publicação do departamento de matemática
da universidade do minho

publicado pelo departamento de matemática
da universidade do minho
campus de gualtar, 4710-054
braga, portugal

segunda edição, setembro de 2005

ISBN 972-8810-08-3

número 9

análise numérica: um curso prático com MATLAB®

maria irene falcão maria joana soares

Conteúdo

Prefácio

A crescente disponibilidade e acessibilidade de computadores pessoais veio permitir o recurso à utilização desta poderosa ferramenta na leccionação de um cada vez maior número de cursos. Tal é especialmente importante em disciplinas da área de Análise Numérica, onde a programação, pelos alunos, dos diversos métodos estudados é imprescindível para uma boa compreensão desses mesmos métodos. Só ao programar e utilizar os programas por si elaborados na resolução de problemas de dimensão razoável poderá o aluno aperceber-se das potencialidades (e também das limitações) dos métodos numéricos apresentados nas aulas teóricas.

De entre os diversos "ambientes de computação" actualmente disponíveis, um dos mais frequentemente utilizados é o MATLAB®. Este apresenta características que o tornam especialmente adequado para ensinar uma disciplina introdutória de Análise Numérica:

- é extremamente fácil de utilizar, sendo os dados introduzidos e manipulados de uma forma simples, especialmente no caso de vectores ou matrizes;
- inclui uma linguagem de programação de alto nível e bastante acessível; a eliminação de "declarações" e a capacidade de "vectorização" de muitas operações, com consequente diminuição do número de ciclos, torna os programas, nesta linguagem, bastante sucintos;
- tem um grande número de funções matemáticas pré-definidas;
- 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; por exemplo, existe uma *toolbox* de *splines*, uma de optimização, etc.; existe ainda uma *toolbox* para efectuar cálculo simbólico, embora, neste curso, a utilização desta facilidade não tenha sido considerada.

Estes textos têm como principal objectivo servir de apoio às aulas práticas, baseadas na utilização do programa MATLAB, de um curso introdutório de Análise Numérica. Os exercícios propostos foram seleccionados com base na nossa experiência de leccionação, com essa abordagem, das disciplinas de Análise Numérica I e II para a Licenciatura em Ensino de Matemática da Universidade do Minho, ao longo dos últimos anos.

Os temas abordados são os que constam do programa, actualmente em vigor, para uma primeira disciplina semestral de Análise Numérica, leccionada pelo Departamento de Matemática da UM, quer à Licenciatura em Matemática e Ciências de Computação, quer à recém criada Licenciatura em Matemática. Estas duas licenciaturas oferecem, no seu *curriculum*, disciplinas opcionais da área de Análise Numérica, onde serão estudados aprofundadamente alguns tópicos aqui não referidos ou apenas muito ligeiramente tratados (caso, por exemplo, da interpolação de Hermite e das funções *spline*, da quadratura Gaussiana, de métodos para a determinação de valores e vectores próprios, da aproximação de mínimos quadrados e minimax, etc.). Os temas para os quais se apresenta uma selecção de exercícios, correspondendo a outros tantos capítulos, são os seguintes:

1. **Iniciação ao MATLAB** – características do sistema MATLAB; manipulação de vectores e matrizes; programação em MATLAB (*scripts* e funções).

2. **Aritmética computacional** – sistemas de numeração de vírgula flutuante; arredondamentos; operações de vírgula flutuante; erro absoluto e erro relativo; condicionamento e estabilidade.

3. **Interpolação** – forma de Lagrange do polinómio interpolador; erro em interpolação polinomial; forma de Newton com diferenças divididas, com diferenças ascendentes e descendentes; funções *spline* cúbicas interpoladoras.

4. **Quadratura** – regras de Newton-Cotes; regras compostas; breve referência a regras de quadratura Gaussiana (Gauss-Legendre).

5. **Sistemas de equações lineares** – resolução de sistemas triangulares; método de eliminação de Gauss; escolha de *pivot*; decomposição LU; decomposição de Cholesky; métodos iterativos de Jacobi e de Gauss-Seidel.

6. **Equações não lineares** – método das iterações sucessivas; métodos da bissecção, secante e Newton.

7. **Equações diferenciais ordinárias** – método de Euler; métodos baseados na série de Taylor; métodos de Runge-Kutta; métodos preditores-correctores.

No primeiro capítulo, um pouco mais extenso, faz-se uma introdução ao estudo da linguagem MATLAB, tendo em vista a sua utilização numa disciplina de Análise Numérica, o que se reflecte na selecção dos tópicos apresentados. Este primeiro capítulo pode ser usado pelos alunos como um manual básico do MATLAB, não dispensando, no entanto, a eventual consulta do respectivo guia do utilizador [Mat].

No início de cada um dos restantes capítulos, é apresentado um breve resumo teórico com os resultados básicos relativos ao tema aí tratado. O objectivo é, essencialmente, indicar as notações e definições por nós adoptadas, podendo ser visto como um formulário um pouco alargado, e não devendo *nunca* ser interpretado pelos alunos como um substituto dos textos de apoio ou da bibliografia adoptada para a disciplina.

Segue-se uma selecção de exercícios destinados a facilitar a compreensão dos conceitos introduzidos nas aulas teóricas. Como já referimos, é nosso objectivo que os alunos programem, eles próprios, a maioria dos métodos numéricos estudados. Tendo em vista esse objectivo, são, para alguns dos métodos, apresentados algoritmos numa linguagem especialmente adaptada à sua implementação em MATLAB, sendo, para outros, fornecida uma descrição pormenorizada da forma como deverá ser construída a respectiva função em MATLAB. A nossa experiência diz-nos, no entanto, que, mesmo com este tipo de ajuda, os alunos têm alguma dificuldade inicial em escrever os programas; por essa razão, decidimos disponibilizar as listagens de algumas das funções pedidas. Julgamos que, com o estudo cuidadoso dessas funções e sua adaptação a outros objectivos, o aluno irá desenvolvendo progressivamente a capacidade de escrever programas mais elaborados. Os exercícios cuja resolução é fornecida são assinalados com o símbolo ✔.

Na escrita das funções em MATLAB, embora tenha havido alguma preocupação em tirar partido das potencialidades dessa linguagem (nomeadamente, no que diz respeito à

capacidade de "vectorização"), houve também algum cuidado em facilitar a sua leitura por parte dos alunos, tentando seguir-se de perto o algoritmo estudado nas aulas teóricas. Isto significa que, nalguns casos, houve necessidade de sacrificar um pouco a eficiência, em nome de uma maior clareza.

Uma vez que todo este curso foi pensado para a utilização do programa MATLAB, as funções ou comandos a construir e/ou utilizar referem-se sempre a essa linguagem, dispensando-nos de o referir explicitamente.

No final de cada capítulo encontra-se uma pequena selecção de exercícios, designados por **Trabalhos**, que, sendo ligeiramente mais elaborados, poderão servir como trabalhos a propor aos alunos para realização fora do horário lectivo. É também dada uma pequena lista com a indicação das funções pré-definidas em MATLAB especialmente relevantes para o assunto desse capítulo.

Como referências básicas para o curso, indicamos os dois excelentes livros em português, [Pin95] e [Val96], e ainda os livros [Atk89], [BF98], [Cd80], [HH91] e [SB80]. O livro de Van Loan [Van97], pela sua abordagem ao estudo de computação científica com o MATLAB, influenciou de forma decisiva a escrita destas notas. As seguintes referências disponíveis *on-line* contêm algumas reflexões interessantes sobre o papel da Análise Numérica nos dias de hoje: [Atk], [Tre92], [Tre00].

Em cada capítulo, é indicada também alguma bibliografia adicional e, sempre que possível, são fornecidos apontadores para páginas *web* que julgamos particularmente interessantes para aprofundar o tema respectivo.

Todas as funções cuja listagem é apresentada no texto estão também disponíveis no endereço url:

www.math.uminho.pt/analise_numerica

Quaisquer comentários ou sugestões, que desde já agradecemos, podem ser enviados para: jsoares@math.uminho.pt ou mif@math.uminho.pt.

Maria Irene Falcão
Maria Joana Soares

1. Iniciação ao MATLAB

Matrizes e vectores
Programar em MATLAB
Gráficos

1.1 Introdução

O MATLAB é uma *package* destinada ao estudo de problemas científicos e de engenharia, ou, de uma forma mais geral, ao estudo de quaisquer problemas em que haja um trabalho computacional significativo a realizar que possa (ou deva) utilizar matrizes. Este *software* é produzido pela companhia americana The Math Works, Inc. e o seu nome deriva do inglês "matrix laboratory".

O MATLAB trabalha com matrizes quadradas ou rectangulares, com elementos reais ou complexos, e derivou dos projectos 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, a *package* permite o acesso a um número crescente de rotinas incorporadas, potencialidades gráficas a 2 e 3 dimensões e a possibilidade de configurar o *software* às necessidades de cada utilizador.

Este capítulo pretende ser uma primeira introdução ao MATLAB. Nos capítulos seguintes serão referidas outras potencialidades desta ferramenta, relacionadas com os tópicos abordados.

Nota: *Estes textos foram originalmente escritos para a versão 6 do MATLAB e foi também esta a versão por nós utilizada na resolução dos exercícios propostos. Tendo sido entretanto disponibilizada a versão 7, assinalamos algumas das especificidades desta nova versão,*

particularmente relevantes para este curso, usando para o efeito o símbolo 7.

1.1.1 Iniciar uma sessão em MATLAB

Quando se invoca o MATLAB, é criada uma 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.

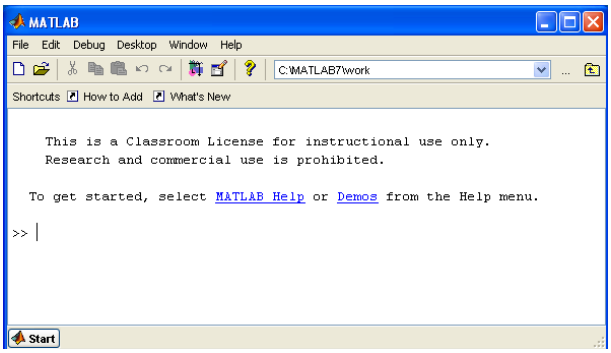


Figura 1.1. A janela de comandos do MATLAB

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

```
>> whos
  Name        Size                    Bytes  Class
  a           2x2                        32  double array
  b           1x1                        16  double array (complex)
  c           2x1                        16  double array
  d           1x4                        32  double array

Grand total is 11 elements using 96 bytes
>> who
Your variables are:
a  b  c  d
```

A mesma informação pode ser visualizada recorrendo à opção **Workspace** do menu **Desktop**.

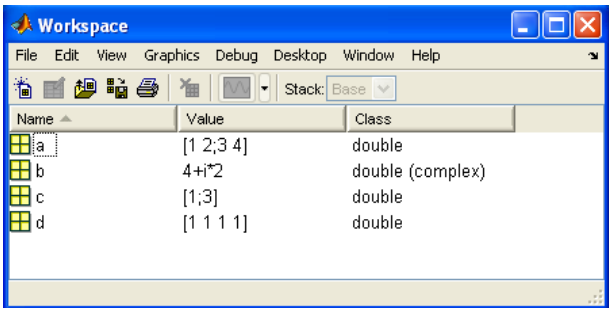


Figura 1.2. Workspace de uma sessão de trabalho

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*.

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*.

Alternativamente, podem ser usadas as opções do menu **File** para realizar estas tarefas.

Finalmente, seleccionando a opção **Command History** do menu **Desktop** pode ver-se um historial de todas as sessões de trabalho realizadas, desde que se apagou pela última vez esta informação (opção **Clear Command History** do menu **Edit**). Verifique o que acontece se seleccionar uma das linhas do *Command History*!

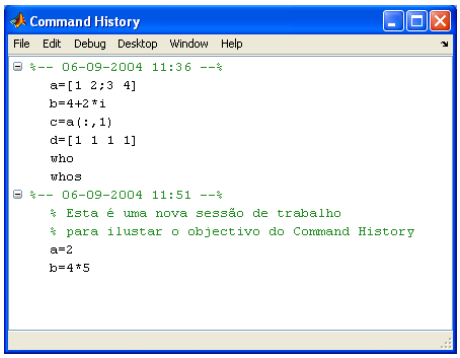


Figura 1.3. Command History do MATLAB

1.1.2 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, respectivamente

```
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 31 caracteres. 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 1.2.7).

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, usando o símbolo ,.

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

b =
     8
```

1.1.3 Definição e manipulação de matrizes

O MATLAB trabalha essencialmente com um tipo de objecto, uma matriz numérica rectangular com elementos eventualmente complexos. Escalares são entendidos como matrizes 1×1 e os vectores como 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 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 exactamente a mesma matriz:

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

Para obter um determinado elemento da matriz A, basta indicar a sua linha e coluna. Por exemplo,

```
>> a23=A(2,3)
a23 =
     2
```

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. Assim, 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

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

ou apenas

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

para se obter a nova matriz

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

Para obter submatrizes de uma dada matriz deve usar-se o símbolo : para indicar quais as linhas e colunas da matriz inicial a considerar. Por exemplo, considerando a matriz

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

a declaração

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

resultará numa matriz que contém as linhas 2 e 3 e as colunas 1 a 3 da matriz inicial,

```
B =
     4     5     6
     7     8     9
```

enquanto a atribuição

```
>> C=A(:,1:2)
```

resultará numa matriz que tem as mesmas linhas e as 2 primeiras colunas de A.

```
C =
     1     2
     4     5
     7     8
    -1    -2
```

Também é possível apagar linhas e colunas de uma matriz, usando os símbolos [ ]. Para retirar a segunda linha da matriz A definida anteriormente basta fazer

```
>> A(2,:)=[]
```

```
A =
     1     2     3
     7     8     9
    -1    -2    -3
```

Vectores com componentes inteiras podem também ser usados como índices. Por exemplo,

```
>> D=A([2 4],:)
```

produzirá uma matriz D cujas linhas são as linhas 2 e 4 da matriz A.

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

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

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

a atribuição

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

resultará na matriz

```
 A =
     2     2     2
     4     5     6
     7     8     9
     4     4     4
```

O símbolo : permite ainda definir vectores e/ou matrizes de modo muito simples. Por exemplo,

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

x =
    -2    -1     0     1     2

>> y=[0:3:9;2:2:8;5:5:20]

y =
     0     3     6     9
     2     4     6     8
     5    10    15    20
```

1.1.4 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
- - subtracção
- * multiplicação
- $/$ divisão à direita [1]
- $\backslash$ 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 Secção 2.6.

Por defeito, o MATLAB efectua todas as operações em precisão dupla.[2] 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 aproximadamente 5 algarismos significativos. É possível alterar este formato, usando formatos que permitem mais algarismos significativos ou usam a notação científica, como se indica na tabela seguinte.

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

[1] As operações com matrizes tornam conveniente o uso de dois símbolos para a divisão; ver Secção 1.2.4.

[2] Mais pormenores sobre o sistema de numeração usado pelo MATLAB serão fornecidos no Capítulo 2.

Por exemplo:

```
>> X=[4/3 sqrt(2)/2.5E4]
X =
    1.3333    0.0001

>> format short;X
X =
    1.3333    0.0001

>> format short e;X
X =
  1.3333e+000  5.6569e-005

>> format short g;X
X =
       1.3333  5.6569e-005

>> format long;X
X =
   1.33333333333333   0.00005656854249
```

7 A versão 7 do MATLAB permite já trabalhar em aritmética de precisão simples ou inteira, especialmente adequada quando é necessário processar uma elevada quantidade de dados deste tipo. Dadas as especificidades deste curso, usaremos, de um modo geral, a aritmética dupla tradicional do MATLAB.

1.2 Operações com matrizes

1.2.1 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.[3] Assim, as declarações

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

```
>> 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
```

1.2.2 Soma de matrizes

Os símbolos $+$ e $-$ designam soma e subtracção 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. Neste caso, a matriz resultante é a que se obtém somando ou subtraindo o escalar a todos os elementos da outra matriz. As declarações

```
>> D=A-1;
>> D=-1+A
```

originam exactamente a mesma matriz

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

1.2.3 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.
```

1.2.4 Quociente de matrizes

No MATLAB existem dois símbolos para representar a "divisão" de duas matrizes: $\backslash$ e $/$. 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, i.e.

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

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

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

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

Então, se

```
>> X=A\B
```

obtém-se

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

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 $8\backslash 2 = 2/8 = 0.25$.

O problema da resolução de sistemas será abordado no Capítulo 5.

1.2.5 Operações elemento a elemento

As operações elemento a elemento 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.

Por exemplo, se

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

então

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

Para que estas operações possam ser executadas, as matrizes (ou vectores) X e Y têm que ter 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
```

1.2.6 Matrizes especiais

O MATLAB tem incorporadas algumas funções muito úteis que permitem definir matrizes muito usadas. Para definir, por exemplo a matriz identidade, a matriz nula ou a matriz "constante" podem usar-se as funções **eye**, **zeros** ou **ones**[4], cujo modo de utilização pode ser obtido invocando o **help** do MATLAB.

```
>> help eye
 EYE Identity matrix.
    EYE(N) is the N-by-N identity matrix.
    EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1's on
    the diagonal and zeros elsewhere.
    EYE(SIZE(A)) is the same size as A.
                ...
>> help zeros
 ZEROS  Zeros array.
    ZEROS(N) is an N-by-N matrix of zeros.
    ZEROS(M,N) or ZEROS([M,N]) is an M-by-N matrix of zeros.
    ZEROS(M,N,P,...) or ZEROS([M N P ...]) is an M-by-N-by-P-by-...
    array of zeros.
    ZEROS(SIZE(A)) is the same size as A and all zeros.
                ...
```

[4] O nome das funções, tal como já foi referido, deve ser escrito em minúsculas.

```
>> help ones
 ONES   Ones array.
    ONES(N) is an N-by-N matrix of ones.
    ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones.
    ONES(M,N,P,...) or ONES([M N P ...]) is an M-by-N-by-P-by-...
    array of ones.
    ONES(SIZE(A)) is the same size as A and all ones.
                 ...
```

As declarações

```
>> A1=eye(3)
>> A2=zeros(2,3)
>> A3=2*ones(size(A2))
```

produzem as matrizes

```
A1 =
     1     0     0
     0     1     0
     0     0     1

A2 =
     0     0     0
     0     0     0

A3 =
     2     2     2
     2     2     2
```

Nota: Uma outra forma de obter a mesma matriz $A3$ seria através do uso do comando `A3=repmat(2,size(A2))`.

1.2.7 Funções

O MATLAB contém um conjunto grande de funções já definidas. Algumas dessas funções são funções matriciais, como por exemplo det(A), que calcula o determinante de uma

matriz quadrada A, 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 5 categorias: funções matemáticas elementares, matrizes elementares e manipulação de matrizes, funções matriciais, polinómios e análise de dados. Uma lista completa de todas as funções existentes em cada uma destas categorias pode ser obtida invocando o **help**: `help elfun`, `help elmat`, `help matfun`, `help polyfun`, `help datafun`.[5]

➠ **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*.

[5] As diversas categorias de funções encontram-se em MATLAB7\TOOLBOX\MATLAB.

➟ Função Exponencial

exp - *Exponencial.*
log - *Logaritmo neperiano.*
log10 - *Logaritmo na base 10.*
sqrt - *Raiz quadrada.*

➟ Funções Complexas

abs - *Módulo.*
conj - *Conjugado.*
imag - *Parte imaginária.*
real - *Parte real.*

➠ Matrizes Elementares e Manipulação de Matrizes

➟ Matrizes Elementares

zeros - *Matriz nula.*
ones - *Matriz com todos os elementos 1.*
eye - *Matriz identidade.*

➟ 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.*

➟ Manipulação de Matrizes

diag - *Criar ou extrair diagonais.*
reshape - *Redimensionar uma matriz.*
tril - *Extrair parte triangular inferior.*
triu - *Extrair parte triangular superior.*

➠ Funções Matriciais

➠ Análise de Matrizes

cond - *Número de condição de uma matriz.*
norm - *Norma matricial ou vectorial.*
rank - *Característica de uma matriz.*
det - *Determinante de uma matriz.*
trace - *Traço de uma matriz.*
null - *Núcleo de uma matriz.*

➠ Equações Lineares

\ e / - *Solução de um sistema linear; use "help slash".*
chol - *Decomposição de Cholesky.*
linsolve - *Solução de um sistema linear.* 7
lu - *Decomposição LU.*
inv - *Inversa de uma matriz.*

➠ Valores Próprios

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

➠ Funções

expm - *Exponencial de uma matriz.*
logm - *Logaritmo de uma matriz.*
sqrtm - *Raiz quadrada de uma matriz.*

➠ Polinómios

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

➠ **Análise de Dados**

max	- *Máximo.*
min	- *Mínimo.*
mean	- *Média.*
median	- *Mediana.*
std	- *Desvio padrão.*
sort	- *Ordenação (ascendente).*
sum	- *Soma dos elementos.*
prod	- *Produto dos elementos.*
cross	- *Produto vectorial.*
dot	- *Produto escalar.*

Para conhecer o modo de utilização de cada função, basta fazer **help** seguido do nome da função pretendida. Por exemplo,

```
>> help triu
```

indica

```
 TRIU Extract upper triangular part.
    TRIU(X) is the upper triangular part of X.
    TRIU(X,K) is the elements on and above the K-th diagonal of
    X.  K = 0 is the main diagonal, K > 0 is above the main
    diagonal and K < 0 is below the main diagonal.

    See also TRIL, DIAG.
```

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

```
>> lookfor upper
```

tem como resultado

```
TRIU Extract upper triangular part.

UPPER  Convert string to uppercase.
```

1.3 Programar em MATLAB

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 podem criar-se usando o editor de texto associado ao MATLAB. Para aceder a este editor basta seleccionar a opção **New** seguida de **M-file** do menu **File**, para criar um novo ficheiro-M, ou **Open** para editar um ficheiro-M já existente. Qualquer texto a seguir ao símbolo % é considerado comentário.

Há dois tipos de ficheiros-M que podem ser usados: *scripts* e funções.

1.3.1 *Scripts*

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 calcular o raio espectral da matriz[6]

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

Para o efeito, no menu do MATLAB, seleccione a opção **New M-file** do menu **File**, para criar um ficheiro de texto chamado, por exemplo, *raio_script.m* com a seguinte sequência de instruções

[6] O valor $\rho(A) = \max_i\{|\lambda_i| : \lambda_i \text{ é valor próprio de } A\}$ é chamado o raio espectral de A.

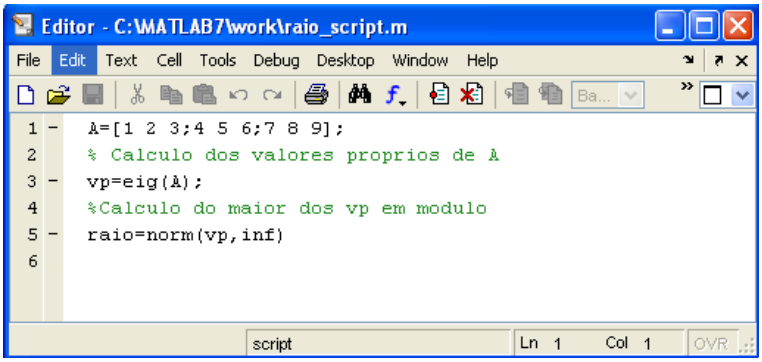


Figura 4. Ficheiro-M raio_script.m

Para executar esta *script* basta fazer, na janela do MATLAB

```
>> raio_script
```

obtendo-se

```
raio =
   16.1168
```

A matriz A, o vector vp e a variável $raio$ estão disponíveis e podem ser de novo utilizadas; por exemplo, se fizer

```
>> dobro_raio=2*raio
```

obterá

```
dobro_raio=
   32.2337
```

1.3.2 Funções

Um ficheiro-M que contém a palavra **function** no início da primeira linha é chamado uma **função**. 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 função é definida do seguinte modo:

function *parâmetros_saida=nome_função(parâmetros_entrada)*

Para resolver o problema do cálculo do raio espectral de uma dada matriz pode criar-se uma função *raio_function.m* do género

```
function raio=raio_function(A)
% RAIO_FUNCTION Calcula o raio espectral
%               de uma dada matriz
vp=eig(A);
raio=norm(vp,inf);
```

Listagem 1.1. Exemplo de uma função

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

```
>> help raio_function

  RAIO_FUNCTION Calcula o raio espectral
                de uma dada matriz
```

Para calcular o raio espectral da matriz `A` definida anteriormente basta fazer

```
>> raio_function(A)

ans =
   16.1168
```

Neste caso as variáveis vp e $raio$ não estão definidas no espaço de trabalho.

```
>> raio
??? Undefined function or variable 'raio'.

>> vp
??? Undefined function or variable 'vp'.
```

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 equacao ax^2+bx+c=0,
d=sqrt(b*b-4*a*c);
x1=(-b+d)/(2*a);
x2=(-b-d)/(2*a);
```

Listagem 1.2. 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
```

1.3.3 *Input* e *Output*

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

variavel_numerica=**input***('texto')*

ou

variavel_string=**input***('texto','s').*

No primeiro caso, aparece no ecrã o texto *texto* 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/NAO)','s')
```

Se for introduzido o valor 10 no primeiro caso e SIM no segundo, então as instruções anteriores originam $vn = 10$ e $vs =$ SIM.

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

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

escreve a matriz A no ecrã.

Para escrever uma frase no ecrã deve rodear-se essa frase do símbolo '. Por exemplo,

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

Quando se pretender escrever um conjunto de várias frases, devem separar-se as frases por vírgulas e rodear o conjunto com os símbolos [e]. Por exemplo,

```
>> disp([' Estou a escrever esta frase ',' ha ', ' minutos' ] )
```

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 ',' ha ',num2str(n), ' minutos' ] )
```

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

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

Para os alunos mais curiosos e que gostam de escrever programas com um formato de entrada/saída atractivo, aqui fica o convite para invocarem o **help** do MATLAB para conhecerem melhor o uso das instruções **fprintf**, **sprintf**, **fwrite**, **fscanf**, etc.

1.3.4 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, pelo que nos limitaremos a descrever a sua forma geral.

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 $\mathbf{m}:\mathbf{n}$ ou $\mathbf{m}:\mathbf{i}:\mathbf{n}$, sendo $\mathbf{m}$ o valor inicial, $\mathbf{i}$ o valor incremental e $\mathbf{n}$ o valor final da *variavel*.

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

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

Listagem 1.3. Ciclos **for**

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 seguintes operadores de relação.

Operadores relacionais

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

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

Operadores lógicos

Operador	Descrição
&	e
|	ou
$\sim$	negação
xor	ou exclusivo
any	verdadeiro se algum elemento de um vector é não nulo
all	verdadeiro se todos os elementos de um vector são não nulos

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 1.4. Ciclos **while**

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

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

correspondem a uma versão vectorizada do exemplo ilustrativo do ciclo **for** considerado anteriormente.

Instruções $\mathbf{if - elseif - else}$

A instrução **if** permite alterar a ordem de execução de uma sequência de instruções, se determinada condição for (ou não) satisfeita. Tem a forma

if *expressão1*
    instruções
elseif *expressão2*
    instruções
elseif *expressão3*
    instruções
      ...
else
    instruções
end

Cada uma das expressões *expressao1*, *expressao2*, 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 com os operadores lógicos.

Suponhamos que pretendemos definir uma matriz de ordem $m \times n$, cujos elementos diagonais são iguais a 3, os elementos acima da diagonal são 5 e abaixo são 4 e além disso pretendemos obter a soma de todos os elementos da matriz. A *script exemplo.m* definida abaixo pode ser usada para resolver o problema.

