressão

$$a \cdot b = \sum_{n=1}^{N} a(n)b(n)$$

e que em linguagem do MATLAB se escreve simplesmente `a*b'`.

O programa que se segue pretende demonstrar a diferença de velocidade de processamento entre a versão que calcula o produto interno fazendo uso de um ciclo `for`, e outra que o calcula na forma vectorizada.

```
% Comparação do tempo de CPU no cálculo de um produto interno.
N = 10000;
a = rand(1,N);
b = rand(1,N);
nciclos = 100;
% Cálculo utilizando um ciclo for
t = cputime;
for c = 1:nciclos,
       y = 0;
       for n = 1:N,
           y = y+a(n)*b(n);
       end
end
tfor = cputime-t
% Cálculo de forma vectorizada
t = cputime;
for c = 1:nciclos,
      y = a*b';
end
tmat= cputime-t
% Velociade de cálculo relativa
% Num processador 1,6 GHz Intel Core i5 a versão com reserva de memória
% foi cerca de 30 vezes mais rápida
rel= tfor/tmat
```

Pelo exemplo anterior pode-se concluir que a *vectorização* dos algoritmos é a única forma de utilizar o MATLAB de forma eficiente.

Há muitas coisas relativas ao MATLAB que não foram referidas nesta breve introdução. Esta ficha é dedicada a algumas delas, mas muitas outras continuam a faltar. As referências bibliográficas no final da ficha permitem, aos mais interessados, continuar a explorar o MATLAB.

Cadeias de caracteres

Os dados de texto (cadeias de caracteres ou *strings*) podem ser gerados usando plicas como delimitadores e podem ser atribuídos a variáveis da maneira usual:

```
>> s = 'Para a frente é que é caminho'
s =
    'Para a frente é que é caminho'
```

As plicas dentro de uma sequência são representadas por um par de plicas:

```
>> 'P''ra a frente é que é caminho'
ans =
    'P'ra a frente é que é caminho'
```

Matrizes de células e estruturas

As *matrizes de células* são matrizes que contêm elementos de tipos arbitrários. Elas são identificadas por chavetas em vez de parêntesis rectos:

```
>> c = {[3,4],18.2,[1,2;2,1],'string'};
```

define a matriz de células `c`. Podemos aceder aos elementos de `c` ca seguinte forma:

```
>> c{3}
ans =
     1   2
     2   1
```

Uma *estrutura* é uma matriz de várias variáveis onde cada uma tem seu próprio tipo e identificador.

```
>> p.pol = 'x^2-3*x+2'; p.coef = [1,-3,2]; p.zeros = [1,2]; p.min = [-1/4]
p =
   struct with fields:
      pol: 'x^2-3*x+2'
      coef: [1 -3 2]
      zeros: [1 2]
      min: -0.2500
```

define uma (matriz de) estrutura `p`. Os elementos da estrutura são acedidos por `estrutura.identificador`, por exemplo:

```
>> p.zeros
ans =
     1     2
```

Manipulação de entrada, saída e uso de ficheiros

O comando `disp` pode ser usado para exibir expressões ou o conteúdo das variáveis no ecrã durante a execução de funções ou ficheiros *script*:

```
>> disp('cadeia de caracteres');
cadeia de caracteres

>> A = [3,2;2,3];
>> disp(A);
A =
     3     2
     2     3
```

O comando `input` pode ser usado para solicitar o valor de uma variável ao utilizador. Por exemplo, a instrução `r = input('Valor de r = ');` exibe `Valor de r =` no ecrã e aguarda que o utilizador insira uma expressão a ser atribuída a `r`.

Para exibir uma saída formatada no ecrã, pode usar-se a função `fprintf`, com uma sintaxe semelhante à usada na linguagem C:

```
>> fprintf ('aproxiamção para pi: %6.4f\n',pi)
aproxiamção para pi: 3.1416
```

Também é possível manipular entradas e saídas com ficheiros externos. Aqui vamos apenas fazer uma breve descrição de alguns dos comandos mais importantes do MATLAB.