```
soma=0;
for i=1:m
  for j=1:n
    if i==j
      a(i,j)=3;
    elseif i<j
      a(i,j)=5;
    else
      a(i,j)=4;
    end
    soma=soma+a(i,j);
  end
end
```

Listagem 1.5. Uso de **if − elseif − else**

Para se obter a matriz de ordem 4×6 nas condições indicadas, pode fazer-se

```
>> m=4;n=6;
>> exemplo
>> a
>> soma
```

Nota: Este exemplo pretende apenas ilustrar o uso da instrução **if**. Na verdade, é possível resolver o problema em causa de uma forma muito simples, usando funções já definidas no MATLAB. As seguintes instruções resolvem exactamente o mesmo problema!

```
>> a=3*eye(6,4)+triu(5*ones(6,4),1)+tril(4*ones(6,4),-1);
>> soma=sum(sum(a));
```

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 1.6. 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 1.8. Este ficheiro-M escreve, apenas, 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 1.8. 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.

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

Listagem 1.9. Uso de **error** e **return**

A Listagem 1.9 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 vector.

1.4 Gráficos

O MATLAB dispõe de um grande número de facilidades gráficas que podem ser usadas directamente ou acedidas a partir de funções ou *scripts*. Tendo em conta os objectivos específicos destes apontamentos, centraremos a nossa atenção apenas em gráficos básicos 2-D. Todos os pormenores relativos às ferramentas de visualização incluídas nesta versão do MATLAB podem ser obtidos no guia *Using MATLAB Graphics* ou invocando 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 vectores (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 vector 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

$$\textbf{axis}([xmin \;\; xmax \;\; ymin \;\; ymax])$$

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)
  plot (x_2, y_2)
  plot (x_3, y_3)
hold off

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

$$\textbf{plot}\ (x_1, y_1, x_2, y_2, x_3, y_3)$$

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

Na Figura 1.4 estão representados simultaneamente os gráficos das funções

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

Estes gráficos foram obtidos recorrendo à *script* apresentada na Listagem 1.10, 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('Aproximacoes para sen(t)')
title('Exemplo 1')
text(3.5,0,'sen(t)')
gtext('Aproximacao linear')
gtext('Primeiros 3 termos')
gtext('da serie de Taylor')
```

Listagem 1.10. Representação gráfica de funções

7 A versão 7 do MATLAB contém uma nova *interface* gráfica que permite criar e editar gráficos sem usar código. Use o menu **Help** para obter informações.

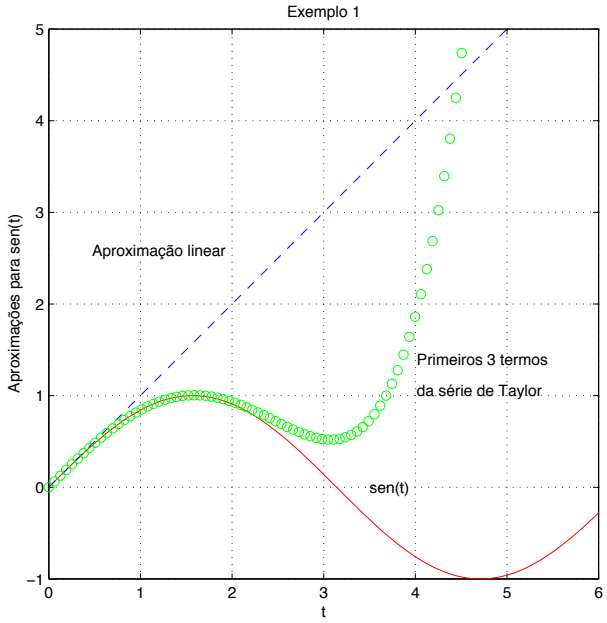


Figura 1.4. Gráfico obtido a partir da *script* anterior

1.5 Funções de novo!

1.5.1 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).

1.5.2 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.

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

function y=mean(x,n)
% MEAN calcula a media de um conjunto de valores
%      Nao ha conflito com a funcao interna MEAN do MATLAB
%      porque esta funcao e executada primeiro.
disp('Funcao invocada:subfuncao mean da funcao exemplo_subfuncoes')
y=sum(x)/n;

function y=median(x,n)
% MEDIAN calcula a mediana de um conjunto de valores
%        Nao ha conflito com a funcao interna MEDIAN do MATLAB
disp('Funcao invocada:subfuncao median da funcao exemplo_subfuncoes')
xordenado=sort(x);
if rem(n,2)==1
    % Se n e impar, a mediana e o elemento na posicao (n+1)/2
    y=xordenado((n+1)/2);
else
    % Se n par, a mediana e a media entre os elementos
    % nas posicoes n/2 e n/2+1
    y=(xordenado(n/2)+xordenado(n/2+1))/2;
end
```

Listagem 1.11. Funções e subfunções

No exemplo anterior, 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];
```

```
>> [minha_media,minha_mediana]=exemplo_subfuncoes(x)
>> outra_media=mean(x)
>> outra_mediana=median(x)
```

obtém-se

```
    Funcao invocada:subfuncao mean da funcao exemplo_subfuncoes
    Funcao invocada:subfuncao median da funcao exemplo_subfuncoes
minha_media =
    3.6667
minha_mediana =
   3.5000
outra_media =
    3.6667
outra_mediana =
    3.5000
```

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 media e a mediana de uma amostra
           Este exemplo permite ilustrar o uso de subfuncoes

>> help exemplo_subfuncoes/mean

  MEAN calcula a media de um conjunto de valores
       Nao ha conflito com a funcao interna MEAN do MATLAB
       porque esta funcao e executada primeiro.
```

1.5.3 Quando uma função é um argumento

Muitas funções em MATLAB têm como argumentos outras funções. Existem várias formas de passar uma função como argumento, dependendo da forma como esta foi escrita.

Consideremos, por exemplo, o problema de representar graficamente a chamada *função de Runge*

$$f(x) = \frac{1}{1+25x^2},\ x \in [-1,1].$$

Claro que podemos recorrer, como fizemos anteriormente, à função **plot**, fazendo

```
>> x=linspace(-1,1);,y=1./(1+25*x.^2);
>> plot(x,y)
```

No entanto, a função **fplot** permite ao utilizador obter de uma forma mais eficiente o gráfico de uma função, definida num dado intervalo. A maneira mais imediata de usar **fplot** é criar um ficheiro-M chamado, por exemplo, *runge.m* com o código que define a função em causa.

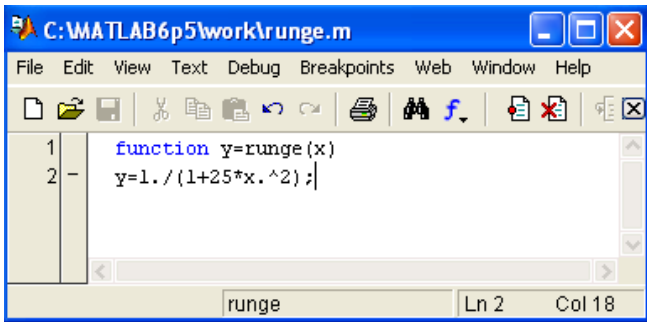


Figura 1.5. Ficheiro-M **runge.m**

O gráfico da função de Runge pode ser obtido fazendo

```
>> fplot('runge',[-1,1])
```

Neste caso, a função que é argumento da função **fplot** é fornecida como uma *string* de caracteres que identifica o ficheiro-M adequado.

Como forma alternativa para resolver o problema de passar uma função como argumento poderemos usar as chamadas *function_handles*. *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 ser criada colocando o caracter **@** atrás do nome da função definida anteriormente através de um ficheiro-M.

No caso do exemplo de Runge, pode fazer-se

```
>> f_handle=@runge;
>> fplot(f_handle,[-1,1])
```

ou ainda

```
>> fplot(@runge,[-1,1])
```

7 Finalmente, convém referir que, na versão 7 do MATLAB, existe uma forma muito simples de criar funções (definindo automaticamente uma *function_handle*): trata-se das chamadas *funções anónimas*.

A sintaxe para criar uma função anónima e a respectiva *function_handle* é a seguinte:

$$f\_anonima = @(argumentos)\ expressão$$

onde: *f_anonima* é nome da função; @ é o operador do MATLAB para construir uma *function_ handle*; *argumentos* é uma lista de argumentos de entrada, separados por vírgulas; *expressão* é o código que define a função (o qual consiste em qualquer expressão matemática válida do MATLAB).

As funções anónimas permitem escrever funções simples de uma forma rápida e sem recurso à criação de um ficheiro-M. Por exemplo, poder-se-ia construir a função de Runge, do seguinte modo

```
>> f_anonima=@(x) 1./(1+25*x.^2)
```

e a função $h(x, y) = (x^2 + y^2, x + y)$ poderia ser definida da seguinte forma

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

Vejamos agora como podemos avaliar uma função, quando esta é definida por qualquer uma das formas atrás referidas.

Continuando com o exemplo de Runge, suponhamos que pretendemos calcular $f(1)$. Como esta função está definida no ficheiro-M *runge.m* como uma função, basta fazer

```
>> runge(1)
```

7 Na nova versão 7 do MATLAB, quando uma função é definida através de uma função anónima ou como *function_handle*, pode ser avaliada tal como se fosse criada com um ficheiro-M.

Por exemplo, qualquer uma da instruções

```
>> f_handle(1)
>> f_anonima(1)
```

produziria o valor da função de Runge no ponto $x = 1$. De modo análogo, fazendo

```
>> h(1,2)
```

obtém-se

```
ans =
     5     3
```

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. Mais informações acerca deste tipo de funções, podem ser obtidas no Capítulo *Function Handles* do manual do MATLAB.

1.5.4 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. Ao longo deste curso iremos recorrer pontualmente a esta técnica de programação.

Apresenta-se, a título de exemplo, a função **fibfun** que se encontra na Versão 6 do MATLAB. Esta função permite obter valores da sucessão

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_n &= F_{n-1} + F_{n-2},\ n > 2 \end{aligned}$$

i.e. determina recursivamente o n-ésimo número de Fibonacci.

```
function f = fibfun(n)
% FIBFUN For calculating Fibonacci numbers.
% Incidentally, the name Fibonacci comes from Filius Bonassi,
% or "son of Bonassus".
if n > 2
    f = fibfun(n - 1) + fibfun(n - 2);
else
    f = 1;
end;
```

Listagem 1.12. Função **fibfun** do MATLAB.

1.6 Notas e referências

A melhor forma de aprender a programar em MATLAB é programando! A análise de programas "bem escritos", como os que o próprio MATLAB apresenta, pode ajudar na difícil tarefa de bem programar.

O objectivo deste capítulo inicial é, fundamentalmente, familiarizar os alunos que estudam Análise Numérica com as principais características do MATLAB no que respeita a esta área da Matemática. Ao longo destes textos são apresentadas algumas funções que, embora respeitando o estilo de programação associado ao MATLAB, privilegiam acima de tudo a clareza (do algoritmo) e a simplicidade (do código).

Nos capítulos seguintes serão propostos vários trabalhos que envolvem programação em MATLAB. Nestes trabalhos pretende-se que o princípio atrás mencionado esteja também presente. Para ajudar o aluno nesta tarefa, apresentamos resumidamente algumas "regras" para conseguir um estilo de programação adequado aos propósitos de uma primeira disciplina de Análise Numérica.

① A estrutura geral de uma função deve ser a seguinte:

```
function Parametros_saida=nome_funcao(Parametros_entrada)
%
% Comentarios que especificam a funcao
%

                Codigo da funcao
```

② Os comentários que especificam a função devem incluir todos os detalhes sobre os parâmetros de entrada e de saída.

③ Ao longo da função devem ser incluídos comentários pertinentes que facilitem o entendimento do código. Muitas vezes estes comentários podem ser dispensados se forem usadas variáveis com nomes sugestivos.

④ Deve ser dado especial cuidado à formatação. Todos os ciclos devem ser indentados.

⑤ Recomenda-se um estilo de programação que permita detectar eventuais erros nos parâmetros de entrada.

⑥ Um programa deve estar escrito de forma clara e ter uma organização lógica, de modo a facilitar a sua leitura e entendimento. Além disso, num programa bem estruturado, a detecção de erros é feita com mais facilidade.

⑦ Todos os programas devem ser testados, usando para o efeito exemplos cuja solução seja conhecida.

⑧ A vectorização dos algoritmos deve ser incentivada. Esta técnica tem várias vantagens para além de aumentar a velocidade de execução. Recomenda-se vivamente a leitura do Capítulo 20 de [HH00], onde são apresentadas várias outras sugestões para optimizar ficheiros-M.

➠ Referências

O manual do MATLAB [Mat] é, sem dúvida, de consulta obrigatória para esclarecimentos adicionais. Os livros [HH00] e [Van97] constituem também uma referência importante nesta

matéria. Não podemos deixar de referir o guia *MATLAB primer* de Kermit Sigmon. Estes textos podem ser obtidos *on-line* e têm sido usados por diversos alunos para aprender a programar em MATLAB.

Nos seguintes endereços encontrará informações bastante úteis para quem usa o MATLAB.

`http://www.mathworks.com`
`http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html`
`http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do`
`http://www.ma.man.ac.uk/~higham/mg`

1.7 Exercícios

Exercício 1.1.

a) Construa, de uma forma simples, os seguintes vectores:

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

b) Usando os vectores definidos na alínea anterior, construa:

$$U = (3,3,3,3,3,3,3,3,3,3); \qquad V = (e^1, e^2, e^3, e^4, e^5, e^6, e^7, e^8, e^9, e^{10});$$
$$X = (1, 1.25, 1.5, 1.75, 2); \qquad Y = (4,16,36,64,100,144,196,256);$$
$$Z = (-6,-4,-2,0,2,4,6,8,10); \qquad W = (0,0,0,1,1,1).$$

Exercício 1.2. Construa o vector $z = (10,20,30,40,50,60,70,80)$. Indique que vectores se obtêm se efectuarmos cada um dos seguintes comandos:

a) $u = z(1:2:7)$; $v = z(7:-2:1)$; $w = z([3\ 4\ 8\ 1])$.

b) $z(1:2:7) = \text{zeros}(1,4)$; $z(7:-2:1) = \text{ones}(1,4)$; $z(1:3) = [\,]$.

Exercício 1.3. Construa as seguintes matrizes:

a) $A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$; *b)* $B = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$;

c) $C = (c_{ij})$, quadrada de ordem 15 e tal que

$$c_{ij} = \begin{cases} 4 & i = j; \\ -1 & i = j - 1; \\ 1 & i = j + 1; \\ 0 & |i - j| > 1 \end{cases}$$

Exercício 1.4. Forme a seguinte matriz

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 5 & 6 & 7 & 8 \\ 1 & 4 & 9 & 16 & 25 \\ 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

e, a partir dela, construa:

$$U = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 6 & 7 & 8 \\ 0 & 0 & 0 & 16 & 25 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \quad L = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 & 0 \\ 1 & 4 & 9 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 \end{pmatrix}.$$

Exercício 1.5. Seja C a matriz do Exercício 1.3.

a) Calcule $\det C$ e C^{-1}.

b) Resolva o sistema $Cx = b$, onde $b = (1, 2, \ldots, 15)^{\mathsf{T}}$.

Exercício 1.6. Escreva uma *script* em MATLAB para gerar uma matriz tridiagonal de ordem $n \times n$ que tem o valor d ao longo da diagonal principal e o valor c nas diagonais abaixo e acima da diagonal principal. Esta *script* deve permitir o uso de qualquer valor de d, c e n.

Exercício 1.7. 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 $h_{i,j} = 1/(i+j+1)$.

(Nota: Faça `edit hilb.m` e analise a função análoga **hilb** do MATLAB.)

Exercício 1.8. Dada a matriz A de ordem 4×5, escreva uma *script* para obter a soma de cada coluna de A, usando:

a) a instrução **for**;

b) a função **sum**.

Exercício 1.9. A seguinte *script* define um vector b, usando um ciclo **for**.

```
% Definir o vector b=(b_i)=sqrt i; i=1,...,10000
% usando um ciclo FOR
%
clear;
tic;
for i=1:10000
   b(i)=sqrt(i);
end
t=toc;
disp(['Tempo de execucao do ciclo FOR e',num2str(t)]);
```

a) Apresente uma versão "vectorizada" desta *script*.

b) Compare o tempo de execução das duas *scripts*.

(Use as funções **tic** e **toc** e execute várias vezes as suas *scripts* para ter uma ideia do tempo médio de execução de cada uma delas.)

Exercício 1.10. 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 interactivamente ao utilizador, através do uso do comando **input**; comece, por isso, por digitar `help input` para relembrar como usar este comando.

Exercício 1.11.

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.

Nota: Uma vez mais, tente escrever a sua função da forma o mais *vectorizada* possível (isto é, sem recurso a ciclos).

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 1.12. Escreva uma função da forma

`function triangulo(p1,p2,p3)`

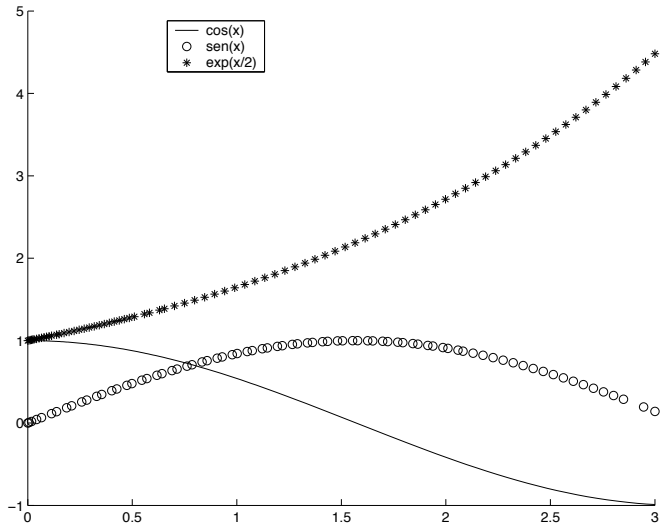
que, dados três pontos $p1$, $p2$ e $p3$, desenhe o triângulo com vértices nesses pontos. A função deverá enviar uma mensagem de erro, caso os pontos não definam um triângulo.

Exercício 1.13.

a) Use o comando **plot** para esboçar o gráfico da função $f(x) = \text{sen}(2\pi x)$, para $x \in [0, 1]$.

b) Repita a alínea anterior, usando agora o comando **fplot**.

Exercício 1.14. Obtenha a seguinte figura.



Exercício 1.15. Complete a seguinte função

```
function meuplot(funcao,dfuncao,lim,ponto)
% MEUPLOT desenha o grafico da funcao FUNCAO, no intervalo  LIM,
%        e da recta tangente ao grafico no ponto de abcissa PONTO
%
%        A equacao da recta tangente e dada por
%        y-funcao(ponto)=dfuncao(ponto)*(x-ponto)
%
%        FUNCAO e DFUNCAO podem ser especificadas como uma
%        funcao anonima ou atraves de uma function handle.
%
% Exemplo:
%         meuplot(@runge,@derivadarunge,[-1 1],0.5)
%
%         f=@(x) 1./(1+25*x.^2), df=@(x) -50*x./((1+25*x.^2)^2)
%         meuplot(f,df,[-1 1],0.5)
%
```

Exercício 1.16.

a) Utilize o comando `help polyfun` para obter uma lista de funções para operações com polinómios.

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

(i) calcule

$$p * q; \quad p'; \quad p(5); \quad [p(1), p(2), p(3)];$$

(ii) determine as raízes de p e de q;

(iii) sobreponha os gráficos dos polinómios p e q (no intervalo $[-1, 1]$).

Exercício 1.17. Considere um polinómio de grau $n - 1$

$$p(x) = a_n + a_{n-1}x + a_{n-2}x^2 + \ldots + a_1 x^{n-1}$$

e suponha que se pretende calcular o seu valor num determinado ponto u.

a) Considere o algoritmo

$p = a_n$
para $k = 1 : n - 1$
$\quad p = p + a_{n-k} * u^k$
fim

Indique o número de adições/subtracções e multiplicações exigidas por este algoritmo.

b) Considere agora o polinómio escrito na *forma encaixada* ou de *Horner*

$$p(x) = (\ldots((a_1 x + a_2)x + a_3)x + \ldots + a_{n-1})x + a_n$$

e o correspondente algoritmo para o cálculo de $p(u)$

$p = a_1$
para $k = 2 : n$
$\quad p = p * u + a_k$
fim

Indique o número de adições/subtracções e multiplicações/divisões exigidas por este algoritmo.

c) Escreva uma função

```
valpol=horner(a,u)
```

destinada a calcular o valor de um determinado polinómio $p(x) = a_1x^n + a_2x^{n-1} + \ldots + a_{n-1}x + a_n$ num certo ponto u, usando a forma de Horner. Mais precisamente, a sua função deve ter:

- como parâmetros de entrada:
  - um vector $a = (a_1, \ldots, a_n)$ (coeficientes de um dado polinómio);
  - um escalar u;
- como parâmetro de saída: um escalar $valpol = (\ldots((a_1u + a_2)u + a_3)u + \ldots + a_{n-1})u + a_n$.

d) Modifique convenientemente a função anterior de forma a que ela admita como parâmetro de entrada um determinado vector $u = (u_1, \ldots, u_m)$, tendo então como parâmetro de saída um vector $valpol = (p(u_1), \ldots, p(u_m))$ com os valores de p em cada uma das componentes de u.

e) Teste a função **horner** no cálculo de $p([-1\ 2\ 3])$ com $p(x) = 5x^3+7x^2-2x+1$.

2. Aritmética Computacional

Sistemas de numeração de vírgula flutuante
Erros, estabilidade e condicionamento

2.1 Notações, definições e resultados básicos

2.1.1 Sistema de numeração $F(b,t,m,M)$

Um sistema de numeração de vírgula flutuante $F(b,t,m,M)$ é caracterizado por quatro parâmetros: b - base; t - número de dígitos da mantissa; m - valor mínimo do exponente; M - valor máximo do expoente. Constituem o sistema $F(b,t,m,M)$, para além do número zero, todos os números que se puderem exprimir na forma

$$\pm(.d_1 d_2 \dots d_t)_b \times b^e, \tag{2.1}$$

com $d_1, d_2, \dots d_t \in \{0, 1 \dots, b-1\}$, $d_1 \neq 0$, e $e \in \mathbb{Z}$ tal que $m \leq e \leq M$. [1]

Estes são os chamados números *normalizados*. Um sistema $F(b,t,m,M)$ pode ainda admitir os chamados números *desnormalizados* ou *subnormais*, que são os números obtidos deixando de impor a condição $d_1 \neq 0$ em (2.1) quando o expoente assume o valor mínimo.

O maior número de $F(b,t,m,M)$ é

$$\Omega := (1 - b^{-t}) b^M, \tag{2.2}$$

[1] Em (2.1), a notação $(.d_1 d_2 \dots d_t)_b$ designa $d_1\, b^{-1} + d_2\, b^{-2} + \cdots + d_t\, b^{-t}$.

dito *nível de overflow*. O menor número positivo normalizado, chamado *nível de underflow*, é dado por

$$\omega := b^{m-1}. \tag{2.3}$$

O menor número positivo de um sistema que admita números desnormalizados é b^{m-t}.

Se nada for dito em contrário, quando nos referirmos a um sistema $F(b,t,m,M)$, consideramos apenas os números normalizados.

Ao conjunto

$$R_F := [-\Omega, -\omega] \cup \{0\} \cup [\omega, \Omega] \tag{2.4}$$

chamamos *conjunto dos números representáveis*.

Arredondamento: Dado $x \in R_F$, $fl(x)$ designa o número de $F(b,t,m,M)$ obtido (salvo indicação em contrário) somando $\frac{1}{2}b^{-t}$ à mantissa e truncando o resultado para t dígitos.

A *unidade de erro de arredondamento* do sistema é

$$\mu := \frac{1}{2}b^{1-t}. \tag{2.5}$$

Chama-se *epsilon da máquina*, e denota-se por ϵ, a diferença entre o número de $F(b,t,m,M)$ imediatamente superior a 1 e o número 1, isto é,

$$\epsilon := b^{1-t}. \tag{2.6}$$

Operações de vírgula flutuante: Representaremos as operações de vírgula flutuante pelo símbolo usual rodeado por $\circ$; por exemplo $\oplus$, $\otimes$. Admitimos que o resultado de uma operação de vírgula flutuante é obtido por arredondamento do resultado da operação exacta, isto é, $x \oplus y = fl(x+y)$, $x \otimes y = fl(x \times y)$, etc.

2.1.2 Norma IEEE 754

Com o objectivo de uniformizar as operações nos sistemas de vírgula flutuante foi publicada, em 1985, a norma IEEE 754. [2] Esta norma especifica dois formatos básicos para representação de números em sistema de vírgula flutuante: *simples* e *duplo*. O formato simples corresponde ao sistema $F(2,24,-125,128)$ e o duplo corresponde a $F(2,53,-1021,1024)$. Ambos os sistemas admitem números desnormalizados.

[2] IEEE- Institute for Electrical and Electronics Engineers.

O sistema de numeração IEEE admite ainda os "números" especiais $+\infty$ e $-\infty$ ($\mathbf{Inf}$ e $-\mathbf{Inf}$), para representar, por exemplo, o resultado da divisão de um número por zero, bem como o símbolo especial $\mathbf{NaN}$ (Not a Number), para representar o resultado de operações não definidas matematicamente, tais como $0/0$, $\infty - \infty$, etc.

A norma IEEE 754 especifica também as regras de arredondamento a utilizar. Por defeito, é utilizado o chamado *arredondamento para par*, isto é, dado $x \in R_F$, $fl(x)$ é escolhido como o número de máquina mais próximo de x, sendo, em caso de "empate", escolhido aquele que tem o último *bit* da mantissa igual a zero; se $x > \Omega$, $fl(x) = \mathbf{Inf}$ e se $x < -\Omega$, $fl(x) = -\mathbf{Inf}$. Se $2^{m-t} \leq x < \omega$ (onde $m = -125$, $t = 24$, no formato simples, e $m = -1021$, $t = 53$, no formato duplo), $fl(x)$ será o número desnormalizado mais próximo de x; se $x < 2^{m-t}$, ter-se-á $fl(x) = 0$.

Nota: *Por defeito, o MATLAB trabalha no sistema de numeração de norma IEEE em formato duplo, isto é, no sistema* $F(2, 53, -1021, 1024)$. *Na nova versão 7 do MATLAB é possível também trabalhar em precisão simples e com dados de tipo inteiro; para mais pormenores, veja [Mat].*

2.1.3 Erro absoluto, erro relativo, algarismos significativos e casas decimais de precisão

Ao valor

$$E_{\tilde{x}} := x - \tilde{x} \tag{2.7}$$

chamamos *erro absoluto* do valor aproximado $\tilde{x}$ para x. Para $x \neq 0$, o valor

$$R_{\tilde{x}} := \frac{x - \tilde{x}}{x} \tag{2.8}$$

constitui o chamado *erro relativo* do valor aproximado $\tilde{x}$ para x.[3]

Dizemos que $\tilde{x}$ é uma aproximação para x com p *casas decimais de precisão*, se p é o maior inteiro tal que

$$|x - \tilde{x}| \leq 0.5 \times 10^{-p}. \tag{2.9}$$

[3]Muitas vezes estamos interessados apenas no valor absoluto destas quantidades, designado-as pelos mesmos nomes, caso tal seja claro pelo contexto.

Dizemos que $\tilde{x}$ é uma aproximação para x com q *algarismos (decimais) significativos*, se q é o maior inteiro para o qual se tem

$$|x - \tilde{x}| \leq 0.5 \times 10^{-q} \times 10^{e}, \tag{2.10}$$

onde e é o expoente de x na notação normalizada (na base decimal).

2.1.4 Condicionamento e estabilidade

Um problema diz-se *mal condicionado* se for muito sensível a perturbações introduzidas nos seus dados, isto é, se "pequenas" alterações nos dados produzirem "grandes" alterações na sua solução (independentemente do método escolhido para resolver o problema). Se tal não acontecer, o problema diz-se *bem condicionado*.

Um método numérico diz-se *instável* se, no decurso dos cálculos inerentes à aplicação do método, os erros se amplificarem de forma inaceitável; diz-se *estável*, caso contrário.

Número de condição de uma função

Sendo f uma função continuamente diferenciável na vizinhança de um ponto x e sendo $f(x) \neq 0$, à quantidade

$$\text{cond}(f(x)) := \frac{|xf'(x)|}{|f(x)|} \tag{2.11}$$

chamamos *número de condição de f em x*. Temos, supondo $x \neq 0$ e sendo $\tilde{x}$ pertencente à vizinhança de x onde f é diferenciável

$$\frac{|f(x) - f(\tilde{x})|}{|f(x)|} \approx \text{cond}(f(x)) \frac{|x - \tilde{x}|}{|x|}.$$

De modo análogo, se f é uma função de duas variáveis suficientemente diferenciável na vizinhança de (x, y) e $(\tilde{x}, \tilde{y})$ está nessa vizinhança, tem-se

$$\frac{|f(x,y) - f(\tilde{x},\tilde{y})|}{|f(x,y)|} \approx \frac{|x\frac{\partial f(x,y)}{\partial x}|}{|f(x,y)|} \frac{|x - \tilde{x}|}{|x|} + \frac{|y\frac{\partial f(x,y)}{\partial y}|}{|f(x,y)|} \frac{|y - \tilde{y}|}{|y|}.$$

As quantidades

$$\frac{|x\frac{\partial f(x,y)}{\partial x}|}{|f(x,y)|} \quad \text{e} \quad \frac{|y\frac{\partial f(x,y)}{\partial y}|}{|f(x,y)|} \tag{2.12}$$

são os *números de condição de f em (x, y)* relativamente à variável x e à variável y, respectivamente.

2.2 Notas e referências

➠ Funções pré-definidas

Função	Objectivo
abs	Valor absoluto
ceil	Arredondamento para o inteiro mais próximo (na direcção de $+\infty$)
base2dec	Mudança de uma dada base para a base decimal
bin2dec	Mudança da base binária para a base decimal
dec2base	Mudança da base decimal para outra base
dec2bin	Mudança da base decimal para a base binária
eps	*epsilon* da máquina
fix	Arredondamento para o inteiro mais próximo (na direcção de zero)
floor	Arredondamento para o inteiro mais próximo (na direcção de $-\infty$)
Inf	Representação na aritmética IEEE de $+\infty$
NaN	Representação na aritmética IEEE de "Not-a-Number"
realmax	Nível de *overflow*
realmin	Nível de *underflow*
rem	Resto da divisão
round	Arredondamento para o inteiro mais próximo

➠ Referências

Para mais pormenores sobre a norma IEEE 754, veja, e.g., [IEE85], [Ove01] ou [Gol91]; o livro clássico de Wilkinson [Wil63], apesar de bastante antigo, continua a ser uma referência importante sobre o tema deste capítulo; outro livro bastante interessante sobre este tópico é o de Higham [Hig96].

Nos seguintes endereços encontrará exemplos curiosos de casos verídicos de problemas causados por erros de arredondamento:

`http://www5.in.tum.de/~huckle/bugse.html`

`http://catless.ncl.ac.uk/Risks/`

2.3 Exercícios

Exercício 2.1. Use a função **help** do MATLAB para obter mais informação sobre as funções pré-definidas **bin2dec**, **dec2bin**, **base2dec** e **dec2base**. Utilize-as para:

a) obter a representação nas bases decimal, octal e hexadecimal, dos seguintes números representados na base binária: $(1011011)_2$ e $(1111111110000)_2$;

b) obter a representação nas bases binária, octal e hexadecimal dos seguintes números representados no sistema decimal: 1325 e 128.

✔ Exercício 2.2.

a) Escreva uma função, $digitos = \mathbf{fracDec2Bin}(frac, n)$, para fazer a conversão de um número fraccionário (i.e., de um número de módulo inferior a 1), do sistema decimal para o sistema binário. Mais precisamente, a sua função deve

- aceitar como argumentos:
  - um número $frac$ (de valor absoluto inferior a 1);
  - um inteiro n (número de dígitos desejado para a expansão);
- dar como resultado: o vector $(d_1, d_2, \ldots, d_n)$ com os primeiros n dígitos da expansão de $frac$ na base binária.

b) Modifique a função anterior de forma a que, se a expansão de $frac$ no sistema binário tiver um número de dígitos $m < n$ (isto é, se $d_{m+1} = d_{m+2} = \ldots = 0$), seja dada uma mensagem indicando que todos os dígitos foram calculados e, como resultado, seja dado o vector com esses dígitos.

c) Teste a sua função na conversão dos números: 0.125, $\frac{1}{3}$, 0.1 e 0.2.

Exercício 2.3.

a) Escreva um pequena *script* para efectuar a seguinte operação:

$$8\,000 - \sum_{k=1}^{80\,000} 0.1.$$

b) Repita a alínea anterior, para calcular:

$$10\,000 - \sum_{k=1}^{80\,000} 0.125.$$

c) Comente os resultados das alíneas anteriores.

✔ Exercício 2.4. Considere o sistema de numeração $F(2, 3, -1, 2)$.

a) Determine os níveis de *overflow* e *underflow* para este sistema.

b) Quantos números distintos constituem o sistema? Explicite-os e represente-os graficamente, usando a função **plot**.

c) Qual a unidade de erro de arredondamento do sistema?

d) Determine $fl(0.125)$, $fl(0.25)$, $fl(4.0)$ e $fl(1.82)$.

Exercício 2.5. Considere a função em MATLAB apresentada na Listagem 2.1.

a) Tente perceber qual o objectivo dessa função e preencha, devidamente, os espaços assinalados com os símbolos ????.

b) Teste a sua função com alguns exemplos por si escolhidos.

Exercício 2.6.

a) Obtenha informação sobre as constantes **realmax**, **realmin** e **eps** definidas em MATLAB e determine o seu valor.

b) Tendo em atenção que está a trabalhar num sistema de norma IEEE 754 em formato duplo, justifique os valores de **realmax** e **realmin** obtidos.

c) Qual a relação entre **eps** e a unidade de erro de arredondamento do sistema em que está a trabalhar?

```
function y=fl(x,t,emin,emax)
%FL ????
% Y=FL(X,T,EMIN,EMAX) ????
%
% PARAMETROS DE ENTRADA:
%    X: ????
%    T: ????
%    EMIN: ????
%    EMAX: ????
%
% PARAMETRO DE SAIDA:
%    Y: ????
%-------------------------------------------------------------------------
%             PARAMETROS T, EMIN E EMAX, POR DEFEITO
%-------------------------------------------------------------------------
if nargin==3
    emax=99;
elseif nargin==2
    emin=-99;emax=99;
elseif nargin==1
    t=4; emin=-99; emax=99;
end
% VERIFICACAO DO PARAMETRO  T (so consideramos t<=15)
if t>15
    error('t deve ser inferior a 16')
end
%-------------------------------------------------------------------------
format long
Omega=(1-10^(-t))*10^emax;  % ????
omega=10^(emin-1);          % ????
if abs(x)>Omega
    disp('????')
    y=sign(x)*Inf;
elseif abs(x)<omega
    disp('????')
    y=0;
elseif x==0
    y=x;
else
expoente=floor(log10(abs(x)))+1; % ????
y=round(x*10^(t-expoente))/10^(t-expoente);
end
```

Listagem 2.1. Função fl

Exercício 2.7.

a) Justifique os seguintes resultados obtidos em MATLAB:

```
>> (1+2^(-52))-1
ans =
  2.2204e-016
>> (1+2^(-53))-1
ans =
     0
>> 1+(2^(-53)-1)
ans =
  1.1102e-016
>> 2^(-1074)
ans =
  4.9407e-324
>> 2^(-1075)
ans =
     0
```

b) Que valor espera obter se fizer

```
>> 2^1024
```

no seu computador? Verifique.

Exercício 2.8. Corra o seguinte código em MATLAB e explique o resultado:

```
>> x=2;
>> for i=1:54
x=sqrt(x);
end;
>> for i=1:54
x=x.^2;
end
>> x
```

Exercício 2.9.

a) Uma estimativa para a unidade de erro de arredondamento de um sistema pode ser obtida recorrendo ao seguinte algoritmo

```
μ ← 1
δ ← 2
enquanto (δ > 1)
        μ ← μ/2
        δ ← 1 + μ
fim
```

Algoritmo 2.1. Unidade erro arredondamento

Justifique o uso do algoritmo.

b) Indique circunstâncias em que o algoritmo anterior não produza o valor exacto da unidade de erro de arredondamento $\mu = \frac{1}{2}b^{1-t}$ de um sistema $F(b, t, m, M)$.

c) Escreva uma *script* que utilize o algoritmo acima para calcular a unidade de erro de arredondamento do seu sistema computacional.

Exercício 2.10. Considere o sistema $F(10, 4, -99, 99)$.

a) Dados $x = 0.8348$, $y = 0.4316 \times 10^{-4}$ e $z = 0.4721 \times 10^{-4}$, calcule $(x \oplus y) \oplus z$ e $x \oplus (y \oplus z)$. Que conclui quanto à associatividade da adição num sistema de vírgula flutuante?

b) Sejam $x = 0.5411$, $y = 0.7223$ e $z = 0.6134$. Verifique que $x \otimes (y \oplus z) \neq (x \otimes y) \oplus (x \otimes z)$.

c) Considere a equação $3 \oplus x = 3$. Indique várias das suas soluções no sistema considerado.

Nota: Na resolução deste exercício pode usar a função $\mathbf{fl}$ do Exercício 2.5.

Exercício 2.11. Justifique que num sistema de vírgula flutuante de base b, a divisão ou multiplicação por uma potência de b, se não conduzir a *overflow* ou *underflow*, é uma operação exacta.

Exercício 2.12. Diga, justificando, se são verdadeiras ou falsas as seguintes afirmações, considerando que está a trabalhar num sistema de vírgula flutuante IEEE (com o arredondamento usual):

a) $x \leq y \Longrightarrow fl(x) \leq fl(y)$;

b) $x < y \Longrightarrow fl(x) < fl(y)$;

c) $x \leq y \Longrightarrow x \leq fl(\frac{x+y}{2}) \leq y \quad (x, y \in F)$.

d) Mostre, através de um exemplo, que a afirmação contida na alínea *c)* pode ser falsa se trabalharmos num sistema de vírgula flutuante de base 10.

Exercício 2.13. Determine, em cada caso, o erro absoluto, o erro relativo, o número de algarismos significativos e o número de casas decimais correctas do valor aproximado $\tilde{x}$ para x:

a) $x = 1/3, \quad \tilde{x} = 0.3333$;

b) $x = 10.375, \quad \tilde{x} = 10.373$;

c) $x = 0.000\,0234, \quad \tilde{x} = 0.000\,0212$;

d) $x = 0.721 \times 10^{-6}, \quad \tilde{x} = 0.724 \times 10^{-6}$.

Exercício 2.14. Escreva aproximações com 3 e 5 algarismos significativos para os números $1/6$, $1/11$, $\pi/100$, e^3 e $\ln 5$.

Exercício 2.15. Sejam $\tilde{x}$ e $\tilde{y}$ valores aproximados para x e y, respectivamente ($x, y \neq 0$), e sejam $S = x + y$, $P = x.y$ e $Q = x/y$. Sejam $\widetilde{S}, \widetilde{P}$ e $\widetilde{Q}$ os valores aproximados para S, P e Q, obtidos usando os valores $\tilde{x}$ e $\tilde{y}$ em vez de x e y e admitindo que as operações são efectuadas exactamente. Designando por $E_{\tilde{u}}$ e $R_{\tilde{u}}$, respectivamente, o erro absoluto e erro relativo no valor aproximado $\widetilde{u}$ para u, mostre que:

a) $E_{\widetilde{S}} = E_{\tilde{x}} + E_{\tilde{y}} \quad$ e $\quad R_{\widetilde{S}} = \frac{x}{x+y} R_{\tilde{x}} + \frac{y}{x+y} R_{\tilde{y}}$;

b) $E_{\widetilde{P}} = E_{\tilde{x}}\tilde{y} + E_{\tilde{y}}\tilde{x} + E_{\tilde{x}}E_{\tilde{y}} \quad$ e $\quad R_{\widetilde{P}} \approx R_{\tilde{x}} + R_{\tilde{y}}$;

c) $E_{\widetilde{Q}} = \dfrac{E_{\tilde{x}}\ \tilde{y} - E_{\tilde{y}}\ \tilde{x}}{\tilde{y}\,(\tilde{y} + E_{\tilde{y}})} \quad$ e $\quad R_{\widetilde{Q}} \approx R_{\tilde{x}} - R_{\tilde{y}}$.

d) Qual das operações referidas pode sofrer de instabilidade, e em que caso?

Exercício 2.16. Considere as funções

$$f(x) = \frac{1 - \cos x}{\operatorname{sen} x} \qquad \text{e} \qquad g(x) = \frac{\operatorname{sen} x}{1 + \cos x}.$$

a) Prove que $f(x) = g(x)$, para $x \neq k\pi$, $k \in \mathbb{Z}$.

b) Calcule o valor das duas expressões para $x = 10^{-9}\pi$. Justifique os resultados obtidos.

Exercício 2.17. Considere o desenvolvimento em série da função exponencial

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots.$$

a) Utilize este desenvolvimento, com 101 termos, para calcular uma aproximação para o valor de e^{-25}.

b) Obtenha uma aproximação para e^{25} usando a série referida e calcule, então, e^{-25} através da fórmula $e^{-25} = \frac{1}{e^{25}}$.

c) Compare os resultados obtidos nas alíneas anteriores com o valor de e^{-25} dado usando a função $\mathbf{exp}$ do MATLAB e explique-os.

Exercício 2.18.

a) Determine, usando aritmética de quatro dígitos, as duas raízes da equação $x^2 + 800\,x + 1 = 0$:

(i) usando a fórmula resolvente habitual para ambas as raízes;

(ii) usando a fórmula resolvente para a raiz de maior valor absoluto e uma fórmula alternativa para o cálculo da outra raiz.

b) Calcule, recorrendo ao MATLAB, as raízes da equação e comente os resultados obtidos.

Exercício 2.19. Encontre fórmulas alternativas para calcular as expressões abaixo indicadas, de modo a evitar o efeito do cancelamento subtractivo:

a) $\sqrt{1+x}-1, \quad x \approx 0$;

b) $1-\cos x;, \quad x \approx 0$;

c) $\operatorname{sen}(x+\delta)-\operatorname{sen} x, \quad |\delta| \ll |x|$;

d) $\frac{1}{1-x}-\frac{1}{1+x}; \quad x \approx 0$.

Exercício 2.20.

a) Calcule o número de condição das funções $f(x)=\sqrt{x}$ e $f(x)=x^2$ e comente sobre o condicionamento dessas funções.

b) Calcule o número de condição das funções $f(x)=\exp(x)$ e $f(x)=\ln x$ e diga para que valores de x o cálculo dessas funções é um problema mal condicionado.

Exercício 2.21. Suponha que se pretende calcular $\ln x - \ln y$ para valores de x e y muito próximos. Poderá haver problemas de cancelamento subtractivo nesse cálculo? Diga, justificando, se poderemos resolver o problema, calculando a expressão equivalente $\ln(x/y)$.

Exercício 2.22. Determine estimativas para os erros (em valor absoluto) cometidos nos cálculos dos valores abaixo indicados, quando os argumentos são arredondados para duas casas decimais:

$$\cos(1.432); \quad \ln(2.347); \quad e^{6.135}; \quad \operatorname{sen}(4.317 \times 1.234)$$

Exercício 2.23. Considere o sistema de equações

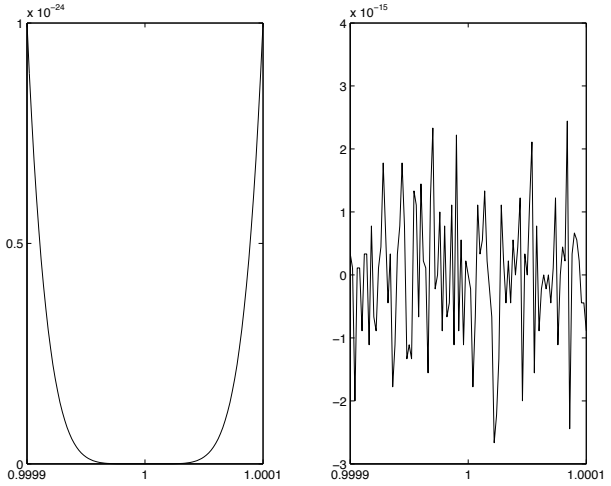
$$\begin{cases} 3.021x + 2.714y + 6.913z = 12.648 \\ 1.031x - 4.273y + 1.121z = -2.121 \\ 5.084x - 5.832y + 9.155z = 8.407 \end{cases}$$

a) Determine a sua solução, usando o MATLAB.

b) Altere o coeficiente -4.273 para -4.275 e resolva o sistema resultante.

c) Que conclusão pode tirar quanto ao sistema em causa?

Exercício 2.24. Na figura seguinte apresentam-se dois gráficos do polinómio $p(x) = (x-1)^6$, para $x \in [0.999, 1.001]$. O gráfico do lado esquerdo foi obtido usando a expressão indicada para p e, no do lado direito, usou-se a expressão expandida $p(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$. Comente os resultados obtidos.



2.4 Trabalhos

Trabalho 2.1. A média de uma amostra de n valores $x_i; i = 1, \ldots, n$, é dada por

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i,$$

sendo o desvio padrão amostral dado por

$$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \right)^{1/2} . \tag{2.13}$$

Para maior eficiência, é frequentemente sugerido o uso da seguinte fórmula alternativa para o cálculo do desvio padrão

$$s = \left(\frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - n\overline{x}^2 \right) \right)^{1/2} . \tag{2.14}$$

Escreva uma função, $[media, desvio1, desvio2] = \mathbf{mediaDesvios}(x)$, destinada a calcular a média e o desvio padrão de uma amostra, sendo usadas as duas fórmulas (2.13) e (2.14) para o cálculo do desvio padrão.

Teste a sua função para várias amostras $\{x_i\}$. Em particular, tente encontrar uma amostra para a qual as duas fórmulas do cálculo do desvio padrão produzam valores bastante diferentes. Justifique a diferença dos resultados.

Trabalho 2.2. Escreva uma *script* destinada a calcular aproximações para o número de Nepper e, usando a definição

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n} \right)^n$$

e indicar o erro nessas aproximações. Mais especificamente, a sua *script* deverá produzir uma tabela da forma

n	Aproximação para e	Erro

para os valores de $n = 10^k; k = 1, 2 \ldots, 20$. Comente os resultados obtidos.

Trabalho 2.3. Relembrando as expansões em série das funções $\arctan x$ e $\operatorname{arcsen} x$

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots ,$$

e

$$\operatorname{arcsen} x = x + \frac{1}{2}\frac{x^3}{3} + \frac{1 \times 3}{2 \times 4}\frac{x^5}{5} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6}\frac{x^7}{7} + \ldots ,$$

obtenha duas fórmulas alternativas para o cálculo de π.

Escreva uma *script* para calcular aproximações para π usando as fórmulas referidas e considerando um número de termos em cada série sucessivamente igual a $10, 20, \ldots, 100, 200, 300, \ldots, 1000$. Comente os resultados obtidos.

Trabalho 2.4. A seguinte fórmula, conhecida por *fórmula de Stirling*, fornece uma estimativa para $n!$:

$$n! \approx n^n e^{-n} \sqrt{2\pi n}.$$

Escreva uma *script* para produzir uma tabela da forma

n	$n!$	Aproximação de Stirling	Erro absoluto	Erro relativo

para os valores de $n = 1, 2, \ldots, 10$. Comente os resultados obtidos.

Trabalho 2.5. Considere a família de integrais

$$I_n = \int_0^1 \frac{t^{n-1}}{5+t} dt, \quad n = 1, 2, \ldots.$$

a) Mostre que

$$I_{n+1} = \frac{1}{n} - 5I_n, \tag{2.15}$$

onde $I_1 = \ln(1.2)$.

b) Use a fórmula recursiva (2.15), no MATLAB, para calcular I_{20}.

c) Rearrange a fórmula (2.15) de modo a obter

$$I_n = \frac{1}{5}(1 - I_{n+1}) \tag{2.16}$$

e use esta nova fórmula para recalcular I_{20}, partindo de $I_{40} \approx 0$ (aproximação apenas com duas casas decimais correctas).

d) Sabendo que o valor de I_{20} (com 10 c.d. de precisão) é

$$I_{20} = 0.0084004954$$

explique pormenorizadamente os resultados obtidos.

2.5 Resoluções

Resolução do Exercício 2.2. *a)* e *b)*

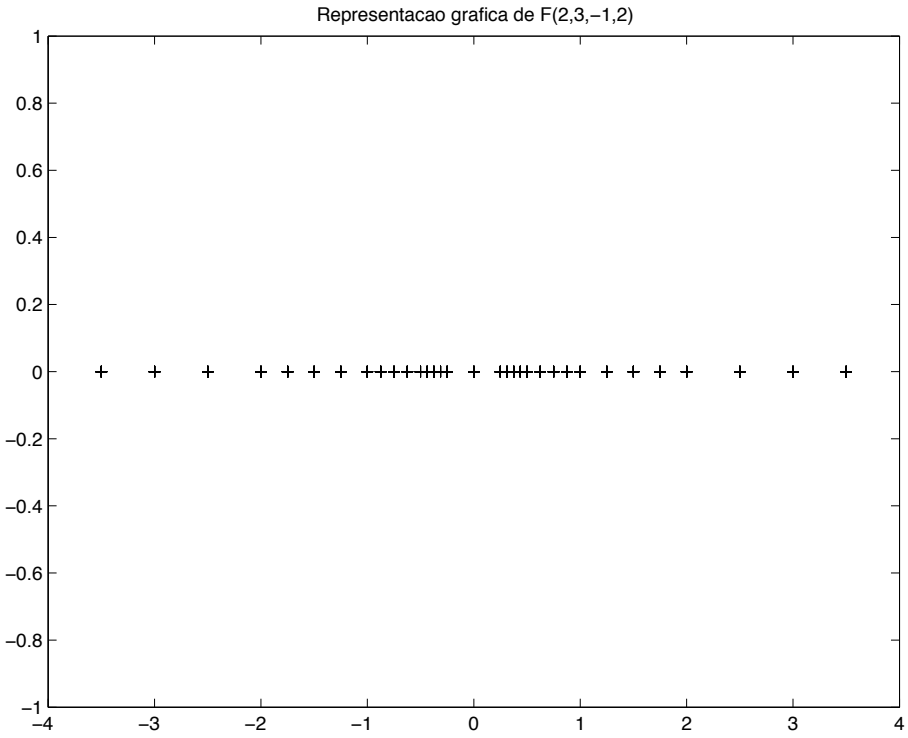
```
function digitos=fracDec2Bin(frac,n)
%FRACDEC2BIN  Determina representacao binaria de um numero fraccionario
%
% DIGITOS=FRACDEC2BIN(FRAC,N) Determina os primeiros N digitos da
% representacao, na base 2, do numero fraccionario FRAC
%
% PARAMETROS DE ENTRADA:
%   FRAC: numero real de valor absoluto menor que 1
%   N: inteiro positivo (numero maximo de digitos desejado)
%
% PARAMETRO DE SAIDA:
%   DIGITOS: vector com (maximo de) N digitos da representacao binaria de FRAC
%
% Se o numero nao e fraccionario, interrompemos o programa
if (abs(frac)>=1.0 || frac==0)
    error('Numero nao e fraccionario')
% Se o numero e negativo, consideramos o seu modulo
elseif frac<0
    frac=abs(frac);
    disp('O numero e negativo; damos os digitos do seu modulo')
end
% Inicializacao do vector de digitos
digitos=[];
for ndigitos=1:n
    % Calculo da parte inteira do dobro de frac
    dobro=2*frac;
    parteinteira=fix(dobro);
    % Parte int. do dobro 'e o digito a acrescentar
    digitos=[digitos parteinteira];
    % Nova parte fraccionaria
    frac=dobro-parteinteira;
    % Se a nova parte fraccionaria e zero, os restantes digitos sao todos zero
    % e nao vale a pena continuar
    if frac==0
      disp(['A rep. binaria do numero tem menos do que ',num2str(n),' digitos'])
      break
    end
end
```

Listagem 2.3. Função **fracDec2Bin**

Resolução do Exercício 2.4. *a) b)* e *c)*