- `ID = fopen(nome, permissão)` abre um ficheiro especificado por `nome` com as permissões especificadas por `permissão ('r'..read, 'w'..write, 'a'..append, ...)` e um valor é guardado em `ID` com o qual o ficheiro pode ser acedido mais tarde.
- `fclose(ID)` fecha o ficheiro especificado por `ID`.
- `fprintf(ID, formato, A,...)` grava dados formatados no ficheiro especificado por `ID`. A opção `formato` é uma cadeira de caracteres que contém as especificações de formatação.
- `[A, count] = fscanf(ID, formato, tamanho)` lê dados formatados do ficheiro especificado por `ID`. A cadeira de caracteres `formato` contém as especificações de formatação dos dados. O parâmetro `tamanho` coloca um limite no número de elementos a serem lidos do ficheiro (opcional). Os dados lidos no ficheiro são armazenados em `A`. Se o parâmetro de saída (opcional) `count` for usado, é atribuído o número de elementos lidos com sucesso.

É possível criar uma tabela em MATLAB a partir de um ficheiro, usando o comando `readtable`, e de uma matriz numérica, usando o comando `readmatrix`.

- `A = readmatrix (ID)` cria uma matriz `A` lendo dados (numéricos) orientados por coluna do ficheiro especificado por `ID` . Por exemplo, o comando `M = readmatrix('dados.xlsx','Sheet','2020')` atribui à matriz `M` os valores contidos no ficheiro excel `dados.xlsx`, na folha `2020`.

Controle de tempo

Para comparar diferentes algoritmos que foram programados para resolver o mesmo problema, é geralmente útil avaliar o tempo necessário executar cada um. Isso pode ser feito usando os comandos `tic` e `toc`. O comando `tic` inicia um cronómetro, `toc` lê o tempo do cronómetro e devolve o tempo decorrido (em segundos).

Para executar um conjunto de comandos e exibir o tempo decorrido, use:

```
tic;
...
comandos;
...
toc
```

Os comandos LOAD e SAVE

Com estes comandos é possível guardar num ficheiro variáveis do MATLAB que estejam na área de trabalho. O comando `save` permite guardar as variáveis para disco e o comando `load` permite carregá-las para a área de trabalho. O exemplo seguinte ilustra a funcionalidade destes comandos.

```
>> A = rand(2)
A=
    0.9932    0.8987
    0.8426    0.5093
>> b = rand(1,3)
b =
    0.7540    0.9537    0.7044
>> save ficheiro A b
>> load ficheiro
```

Os comandos `load` e `save` permitem igualmente guardar ficheiros em formato ASCII.

```
>> save ficheiro A b -ascii
```

sendo mais fácil para o utilizador verificar o conteúdo com um simples editor de texto.

Imagens em MATLAB

O MATLAB possui um conjunto de funções para manipulação e visualização de imagens a cores ou em níveis de cinzento. As imagens são representadas sob a forma de matrizes. No caso de imagens só com níveis de cinzento, estas podem ser armazenadas numa matriz bidimensional em que cada elemento representa o nível de cinzento. Os seguintes comandos mostram como visualizar uma imagem armazenada num ficheiro em disco.

```
>> I = imread('corto.jpg');
>> image(I)
>> axis image
```

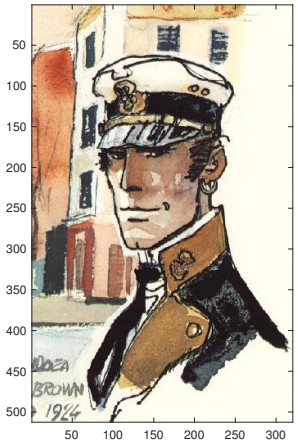


Figura 12: Saída de `image(I)`.