```
% SCRIPT SExercicio24 (para resolver o Exercicio 2.4)
disp('RESOLUCAO DO EXERCICIO 2.4')
disp('____________________')
disp('    Alinea a)      ')
disp('____________________')
b=2;
t=3;
m=-1;
M=2;
disp(' Nivel de overflow')
Omega=(1-b^(-t))*b^M
disp('Nivel de underflow')
omega=b^(m-1)
disp('____________________')
disp('    Alinea b)      ')
disp('____________________')
% Numero de mantissas diferentes: (b-1)*b*...*b (t factores)
nummantissas=(b-1)*b^(t-1);
% Numero de expoentes diferentes
numexpoentes=M-m+1;
% Numero de numeros distintos  (positivos, negativos e zero)
disp('Numero de elementos de F')
N=2*(nummantissas*numexpoentes)+1
% Diferentes mantissas
mantissas=[2^(-1),2^(-1)+2^(-3),2^(-1)+2^(-2),2^(-1)+2^(-2)+2^(-3)];
% Diferentes expoentes
expoentes=[-1,0,1,2];
% Numeros positivos
positivos=kron(2.^expoentes,mantissas);% Usa a funcao KRON
negativos=-positivos;
disp('Numeros do sistema')
numeros=[negativos,0,positivos];
numeros=sort(numeros) % Para ordenar os numeros por ordem crescente
% Grafico dos numeros de F(2,3,-1,2)
plot(numeros,zeros(N),'+')
title('Representacao grafica de F(2,3,-1,2)')
%
disp('____________________')
disp('    Alinea c)      ')
disp('____________________')
disp('Unidade de erro de arredondamento')
mu=(1/2)*b^(1-t)
```

Listagem 2.3. *Script* **SExercicio24**


Representacao grafica de F(2,3,−1,2)
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−4
−3
−2
−1
0
1
2
3
4

3. Interpolação

Forma de Lagrange
Forma de Newton
Funções spline

3.1 Notações, definições e resultados básicos

3.1.1 Interpolação polinomial

Dados n pontos $(x_i, y_i); i = 1, \ldots, n$ (x_i distintos), existe um único polinómio P_{n-1} de grau não superior a $n-1$ que satisfaz as condições de interpolação

$$P_{n-1}(x_i) = y_i;\ \ i = 1, \ldots, n. \tag{3.1}$$

O polinómio P_{n-1} é chamado *polinómio interpolador* dos pontos $(x_i, y_i); i = 1, \ldots, n$. Os números x_i são chamados *nós* ou *abcissas* de interpolação. Em geral y_i são os valores de uma dada função y nos nós x_i, isto é, $y_i = y(x_i); i = 1, \ldots, n$. Nesse caso, dizemos que P_{n-1} interpola a função y nos nós x_i.

Forma cardinal do polinómio interpolador

O polinómio interpolador é dado por

$$P_{n-1}(x) = \sum_{i=1}^{n} L_i(x) y_i, \tag{3.2}$$

onde L_i são os polinómios definidos por

$$L_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right); i = 1, \ldots, n. \tag{3.3}$$

A forma (3.2) é a chamada *forma de Lagrange* do polinómio interpolador e L_i são os chamados *polinómios de Lagrange* (relativos aos nós $x_1, \ldots, x_n$).

Forma de Newton (com diferenças divididas)

O polinómio interpolador P_{n-1} admite também a seguinte expressão, conhecida como *forma de Newton com diferenças divididas*:

$$P_{n-1}(x) \quad = y_1 + \sum_{k=2}^{n} [y_1, \ldots, y_k](x - x_1) \ldots (x - x_{k-1}) \,. \tag{3.4}$$

Em (3.4), $[y_1, \ldots, y_k]$ é a chamada *diferença dividida* de ordem $k-1$ relativa aos pontos $(x_1, y_1), \ldots, (x_k, y_k)$.[1] As diferenças divididas podem calcular-se recursivamente do seguinte modo:

$$[y_i, \ldots, y_{i+\ell}] = \begin{cases} \frac{[y_{i+1}, \ldots, y_{i+\ell}] - [y_i, \ldots, y_{i+\ell-1}]}{x_{i+\ell} - x_i}, & \ell \geq 1, \\ y_i, & \ell = 0 \,. \end{cases} \tag{3.5}$$

Note-se que o valor do polinómio pode ser calculado eficientemente usando a seguinte forma encaixada:

$$P_{n-1}(x) = a_1 + (x - x_1)\Big(a_2 + (x - x_2)\Big(\ldots \Big(a_{n-1} + a_n(x - x_{n-1}) \Big) \ldots \Big) \Big), \tag{3.6}$$

onde, por simplicidade, usámos a notação $a_k := [y_1, \ldots, y_k]$.

Erro em interpolação de Lagrange

Teorema 3.1 *Sejam $a \leq x_1 < x_2 < \ldots < x_n \leq b$ e suponhamos que $y \in C^n[a, b]$. Seja P_{n-1} o polinómio de grau não superior a $n-1$ interpolador de y nos nós $x_1, \ldots, x_n$. Então,*

[1] Ou diferença dividida de ordem $k-1$ de y relativa aos pontos $x_1, \ldots, x_k$, quando $y_i = y(x_i)$; neste caso é mais usual a notação $y[x_1, \ldots, x_k]$.

é válida a seguinte fórmula para o erro de interpolação

$$\forall x \in [a,b], \qquad e_{n-1}(x) := y(x) - P_{n-1}(x) = \frac{y^{(n)}(\xi(x))}{n!}\prod_{i=1}^{n}(x-x_i), \tag{3.7}$$

para um certo número $\xi(x)$ *no intervalo* (a,b).

Temos também a seguinte fórmula para o erro, em termos de diferenças divididas:

$$e_{n-1}(x) = y[x_1,\ldots,x_n,x]\prod_{i=1}^{n}(x-x_i) \approx y[x_1,\ldots,x_n,x_{n+1}]\prod_{i=1}^{n}(x-x_i), \tag{3.8}$$

onde x_{n+1} é mais um ponto no intervalo $[a,b]$.

Caso de nós igualmente espaçados

Se os nós x_i são igualmente espaçados com espaçamento h, isto é, se $x_i = x_1 + (i-1)h$; $i = 1,\ldots,n$, podem obter-se formas do polinómio interpolador mais eficientes em termos de diferenças finitas. Como exemplos, temos:

(i) Forma de Newton com diferenças descendentes

$$\begin{aligned} P_{n-1}(x_1+sh) &= y_1 + \sum_{k=1}^{n-1}\frac{s(s-1)\ldots(s-k+1)}{k!}\triangle^k y_1 \\ &= y_1 + s\{\triangle y_1 + \frac{(s-1)}{2}\{\triangle^2 y_1 + \frac{(s-2)}{3}\{\triangle^3 y_1 + \ldots \\ &\qquad \ldots\{\triangle^{n-2}y_1 + \frac{(s-n+2)}{n-1}\triangle^{n-1}y_1\}\ldots\}\}\}, \end{aligned} \tag{3.9}$$

onde $\triangle^k$ é o operador usual de diferenças descendentes definido por

$$\triangle y_i := y_{i+1} - y_i; \qquad \triangle^k y_i := \triangle(\triangle^{k-1}y_i).$$

(ii) Forma de Newton com diferenças ascendentes

$$\begin{aligned} P_{n-1}(x_n+th) &= y_n + \sum_{k=1}^{n-1}\frac{t(t+1)\ldots(t+k-1)}{k!}\triangledown^k y_n \\ &= y_n + t\{\triangledown y_n + \frac{(t+1)}{2}\{\triangledown^2 y_n + \frac{(t+2)}{3}\{\triangledown^3 y_n + \ldots \\ &\qquad \ldots\{\triangledown^{n-2}y_n + \frac{(t+n-2)}{n-1}\triangledown^{n-1}y_n\}\ldots\}\}\}, \end{aligned} \tag{3.10}$$

onde $\bigtriangledown^k$ é o operador usual de diferenças ascendentes definido por

$$\bigtriangledown y_i := y_i - y_{i-1}; \qquad \bigtriangledown^k y_i := \bigtriangledown(\bigtriangledown^{k-1} y_i).$$

Casos particulares do erro (nós igualmente espaçados)

Quando os nós são os pontos igualmente espaçados

$$x_i = a + (i-1)h;\ i = 1, \ldots n,\ h = \frac{b-a}{n-1},$$

supondo que $f \in C^n[a,b]$ e que $\max_{x\in[a,b]} |f^{(n)}(x)| \leq M_n$, têm-se os seguintes majorantes para o erro (casos $n = 2, 3, 4$):

$$|f(x) - P_1(x)| \leq \frac{h^2}{8} M_2 \tag{3.11}$$

$$|f(x) - P_2(x)| \leq \frac{\sqrt{3}\, h^3}{27} M_3 \tag{3.12}$$

$$|f(x) - P_3(x)| \leq \frac{3h^4}{128} M_4. \tag{3.13}$$

Convergência de uma tabela de diferenças finitas

Seja y uma função tabelada para valores do argumento em progressão aritmética, com os valores tabelados dados com d casas decimais de precisão; diremos que as diferenças relativas a esses valores *convergem* se, para algum inteiro n, as diferenças de ordem n não forem, em módulo, superiores a $2^{n-1} \times 10^{-d}$, isto é, se essas diferenças se "comportarem"como as de um polinómio (de grau $n-1$) cujos valores tenham sido arredondados para d casas decimais. Nesse caso, fará sentido interpolar y por um polinómio de grau $n-1$.

3.1.2 Funções *spline*

Seja k um inteiro não negativo e seja dada uma partição

$$\Omega_n : a = x_1 < \ldots < x_n = b$$

de um intervalo $[a,b]$ em $n-1$ subintervalos $I_i := [x_i, x_i + 1); i = 1, \ldots, n-2$, $I_{n-1} := [x_{n-1}, x_n]$. Uma função $s : [a,b] \longrightarrow \mathbb{R}$ diz-se uma função *spline* de grau k com nós Ω_n se:

- Em cada intervalo $I_i; i = 1, \ldots, n-1$, s é um polinómio de grau não superior a k;

- $s \in C^{(k-1)}[a, b]$.[2]

De entre as funções *spline*, são especialmente importantes as funções *spline cúbicas*. O problema da determinação de uma função *spline* cúbica interpoladora de certos valores nos nós, isto é, satisfazendo as condições de interpolação

$$s(x_i) = y_i; \ i = 1, \ldots, n,$$

não está completamente definido, sendo necessário especificar duas condições adicionais. As escolhas mais usuais são as seguintes, tomando as respectivas funções *spline* os nomes indicados:

- *Spline completa*

$$s'(x_1) = y'_1; \quad s'(x_n) = y'_n, \tag{3.14}$$

  para valores y'_1 e y'_n dados.[3]

- *Spline natural*

$$s^{(2)}(x_1) = s^{(2)}(x_n) = 0. \tag{3.15}$$

- *Spline sem-nó*

$$s^{(3)}(x_2^+) = s^{(3)}(x_2^-), \quad s^{(3)}(x_{n-1}^+) = s^{(3)}(x_{n-1}^-). \tag{3.16}$$

[2]Entende-se por $C^{-1}[a, b]$ o conjunto das funções seccionalmente contínuas em $[a, b]$.

[3]Se $y_i = y(x_i)$ e se conhecermos $y'(x_1)$ e $y'(x_n)$, devemos tomar $y'_1 = y'(x_1)$ e $y'_n = y'(x_n)$.

3.2 Notas e referências

➠ Funções pré-definidas

Função	Objectivo
diff	Diferenças finitas
polyfit	Polinómio de mínimos quadrados
polyval	Valores de polinómios
ppval	Valores de funções polinomiais segmentadas
spline	*Splines* cúbicas
unmkpp	Pormenores sobre função polinomial segmentada

➠ Referências

Os livros de Davis [Dav75] e Powell [Pow81] são boas referências para o estudo do problema de interpolação; para um estudo mais aprofundado de funções *spline*, recomendamos os livros de de Boor [dB78] e de Schumaker [Sch81].

3.3 Exercícios

Exercício 3.1. Considere os pontos $x_1 = 0, x_2 = 0.4$ e $x_3 = 0.6$ e suponha que se pretende obter uma aproximação para o valor de uma dada função f no ponto $x = 0.25$, usando:

(i) interpolação linear ; (ii) interpolação quadrática.

Determine essa aproximação, recorrendo à forma de Lagrange do polinómio interpolador, obtenha um majorante para o respectivo erro e compare-o com o erro efectivamente cometido, quando:

a) $f(x) = \operatorname{sen} x$; *b)* $f(x) = \sqrt{x+2}$; *c)* $f(x) = \ln(x+1)$.

Exercício 3.2. Mostre que o cálculo do valor do polinómio interpolador de grau $\leq n-1$ num determinado ponto, obtido pela forma de Lagrange (3.3), requer $2n^2 - n - 1$ adições/subtracções e $2n^2 - 2n$ multiplicações/divisões.

Exercício 3.3.

a) Pretende-se construir uma tabela de valores da função $f(x) = \cos x$, para $0 \leq x \leq 1$, de modo que ao usar interpolação quadrática entre quaisquer dois pontos dessa tabela, se obtenha uma precisão de 3 casas decimais. Sabendo que se pretende uma tabela de pontos igualmente espaçados, qual deverá ser o número mínimo de entradas da mesma?

b) Determine, por interpolação quadrática, uma aproximação para $\cos 0.23$ com 3 casas decimais de precisão.

✔ Exercício 3.4.

a) Seja dada uma tabela de pontos $(x_i, y_i); i = 1, \ldots, n$. Escreva uma função $DD = \mathbf{tabDifDiv}(x, y)$, destinada a construir a tabela das diferenças divididas dos valores tabelados. A sua função deve

- aceitar como argumentos:
  - um vector $x = (x_1, \ldots, x_n)$ (abcissas, com x_i distintos);
  - um vector $y = (y_1, \ldots, y_n)$ (ordenadas - geralmente, valores de uma determinada função nos pontos x_i);
- dar como resultado: uma matriz quadrada de ordem n, DD, cuja coluna k seja formada pelas diferenças divididas de ordem $k-1$ dos valores (x_i, y_i).

A primeira coluna da matriz DD deve ser o vector y^{T} e, na coluna k, uma vez que o vector das diferenças de ordem $k-1$ tem apenas $n-k+1$ elementos, os elementos nas posições abaixo da posição $n-k+1$ serão definidos como zero.

b) Teste a sua função para $x = (0.0, 0.1, 0.2, \ldots, 1.0)$ e $y(x) = x^2$. Neste caso, o que espera que aconteça com as colunas $4, 5, \ldots, 11$ da matriz DD?

Exercício 3.5. Forme uma tabela dos valores da função $y(x) = \operatorname{sen} x$ nos pontos $x_1 = 0.2$, $x_2 = 0.4$, $x_3 = 0.5$, $x_4 = 0.55$, $x_5 = 0.6$, $x_6 = 0.63$ e $x_7 = 0.67$.

a) Recorrendo à função **tabDifDiv**, determine a tabela das diferenças divididas da função tabelada.

b) Usando valores adequados da tabela anterior, obtenha, por interpolação cúbica, uma estimativa para sen 0.32.

c) Obtenha uma estimativa para o erro cometido na alínea anterior. Compare com o erro efectivamente cometido.

Exercício 3.6. Mostre que o cálculo da tabela de diferenças divididas (para uma tabela correspondente a n entradas) requer $n(n-1)$ adições/subtracções e $n(n-1)/2$ divisões.

✔ Exercício 3.7.

a) Construa uma função $valpol = \mathbf{polDifDiv}(x, y, z)$ com o objectivo de calcular o valor, em z, do polinómio interpolador dos pontos (x_i, y_i), obtido usando a forma de Newton com diferenças divididas. Mais precisamente, a sua função deve

- aceitar como argumentos:
  - um vector $x = (x_1, \ldots, x_n)$ (abcissas - x_i distintos);
  - um vector $y = (y_1, \ldots, y_n)$ (ordenadas);
  - um escalar z (ponto onde se pretende avaliar o polinómio interpolador);
- usar a função $DD = \mathbf{tabDifDiv}(x, y)$ para gerar a tabela de diferenças divididas dos valores tabelados;
- tendo em atenção que a primeira linha da matriz DD contém as diferenças divididas necessárias à construção do polinómio interpolador, usá-las para determinar o valor $valpol = P_{n-1}(z)$, recorrendo à forma encaixada (3.6).

b) Modifique a função definida na alínea anterior, de modo a que o argumento z possa ser um determinado vector $z = (z_1, \ldots, z_m)$, devendo, nesse caso, obter-se como resultado $valpol = (P_{n-1}(z_1), \ldots, P_{n-1}(z_m))$.

Exercício 3.8. Considere os seguintes valores da função $y(x) = e^x$, tabelados com 4 casas decimais de precisão:

x	-1.5	0.0	1.0	1.6	2.1
e^x	0.2231	1.0000	2.7183	4.9530	8.1662

Seja P_4 o polinómio de grau ≤ 4 interpolador dos pontos dessa tabela.

a) Use P_4 para estimar o valor de e^z para $z = -2.0, 0.1, 1.65, 2.3, 4.0$.

b) Esboce os gráfico da função $y(x) = e^x$ e do polinómio P_4, no intervalo $[-2, 4]$.

Exercício 3.9. Considere a seguinte tabela de valores de uma determinada função y:

x	0.2	0.25	0.4	0.55	0.6	0.7
y	0.3624	0.3153	0.1700	0.0208	-0.0292	-0.1288

a) Forme a tabela de diferenças divididas dos valores tabelados.

b) Estime, por interpolação por um polinómio de grau 3, o valor de $y(0.3)$.

c) Obtenha uma estimativa para o erro cometido na alínea anterior.

d) Esboce o gráfico do polinómio interpolador dos pontos da tabela dada.

Exercício 3.10. Escreva uma função $DF = \mathbf{tabDifFin}(y)$, destinada a construir a tabela de diferenças finitas de um vector y. Mais precisamente, esta função deve

- aceitar como argumento: um vector $y = (y_1, \ldots, y_n)$;
- dar como resultado: uma matriz quadrada de ordem n, DF, cuja coluna k seja formada pelos valores das diferenças finitas de ordem $k-1$ de y.

Nota: À semelhança do que foi feito para a tabela de diferenças divididas, na coluna k, os elementos abaixo da posição $n - k + 1$ serão definidos como zero.

Exercício 3.11. Considere os seguintes valores da função $f(x) = \operatorname{sen} x$, tabelados com 4 casas decimais:

x	0.2	0.3	0.4	0.5	0.6	0.7	0.8
$\operatorname{sen} x$	0.1987	0.2955	0.3894	0.4794	0.5646	0.6442	0.7174

a) Construa, utilizando a função $\mathbf{tabDifFin}$, a tabela de diferenças finitas para a função tabelada.

b) Verifique que as diferenças convergem e indique qual o grau adequado para fazer interpolação nos pontos dessa tabela.

Exercício 3.12.

a) Escreva uma função $valpol = \mathbf{polDifDes}(a, h, y, z)$ que tenha como objectivo calcular o valor, num determinado ponto z, do polinómio interpolador dos pontos $(x_i, y_i); i = 1, \ldots, n$, quando as abcissas são os pontos igualmente espaçados

$$x_i = a + (i-1)h; i = 1, \ldots, n,$$

e fazendo uso da forma de Newton com diferenças descendentes (3.9). Mais precisamente, a sua função deve

- aceitar como argumentos:
  - um escalar a (ponto inicial de interpolação);
  - um escalar h (espaçamento entre os pontos de interpolação);
  - um vector $y = (y_1, y_2, \ldots, y_n)$ (ordenadas dos pontos a interpolar);
  - um escalar z (ponto onde se pretende obter o valor do polinómio interpolador) ;
- usar a função $DF = \mathbf{tabDifFin}(y)$ para gerar a tabela de diferenças finitas do vector y;
- tendo em atenção que a primeira linha da matriz DF contém as diferenças finitas necessárias à construção do polinómio interpolador, usar a fórmula (3.9) para determinar o valor de $valpol = P_{n-1}(z)$.

b) Modifique convenientemente a função definida na alínea anterior de forma a que ela passe a admitir, como parâmetro de entrada, um vector z em vez de um escalar, dando como resultado também um vector.

Exercício 3.13. A *função gama* $\mathbf{\Gamma}$ é uma função especial com muitas aplicações em diversas áreas de matemática, e é definida por

$$\mathbf{\Gamma}(x) = \int_0^x t^{x-1} e^{-t} dt, \quad x > 0.$$

Para valores inteiros do argumento, pode provar-se que $\mathbf{\Gamma}(n) = (n-1)!,\ n \in \mathbb{N}$. A tabela seguinte contém valores da função $\mathbf{\Gamma}$ em 6 pontos igualmente espaçados no intervalo $[1, 2]$.

x	$\mathbf{\Gamma}(x)$
1.0	1.0000000000
1.2	0.9181687424
1.4	0.8872638175
1.6	0.8935153493
1.8	0.9313837710
2.0	1.0000000000

a) Determine o polinómio de grau 5 interpolador nos pontos dessa tabela e use-o para estimar o valor de $\mathbf{\Gamma}(x)$ para $x = 1.1, 1.3, 1.5, 1.7$ e 1.9.

b) Obtenha informação sobre a função pré-definida $\mathbf{gamma}$ e use-a para calcular o erro das aproximações obtidas na alínea anterior.

c) Esboce o gráfico do polinómio obtido na alínea *a)* e da função $\mathbf{\Gamma}$, no intervalo $[1, 2]$.

Nota: Para esboçar o gráfico da função $\mathbf{\Gamma}$, use a função pré-definida $\mathbf{gamma}$.

Exercício 3.14. Considere a seguinte tabela de valores da função $f(x) = \operatorname{sen} x - x \ln x$, arredondados para 5 casas decimais:

x	1.72	1.74	1.76	1.78
$f(x)$	0.05609	0.02196	-0.01280	-0.04818

a) Forme a tabela de diferenças finitas para os dados tabelados.

b) Com base na tabela anterior, determine um valor aproximado para $f(1.75)$, usando interpolação por um polinómio de grau adequado.

c) Pretende-se calcular $f(x)$, $x \in [1.72, 1.78]$, com erro não superior a 10^{-7}, usando interpolação por um polinómio do segundo grau. Será possível, com base na tabela dada? Em caso negativo, que alterações deveriam ser introduzidas na tabela?

Exercício 3.15. Considere o seguinte extracto de uma tabela de valores de uma dada função f:

x	0	1	2	3	4	5
$f(x)$	1.0	2.5	25.0	104.5	289.0	638.5

a) Existe um e um só polinómio de grau não superior a 5 que interpola f nos pontos considerados. Sem o determinar, indique qual é, efectivamente, o seu grau e qual é o coeficiente do seu termo de maior grau.

b) Determine uma aproximação para $f(2.5)$, usando o polinómio referido na alínea anterior.

Exercício 3.16. Obtenha informação sobre a função pré-definida **polyfit**. Dados dois vectores $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$, qual o resultado de invocar o comando $pp = \mathbf{polyfit}(x, y, n-1)$? Use a função **polyfit** para resolver alguns dos exercícios anteriores.

Exercício 3.17. Escreva uma função, $\mathbf{plotPolInt}(f, a, b, n)$, que

- aceite como argumentos:
  - uma dada função f;
  - os extremos a e b de um certo intervalo;
  - um inteiro n;
- esboce, em sobreposição, os gráficos da função f e do polinómio P_{n-1} de grau $\leq n-1$ interpolador dessa função em n pontos igualmente espaçados no intervalo $[a, b]$.

Exercício 3.18. Considere a *função de Runge*

$$f(x) = \frac{1}{1+25x^2}, \quad x \in [-1, 1].$$

Para $n \in \mathbb{N}$, seja P_{n-1} o polinómio interpolador de f em n pontos igualmente espaçados no intervalo $[-1, 1]$. Escreva uma *script* que esboce o gráfico de f, P_2, P_4, P_{10} e P_{20}. Comente os resultados obtidos.

Exercício 3.19. Determine k de modo que a seguinte função seja uma função *spline* cúbica:

$$s(x) = \begin{cases} x^3 - x^2 + 3x + 1, & 0 \leq x \leq 1, \\ -x^3 + kx^2 - 3x + 3, & 1 \leq x \leq 2. \end{cases}$$

Exercício 3.20.

a) Obtenha informação sobre a função pré-definida **spline**. Sendo $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ e $z = (u_1, \ldots, u_m)$, qual é o resultado de invocar o comando $valspl = \mathbf{spline}(x, y, z)$, no MATLAB? E qual será o resultado de $spl = \mathbf{spline}(x, y)$?

b) O que acontece no caso em que o vector y tem mais duas componentes do que o vector x, isto é, $y = (y_1, \ldots, y_{n+2})$?

c) Obtenha ajuda sobre as funções **unmkpp** e **ppval**.

Exercício 3.21. Considere a seguinte tabela de pontos

x	0	1	2	3
y	0.0	0.5	2.0	1.5

a) Seja S_c a função *spline* cúbica interpoladora dos pontos dessa tabela, satisfazendo as condições finais: $S_c'(0) = 0.2$ e $S_c'(3) = -1$.

(i) Determine, usando a função **spline**, os valores $S_c(0.5)$, $S_c(1.2)$ e $S_c(2.8)$.

(ii) Esboce o gráfico de S_c e assinale, sobre esse gráfico, os pontos da tabela.

(iii) Usando a função **unmkpp**, determine a expressão de cada uma das cúbicas que formam S_c.

b) Repita a alínea anterior, mas construindo S_{sn}, a função *spline* sem-nó interpoladora dos pontos da tabela.

c) Seja P_3 o polinómio cúbico interpolador dos pontos da tabela dada. Sobreponha ao gráfico de S_{sn} o gráfico de P_3. Que conclui? Como justifica tal resultado?

Exercício 3.22. Seja S_n a função *spline* cúbica completa interpoladora da função $f(x) = \exp(x)$ em $n + 1$ pontos igualmente espaçados no intervalo $[0, 1]$.

a) Obtenha os valores da função *spline* com 11 nós, S_{10}, nos pontos 0.21, 0.33, 0.75 e 0.99 e compare-os com o valores de f nesses pontos.

b) Esboce os gráficos da função f e da função *spline* S_{10}, no intervalo $[0, 1]$, e marque também os pontos de interpolação.

Exercício 3.23. Considere novamente a função de *Runge*

$$f(x) = \frac{1}{1 + 25x^2} \quad x \in [-1, 1].$$

Seja S_n a função *spline* cúbica completa interpoladora de f nos $n+1$ nós igualmente espaçados

$$x_k = -1 + (k-1)h,\ k = 1, \ldots, n+1;\ h = 2/n.$$

Esboce o gráfico de f e de cada uma das funções *spline* S_4, S_{10} e S_{20}.

3.4 Trabalhos

Trabalho 3.1. Considere a seguinte tabela:

x	y	x	y
10.0	0.42	12.16	3.40
10.6	0.52	12.44	4.62
11.0	0.55	12.5	4.64
11.6	0.65	13.0	4.64
12.0	1.52	13.2	4.64
12.04	1.87	13.5	4.64
12.08	2.35	14.0	4.64

a) Represente graficamente os 14 pontos dessa tabela.

b) Escolha 6 pontos da tabela e construa o polinómio P_5 e a função *spline* cúbica sem-nó interpoladores desses pontos. Represente-os graficamente, marcando os pontos da tabela.

c) Repita a alínea anterior escolhendo 9 pontos de interpolação (e construindo o polinómio P_8 e a função spline sem-nó); finalmente, use todos os pontos da tabela para construir P_{13} e a função *spline* sem-nó interpoladores desses pontos.

d) Comente os resultados obtidos.

Trabalho 3.2. **Interpolação inversa**

Se tivermos uma tabela de pontos (x_i, y_i) com $y_i = f(x_i)$ valores de uma função $y = f(x)$ nos nós x_i, e se soubermos que a função f é invertível, podemos trocar o papel das abcissas $x_1, \ldots, x_n$ e das ordenadas $y_1, \ldots, y_n$ e construir o polinómio interpolador dos valores $x_1, \ldots, x_n$ nos nós $y_1, \ldots, y_n$, ou seja, construir o polinómio interpolador da função inversa $x = f^{-1}(y)$. Diz-se, neste caso, que se trata de *interpolação inversa*.

a) Justifique por que razão este processo não funciona se f não for monótona no intervalo de interpolação.

b) Use interpolação inversa para determinar uma estimativa para $\sqrt[3]{8.1232}$, supondo que a sua "máquina" não calcula valores de raízes cúbicas.

Trabalho 3.3. Considere a seguinte tabela

x	0.00	0.95	1.00	1.05	2.00
f	2.00	1.00	1.05	1.00	0.00

a) Seja P_4 o polinómio interpolador dos pontos dessa tabela. Calcule o valor de $P_4(1.80)$.

b) Altere a tabela dada, fazendo $f(1.00) = 0.95$ e seja Q_4 o correspondente polinómio interpolador. Indique o valor de $Q_4(1.80)$.

c) Esboce os gráficos de P_4 e Q_4.

d) Comente os resultados obtidos.

Trabalho 3.4. Chamam-se *nós de Chebyshev* (de grau n) os pontos definidos por

$$z_k^{(n)} = \cos\left(\frac{(2k-1)\pi}{2n}\right); k = 1, \ldots, n. \tag{3.17}$$

Estes pontos são os zeros do chamado *polinómio de Chebyshev* de grau n, definido por $T_n(x) = \cos(n \arccos x), x \in [-1, 1]$; os polinómios de Chebyshev constituem uma importante família de polinómios ortogonais, com muitas aplicações em diversas áreas da matemática.

a) Usando a função **plot**, represente graficamente, no intervalo $[-1, 1]$, os nós de Chebyshev de grau 11.

b) Considere o produto

$$\pi(x) = (x - x_1) \dots (x - x_{11})$$

com as duas escolhas de pontos seguintes:

(i)
$$x_k = -1 + (k - 1)/5; k = 1, \dots, 11;$$

(ii)
$$x_k = z_k^{(11)}; k = 1, \dots, 11.$$

Esboce os gráficos de $|\pi(x)|$ para cada uma dessas escolhas de nós. Que observa?

Nota: De facto, pode mostrar-se que a escolha $x_k = z_k^{(n)}$ minimiza

$$\max_{x \in [-1,1]} |(x - x_1) \dots (x - x_n)| .$$

c) Considere novamente a *função de Runge* $f(x) = \frac{1}{1+25x^2}$, $x \in [-1, 1]$ e, para $n \in \mathbb{N}$, seja P_{n-1} o polinómio de grau $\leq n - 1$ interpolador de f nos nós de Chebyshev de grau n, isto é, nos pontos $x_k = z_k^{(n)}; k = 1, \dots, n$. Construa os polinómios P_2, P_4 e P_{10} e P_{20} e esboce o gráficos de f e de cada um desses polinómios.

d) Relembrando os resultados do Exercício 3.18, faça um pequeno comentário sobre os resultados obtidos.

3.5 Resoluções

Resolução do Exercício 3.4. *a)* e *b)*

```
function DD=tabDifDiv(x,y)
%TABDIFDIV  Constroi tabela de diferencas divididas
%
% DD=TABDIFDIV(X,Y)   constroi uma matriz DD cujas colunas sao
%   as diferencas divididas relativas a uma tabela de pontos (Xi,Yi).
%
% PARAMETROS DE ENTRADA:
%    X: vector de N componentes distintas (abcissas);
%    Y: vector da mesma dimensao que X;
%
% PARAMETRO DE SAIDA:
%    DD: matriz quadrada N*N tendo, na coluna K,  as diferencas divididas
%        de ordem K-1 relativas aos pontos (Xi,Yi).
%        Na coluna K, como ha apenas N-K+1 diferencas divididas,
%        os elementos abaixo da posicao N-K+1 sao definidos como zero.

%---------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%---------------------------------------------------------------------
% Verificar se X e Y sao vectores da mesma dimensao e se os elementos
% de X sao distintos
n=length(x);
if length(y) ~= n
   error('Os vectores x e y devem ser da mesma dimensao.')
elseif ~all(diff(sort(x)))
   error('As abcissas devem ser distintas.');
end
%---------------------------------------------------------------------
% Escrever x e y como vectores coluna, caso entrem como linha!
if size(x,1) == 1, x = x'; end
if size(y,1) == 1, y = y'; end
%---------------------------------------------------------------------
%             FORMACAO DA MATRIZ DD DAS DIFERENCAS DIVIDIDAS
%---------------------------------------------------------------------
DD=zeros(n); % Iniciar a matriz DD com zeros
DD(:,1)=y;   % Tornar a primeira coluna de DD igual ao vector y
% Calculo recursivo das dif. divididas e armazenamento nas colunas de DD
for k=2:n
    i=1:n-k+1;
    l=k:n;
    DD(i,k)=(DD(i+1,k-1)-DD(i,k-1))./(x(l)-x(i));
end
```

Listagem 3.1. Função **tabDifDiv**

Resolução do Exercício 3.7. *a)* e *b)*

```
function valp=polDifDiv(x,y,z)
%POLDIFDIV  calculo do polinomio interpolador
%
% VALP=POLDIFDIV(X,Y,Z)   Calcula P(Z), onde P e o polinomio interpolador
%      dos pontos (X,Y). O polinomio P (de grau <= N-1, onde N=length(X))
%      'e calculado usando a forma de Newton com  diferencas divididas.
%
% PARAMETROS DE ENTRADA:
%    X: vector de N componentes distintas (abcissas);
%    Y: vector  com a mesma dimensao que X;
%    Z: vector de M componentes com os pontos onde se pretende calcular
%       os valores do polinomio intetnrpolador.
%
% PARAMETRO DE SAIDA:
%    VALP: VALP=(P(Z_1),...,P(Z_M)), onde P e o polinomio de grau <=N-1
%          interpolador dos pontos (X_i,Y_i).
%
%  Esta funcao invoca a funcao TABDIFDIV

%-------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%-------------------------------------------------------------------------
% As verificacoes relativas a X e Y sao feitas quando invocarmos
% a funcao TABDIFDIV
%-------------------------------------------------------------------------
%    USO DA FUNCAO TABDIFDIV PARA CALCULAR DIFERENCAS DIVIDIDAS
%-------------------------------------------------------------------------
DD=tabDifDiv(x,y);
% Extrair linha 1 de DD (contem as dif.div. necessarias ao calculo de VALP)
dfdv=DD(1,:);
%-------------------------------------------------------------------------
%                     CALCULO DE VALP
%-------------------------------------------------------------------------
n=length(dfdv);
valp=ones(size(z)); % Inicializcao do vector VALP
valp(:)=dfdv(n);
for i=n-1:-1:1
    valp=dfdv(i)+valp.*(z-x(i)); % Uso da forma encaixada
end
```

Listagem 3.2. Função **polDifDiv**

4. Quadratura

Regras de Newton-Cotes
Regras compostas
Regras de Gauss-Legendre

4.1 Notações, definições e resultados básicos

4.1.1 Regras de Newton-Cotes

Consideremos o problema de estimar o valor de um certo integral

$$I = \int_a^b f(x)dx, \tag{4.1}$$

o qual supomos existir, e suponhamos que tal é feito usando uma regra de quadratura da forma

$$I \approx Q_n(f) = \sum_{i=1}^{n} w_i f(x_i). \tag{4.2}$$

Os pontos x_i são chamados *abcissas* (ou *pontos de quadratura*) da regra de quadratura e os coeficientes w_i são chamados *pesos* dessa regra. Ao erro resultante da substituição do integral pelo valor dado pela regra, chamamos *erro de quadratura* da regra em questão. [1]

[1] Referimo-nos, aqui, ao erro de truncatura, não considerando os eventuais erros de arredondamento cometidos ao efectuar os cálculos.

Diz-se que uma regra de quadratura tem precisão m (ou é de grau m) se for exacta para todos os polinómios de grau não superior a m e existir, pelo menos, um polinómio de grau $m+1$ que não é integrado exactamente por essa fórmula.

As *regras de quadratura de Newton-Cotes (fechadas)* são regras do tipo (4.2), nas quais:

- se consideram para abcissas os n pontos igualmente espaçados no intervalo $[a, b]$,

$$x_i = a + (i-1)h;\ i = 1, \ldots, n;\ h = \frac{b-a}{n-1}; \tag{4.3}$$

- os pesos w_i são determinados substituindo a função integranda f pelo polinómio P_{n-1} de grau não superior a $n-1$ que interpola f nas n abcissas x_i.

Apresentamos, de seguida, uma tabela com os pesos e a expressão do erro de quadratura[2] das fórmulas de Newton-Cotes fechadas para $n = 2, \ldots, 9$. Para simplificar a tabela, consideramos as fórmulas escritas na forma

$$Q_n(f) = \mathbf{C}_n h \left[w_1 f_1 + \ldots + w_n f_n\right]$$

e indicamos, em cada caso, o valor de $\mathbf{C}_n$ e dos coeficientes $w_i; i = 1, \ldots, \lfloor (n+1)/2 \rfloor$; os restantes coeficientes w_i podem ser obtidos por simetria, isto é, $w_{n+1-i} = w_i$. Indicamos também o nome por que são conhecidas algumas dessas regras.

Fórmulas de Newton-Cotes

n	$\mathbf{C}_n$	w_1	w_2	w_3	w_4	w_5	$E_n(f)$	Nome
2	$\frac{1}{2}$	1					$-\frac{h^3}{12} f^{(2)}(\eta)$	Trapézio
3	$\frac{1}{3}$	1	4				$-\frac{h^5}{90} f^{(4)}(\eta)$	Simpson
4	$\frac{3}{8}$	1	3				$-\frac{3h^5}{180} f^{(4)}(\eta)$	Três oitavos
5	$\frac{2}{45}$	7	32	12			$-\frac{8h^7}{945} f^{(6)}(\eta)$	Milne
6	$\frac{5}{288}$	19	75	50			$-\frac{275h^7}{12\,096} f^{(6)}(\eta)$	—
7	$\frac{1}{140}$	41	216	27	272		$-\frac{9h^9}{180} f^{(8)}(\eta)$	Weddle
8	$\frac{7}{17\,280}$	751	3577	1323	2989		$-\frac{8\,183h^9}{518400} f^{(8)}(\eta)$	—
9	$\frac{4}{14175}$	989	5888	-928	10496	-4540	$-\frac{2368h^{11}}{467775} f^{(10)}(\eta)$	—

[2] Supondo que a função integranda é suficientemente diferenciável em $[a, b]$.

4.1.2 Regras compostas

As regras de Newton-Cotes para valores de n elevado têm pesos negativos e positivos, sendo sujeitas a problemas de instabilidade, pelo que são raramente utilizadas. Em vez disso, poderá recorrer-se ao uso de regras compostas. A ideia das regras compostas é subdividir o intervalo de integração $[a, b]$ num certo número de subintervalos e aplicar as regras simples em subintervalos. Temos, em particular os seguintes resultados.

Teorema 4.1 (Regra do Trapézio Composta)
Seja $f \in C^2[a, b]$ e sejam $x_i = a + (i - 1)h$; $i = 1, \ldots, N + 1$, com $h = \frac{b-a}{N}$. Então, tem-se

$$\int_a^b f(x)dx = T_N(f) + E_{T_N}(f), \tag{4.4}$$

onde

$$T_N(f) := \frac{h}{2}\left[f_1 + 2\left(f_2 + \cdots + f_N\right) + f_{N+1}\right] \quad \text{e} \quad E_{T_N}(f) = -\frac{(b-a)h^2}{12} f^{(2)}(\eta). \tag{4.5}$$

Teorema 4.2 (Regra de Simpson composta)
Seja $f \in C^4[a, b]$ e sejam $x_i = a + (i - 1)h$; $i = 1, \ldots, 2m + 1$, com $h = (b - a)/2m$. Então,

$$\int_a^b f(x)dx = S_{2m}(f) + E_{S_{2m}}(f), \tag{4.6}$$

onde

$$S_{2m}(f) := \frac{h}{3}\left[f_1 + 4\left(f_2 + \cdots + f_{2m}\right) + 2\left(f_3 + \cdots + f_{2m-1}\right) + f_{2m+1}\right] \tag{4.7}$$

e

$$E_{S_{2m}}(f) = -\frac{(b-a)h^4}{180} f^{(4)}(\eta). \tag{4.8}$$

4.1.3 Regras de Gauss-Legendre

A regra de Newton-Cotes fechada com n pontos é exacta para polinómios de grau $\leq n-1$ ou, sendo n ímpar, para polinómios de grau $\leq n$. É, no entanto, possível, obter regras do

tipo (4.2) exactas para polinómios de grau $2n-1$, escolhendo as abcissas apropriadamente. Esta é a ideia básica das regras de quadratura Gaussiana. Em particular, têm-se as chamadas regras de Gauss-Legendre para estimar integrais da forma $I = \int_{-1}^{1} f(x)dx$. Essas regras são dadas por

$$I = \int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i),$$

onde as abcissas são os zeros do *polinómio ortogonal de Legendre* de grau n e os pesos w_i são dados por

$$w_i = \int_{-1}^{1} L_i^2(x)dx; i = 1, \ldots, n,$$

com $L_i(x)$ os polinómio de Lagrange relativos aos pontos x_i. As abcissas e os pesos das regras de Gauss-Legendre, para os primeiros valores de n, são dados na tabela seguinte (com 10 c.d. de precisão).

Abcissas e pesos das fórmulas de Gauss-Legendre

n	x_i	w_i
2	$\pm$0.5773502692	1.0000000000
3	0.0000000000	0.8888888889
	$\pm$0.7745966692	0.5555555556
4	$\pm$0.3399810436	0.6521451549
	$\pm$0.8611363115	0.3478548451
5	0.0000000000	0.5688888889
	$\pm$0.5384693101	0.4786286705
	$\pm$0.9061798459	0.2369268851
6	$\pm$0.2386191861	0.4679139346
	$\pm$0.6612093865	0.3607615730
	$\pm$0.9324695142	0.1713244924

É a seguinte a fórmula do erro de quadratura da regra de Gauss-Legendre com n pontos, supondo que $f \in C^{2n}[-1,1]$:

$$E_n(f) = C_n \frac{f^{(2n)}(\eta)}{(2n)!}, \tag{4.9}$$

onde

$$C_n = \frac{2^{2n+1}(n!)^4}{(2n+1)((2n)!)^2}.$$

4.2 Notas e referências

Funções pré-definidas

Função	Objectivo
dblquad	Integração dupla
quad	Quadratura (ordem baixa)
quadl	Quadratura (ordem alta)
triplequad	Integração tripla

Referências

Como referências sobre o tema de quadratura, sugerimos os livros [DR84], [Eng80] e [Eva93]; a referência [PdUK83], que contém documentação sobre a *package* de quadratura **QUADPACK**, é também importante; para mais informação sobre quadratura Gaussiana, nomeadamente para consulta de tabelas mais completas de pesos e abcissas deste tipo de regras, veja, [SS96] e [AS66].

4.3 Exercícios

Exercício 4.1. Considere o integral

$$I = \int_0^1 \left(1 + e^{-x}\text{sen}\, 4x\right)\, dx.$$

a) Determine uma aproximação para I, usando: (*i*) a regra do trapézio; (*ii*) a regra de Simpson.

b) Sabendo que $I = 1.3082506046$ (10 c.d.), comente os resultados obtidos.

c) Esboce o gráfico da função $f(x) = 1+e^{-x}\text{sen}\,4x$ no intervalo $[0, 1]$ e sobreponha-lhe o gráfico de P_1 (polinómio linear interpolador de f em $x = 0$ e $x = 1$). Repita o exercício considerando o polinómio P_2 interpolador nos três pontos $x = 0$, $x = \frac{1}{2}$ e $x = 1$. Tendo em conta os gráficos obtidos, justifique os resultados das alíneas anteriores.

Exercício 4.2.

a) Deduza a chamada *Regra do Ponto Médio*,

$$\int_a^b f(x)\,dx = h\,f(\frac{a+b}{2}) + \frac{h^3}{24}\,f''(\xi)\,, \quad \xi \in (a,b)\,, \tag{4.10}$$

onde $h = b - a$ e se supõe $f \in C^2[a, b]$.

b) Interprete geometricamente a regra anterior.

c) A partir da regra (4.10), deduza a *Regra do Ponto Médio Composta*

$$\int_a^b f(x)dx = h\sum_{i=1}^{N} f_{i+\frac{1}{2}} + \frac{h^2}{24}(b-a)f''(\eta),$$

onde $x_{i+\frac{1}{2}} = a+(2i-1)\frac{h}{2}, i = 1, \ldots, N, h = \frac{b-a}{N}, f_{i+\frac{1}{2}} := f(x_{i+\frac{1}{2}})$ e $\eta \in (a, b)$.

d) Use a regra definida na alínea anterior, com $N = 10$, para estimar novamente o valor do integral do Exercício 4.1.

Exercício 4.3. Determine o número N de subintervalos que é necessário considerar para, usando a regra do trapézio composta, encontrar uma aproximação para o valor do integral

$$I = \int_2^7 \frac{1}{x}\,dx$$

com 6 casas decimais de precisão.

Nota: Nesta questão, ignoramos a contribuição dos erros de arredondamento.

✔ Exercício 4.4. Escreva uma função, $integral = \mathbf{regTrapezio}(f, a, b, N)$, para calcular uma aproximação para o valor de um integral $I = \int_a^b f(x)dx$, usando a regra do trapézio composta com N subintervalos. Essa função deve

- aceitar como argumentos:
  - uma função f (que pretendemos integrar);
  - dois números reais a e b, com $a < b$ (extremos do intervalo de integração);
  - um inteiro positivo N (número de subintervalos em que se pretende partir o intervalo de integração $[a, b]$);
- dar como resultado: uma aproximação $integral$ para I, obtida usando a fórmula da regra do trapézio composta (4.5).

Exercício 4.5. Considere o integral

$$I = \int_1^6 \left(2 + \operatorname{sen}\left(2\sqrt{x}\right)\right)\, dx.$$

a) Obtenha aproximações para I, usando a regra do trapézio composta com N subintervalos, T_N, para os seguintes valores sucessivos de N: $10, 20, 40, 80$ e 160.

b) Sabendo que o valor do integral (com 10 c.d.) é $I = 8.1834792077$, calcule $|E_{T_N}|$ para os sucessivos valores de N e diga se os resultados confirmam a ordem de aproximação $\mathcal{O}(h^2)$ ($h = (b-a)/N$) da regra do trapézio composta.

Exercício 4.6.

a) Estabeleça a seguinte estimativa para o erro da regra do trapézio com N subintervalos (N par):

$$|E_{T_N}(f)| \approx \left|\frac{T_N(f) - T_{N/2}(f)}{3}\right|. \tag{4.11}$$

b) Considere o integral

$$I = \int_{-0.1}^{0.1} \cos x dx.$$

Estime o valor de I usando T_4 e T_8 e obtenha, então, uma estimativa para $|E_{T_8}|$. Compare essa estimativa com o erro efectivamente cometido.

Exercício 4.7.

a) Modificando convenientemente a função **regTrapezio**, escreva uma função $[integral, erro] = \mathbf{erroTrapezio}(f, a, b, N)$ para calcular uma aproximação para o valor do integral $I = \int_a^b f(x)dx$, usando a regra do trapézio composta com N subintervalos (N par) e obter uma estimativa para o erro nessa aproximação, usando a fórmula (4.11).

Nota: A sua função deve estar escrita de forma a não repetir desnecessariamente o cálculo de valores da função f em pontos que sejam comuns ao uso das regras com N e com $N/2$ subintervalos.

b) Considere o integral $I = \int_1^2 \frac{1}{x}dx$. Use a função da alínea anterior para calcular um valor aproximado para I e uma estimativa para o respectivo erro, considerando a utilização da regra do trapézio composta com $N = 20$ subintervalos. Compare a estimativa para o erro com o erro efectivamente cometido no cálculo do integral.

Exercício 4.8. Escreva uma função $integral = \mathbf{regSimpson}(f, a, b, N)$ destinada a implementar a regra de Simpson composta com N subintervalos (N par). Usando essa função, repita o Exercício 4.5., mas trabalhando com a regra de Simpson composta.

Exercício 4.9.

a) Seja S_N um valor aproximado para o integral $I = \int_a^b f(x)dx$ obtido usando a regra de Simpson composta com N subintervalos (N múltiplo de 4) e seja $E_{S_N}(f)$ o respectivo erro de truncatura. Estabeleça a seguinte estimativa para o erro:

$$|E_{S_N}(f)| \approx \left| \frac{S_N(f) - S_{N/2}(f)}{15} \right|. \tag{4.12}$$

b) Escreva uma função $[integral, erro] = \mathbf{erroSimpson}(f, a, b, N)$ para calcular uma aproximação para o valor do integral $I = \int_a^b f(x)dx$, usando a regra de Simpson composta com N subintervalos (N múltiplo de 4), e obter uma estimativa para o erro nessa aproximação, usando a fórmula (4.12).

c) Considere novamente o integral $I = \int_{-0.1}^{0.1} \cos x \, dx$. Use a função definida na alínea anterior para calcular uma estimativa para I, usando $N = 8$ subintervalos, e estimar o respectivo erro. Compare essa estimativa com o erro efectivamente cometido.

Exercício 4.10. Mostre que a precisão de uma regra de quadratura do tipo

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^{n} A_i f(x_i)$$

é, no máximo, $2n - 1$.

Exercício 4.11. Determine o valor das constantes w_1 e w_2 de modo que a seguinte fórmula de quadratura

$$\int_{-1}^{1} f(x)dx \approx w_1 f(-1) + w_2 f(-\frac{1}{3}) + w_2 f(\frac{1}{3}) + w_1 f(1)$$

tenha a maior precisão possível e indique qual é essa precisão.

Exercício 4.12. Suponha que se pretende calcular o valor do integral $I = \int_a^b f(t)dt$. Note que, através de uma mudança de variável, definida por $t = \frac{a+b}{2} + \frac{b-a}{2}x$, se obtém

$$\int_a^b f(t)dt = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{a+b}{2} + \frac{b-a}{2}x\right) dx.$$

a) Deduza, então, que se pode utilizar a seguinte regra de quadratura para estimar o valor do integral:

$$\int_a^b f(t)dt \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{a+b}{2} + \frac{b-a}{2}x_i\right), \tag{4.13}$$

onde w_i e x_i são, respectivamente, os pesos e as abcissas da fórmula de quadratura de Gauss-Legendre com n pontos.

b) Utilize a fórmula de Gauss-Legendre com três pontos para estimar o valor do integral $I = \int_1^5 \frac{1}{t} dt$. Compare a estimativa obtida com o valor exacto do integral.

Exercício 4.13. Escreva uma função $[w, x] = \mathbf{pesosAbcissasGL}(n)$ que lhe permita ter acesso aos pesos e abcissas das fórmulas de Gauss-Legendre, para $2 \leq n \leq 6$. Mais precisamente, essa função deve

- aceitar como argumento: um inteiro n satisfazendo $2 \leq n \leq 6$;
- para cada valor de n, dar como resultados: dois vectores w e x, de dimensão n, contendo, respectivamente, os pesos e as abcissas da fórmula de Gauss-Legendre com n pontos.

✔ Exercício 4.14. Escreva uma função $integral = \mathbf{regQuadGL}(f, a, b, n)$ com o objectivo de estimar o valor de um integral do tipo $\int_a^b f(t)dt$, por meio da utilização da regra de Gauss-Legendre com n pontos ($2 \leq n \leq 6$). A função deve

- aceitar como argumentos:
  - uma função f (que pretendemos integrar);
  - dois números reais a e b, com $a < b$ (extremos do intervalo de integração);
  - um inteiro n, com $2 \leq n \leq 6$;
- invocar a função **pesosAbcissasGL** e calcular uma estimativa $integral$ para $I = \int_a^b f(t)dt$, usando a fórmula de Gauss-Legendre com n pontos, adaptada ao intervalo $[a, b]$, ou seja, a fórmula (4.13).

Exercício 4.15. Utilize a função definida no exercício anterior para estimar o valor dos seguintes integrais, usando diferentes valores de n:

$$a) \quad \int_0^2 \text{sen}\, t dt \qquad b) \quad \frac{1}{\pi} \int_0^\pi \cos(0.6\,\text{sen}\,t)dt \qquad c) \quad \int_0^1 te^{-t^2} dt$$

Exercício 4.16. Numa carta enviada a Hardy, o matemático indiano Ramanujan fez a seguinte conjectura: "*Dados dois inteiros positivos a e b, o número de inteiros entre a e b que são ou quadrados perfeitos ou soma de dois quadrados perfeitos é dado, aproximadamente, por:*

$$0.764 \int_a^b \frac{1}{\sqrt{\ln x}}\, dx."$$

Teste a afirmação de Ramanujan para os pares de números $(1,6), (3,27), (5,15), (5,16), (9,23)$ e $(12,28)$, usando a regra de quadratura de Gauss-Legendre com $n = 6$ para estimar o valor dos integrais.

Exercício 4.17. Obtenha informação sobre as funções **quad** e **quadl** e utilize-as para estimar os integrais cujo cálculo é requerido nas alíneas seguintes.

a) A função *erro*, erf, é definida por

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt\,.$$

Determine uma estimativa para erf(0.5).

Use a função pré-definida **erf** para estimar novamente o valor erf(0.5), comparando com o valor obtido por integração.

b) Tendo em conta que a área do círculo unitário é $A = \pi$, deduza que

$$\frac{\pi}{4} = \int_0^1 \sqrt{1 - x^2} dx\,.$$

Obtenha uma estimativa para π, usando o integral anterior.

4.4 Trabalhos

Trabalho 4.1. **Regra do Trapézio Corrigida**

Pode estabelecer-se a seguinte regra de quadratura, conhecida por *Regra do Trapézio Corrigida*

$$\int_a^b f(x)dx = \frac{h}{2}\left(f(a) + f(b)\right) + \frac{h^2}{12}\left(f'(a) - f'(b)\right) + \frac{h^5}{720} f^{(4)}(\xi), \qquad (4.14)$$

onde $h = b - a$, $\xi \in (a, b)$ e, naturalmente, se supõe que $f \in C^4[a, b]$.

a) Deduza, a partir da regra (4.14), a chamada *Regra do Trapézio Corrigida Composta*:

$$\int_a^b f(x)dx = \frac{h}{2}\left[f_1 + 2\left(f_2 + \cdots + f_N\right) + f_{N+1}\right] + \frac{h^2}{12}\left(f'(a) - f'(b)\right) + \frac{(b-a)h^4}{720} f^{(4)}(\eta), \quad (4.15)$$

onde $x_i = a + (i-1)h; i = 1, \ldots, N+1; h = \frac{b-a}{N}$, $f_i := f(x_i)$ e $\eta \in (a, b)$.

b) Mostre que, se f é periódica de período $b - a$ e $f \in C^4[a, b]$, então

$$\int_a^b f(x)dx = \frac{h}{2}\left[f_1 + 2\left(f_2 + \ldots + f_N\right) + f_{N+1}\right] + \frac{(b-a)h^4}{720} f^{(4)}(\eta) \quad (4.16)$$

e deduza que a regra do trapézio composta é especialmente adaptada à integração, entre a e b, deste tipo de funções.

c) Se $TC_N(f)$ designa o valor dado pela regra do trapézio corrigida composta, com N subintervalos (N par), e $E_{TC_N}(f)$ o respectivo erro, estabeleça a seguinte estimativa

$$|E_{TC_N}(f)| \approx \left|\frac{TC_N(f) - TC_{N/2}(f)}{15}\right|. \quad (4.17)$$

d) Escreva uma função $[integral, erro] = \mathbf{regTrapezioCorr}(f, a, b, N)$ para calcular uma aproximação para o valor de um integral $I = \int_a^b f(x)dx$, usando a regra do trapézio corrigida composta com N subintervalos (N par) e estimar o respectivo erro pelo uso da fórmula (4.17).

e) Considere o integral

$$I = \int_0^1 e^{-x}dx.$$

Calcule

$$\log_2 \frac{|E_{TC_N}(f)|}{|E_{TC_{N/2}}(f)|} \quad (4.18)$$

para $N = 10, 20, 40, 80$ e diga se os resultados confirmam a ordem de convergência da regra do trapézio composta corrigida.

Trabalho 4.2. **Tabela de Romberg**

Seja $T_N := T_N(f)$ a aproximação para $I = \int_a^b f(x)dx$ obtida usando a regra do trapézio composta com N subintervalos e seja $h = (b-a)/N$. Pode provar-se que, se $f \in C^{2M+2}[a,b]$, então

$$I - T_N = c_1 h^2 + c_2 h^4 + \ldots + c_M h^{2M} + \mathcal{O}(h^{2M+2}), \tag{4.19}$$

com as constantes c_i independentes de h. Tem-se, então, se usarmos $2N$ subintervalos:

$$I - T_{2N} = \frac{1}{4} c_1 h^2 + \frac{1}{16} c_2 h^4 + \ldots + \frac{1}{4^M} c_M h^{2M} + \mathcal{O}(h^{2M+2}). \tag{4.20}$$

Multiplicando (4.20) por 4 e subtraindo de (4.19), vem

$$I - \frac{4T_{2N} - T_N}{3} = d_2 h^4 + d_3 h^6 + \ldots + d_M h^{2M} + \mathcal{O}(h^{2M+2}). \tag{4.21}$$

Denotando por T^1_{2N} o valor

$$T^1_{2N} := \frac{4T_{2N} - T_N}{3}$$

temos

$$I - T^1_{2N} = d_2 h^4 + d_3 h^6 + \ldots + d_M h^{2M} + \mathcal{O}(h^{2M+2}). \tag{4.22}$$

Virá, então, usando $4N$ intervalos,

$$I - T^1_{4N} = \frac{1}{16} d_2 h^4 + \frac{1}{128} d_3 h^6 + \ldots + \frac{1}{4^M} d_M h^{2N} + \mathcal{O}(h^{2M+2}). \tag{4.23}$$

Multiplicando esta equação por 16 e subtraindo de (4.22), vem

$$I - T^2_{4N} = e_3 h^6 + \ldots + e_M h^{2M} + \mathcal{O}(h^{2M+2}), \tag{4.24}$$

onde usámos a notação

$$T^2_{4N} := \frac{16T^1_{4N} - T^1_{2N}}{15}.$$

Este processo pode, naturalmente, continuar, dependendo apenas da suavidade de f. Introduzindo a notação $T^0_N := T_N(f)$ obter-se-ão, então, aproximações para I, dadas por

$$T^j_{2^k N} := \frac{4^j\, T^{j-1}_{2^k N} - T^{j-1}_{2^{k-1} N}}{4^j - 1}; j = 1, 2, \ldots, M; k = j, j+1, \ldots, M.$$

Tem-se

$$|I - T^j_{2^k N}| = \mathcal{O}\left((\frac{b-a}{2^k N})^{2j+2}\right); j = 1, 2, \ldots M; k = j, j+1, \ldots, M. \qquad (4.25)$$

É costume visualizar estas aproximações para I como entradas numa tabela triangular, chamada *Tabela de Romberg*:

$$\begin{array}{llll} T^0_N \equiv T_N & & & \\ T^0_{2N} \equiv T_{2N} & T^1_{2N} & & \\ T^0_{4N} \equiv T_{4N} & T^1_{4N} & T^2_{4N} & \\ \vdots & \vdots & \vdots & \\ T^0_{2^M N} \equiv T_{2^M N} & T^1_{2^M N} & T^2_{2^M N} & \cdots \quad T^M_{2^M N} \end{array} \qquad (4.26)$$

a) Escreva uma função $TR = \mathbf{tabRomberg}(f, a, b, N, M)$ destinada a construir uma tabela de Romberg (4.26) relativa ao cálculo de aproximações para $I = \int_a^b f(x)dx$. O valor de N é o número de intervalos inicial da regra do trapézio composta e $M+1$ é o número de colunas da tabela. A tabela será armazenada numa matriz TR, triangular inferior.

b) Seja $I = \int_0^1 e^{2x}dx$. Determine a tabela de Romberg para o cálculo de I, considerando $N = 1$ e $M = 5$.

c) Obtenha uma tabela com os erros em cada uma das aproximações para I e efectue os cálculos necessários para verificar (numericamente) se estão de acordo com as ordens teoricamente previstas.

Trabalho 4.3. **Regra de Simpson adaptativa**

Suponha que se pretende obter uma aproximação para

$$I = \int_a^b f(x)dx$$

com um erro inferior a uma certa tolerância ϵ. Sejam S_2 e S_4 aproximações para I obtidas usando a regra de Simpson com $N = 2$ e $N = 4$ subintervalos, respectivamente. Então, como vimos no Exercício 4.9, podemos tomar o valor de

$$E = \frac{|S_4 - S_2|}{15}$$

como uma estimativa para o erro na aproximação S_4, isto é, considerar

$$|I - S_4| \approx E.$$

Assim, se $E < \epsilon$, podemos considerar que S_4 é a aproximação desejada para I. Caso isso não se verifique, repetimos o procedimento anterior para cada um dos subintervalos $[a, (a+b)/2]$ e $[(a+b)/2, b]$, determinando se (a estimativa para) o erro na aproximação em cada um desses subintervalos é inferior a $\epsilon/2$. Se tal acontecer, a soma das aproximações será uma aproximação para I com uma tolerância ϵ. Se a estimativa para o erro da aproximação relativa a algum dos subintervalos não for inferior a $\epsilon/2$, esse intervalo será novamente dividido, sendo a aproximação de cada novo subintervalo analisada para ver se o erro é inferior a $\epsilon/4$. Este procedimento repetir-se-á até que a aproximação de cada subintervalo esteja dentro da tolerância desejada. O processo que acabámos de descrever constitui a chamada regra de Simpson adaptativa.

a) Escreva uma função $integral = \mathbf{regSimpsonAdapt}(f, a, b, tol)$ para calcular uma aproximação para $I = \int_a^b f(x)dx$ com tolerância tol, usando a regra de Simpson adaptativa.

A sua função deverá ser usada recursivamente (isto é, deverá invocar-se a ela própria) e deve fazer uso da função **erroSimpson** para calcular (em cada intervalo) os dois integrais necessários à obtenção da estimativa para o erro.

b) Considere o cálculo de

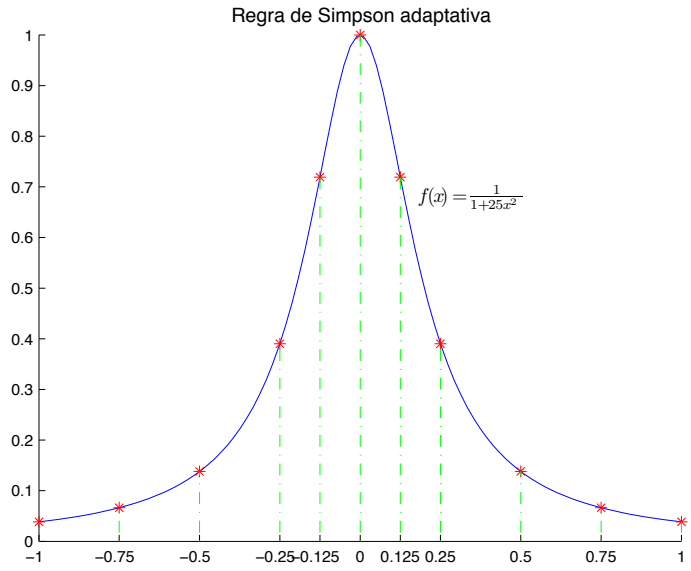
$$I = \int_{-1}^{1} \frac{1}{1 + 25x^2} dx.$$

Usando a função **regSimpsonAdapt**, determine uma estimativa para I com precisão de 4 casas decimais.

c) Esboce o gráfico de $f(x) = \frac{1}{1+25x^2}$ no intervalo $[-1, 1]$ e marque sobre ele os pontos usados na obtenção da estimativa da alínea anterior, isto é, obtenha um gráfico semelhante ao apresentado na figura seguinte.

Nota: Para poder armazenar os pontos usados, poderá ter de definir uma variável global.

d) Para obter uma aproximação para I, seria razoável usar uma regra de quadratura de Newton-Cotes fechada com n grande? Porquê?


Regra de Simpson adaptativa
$f(x) = \frac{1}{1+25x^2}$
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−1
−0.75
−0.5
−0.25
−0.125
0
0.125
0.25
0.5
0.75
1

4.5 Resoluções

Resolução do Exercício 4.4.