O comando `imread` carrega a imagem `corto.jpg` para a matriz `I`, e o comando `image` mostra a figura no ecrã. Podemos facilmente criar imagens sintéticas com o MATLAB e visualizá-las com os comandos anteriores. Vejamos um exemplo de uma imagem com a metade inferior a branco e a superior a preto:

```
>> M = zeros(100);
>> M(1:50,:) = 1;
>> colormap gray
>> imagesc(M);
```

O *script* `joker.m` seguinte considera uma imagem digital a cores `I`, e transforma-a numa imagem em níveis de cinzento `CI`. Posteriormente usa o comando `[U,S,V] = svd(CI)` para obter a decomposição SVD de `CI` e obtém diferentes imagens conforme o grau de compressão de informação armazenada para cada valor de `k`, associado à característica da matriz obtida.

```
% Ficheiro correspondente ao script joker.m
clear; clc; close all;
I = imread('joker.jpg');
I = im2double(I);    % converter uint8 para double
subplot(221), imshow(I), axis image, title('Original cores');
CI = rgb2gray(I);    % converter imagem RGB em níveis de cinzento
subplot(222), imshow(CI), axis image, title(['Original cinza; car = ',num2str(rank(CI))]);
[U,S,V] = svd(CI);   % obtém a decomposição SVD da matriz em tons de cinza
k = 10; CIk = U(:,1:k)*S(1:k,1:k)*(V(:,1:k))';
subplot(223), imshow(CIk), title(['Compressão; car = ',num2str(rank(CIk))]);
k = 100; CIk = U(:,1:k)*S(1:k,1:k)*(V(:,1:k))';
subplot(224), imshow(CIk), title(['Compressão; car = ',num2str(rank(CIk))]);
```

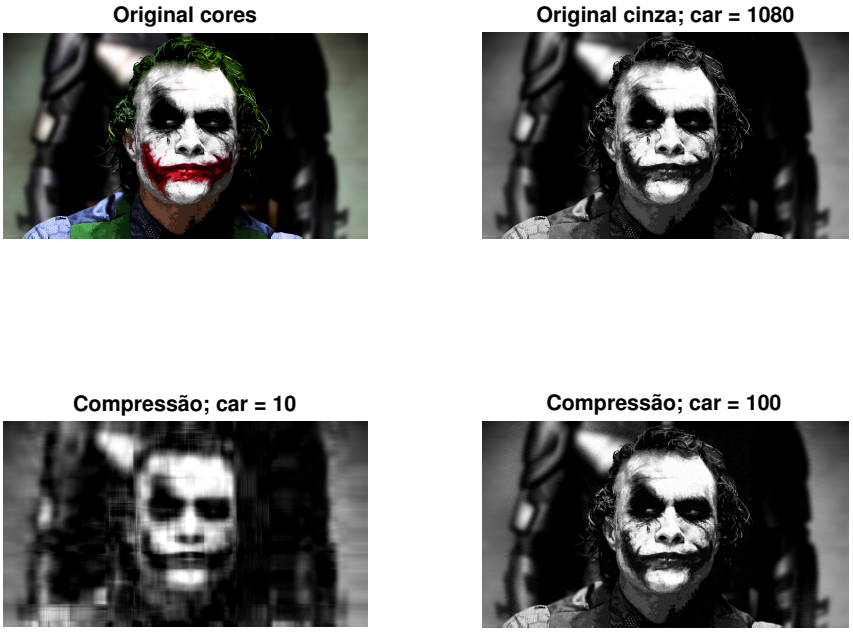


Figura 13: Saída de `joker.m`.

Algumas referências bibliográficas

O material exposto nestas fichas fez uso (muitas vezes de forma literal) de conteúdos obtidos nas seguintes referências bibliográficas (algumas delas disponibilizam os códigos MATLAB usados):

1. MATLAB Primer, thirtieth printing, March 2018 [on-line course].
2. David F. Griffiths, An Introduction to Matlab, The University of Dundee, 2015.
3. Desmond J. Higham and Nicholas J. Higham, MATLAB Guide, third Edition, SIAM, 2017.
4. Andrew Knight, Basics of MATLAB and Beyond, Chapman and Hall/CRC, 1999.
5. Cleve Moler, Numerical Computing with MATLAB, SIAM Philadelphia, 2004.
6. José Manuel Neto Vieira, Matlab num Instante, Universidade de Aveiro, 2004.