```
function integral=regTrapezio(f,a,b,N)
% REGTRAPEZIO  Calcula integral pela regra do trapezio composta
%
% INTEGRAL=REGRATRAPEZIO(F,A,B,N)
%   calcula uma aproximacao para o valor do integral entre A e B da
%   funcao F, usando a regra do trapezio composta com N subintervalos.
%
% PARAMETROS DE ENTRADA:
%   F: uma funcao;
%   A,B: reais, com A<B (extremos do intervalo de integracao);
%   N: numero inteiro positivo (numero de subintervalos).
%
% PARAMETRO DE SAIDA:
%   INTEGRAL: aproximacao para o integral calculada usando a regra do
%             trapezio composta com N subintervalos.
%
%  A funcao Y=F(X) deve ser ''vectorizada'' (i.e. deve usar os
%  operadores pontuais  .*, ./ e .^ na sua definicao); a funcao deve ser
%  especificada como uma funcao anonima ou atraves de uma function_handle.

%-------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%-------------------------------------------------------------------------
% Verificar se N e inteiro e se A<B
if (length(N)~=1) | (fix(N) ~= N) | (N < 1)
  error('N deve ser inteiro positivo');
elseif (a>=b)
    error('a deve ser menor que b')
end
%
%-------------------------------------------------------------------------
%          DETERMINACAO DE H E CRIACAO DE VECTOR DOS NOS
%-------------------------------------------------------------------------
h=(b-a)/N;
nos=a:h:b;
%-------------------------------------------------------------------------
%                 REGRA DO TRAPEZIO COMPOSTA
%-------------------------------------------------------------------------
fnos=f(nos);         % Vector dos valores da funcao dada nos nos
fnosMeio=fnos(2:end-1);    % Exclusao dos pontos inicial e final
integral=(h/2)*(fnos(1)+2*sum(fnosMeio)+fnos(end));
```

Listagem 4.1. Função **regTrapezio**

Resolução do Exercício 4.14. *b)*

```
function integral=regQuadGL(f,a,b,n)
%REGQUADGL  Calcula integral por regra de Gauss-Legendre
%
% INTEGRAL=REGQUADGL(F,A,B,N)
%   calcula uma aproximacao para o valor do integral entre A e B da
%   funcao F, usando a regra de Gauss-Legendre com N pontos.
%
% PARAMETROS DE ENTRADA:
%   F: uma funcao;
%   A,B: reais, com A<B (extremos do intervalo de integracao);
%   N: numero inteiro positivo entre 2 e 6 (numero de pontos).
%
% PARAMETRO DE SAIDA:
%   INTEGRAL: aproximacao para o integral calculada usando a regra de
%             Gauss-Legendre com N pontos.
%
%  A funcao Y=F(X) deve ser ''vectorizada'' (i.e. deve usar os
%  operadores pontuais  .*, ./ e .^ na sua definicao); a funcao pode ser
%  especificada como uma  funcao anonima ou atraves de uma function_handle.
%
%  Esta funcao necessita da funcao PESOSABCISSASGL

%-------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%-------------------------------------------------------------------------
% Verificar se A<B; a verificacao relativa a N (2 <=N<=6) faz-se ao invocar
% a funcao PESOSABCISSASGL
if (a>=b)
    error('a deve ser menor que b')
end
%
%-------------------------------------------------------------------------
%          USO DE PESOSABCISSASGL PARA OBTER PESOS E ABCISSAS  DA REGRA
%          DE GAUSS-LEGENDRE PARA N ESCOLHIDO
%-------------------------------------------------------------------------
[w x]=pesosAbcissasGL(n);
%-------------------------------------------------------------------------
%                  REGRA DE GAUSS-LEGENDRE
%-------------------------------------------------------------------------
nos=((b-a)/2)*x+(a+b)/2; % Mudanca de variavel e criacao dos n'os em [A,B]
fnos=f(nos);       % Vector com valores da funcao nos n'os
integral=((b-a)/2)*w'*fnos; % Calculo do integral
```

5. Sistemas de Equações Lineares

Método de Gauss
Decomposição LU
Decomposição de Cholesky
Métodos de Jacobi e de Gauss-Seidel
Condicionamento

5.1 Notações, definições e resultados básicos

Ao longo deste capítulo, $\mathbb{R}^{n\times n}$ designa o espaço das matrizes reais, quadradas de ordem n e, salvo indicação em contrário, qualquer matriz referida será um elemento deste espaço.

Considere-se um sistema de n equações lineares em n incógnitas escrito na forma matricial

$$Ax = b, \tag{5.1}$$

onde $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ (matriz do sistema), $b = (b_i) \in \mathbb{R}^n$ (vector dos termos independentes) e $x = (x_i) \in \mathbb{R}^n$ é um vector cujas componentes pretendemos determinar (vector das incógnitas). Seja $(A|b)$ a matriz ampliada do sistema, isto é, a matriz $n \times (n+1)$ formada juntando a A uma última coluna com os elementos de b.

No que se segue, suporemos que o sistema em causa admite solução única.

Os métodos numéricos para a resolução de sistemas são tradicionalmente agrupados em duas categorias: *métodos directos* e *métodos iterativos*. Nos métodos directos, a solução é determinada num número finito de operações aritméticas. Se não existissem erros de arredondamento, tais métodos produziriam a solução exacta; na prática, como os erros de arredondamento são inevitáveis, os métodos conduzem apenas a soluções aproximadas.

Nos métodos iterativos, a partir de uma aproximação inicial para a solução, gera-se uma sequência de aproximações que, sob certas condições, converge para a solução do problema. Na prática, o processo será interrompido ao fim de um certo número de iterações.

5.1.1 Métodos directos

Sistemas triangulares

• *Sistema triangular inferior*

Suponhamos que a matriz do sistema é triangular inferior, isto é, $A = L = (\ell_{ij})$, com $\ell_{ij} = 0$ para $j > i$.[1] A solução de tal sistema pode obter-se facilmente pelo processo da chamada *substituição directa*, cujo algoritmo se apresenta de seguida. Este algoritmo, tal como todos os outros incluídos nestes textos, está escrito numa linguagem adaptada à sua implementação em MATLAB.

função $x = \mathbf{subDirecta}(L, b)$ (1)

% *Resolução de um sistema $Lx = b$, com L triangular inferior*

% **Parâmetros de entrada:** matriz L, triangular inferior de ordem n
vector b dos termos independentes

% **Parâmetro de saída:** vector x, solução do sistema

para $i = 1{:}n$

$$x(i) = \frac{b(i) - L(i, 1{:}i-1) * x(1{:}i-1)}{L(i, i)} \quad (2)$$

fim

% Número de operações: $\frac{n^2}{2} + \mathcal{O}(n)$

Algoritmo 5.1. Substituição Directa

Nota: Para não aumentar as necessidades de armazenamento, as incógnitas que vão sendo calculadas poderão ser sobrepostas aos valores de $b(i)$ (uma vez que estes não necessitam de ser utilizados novamente), isto é, as linhas (1) e (2) do algoritmo poderão ser substituídas, respectivamente, por (1') e (2'):

[1] Note-se que, como estamos a admitir que $A = L$ é invertível, teremos de ter $\ell_{ii} \neq 0; i = 1, \ldots, n$.

função $b = \mathbf{subDirecta}(L, b)$ (1')

$$b(i) = \frac{b(i) - L(i, 1\!:\!i-1) * b(1\!:\!i-1)}{L(i,i)}. \quad (2')$$

Neste caso, à saída, o vector b conterá a solução do sistema.

• *Sistema triangular superior*

Um algoritmo semelhante, no caso em que $A = U = (u_{ij})$ é uma matriz triangular superior, será o seguinte (correspondendo à chamada *substituição inversa*).

função $b = \mathbf{subInversa}(U, b)$

% *Resolução de um sistema $Ux = b$, com U triangular superior*

% **Parâmetros de entrada:** matriz U, triangular superior de ordem n
vector b dos termos independentes

% **Parâmetro de saída:** vector b, solução do sistema

para $i = n\!:\!-1\!:\!1$

$$b(i) = \frac{b(i) - U(i, i+1\!:\!n) * b(i+1\!:\!n)}{U(i,i)}$$

fim

% Número de operações: $\frac{n^2}{2} + \mathcal{O}(n)$

Algoritmo 5.2. Substituição Inversa

Método de Gauss

Relembremos que, dada a matriz ampliada de um sistema $(A|b)$, se efectuarmos sobre essa matriz qualquer operação elementar sobre linhas, obtemos uma matriz que corresponde a um sistema equivalente ao sistema inicial.[2]

A ideia do método de Gauss é fazer uso das operações elementares sobre linhas para converter a matriz ampliada do sistema $(A|b)$ numa matriz da forma $(U|\beta)$, onde U é uma matriz triangular superior. O sistema correspondente poderá, então, ser resolvido por substituição inversa.

[2]Recordemos que as operações elementares sobre linhas são: troca de linhas; multiplicação de uma linha por um escalar diferente de zero; adição a uma linha de outra linha multiplicada por um escalar.

O processo de redução pode ser efectuado em $n-1$ passos, começando com $A^{(0)} := A$ e $b^{(0)} := b$, e terminando num sistema triangular $(U|\beta) = (A^{(n-1)}|b^{(n-1)})$. No início do passo k ($1 \leq k \leq n-1$) ter-se-á uma matriz $(A^{(k-1)}|b^{(k-1)})$, onde

$$A^{(k-1)} = \begin{pmatrix} A_{11}^{(k-1)} & A_{12}^{(k-1)} \\ 0 & A_{22}^{(k-1)} \end{pmatrix},$$

com $A_{11}^{(k-1)}$ uma matriz de ordem $k-1$ triangular superior. O objectivo do passo k é reduzir a zero os elementos da coluna k de $A^{(k-1)}$ situados abaixo da diagonal. Tal será conseguido subtraindo de cada uma das linhas $k+1, \ldots, n$, o produto da linha k, respectivamente por cada um dos números m_{ik} – chamados *multiplicadores* – dados por

$$m_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}; k = 1, \ldots, n-1; i = k+1, \ldots, n. \tag{5.2}$$

Os elementos $a_{kk}^{(k-1)}$ são chamados *pivots* do processo de redução. Apresenta-se de seguida um algoritmo básico do processo de eliminação de Gauss.

função $[A, b] = \mathbf{elimGauss}(A, b)$

% *Redução de um sistema $Ax = b$ à forma triangular*

% **Parâmetros de entrada:** matriz A de ordem n
vector b dos termos independentes

% **Parâmetros de saída:** matriz A triangular superior (ver Nota)
vector b modificado

para $k = 1{:}n-1$

$$mult = A(k+1{:}n, k)/A(k, k) \tag{1}$$

$$A(k+1{:}n, k+1{:}n) = A(k+1{:}n, k+1{:}n) - mult * A(k, k+1{:}n) \tag{2}$$

$$b(k+1{:}n) = b(k+1{:}n) - b(k) * mult \tag{3}$$

fim

% Número de operações: $\frac{2n^3}{3} + \mathcal{O}(n^2)$

Algoritmo 5.3. Eliminação Gaussiana

Nota:

- Como mostram as linhas (1) e (2) do algoritmo precedente, os sucessivos valores $a_{ij}^{(k)}$ e $b_i^{(k)}$ calculados durante o processo são sobrepostos aos valores anteriores, isto é, são armazenados nos "espaços" correspondentes aos valores iniciais a_{ij} e b_i.

- No algoritmo acima, não tornámos explicitamente iguais a zero os elementos abaixo da diagonal, uma vez que apenas nos interessa conhecer a matriz triangular superior resultante da redução, a qual será usada na substituição inversa; assim, à saída, A não é, na realidade, uma matriz triangular inferior.[3]

- Caso estejamos interessados em armazenar os diversos multiplicadores usados no processo de redução, poderemos aproveitar a "parte triangular inferior" de A para os armazenar. Neste caso, deveremos substituir as linhas (1)-(3) do algoritmo anterior, respectivamente por (1')-(3'), como a seguir se indica:

$$A(k+1:n,k) = A(k+1:n,k)/A(k,k) \quad (1')$$

$$A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k) * A(k,k+1:n) \quad (2')$$

$$b(k+1:n) = b(k+1:n) - A(k+1:n,k) * b(k) \quad (3')$$

Se, ao iniciarmos o passo k do método de eliminação de Gauss, for $a_{kk}^{(k-1)} = 0$ (*pivot* nulo), o processo não pode prosseguir. Além disso, se $a_{kk}^{(k-1)}$ for "pequeno" (relativamente aos restantes elementos da matriz), poderá haver problemas de instabilidade no algoritmo; por esse motivo, o método de Gauss deverá ser implementado acompanhado de *escolha de pivot*; existem várias técnicas de escolha de *pivot*, das quais a mais usada é a chamada *escolha parcial de pivot*, que consiste no seguinte: no início do passo k, determina-se a linha ℓ tal que $|a_{\ell,k}^{(k-1)}| = \max_{k\leq i\leq n} |a_{ik}^{(k-1)}|$; se $\ell \neq k$, procede-se à troca das linhas k e ℓ. Note-se que com esta técnica garantimos que todos os multiplicadores têm módulo não superior a 1.

[3] Embora seja muto fácil, usando uma função pré-definida no MATLAB (qual?), obter essa matriz.

Apresenta-se de seguida um algoritmo para a resolução de um sistema $Ax = b$ pelo método de Gauss (redução à forma triangular, com escolha parcial de pivot, seguida de substituição inversa).

função $b = \mathbf{metGauss}(A, b)$

% *Resolução de um sistema $Ax = b$, pelo método de Gauss, com escolha parcial de pivot;*
% *utiliza a função* **subInversa** *(Algoritmo 5.2)*

% **Parâmetros de entrada:** matriz A de ordem n
% vector b dos termos independentes

% **Parâmetro de saída:** vector b, solução do sistema

% *Eliminação com escolha parcial de pivot*

para $k = 1{:}n-1$

% *Escolha de pivot*

Determinar ℓ tal que $|A(\ell, k)| = \max|A(k{:}n, k)|$

$A(k, k{:}n) \leftrightarrow A(\ell, k{:}n)$ % ($\leftrightarrow$ denota troca dos respectivos conteúdos)

$b(k) \leftrightarrow b(\ell)$

% *Eliminação*

$A(k+1{:}n, k) = A(k+1{:}n, k)/A(k, k)$

$A(k+1{:}n, k+1{:}n) = A(k+1{:}n, k+1{:}n) - A(k+1{:}n, k) * A(k, k+1{:}n)$

$b(k+1{:}n) = b(k+1{:}n) - b(k) * A(k+1{:}n, k)$

fim

% *Substituição inversa*

$b = \mathbf{subInversa}(A, b)$

% Número de operações: $\frac{2n^3}{3} + \mathcal{O}(n^2)$

Algoritmo 5.4. Método de Gauss com escolha parcial de pivot

Decomposição LU

Chama-se *decomposição LU* de uma matriz A à factorização de A na forma

$$A = LU, \tag{5.3}$$

onde L é uma matriz triangular inferior de elementos diagonais iguais a 1 e U é triangular superior. Se uma matriz invertível A admite uma decomposição LU, esta decomposição é única. Pode também provar-se que é condição suficiente de existência da decomposição LU de A que todas as submatrizes principais de A contidas nas suas primeiras k linhas e k colunas ($1 \leq k \leq n$) sejam não singulares.

Seja dado um sistema de equações $Ax = b$ e suponhamos que conhecemos a decomposição LU de A, $A = LU$. Então, tem-se

$$Ax = b \iff (LU)x = b \iff L(Ux) = b \iff \left\{ \begin{array}{l} Ly = b \\ Ux = y. \end{array} \right. \tag{5.4}$$

Vemos assim que, para resolver o sistema dado, bastará resolver sucessivamente os dois sistemas triangulares $Ly = b$ e $Ux = y$, o primeiro por substituição directa, e o segundo por substituição inversa.

Cálculo da decomposição LU

Seja

$$A = \underbrace{\begin{pmatrix} 1 & & & & \\ \vdots & \ddots & & & \\ \ell_{k1} & \dots & 1 & & \\ \vdots & & \vdots & \ddots & \\ \ell_{n1} & \dots & \ell_{ni} & \dots & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} u_{11} & \dots & u_{1j} & \dots & u_{1n} \\ & \ddots & \vdots & & \vdots \\ & & u_{jj} & \dots & u_{jn} \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{pmatrix}}_{U}.$$

Igualando os elementos correspondentes de ambos os lados da igualdade anterior, tem-se

$$a_{kj} = \ell_{k1}u_{1j} + \ldots + \ell_{k,k-1}u_{k-1,j} + \boxed{u_{kj}}, \quad \text{se } k \leq j, \tag{5.5}$$

$$a_{kj} = \ell_{k1}u_{1j} + \ldots + \boxed{\ell_{kj}}\, u_{jj}, \quad \text{se } k > j. \tag{5.6}$$

As equações anteriores podem ser usadas para obter as incógnitas $\ell_{kj}; k > j$ e $u_{kj}; k \leq j$, desde que se ordenem estas incógnitas convenientemente. Suponhamos que já determinámos as primeiras $k-1$ linhas de U e as primeiras $k-1$ colunas de L; podemos

então determinar os elementos assinalados com a caixa, os quais formam a $k-$ésima linha de U e a $k-$ésima coluna de L:

$$u_{kj} = a_{kj} - \ell_{k1}u_{1j} - \ell_{k2}u_{2j} - \ldots - \ell_{k,k-1}u_{k-1,j}, \quad k \leq j, \tag{5.7}$$

$$\ell_{kj} = \left(a_{kj} - \ell_{k1}u_{1j} - \ell_{k2}u_{2j} - \ldots - \ell_{k,j-1}u_{j-1,j}\right)/u_{jj}, \quad k > j. \tag{5.8}$$

Este algoritmo para obter a decomposição LU de A é conhecido por *algoritmo de Doolitle*; outra forma alternativa de decompor A no produto de duas matrizes triangulares L e U, consiste em exigir que seja a diagonal da matriz U a ter elementos iguais a 1; o correspondente algoritmo é designado por *algoritmo de Crout*.

Decomposição LU e eliminação Gaussiana

Seja A uma matriz não singular e suponhamos que efectuámos sobre A os $n-1$ passos do processo de eliminação de Gauss, tal como descrevemos anteriormente. Seja $U = A^{(n-1)}$ a matriz triangular superior obtida no final do processo de redução, que supomos ter sido possível efectuar sem escolha de pivot, e seja $L = (\ell_{ik})$ a matriz triangular inferior com elementos diagonais iguais a 1 e tendo, na parte estritamente triangular inferior, os multiplicadores usados no processo de redução, isto é, sejam $\ell_{ik} = m_{ik}; k = 1, \ldots, n-1; i = k+1, \ldots, n$. Então, pode mostrar-se que $A = LU$, isto é, que as matrizes L e U assim obtidas dão a decomposição LU de A. Temos, assim, um outro processo de obter a decomposição LU de uma matriz.

Se efectuarmos escolha parcial de pivot no algoritmo de decomposição (o que é necessário, como referimos, por questões de estabilidade), então ao formarmos as matrizes L e U como acima, no final obter-se-á, não a decomposição LU da matriz A, mas sim da matriz A com as suas linhas trocadas (correspondendo às trocas que foram efectuadas na redução), ou seja, obter-se-á

$$LU = PA,$$

onde P é uma matriz de permutação.[4] Sendo $Ax = b$ e tendo-se $PA = LU$, vem

$$Ax = b \iff P(Ax) = Pb \iff (PA)x = Pb \iff (LU)x = Pb \iff \begin{cases} Ly = Pb \\ Ux = y. \end{cases}$$

[4] Uma matriz P é uma matriz de permutação se for obtida da matriz identidade I por troca de linhas; a pré-multiplicação de uma matriz A por uma matriz de permutação P "troca", em A, as mesmas linhas que foram trocadas para obter P a partir de I; uma matriz de permutação é invertível, tendo-se $P^{-1} = P$.

Assim, se dispusermos da decomposição LU de PA, podemos novamente encontrar a solução do sistema $Ax = b$ resolvendo dois sistemas triangulares, devendo, neste caso, começar por resolver-se o sistema $Ly = Pb$.

Decomposição de Cholesky

Uma matriz simétrica $A \in \mathbb{R}^{n\times n}$ diz-se definida positiva, se $x^{\mathsf{T}}Ax > 0$ para todo o vector não nulo $x \in \mathbb{R}^n$. Se A é definida positiva, é possível encontrar a chamada *decomposição de Cholesky* de A, isto é, a sua decomposição na forma

$$A = R^{\mathsf{T}}R, \tag{5.9}$$

onde $R \in \mathbb{R}^{n\times n}$ é uma matriz triangular superior de elementos diagonais *positivos*. Essa decomposição é, além disso, única. De $A = R^{\mathsf{T}}R$, vem $a_{ij} = r_{1i}r_{1j} + \ldots + r_{i-1,i}r_{i-1,j} + r_{ii}r_{ij};\ j \geq i$, donde

$$r_{ii} = \left(a_{ii} - r_{1i}^2 - \ldots - r_{i-1,i}^2\right)^{1/2}, \tag{5.10}$$

$$r_{ij} = \left(a_{ij} - r_{1i}r_{1j} - \ldots - r_{i-1,i}r_{i-1,j}\right)/r_{ii};\quad j > i. \tag{5.11}$$

Tem-se, então, o seguinte algoritmo para obter a decomposição de Cholesky de uma matriz A simétrica definida positiva.

função $R = \mathbf{decCholesky}(A)$

% *Decomposição de Cholesky de uma matriz A simétrica definida positiva*

% **Parâmetro de entrada:** matriz A, de ordem n, simétrica definida positiva

% **Parâmetro da saída:** matriz R, triangular superior, de elementos diagonais
% positivos e tal que $A = R^{\mathsf{T}}R$

Inicializar R como a matriz nula

para $i = 1{:}n$

$$R(i,i) = (A(i,i) - R(1{:}i-1,i)^{\mathsf{T}} * R(1{:}i-1,i))^{1/2} \tag{1}$$
$$R(i,i+1{:}n) = (A(i,i+1{:}n) - R(1{:}i-1,i)^{\mathsf{T}} * R(1{:}i-1,i+1{:}n))/R(i,i)$$

fim

% Número de operações: $\frac{n^3}{3} + \mathcal{O}(n^2)$

Algoritmo 5.6. Decomposição de Cholesky

Nota: Pode mostrar-se que, sendo A definida positiva, em (1), tem-se sempre

$$A(i,i) - R(1\!:\!i-1,i)^{\mathsf{T}} * R(1\!:\!i-1,i) > 0. \tag{5.12}$$

5.1.2 Métodos iterativos

Métodos de Jacobi e de Gauss-Seidel

Consideremos novamente o problema da resolução de um sistema de n equações lineares a n incógnitas $Ax = b$, onde $A = (a_{ij})$ é não singular e suponhamos que $a_{ii} \neq 0;\ i = 1, \ldots, n$. As equações do sistema podem, naturalmente, ser rearranjadas da seguinte forma

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j);\ i = 1, \ldots, n.$$

O método iterativo de Jacobi para a resolução do sistema é, então, definido por

$$\left.\begin{array}{l} x_i^{(k+1)} = \dfrac{1}{a_{ii}}(b_i - \sum\limits_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)}); i = 1, \ldots, n;\ k = 0, 1, 2, \ldots; \\ x_i^{(0)};\ i = 1, \ldots, n, \quad \text{dados} \end{array}\right\} \tag{5.13}$$

No método de Gauss-Seidel, os valores "actualizados"são utilizados, mal estejam disponíveis. Mais precisamente, o método é definido por

$$\left.\begin{array}{l} x_i^{(k+1)} = \dfrac{1}{a_{ii}}(b_i - \sum\limits_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum\limits_{j=i+1}^{n} a_{ij}x_j^{(k)}); i = 1, \ldots, n;\ k = 0, 1, 2, \ldots; \\ x_i^{(0)};\ i = 1 \ldots, n, \quad \text{dados} \end{array}\right\} \tag{5.14}$$

Formulação Matricial

A matriz A do sistema pode decompor-se como

$$A = D + M + N, \tag{5.15}$$

onde D, M e N são, respectivamente, uma matriz diagonal (contendo a diagonal de A), uma matriz estritamente triangular inferior (com a parte triangular inferior de A) e uma matriz estritamente triangular superior (com a parte triangular superior de A).

Os dois métodos iterativos referidos podem, então, ser descritos matricialmente como

$$x^{(k+1)} = Bx^{(k)} + c, \tag{5.16}$$

onde:

- no método de Jacobi,

$$B = B_J := -D^{-1}(M+N) \quad \text{e} \quad c = c_J := D^{-1}b \tag{5.17}$$

- no método de Gauss-Seidel,

$$B = B_{GS} :== -(D+M)^{-1}N \quad \text{e} \quad c = c_{GS} := (D+M)^{-1}b. \tag{5.18}$$

As matrizes B_J e B_{GS} são ditas, respectivamente, *matriz de iteração de Jacobi* e *matriz de iteração de Gauss-Seidel*.

Convergência

Seja $\epsilon^{(k)}$ o vector erro na iteração k de um dos métodos considerados, isto é, seja $\epsilon^{(k)} := x - x^{(k)}$. Dizemos que o respectivo método converge se, para qualquer aproximação inicial $x^{(0)}$, isto é, para qualquer erro inicial $\epsilon^{(0)}$, tivermos

$$\lim_{k\to\infty} \epsilon^{(k)} = 0.^{5}$$

O teorema seguinte estabelece uma condição necessária e suficiente de convergência dos métodos iterativos considerados.

Teorema 5.1 *É condição necessária e suficiente de convergência dos métodos de Jacobi e de Gauss-Seidel que as respectivas matrizes de iteração tenham raio espectral inferior a 1.*[6]

A rapidez de convergência do método depende do "tamanho" do raio espectral da matriz de iteração, sendo, em geral, tanto mais rápida quanto menor for essa quantidade. Pode, além disso, estabelecer-se o seguinte teorema:

Teorema 5.2 *É condição suficiente de convergência, quer do método de Jacobi, quer do método de Gauss-Seidel, que a matriz do sistema seja de diagonal estritamente dominante.*[7]

[5] Com o significado de $\lim_{k\to\infty} e_i^{(k)} = 0; i = 1, \ldots, n$.

[6] Dada uma matriz quadrada A, chama-se raio espectral de A e denota-se por $\rho(A)$ o número definido por $\rho(A) := \max\{|\lambda_i| : \lambda_i \text{ é valor próprio de } A\}$.

[7] Uma matriz quadrada A diz-se de diagonal estritamente dominante se $|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|;\ i = 1, \ldots, n$.

5.1.3 Normas vectoriais e normas matriciais

Normas vectoriais

Seja $X = \mathbb{R}^n$. Uma aplicação $\|\cdot\| : X \longrightarrow \mathbb{R}$ diz-se uma *norma vectorial* se satisfizer as seguintes propriedades:

N1 $\forall x \in X, \quad \|x\| \geq 0$ e $\|x\| = 0$ se e só se $x = 0$.

N2 $\forall x \in X, \forall \alpha \in \mathbb{R}, \quad \|\alpha x\| = |\alpha| \|x\|$.

N3 $\forall x, y \in X, \quad \|x + y\| \leq \|x\| + \|y\|$.

Exemplos: As normas vectoriais mais frequentemente utilizadas são as seguintes:

$\|x\|_1 = \sum_{i=1}^{n} |x_i|$ (norma 1)

$\|x\|_2 = \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2}$ (norma 2 ou Euclidiana)

$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$ (norma ∞ ou do máximo)

Normas matriciais

Seja $X = \mathbb{R}^{n \times n}$. Uma aplicação de X em $\mathbb{R}$ que satisfaça as propriedades *N1–N3* e ainda a seguinte propriedade adicional

$$\|AB\| \leq \|A\| \, \|B\|, \quad \forall A, B \in X$$

é chamada uma *norma matricial*.

Seja $\|\cdot\|_v$ uma determinada norma vectorial. A função $\|\cdot\|_M : \mathbb{R}^{n \times n} \longrightarrow \mathbb{R}$ definida por

$$\|A\|_M = \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|_v}{\|x\|_v}$$

define uma norma matricial, dita *induzida pela* ou *subordinada à* norma vectorial considerada. Pode provar-se que normas matriciais subordinadas às normas vectoriais $\|\cdot\|_1, \|\cdot\|_2$

e $\|\cdot\|_\infty$ [8] são dadas por

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|$$
$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$$
$$\|A\|_2 = \left(\rho(A^\mathsf{T}A)\right)^{1/2},$$

onde $\rho(A^\mathsf{T}A)$ designa o raio espectral da matriz $A^\mathsf{T}A$.

Todas estas normas satisfazem, naturalmente, a seguinte *condição de compatibilidade* com a correspondente norma vectorial

$$\|Ax\|_p \leq \|A\|_p \|x\|_p, \forall A \in \mathbb{R}^{n\times n}, \forall x \in \mathbb{R}^n; p = 1, 2, \infty.$$

5.1.4 Condicionamento de sistemas

Consideremos um sistema

$$Ax = b \tag{5.19}$$

e suponhamos que $b \neq 0$ e que A é não singular. Sejam $\delta A \in \mathbb{R}^{n\times n}$ e $\delta b \in \mathbb{R}^n$, respectivamente uma matriz e um vector "de perturbação" dos dados do problema, e seja $\tilde{x}$ a solução do problema perturbado, isto é, seja

$$(A + \delta A)\tilde{x} = b + \delta b. \tag{5.20}$$

Seja δ_x o erro na solução induzido pela perturbação dos dados, isto é, seja

$$\delta_x = \tilde{x} - x. \tag{5.21}$$

Temos, então, o seguinte resultado:

[8] Denotamos pelo mesmo símbolo a norma vectorial e a correspondente norma matricial, sendo claro pelo contexto de que norma se trata.

Teorema 5.3 *Se, para uma qualquer das normas consideradas, for* $\|A^{-1}\|\,\|\delta A\| < 1$, *então*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\text{cond}(A)}{1 - \text{cond}(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right), \tag{5.22}$$

onde

$$\text{cond}(A) := \|A\|\,\|A^{-1}\|. \tag{5.23}$$

A quantidade cond(A) dada por (5.23) diz-se *número de condição* da matriz A (relativamente à norma considerada).

Se $\|A^{-1}\|\,\|\delta A\| \ll 1$, então

$$1 - \text{cond}(A)\frac{\|\delta A\|}{\|A\|} = 1 - \|A^{-1}\|\,\|\delta A\| \approx 1,$$

pelo que poderemos, nesse caso, substituir a expressão (5.22)

$$\frac{\|\delta x\|}{\|x\|} \lesssim \text{cond}(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\,|}{\|b\|} \right).$$

Vemos, assim, que:

• um número de condição pequeno é garantia de que o sistema é bem condicionado, isto é, é pouco sensível às perturbações nos dados;

• um número de condição grande é indicador de que o problema poderá ser (e geralmente, é) mal condicionado.

5.2 Notas e referências

➠ Funções pré-definidas

Função	Objectivo
\	$A\backslash b$ dá a solução do sistema $Ax = b$, obtida usando decomposição LU, seguida de substituição directa e inversa
chol	Decomposição de Cholesky
cond	Número de condição
condtes	Estimativa do número de condição
diag	Extracção da diagonal de uma matriz/criação de matriz diagonal
det	Determinante
eig	Valores/vectores próprios
eye	Matriz identidade
hilb	Matriz de Hilbert
inv	Inversa de uma matriz
linsolve	Resolução de um sistema linear 7
lu	Decomposição LU
max	Máximo elemento
norm	Normas de vectores/matrizes
ones	Matriz de elementos iguais a um
tril	Parte triangular inferior de uma matriz
triu	Parte triangular superior de uma matriz
zeros	Matriz de elementos iguais a zero

➠ Referências

Os livros de Demmel [Dem97] e Golub e Van Loan [GV96], bem como o livro clássico de Wilkinson [Wil65], poderão ser interessantes para temas relacionados com este capítulo. Os manuais de utilização das *packages* **LAPACK** [ABB+95] e **LINPACK** [DBMS79] poderão também ser de utilidade.

5.3 Exercícios

Por uma questão de simplicidade, usaremos ao longo dos exercícios as seguintes famílias de matrizes, indexadas pela respectiva dimensão

$$T_n = (t_{ij}), \qquad t_{ij} = \begin{cases} 4, & \text{se } i = j, \\ 1, & \text{se } |i-j| = 1, \\ 0, & \text{restantes valores de } i \text{ e } j \end{cases}$$

$$K_n = (k_{ij}), \qquad k_{ij} = \begin{cases} 2, & \text{se } i = j, \\ -1, & \text{se } |i-j| = 1, \\ 0, & \text{restantes valores de } i \text{ e } j \end{cases}$$

$$H_n = (h_{ij}), \qquad h_{ij} = \frac{1}{i+j-1} \quad \text{(matrizes de Hilbert)}$$

$$S_n = (s_{ij}), \qquad s_{ij} = \begin{cases} 1, & \text{se } i = j, \\ 4, & \text{se } i = j+1, \\ -4, & \text{se } i = j-1, \\ 0, & \text{restantes valores de } i \text{ e } j \end{cases}$$

Exercício 5.1. Introduza as matrizes T_5, K_5, H_5 e S_5, no MATLAB.

Nota: Para a matriz de Hilbert, faça uso da função pré-definida **hilb**.

✔ Exercício 5.2.

a) Escreva uma função $b = \mathbf{subDirecta}(L, b)$ para implementar o Algoritmo 5.1. de substituição directa.

b) Use a função anterior para resolver o seguinte sistema de equações

$$\begin{cases} 5\,x_1 = -10 \\ x_1 + 3\,x_2 = 4 \\ 3\,x_1 + 4\,x_2 + 2\,x_3 = 2 \\ -x_1 + 3\,x_2 - 6x_3 - x_4 = 5 \end{cases}$$

Exercício 5.3.

a) Escreva uma função $b = \mathbf{subInversa}(U, b)$ para implementar o Algoritmo 5.2. de substituição inversa.

b) Use a função anterior para resolver o seguinte sistema de equações

$$\begin{cases} 3\,x_1 - 2\,x_2 + x_3 - x_4 = 8 \\ 4\,x_2 - x_3 + 2\,x_4 = -3 \\ 2\,x_3 + 3\,x_4 = 11 \\ 5\,x_4 = 15 \end{cases}$$

Exercício 5.4.

a) Descreva de que forma se pode obter a matriz inversa de uma dada matriz A, coluna a coluna, resolvendo n sistemas da forma $Ax = b$.

b) Considere a matriz $A = (a_{ij})$ de ordem 10, definida por

$$a_{ij} = \begin{cases} 1, & \text{se } i = j, \\ -1, & \text{se } i < j, \\ 0, & \text{se } i > j. \end{cases}$$

(i) Determine a inversa de A, fazendo uso da função $\mathbf{subInversa}$.

(ii) Use a função pré-definida $\mathbf{inv}$ para calcular a inversa de A e compare o resultado com o resultado da alínea anterior.

Exercício 5.5. Considere o problema da resolução do seguinte sistema de equações lineares, numa máquina com sistema de numeração $F(10, 4, -99, 99)$:

$$\begin{cases} 0.0003\,x_1 + 1.566\,x_2 = 1.569 \\ \\ 0.3454\,x_1 - 2.436\,x_2 = 1.018 \end{cases}$$

a) Resolva o sistema, usando o método de Gauss:

(i) sem escolha de *pivot*;

(ii) com escolha parcial de *pivot*.

b) Comente os resultados obtidos, comparando com a solução dada pelo MATLAB.

Exercício 5.6.

a) Escreva uma função $b = \mathbf{metGauss}(A, b)$ para implementar o Algoritmo 5.4. (método de Gauss com escolha parcial de pivot). Se, ao iniciar-se o passo k do método de eliminação, se verificar $a_{i,k}^{(k-1)} = 0; i = k, \ldots, n$, deverá ser dada uma mensagem indicando que a matriz do sistema é singular, sendo o programa interrompido.

b) Utilize a função anterior na resolução dos sistemas $Ax = b$, onde:

(i)
$$A = \begin{pmatrix} 4 & 1 & 1 & 0 & 1 \\ -1 & -3 & 1 & 1 & 0 \\ 2 & 1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 4 & 0 \\ 0 & 2 & -1 & 1 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix};$$

(ii)
$$A = T_5, \quad b = (1, 6, 12, 18, 19)^{\mathsf{T}};$$

(iii)
$$A = S_5, \quad b = (-4, -7, -6, -5, 16)^{\mathsf{T}};$$

(iv)
$$A = \begin{pmatrix} 1 & 2 & 4 \\ 0 & 3 & -1 \\ -2 & -1 & -9 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ -1 \\ 2 \end{pmatrix}$$

Exercício 5.7. Considere a matriz

$$A = \begin{pmatrix} 2 & 2 & -2 & 4 \\ 1 & 5 & 7 & -10 \\ 0 & 1 & 5 & 3 \\ -1 & 1 & 6 & -5 \end{pmatrix}.$$

a) Obtenha a decomposição LU de A.

b) Resolva o sistema $Ax = b$, com $b = (1, 1, 1, 1)^{\mathsf{T}}$, usando a decomposição obtida na alínea *a)* .

Exercício 5.8. Considere uma matriz tridiagonal

$$A = \begin{pmatrix} d_1 & f_1 & & & \\ e_1 & d_2 & f_2 & & \\ & \ddots & \ddots & \ddots & \\ & & e_{n-2} & d_{n-1} & f_{n-1} \\ & & & e_{n-1} & d_n \end{pmatrix},$$

a qual supomos admitir uma decomposição $A = LU$, onde

$$L = \begin{pmatrix} 1 & & & & \\ \ell_1 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ell_{n-2} & 1 & \\ & & & \ell_{n-1} & 1 \end{pmatrix}, \; U = \begin{pmatrix} u_1 & f_1 & & & \\ & u_2 & f_2 & & \\ & & \ddots & \ddots & \\ & & & u_{n-1} & f_{n-1} \\ & & & & u_n \end{pmatrix}.$$

a) Escreva um algoritmo para determinar a decomposição LU de A, ou seja, para determinar os vectores $\ell = (\ell_1, \ldots, \ell_{n-1})$ e $(u_1, \ldots, u_n)$, em função dos elementos da diagonal, subdiagonal e sobrediagonal de A.

b) Efectue uma contagem do número de operações (multiplicações/divisões e adições/subtracções) envolvidas no algoritmo anterior.

c) Escreva uma função $[\ell, u] = \mathbf{luTrid}(d, e, f)$ para implementar o algoritmo referido. A função deve

- aceitar como argumentos:
  - um vector $d = (d_1, \ldots, d_n)$ (diagonal de uma matriz tridiagonal);
  - um vector $e = (e_1, \ldots, e_{n-1})$ (subdiagonal da mesma matriz);
  - um vector $f = (f_1, \ldots, f_{n-1})$ (sobrediagonal da mesma matriz);
- dar, como resultados: dois vectores $\ell = (\ell_1, \ldots, \ell_{n-1})$ e $u = (u_1, \ldots, u_n)$ correspondentes à subdiagonal de L e diagonal de U da decomposição LU da matriz dada.

d) Obtenha a decomposição LU da matriz K_5, usando a função **luTrid** e use-a para calcular a segunda coluna da matriz K_5^{-1}.

Nota: Pode formar as matrizes L e U e usar as funções **subDirecta** e **subInversa**, ou modificar essas funções de modo a adaptá-las ao caso em questão.

Exercício 5.9. Nas alíneas seguintes, deve fazer uso da função pré-definida **lu**.

a) Obtenha a decomposição LU da matriz PA, onde

$$A = \begin{pmatrix} 17 & 24 & 1 & 8 \\ 23 & 5 & 7 & 14 \\ 4 & 6 & 13 & 29 \\ 10 & 12 & 19 & 21 \end{pmatrix}$$

e P é uma matriz de permutação adequada (correspondente às escolhas de *pivot* usadas no algoritmo) e resolva, então, o sistema $Ax = b$, onde $b = (16, 3, 44, 42)^\mathsf{T}$.

b) Resolva o sistema $S_5 x = b$, com $b = (-4, -7, -6, -5, 16)^\mathsf{T}$.

Exercício 5.10. Dada uma matriz não singular $A \in \mathbb{R}^{n\times n}$ e supondo conhecida uma decomposição $A = LU$, como resolveria, de forma eficiente, cada um dos problemas seguintes?

a) Resolver a equação matricial $AX = B$, com B uma dada matriz de $\mathbb{R}^{n\times m}$.

b) Determinar a solução do sistema $A^k x = b$, com k um determinado inteiro positivo.

c) Calcular $\alpha = c^\mathsf{T} A^{-1} b$, sendo $c \in \mathbb{R}^n$ dado.

Exercício 5.11.

a) Escreva uma função $R = \mathbf{decCholesky}(A)$ para obter a decomposição de Cholesky de uma matriz A simétrica definida positiva, com base no Algoritmo

5.5. A sua função deve começar por testar se a matriz A é simétrica; além disso, tendo em atenção a Nota da página 118, se, ao implementar o Algoritmo 5.5, for violada a condição (5.12) para algum valor de i, deverá ser dada uma mensagem indicando que A não é definida positiva, sendo o programa interrompido.

b) Diga como deveria resolver um sistema $Ax = b$, sabida a decomposição de Cholesky de A.

c) O *critério de Sylvester* afirma que uma matriz simétrica $A \in \mathbb{R}^{n\times n}$ é definida positiva se e só se os menores principais contidos nas primeiras k linhas e k colunas de $A; k = 1, \ldots, n$ são positivos. Considere a matriz

$$A_\epsilon = \begin{pmatrix} \epsilon & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 3 \end{pmatrix}.$$

(i) Determine quais os valores do parâmetro real ϵ para os quais A_ϵ é definida positiva.

(ii) Considere $\epsilon = 2$ e determine a decomposição de Cholesky de A_ϵ.

(iii) Usando a decomposição anterior, calcule a solução do sistema $A_2 x = b$, onde $b = (1, 1, 1)^\mathsf{T}$.

(iv) Escolha um valor do parâmetro para o qual A_ϵ não seja definida positiva e use-a para testar se o seu programa detecta devidamente este caso.

✔ Exercício 5.12.

a) Mostre que o passo k do método de Jacobi para a resolução de um sistema $Ax = b$ pode ser descrito como

$$Dx^{(k+1)} = -(M + N)x^{(k)} + b,$$

onde D, M e N são as matrizes de (5.15), e diga como poderá obter facilmente a matriz $M + N$ no MATLAB.

b) Escreva uma função $x = \mathbf{metJacobi}(A, b, tol, maxit)$ para implementar o método de Jacobi. Mais precisamente, essa função deve

- aceitar como argumentos:

  - uma matriz $A \in \mathbb{R}^{n\times n}$ (matriz do sistema);
  - um vector $b = (b_1, \ldots, b_n)^\mathsf{T}$ (lado direito do sistema);
  - um número real positivo tol (tolerância para o erro);
  - um inteiro $maxit$ (número máximo de iterações a efectuar);
- efectuar diversas iterações do método de Jacobi, tomando como aproximação inicial o vector nulo; o processo iterativo deverá parar quando o vector diferença de duas iterações sucessivas tiver norma infinita inferior à tolerância tol ou quando for atingido o número máximo de iterações fixado pelo utilizador ($maxit$);
- dar como resultado: x, o vector obtido na última iteração.

c) Obtenha ajuda sobre a função **nargin** e utilize-a na função **metJacobi** para que ela possa ser invocada apenas com três ou dois parâmetros de entrada, considerando, por defeito, $maxit = 30$ e $tol = 10^{-6}$.

d) Teste o seu programa na resolução dos seguintes sistemas, usando diversos valores de tol e de $maxit$.

$$(i)\begin{cases} 10x_1 - x_2 + 2x_3 = 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 = 25 \\ 2x_1 - x_2 + 10x_3 - x_4 = -11 \\ 3x_2 - x_3 + 8x_4 = 15 \end{cases} \qquad (ii)\ A = T_{10},\quad b = (3, 3, \ldots, 3)^\mathsf{T}$$

$$(iii)\begin{cases} 2x + 3y = 1 \\ 7x - 2y = 1 \end{cases} \qquad (iv)\ A = T_{10},\quad b = (1, 2, \ldots 1, 2)^\mathsf{T}.$$

Exercício 5.13.

a) Mostre que o passo k do método de Gauss-Seidel para a resolução de um sistema $Ax = b$ pode ser descrito como

$$(D + M)x^{(k+1)} = -Nx^{(k)} + b, \tag{5.24}$$

onde D, M e N são as matrizes usuais. Tendo em conta que $D + M$ é uma matriz triangular inferior, sugira uma forma eficiente de obter o vector $x^{(k+1)}$ a partir da equação (5.24).

b) Escreva uma função, $x = \mathbf{metGSeidel}(A, b, tol, maxit)$, para implementar o método de Gauss-Seidel, com especificações idênticas às referidas para o método de Jacobi.

c) Teste o seu programa na resolução dos mesmos sistemas da questão anterior.

Exercício 5.14.

a) Escreva uma função $raio = \mathbf{raioJacobi}(A)$ para, dada uma matriz A, calcular o raio espectral da respectiva matriz de iteração de Jacobi.

b) Modifique a função anterior para construir uma função idêntica, $raio = \mathbf{raioGSeidel}(A)$, mas relativa à matriz de iteração do método de Gauss-Seidel.

Exercício 5.15.

a) Seja

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}.$$

Mostre que $\rho(B_J) < 1$ e $\rho(B_{GS}) > 1$. Que pode concluir quanto à convergência dos métodos de Jacobi e de Gauss-Seidel aplicados à resolução de um sistema que tenha A como matriz?

b) Seja $b = (1, 3, 5)^{\mathsf{T}}$. Efectue 5 iterações do método de Jacobi para resolver o sistema $Ax = b$, tomando como aproximação inicial o vector $(0, 0, 0)^{\mathsf{T}}$. Repita com o método de Gauss-Seidel e verifique se os resultados confirmam a conclusão da alínea anterior.

c) Seja

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}.$$

Mostre que $\rho(B_J) > 1$ e $\rho(B_{GS}) < 1$. Neste caso, como se comportariam os métodos de Jacobi e de Gauss-Seidel quanto à convergência? Teste a sua

conclusão, resolvendo o sistema com $Ax = b$, com b escolhido por forma que o sistema tenha $x = (1, 1, 1)^{\mathsf{T}}$ como vector solução.

Exercício 5.16. Faça uso das funções **raioJacobi** e **raioGSeidel** para justificar o comportamento dos métodos de Jacobi e Gauss-Seidel na resolução dos sistemas considerados do Exercício 5.12.

Exercício 5.17. Discuta, em função dos valores do parâmetro $\alpha \in \mathbb{R}$, o comportamento dos métodos de Jacobi e de Gauss-Seidel aplicados a um sistema de equações cuja matriz seja

$$A_\alpha = \begin{pmatrix} \alpha & 0 & 1 \\ 0 & \alpha & 0 \\ 1 & 0 & \alpha \end{pmatrix}, \quad \alpha \neq 0.$$

Exercício 5.18. Considere as matrizes

$$A_\epsilon = \begin{pmatrix} 2 & -1 \\ -1 & 0.5 + \epsilon \end{pmatrix}.$$

a) Calcule, recorrendo a funções pré-definidas, o número de condição de A_ϵ para $\epsilon = 1, 0.1, 0.01$ e 0.001 (relativamente à norma $\|\cdot\|_\infty$).

b) Resolva os sistemas

$$\begin{cases} 2x - y = 1 \\ -x + 0.499y = 1 \end{cases}$$

e

$$\begin{cases} 2x - y = 1 \\ -x + 0.501y = 1 \end{cases}$$

e comente os resultados obtidos.

Exercício 5.19. Considere o problema da resolução de um sistema $Ax = b$, onde

$$A = \begin{pmatrix} 0.5 & 0.4 \\ 0.3 & 0.25 \end{pmatrix}$$

e onde se conhece apenas uma aproximação

$$\tilde{b} = \begin{pmatrix} 0.200 & 1.000 \end{pmatrix}^{\mathsf{T}}$$

para o vector b dos termos independentes. Sabendo que os elementos de $\tilde{b}$ são aproximações para os correspondentes elementos de b com 3 casas decimais correctas, indique um majorante para o erro relativo no vector solução

$$\frac{\|x - \tilde{x}\|_\infty}{\|x\|_\infty}.$$

Exercício 5.20. Considere a matriz

$$A = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{pmatrix}.$$

a) Verifique que A é uma matriz singular. Descreva o conjunto de soluções do sistema $Ax = b$, com $b = (0.1, 0.3, 0.5)^{\mathsf{T}}$.

b) Se usássemos o método de eliminação Gaussiana com escolha parcial de *pivot* (com aritmética exacta) para resolver o sistema, em que passo falharia o processo?

c) Tente resolver esse sistema usando a função $\mathbf{metGauss}$ do Exercício 5.6. Como justifica o resultado?

d) Tente novamente resolver o sistema, usando a forma $x = A\backslash b$ usual do MATLAB. Comente os resultados.

Exercício 5.21. Considere o sistema linear $Ax = b$, onde

$$A = \begin{pmatrix} 1.002 & 1 \\ 1 & 0.998 \end{pmatrix} \quad \text{e} \quad b = \begin{pmatrix} 0.002 \\ 0.002 \end{pmatrix},$$

o qual admite como solução exacta $x = (1, -1)^{\mathsf{T}}$. Seja $\tilde{x} = (0.29360067817338, -0.29218646673249)^{\mathsf{T}}$ uma "aproximação" para a solução do sistema. Calcule o vector residual $r = A\tilde{x} - b$, compare $\|r\|_\infty$ com $\|x - \tilde{x}\|_\infty$ e comente.

5.4 Trabalhos

Trabalho 5.1.

a) Modifique a função $\mathbf{metGauss}$ do Exercício 5.6. de forma a que ela permita resolver vários sistemas de equações lineares em simultâneo e forneça também o determinante da matriz do sistema, caso o utilizador o deseje. Mais precisamente, a sua função deve: passar a aceitar como argumentos, duas matrizes $A \in \mathbb{R}^{n\times n}$ e $B \in \mathbb{R}^{n\times m}$; resolver a equação matricial $AX = B$, armazenando a matriz solução em B; calcular o determinante de A, caso a função seja invocada com dois argumentos de saída, $[B, deter] = \mathbf{metGaussMod}(A, B)$.

b) Considere a matriz

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 5 & 3 \\ 4 & 6 & 8 & 0 \\ 3 & 3 & 9 & 8 \end{pmatrix}.$$

Calcule o determinante de A e a matriz A^{-1}, usando a função da alínea anterior.

Trabalho 5.2. Escreva uma função $x = \mathbf{resolveSistema}(A, b)$ destinada a resolver uma sistema de equações $Ax = b$, escolhendo um método adequado. Mais precisamente, a sua função deve

- começar por testar se A é triangular inferior (superior), resolvendo, nesse caso, o sistema por substituição directa (inversa);
- testar se A é tridiagonal, efectuando nesse caso a decomposição LU especialmente desenhada para esse caso, e resolvendo o sistema por substituição directa e inversa;
- verificar se A é simétrica, tentando, nesse caso, obter a sua decomposição de Cholesky; se tal decomposição for obtida, usá-la para resolver o sistema por substituição directa e inversa;
- em qualquer outro caso, usar o método de Gauss com escolha parcial de pivot.

Em cada caso, deverá ser dada uma mensagem indicando qual foi o método utilizado. Teste a sua função para resolver um sistema de cada um dos casos considerados.

Trabalho 5.3. **Método SR(ω)**

O *método de relaxação sucessiva* com parâmetro ω (método SR(ω)), para a resolução de um sistema $Ax = b$, é um método iterativo definido por:

Para $k = 0, 1, 2, \ldots,$

$$\left.\begin{array}{l} x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}); i = 1, \ldots, n, \\ x_i^{(0)};\ i = 1, \ldots, n, \quad \omega \in \mathbb{R}, \quad \text{dados.} \end{array}\right\} \tag{5.25}$$

a) Mostre que o passo k do método SR(ω) pode ser descrito matricialmente como

$$(D + \omega M)x^{(k+1)} = [(1 - \omega)D - \omega N]x^{(k)} + \omega b,$$

onde D, M e N são as matrizes usuais utilizadas na definição matricial dos métodos de Jacobi e Gauss-Seidel.

b) Escreva uma função $x = \mathbf{metSR}(A, b, \omega, tol, maxit)$ para implementar o método de relaxação sucessiva, com especificações idênticas às construídas para os métodos de Jacobi e Gauss-Seidel.

Nota: Tenha em atenção que, neste caso, a função deve aceitar um argumento adicional, valor do parâmetro de relaxação ω.

c) Pode provar-se que é condição necessária de convergência do método SR(ω) que $0 < \omega < 2$; além disso, para $\omega = 1$, o método SR(ω) é simplesmente o método de Gauss-Seidel.

Teste a sua função na resolução de diversos sistemas de equações por si escolhidos, fazendo variar o valor do parâmetro de relaxação e dando exemplos em que, para certos valores do parâmetro, o método convirja mais rapidamente que o método de Gauss-Seidel; ilustre também a afirmação quanto à condição imposta ao parâmetro para que o método possa convergir.

Trace também os gráficos de $\mathbf{r}(n)$ e de cond(H_n), para $n = 1, \ldots, 27$. (Não use o comando $\mathbf{semilogy}$, nestes casos).

Faça um pequeno comentário crítico aos resultados obtidos.

Nota: Para resolver este exercício será conveniente escrever uma *script*.

5.5 Resoluções

Resolução do Exercício 5.2. *a)*

```
function b=subDirecta(L,b)
%SUBDIRECTA resolve sistema triangular inferior (substituicao directa).
%
% B=SUBDIRECTA(L,B)   Determina a solucao de um sistema LX=B, onde L
%   'e uma matriz triangular inferior, por substituicao directa.
%
% PARAMETROS DE ENTRADA:
%    L: matriz quadrada, triangular inferior, sem elementos
%       nulos na diagonal (matriz do sistema)
%    B: vector coluna (termos independentes)
%
% PARAMETRO DE SAIDA:
%    B: solucao do sistema LX=B

%---------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%---------------------------------------------------------------------------
% Verifica se a matriz L'e quadrada, triangular inferior sem elementos
% nulos na diagonal e se B tem a dimensao adequada.
[n,m]=size(L); [nb,mb]=size(b);
if  ( n~=m )
    error('Matriz nao e quadrada')
elseif ( nb~=n | mb~=1 )
    error('Vector b nao tem a dimensao adequada')
elseif  ~( all(diag(L)) )
    error('L tem um elemento nulo na diagonal')
elseif ~( isequal(triu(L,1),zeros(n,n)) )
    error('L nao e triangular inferior')
end
%---------------------------------------------------------------------------
%                     SUBSTITUICAO DIRECTA
%---------------------------------------------------------------------------
for i=1:n
    j=1:i-1;
    b(i)=(b(i)-L(i,j)*b(j))/L(i,i);
end
```

Listagem 5.1. Função **subDirecta**

Resolução do Exercício 5.12. *b)*

```
function x=metJacobi(A,b,tol,maxit)
%METJACOBI Resolve um sistema linear pelo metodo iterativo de Jacobi
%
% X=JACOBI(A,B,TOL,MAXIT)   Determina a solucao de um sistema AX=B,
%   usando o metodo de Jacobi; o processo itrativo para quando a norma
%   infinita de duas iteracoes sucessivas for inferior a TOL ou quando
%   for atingido um numero maximo de iteracoes MAXIT
%
% PARAMETROS DE ENTRADA:
%    A: matriz quadrada (matriz do sistema)
%    B: vector coluna (lado direito do sistema)
%    TOL: real positivo (tolerancia para o erro)
%    MAXIT: inteiro positivo (numero maximo de iteracoes permitidas)
%
% PARAMETRO DE SAIDA:
%    X: solucao aproximada do sistema

%--------------------------------------------------------------------------
%             PARAMETROS MAXIT e TOL POR DEFEITO
%--------------------------------------------------------------------------
if nargin==3
   maxit=30;
elseif nargin==2
    maxit=30; tol=1e-6;
end
%--------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%--------------------------------------------------------------------------
% Verificar se a matriz A e quadrada, sem elementos nulos na diagonal e
% se B tem a dimensao adequada
[n,m]=size(A); [nb,mb]=size(b);
if  ( n~=m )
    error('Matriz nao e quadrada)')
elseif ( (nb~=n) |(mb~=1) )
    error('Vector b nao tem a dimensao adequada')
elseif( all(diag(A))==0 )
    error('A tem um elemento nulo na diagonal')
end
```

(continua)

```
%--------------------------------------------------------------------------
%        FORMACAO DE MATRIZ -(M+N) A USAR NO PROCESSO ITERATIVO
%--------------------------------------------------------------------------
d=diag(A);
A=diag(d)-A; %  Sobrepomos a matriz -(M+N) ‘a matriz A
%--------------------------------------------------------------------------
%                          PROCESSO ITERATIVO
%--------------------------------------------------------------------------
niter=0;        % Inicializacao do contador do numero de iteracoes
erro=tol+1;     % Erro inicial superior a tolerancia
x0=zeros(n,1);  % Inicializacao do processo iterativo com o vector nulo
while erro>=tol & niter < maxit
    x=(A*x0+b)./d; % Calculo da nova iteracao
    niter=niter+1; % Incremento do numero de iteracoes
    erro=norm(x-x0,inf); % Calculo da norma da dif. das duas iteracoes
    %
    % Impressao de  resultados intermedios
    disp(sprintf('\n \n k= %3.0f',niter))
    disp(sprintf('\n \n %12.8f',x))
    disp(sprintf('\n \n erro= %4.2e',erro))
    %
    % Armazenamento em x0 da ultima iteracao calculada
    x0=x;
end
%--------------------------------------------------------------------------
%                   MENSAGEM (NAO CONVERGENCIA)
%--------------------------------------------------------------------------
if niter==maxit
        disp(' ')
        disp('Foi atingido o numero maximo de iteracoes !! ')
end
```

Listagem 5.2. Função **metJacobi**

6. Equações Não Lineares

Método do ponto fixo
Método da bissecção
Método de Newton
Método da secante

6.1 Notações, definições e resultados básicos

Neste capítulo consideramos o problema da determinação de raízes de uma *equação não linear*, isto é, de uma equação da forma

$$f(x) = 0 \tag{6.1}$$

onde f é uma função real de variável real, não linear.

Os métodos usados para resolver (6.1) são *métodos iterativos*: são dadas $m+1$ aproximações iniciais $x_0, \ldots, x_m$ para uma raiz r da equação (6.1) e determina-se, então, uma nova aproximação x_{m+1}, repetindo-se sucessivamente este processo. Mais precisamente, é gerada uma sequência $\{x_k\}$ de aproximações para r através do uso de fórmulas do tipo

$$x_{k+1} = g(k, x_k, \ldots, x_{k-m}); k = m, m+1, \ldots \tag{6.2}$$

Se a função g em (6.2) não depender de k, isto é, se a forma da função iterativa se mantiver de iteração para iteração, o método diz-se *estacionário*.

Seja $e_k := r - x_k$ (erro na aproximação x_k para r). O que se pretende, naturalmente, é ter métodos convergentes, isto é, métodos que satisfaçam

$$\lim_{k\to\infty} x_k = r$$

ou, equivalentemente,

$$\lim_{k\to\infty} e_k = 0.$$

Ordem de convergência

Um conceito importante na discussão de métodos iterativos é o de *ordem de convergência*.

Seja $\{x_k\}$ uma sequência de aproximações obtidas por um determinado método iterativo e suponhamos que $\lim_{k\to\infty} x_k = r$. Seja $e_k := r - x_k$. Se existir um número $p \geq 1$ e uma constante $C > 0$ tais que

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C \tag{6.3}$$

dizemos que o método é de *ordem* p em r. Se $p = 1$, a convergência diz-se *linear* e, se $p > 1$, a convergência é dita *superlinear*, dizendo-se *quadrática* se $p = 2$, *cúbica* se $p = 3$, etc. A constante C é chamada *constante de convergência assimptótica*. É usual escrever a expressão (6.3) de forma assimptótica como

$$|e_{k+1}| \sim C|e_k|^p$$

que nos indica como o erro se comporta quando k é suficientemente grande.
Notas:

- Quando $p = 1$, C deve ser menor ou igual a 1 para que o método convirja, mas para $p > 1$, não é necessário que $C \leq 1$ para haver convergência.
- Nalguns casos é possível estabelecer uma relação do tipo

  $$|e_k| \leq CM^k,\ M < 1, \tag{6.4}$$

  não sendo, no entanto, possível mostrar directamente que a condição (6.3) se verifica (com $p = 1$). Neste caso, dizemos ainda que o método converge linearmente.

Quanto maior for a ordem de convergência de um método iterativo e menor a constante de convergência, maior será, em princípio, a sua rapidez de convergência.

6.1.1 Método do ponto fixo ou das iterações sucessivas

Seja $g : D \subseteq \mathbb{R} \longrightarrow \mathbb{R}$ uma função real de variável real. Dizemos que g é *contractiva* em D, se existe uma constante $0 \leq L < 1$, tal que

$$|g(x) - g(y)| \leq L|x - y|,\ \forall x, y \in D.$$

A constante L é chamada *constante de Lipschitz*.

Seja g uma aplicação de D em si mesmo, isto é, $g : D \longrightarrow D$. Diz-se que $\alpha \in D$ é *ponto fixo* de g se $g(\alpha) = \alpha$. Comecemos por recordar o importante Teorema do Ponto Fixo (num intervalo $I = [a, b]$ de $\mathbb{R}$).

Teorema 6.1 (do ponto fixo de Banach) *Seja $I = [a, b] \subseteq \mathbb{R}$ e seja g uma aplicação de I em si mesmo, contractiva, com constante de Lipschitz L. Então:*

(i) a aplicação g tem um e um só ponto fixo α em I;

(ii) para qualquer valor inicial $x_0 \in I$, a sequência de iterações $\{x_k\}$ definida por

$$x_{k+1} = g(x_k);\ k = 0, 1, 2, \ldots \tag{6.5}$$

converge para α, ponto fixo de g em I;

(iii) o erro $e_k := \alpha - x_k$ satisfaz

$$|\alpha - x_k| \leq \frac{L^{k-m}}{1-L}|x_{m+1} - x_m|;\ 0 \leq m \leq k-1. \tag{6.6}$$

Em particular, tem-se:

- $m = 0$ – *Estimativa "a priori" para o erro na iteração k:*

$$|\alpha - x_k| \leq \frac{L^k}{1-L}|x_1 - x_0| \tag{6.7}$$

- $m = k - 1$ – *Estimativa "a posteriori" para o erro na iteração k:*

$$|\alpha - x_k| \leq \frac{L}{1-L}|x_k - x_{k-1}| \tag{6.8}$$

Voltemos, então, ao problema da determinação de uma raiz da equação não linear (6.1). Suponhamos que, por operações elementares, reescrevemos essa equação, de uma forma equivalente, como

$$g(x) = x \tag{6.9}$$

transformando, assim, o problema da determinação de uma raiz de (6.1) no da determinação de um ponto fixo de g. Se encontrarmos um intervalo I tal que $g(I) \subseteq I$ e g seja contractiva

em I então, tendo em conta o Teorema do Ponto Fixo de Banach, poder-se-á procurar α (ponto fixo de g em I) usando a fórmula iterativa

$$x_{k+1} = g(x_k);\ k = 0, 1, \ldots;\ x_0 \in I. \tag{6.10}$$

A este método chamamos *método do ponto fixo* ou *das iterações sucessivas*.

A verificação de que g é contractiva no intervalo I poderá, por vezes, tornar-se mais simples recorrendo ao seguinte resultado, consequência imediata do Teorema do Valor Médio de Lagrange (veja, e.g., Apêndice B).

Teorema 6.2 *Seja $g \in C^1[a, b]$ tal que $|g'(x)| \leq L < 1$, para todo $x \in (a, b)$. Então, g é contractiva em $I = [a, b]$ com constante de Lipschitz L.*

Tem-se, também o seguinte teorema.

Teorema 6.3 *Seja α um ponto fixo de uma função g e suponhamos que g é continuamente diferenciável num certo intervalo centrado em α e que, além disso, $|g'(\alpha)| < 1$. Então, existe um intervalo $I = [\alpha - \delta, \alpha + \delta]$ para o qual são válidas as hipóteses do teorema do ponto fixo de Banach.*

O resultado anterior mostra, assim, que se α for um ponto fixo de g e $|g'(\alpha)| < 1$, [1] então poderemos aplicar o método das aproximações sucessivas para aproximar α, desde que x_0 seja escolhido "suficientemente"próximo de α. Por estas razões se diz que o método do ponto fixo é um método de convergência *local*.

Pode também mostrar-se que, se $|g'(\alpha)| > 1$, então o método das iterações sucessivas não poderá convergir.

Ordem de convergência do método do ponto fixo

Teorema 6.4 *Seja α um ponto fixo de uma função g e suponhamos que g é continuamente diferenciável num certo intervalo centrado em α e que, além disso,*

$$0 < |g'(\alpha)| < 1.$$

[1] E sendo g' é contínua num intervalo centrado em α.

Seja I o intervalo (que sabemos existir) para o qual se verificam as condições do teorema do ponto fixo, e suponhamos que $x_0 \in I$ e que $x_{k+1} = g(x_k); k = 0, 1, \ldots$. Então, designando por e_k o erro na iteração k, isto é, $e_k = \alpha - x_k$, tem-se

$$\lim_{x\to\infty} \frac{|e_{k+1}|}{|e_k|} = |g'(\alpha)|. \tag{6.11}$$

Por outras palavras, nestas condições o método converge linearmente, com constante de convergência assimptótica $C = |g'(\alpha)|$.

O resultado anterior mostra que a convergência do método do ponto fixo será tanto mais rápida quanto menor for o valor de $C = |g'(\alpha)|$. Pode acontecer que $g'(\alpha) = 0$. Nesse caso, será de esperar que a convergência seja "melhor" que linear. De facto, tem-se o seguinte resultado mais geral:

Teorema 6.5 (convergência do método do ponto fixo) *Seja α um ponto fixo de uma função g e suponhamos que g é p vezes continuamente diferenciável num certo intervalo centrado em α e que, além disso,*

$$g'(\alpha) = g''(\alpha) = \ldots g^{(p-1)}(\alpha) = 0$$

e que $g^{(p)}(\alpha) \neq 0$. Seja $\{x_k\}$ a sequência de iterações obtida por aplicação do método do ponto fixo, com x_0 escolhido suficientemente próximo de α. Então, tem-se

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = \frac{1}{p!}|g^{(p)}(\alpha)|. \tag{6.12}$$

Por outras palavras, nestas condições, a ordem de convergência do método é p e a constante de convergência assimptótica é $C = \frac{1}{p!}|g^{(p)}(\alpha)|$.

6.1.2 Método da bissecção

Como vimos, para aplicar o método do ponto fixo para a determinação de uma raiz de uma equação não linear é importante dispor de uma aproximação inicial "suficientemente" próxima da raiz. O chamado *método da bissecção* é um processo muito simples de procurar essa aproximação para a raiz e baseia-se exclusivamente no uso sucessivo do Teorema do Valor Intermédio.

Suponhamos que encontramos valores a e b, com $a < b$, e tais que $f(a)f(b) < 0$ (por outras palavras, suponhamos que em a e b a função f tem sinais contrários) e que, além disso, f é contínua no intervalo $I = [a, b]$. Então, por aplicação do Teorema do Valor Intermédio, podemos concluir que f admite uma raiz no interior do intervalo $I = [a, b]$. Seja μ o ponto médio desse intervalo, isto é, $\mu = \frac{a+b}{2}$, e calculemos o valor de f nesse ponto. Se $f(\mu) = 0$, então μ é um zero de f. Caso contrário, o sinal de $f(\mu)$ determina em qual dos subintervalos (a, μ) ou (μ, b) se encontra uma raiz. Mais precisamente:

- se $f(a)f(\mu) < 0$, há uma raiz no intervalo (a, μ);
- se $f(a)f(\mu) > 0$, há uma raiz no intervalo (μ, b).

A amplitude do novo intervalo contendo uma raiz é, assim, metade da do intervalo inicial; o processo, pode, então, repetir-se até se encontrar um intervalo de amplitude suficientemente pequena onde se localize uma raiz, tomando-se como aproximação para essa raiz o ponto médio do intervalo.

Note-se que o processo anterior determina uma sequência encaixada de intervalos $[a, b] = I_0 \supset I_1 \supset I_2 \ldots$, existindo, além disso, $r \in I_k$; $k = 0, 1 \ldots$, tal que $f(r) = 0$. Sendo x_k o ponto médio do intervalo $I_{k-1}; k = 1, 2, \ldots$, tem-se

$$|e_k| = |r - x_k| \leq \frac{b-a}{2^k}. \tag{6.13}$$

Concluímos assim que $e_k \to 0$, ou seja, que o método converge. Tendo em atenção a observação que fizemos acerca da convergência linear (fórmula (6.4)), a fórmula (6.13) indica-nos que a convergência deste método é apenas linear com uma "taxa média de decrescimento"do erro de $\frac{1}{2}$. Isto significa que este método converge lentamente, o que constitui uma das suas principais desvantagens.

A sua principal vantagem reside no facto de ele ter convergência garantida, seja qual for a amplitude do intervalo inicial. Isto significa que (contrariamente ao método do ponto fixo) não é necessário começar a iterar "suficientemente"próximo da raiz para haver garantia de convergência. Por essa razão, dizemos que o método da bissecção é um método de convergência *global*.

A fórmula (6.13) permite-nos também determinar "a priori"o número de iterações necessárias para garantir que x_k aproxime r com uma determinada precisão. Mais precisamente, supondo que pretendemos ter $|r - x_k| \leq \delta$, bastará tomar k como (um inteiro)

satisfazendo

$$k \geq \frac{\log(b-a) - \log(\delta)}{\log 2}. \tag{6.14}$$

6.1.3 Método de Newton

Se a função f cuja raiz procuramos for continuamente diferenciável e se a derivada de f puder ser calculada sem grande esforço computacional, poderemos procurar a raiz usando o chamado *método de Newton*, definido por

$$\left.\begin{array}{l} x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}; k = 0, 1, 2, \ldots; \\ \\ x_0 \text{ dado} \end{array}\right\} \tag{6.15}$$

Notas:

- Facilmente se verifica que x_{k+1} dado por (6.15) é a abcissa do ponto de intersecção com o eixo dos xx da recta tangente ao gráfico de f no ponto $(x_k, f(x_k))$;
- o esquema iterativo (6.15) pressupõe que $f'(x_k) \neq 0$, ou seja, que a tangente ao gráfico de f no ponto $(x_k, f(x_k))$ não é paralela ao eixo das abcissas.

O resultado seguinte refere condições suficientes de convergência do método de Newton e dá indicação sobre a sua ordem de convergência.

Teorema 6.6 (Convergência do método de Newton) *Seja r uma raiz da equação $f(x) = 0$ e suponhamos que f é duas vezes continuamente diferenciável numa vizinhança de r e que $f'(r) \neq 0$ (ou seja , que r é um zero simples de f). Então, existe um intervalo $I = [r - \delta, r + \delta]$ tal que, para qualquer aproximação inicial $x_0 \in I$, o método de Newton converge para r, com convergência quadrática e constante de convergência assimptótica dada por*

$$C = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right|. \tag{6.16}$$

Notas:

- A ordem de convergência será 2 se $f''(r) \neq 0$, sendo superior a dois se $f''(r) = 0$ e f for suficientemente suave;
- se r é uma raiz múltipla, então a convergência do método de Newton para essa raiz será apenas linear.

O resultado seguinte estabelece condições suficientes de convergência do método de Newton, para qualquer aproximação inicial num certo intervalo $I = [a, b]$, que são, por vezes, fáceis de verificar na prática.

Teorema 6.7 *Seja f uma função de $C^2[a, b]$ que verifique as seguintes condições:*

1. $f(a)f(b) < 0$
2. $f'(x) \neq 0, \forall x \in [a, b]$
3. $f''(x) \geq 0, \forall x \in [a, b]$ *ou* $f''(x) \leq 0, \ \forall x \in [a, b]$
4. $$\frac{|f(a)|}{|f'(a)|} < b - a \quad \text{e} \quad \frac{|f(b)|}{|f'(b)|} < b - a.$$

Então, o método de Newton converge (com convergência quadrática) para a única solução r, em $[a, b]$, da equação $f(x) = 0$, para qualquer aproximação inicial $x_0 \in [a, b]$.[2]

6.1.4 Método da secante

Um método alternativo ao método de Newton, utilizado, por exemplo, se o cálculo da derivada da função f envolve grande esforço computacional, é o chamado *método da secante*.

Neste caso, partindo de duas aproximações x_{k-1} e x_k para um um zero r de f, vamos considerar como nova aproximação a abcissa do ponto de encontro da recta que une os pontos $(x_{k-1}, f(x_{k-1}))$ e $(x_k, f(x_k))$ com o eixo das abcissas. Mais precisamente, o método da secante gera uma sequência de iterações definidas por

$$\left.\begin{array}{l} x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}; k = 1, 2 \ldots, \\ x_0, x_1 \ \text{ dados} \end{array}\right\} \tag{6.17}$$

No que diz respeito à convergência, tem-se o seguinte resultado.

[2] Se as condições 2.–3. se verificarem para qualquer $x \in \mathbb{R}$ e f tiver um zero r em $\mathbb{R}$, então o método de Newton converge para r para qualquer aproximação inicial $x_0 \in \mathbb{R}$.

Teorema 6.8 (convergência do método da secante) *Seja r uma raiz da equação $f(x) = 0$ e suponhamos que f é duas vezes continuamente diferenciável numa vizinhança de r e que $f'(r) \neq 0$ (ou seja , que r é um zero simples de f). Então, existe um intervalo $I = [r - \delta, r + \delta]$ tal que, para quaisquer aproximações iniciais $x_0, x_1 \in I$, o método da secante converge para r, com ordem de convergência*

$$p = \frac{1 + \sqrt{5}}{2} \tag{6.18}$$

e constante de convergência assimptótica dada por

$$C = \left| \frac{f''(r)}{2f'(r)} \right|^{1/p}. \tag{6.19}$$

6.1.5 Critérios de paragem

Ao implementar qualquer dos métodos iterativos referidos para a a determinação de uma raiz da equação $f(x) = 0$, há necessidade de definir critérios de paragem do processo iterativo. Os critérios de paragem usuais são do tipo:

1. $|x_{k+1} - x_k| < \delta_1$
2. $|x_{k+1} - x_k| < \delta_2 |x_{k+1}|$
3. $|f(x_k)| < \delta_3$

Tendo em atenção que o processo pode divergir ou convergir de forma muito lenta, deverá também ser sempre incluído um critério de paragem do tipo

$$k = maxit$$

onde $maxit$ é o número máximo de iterações que o utilizador pretende que sejam efectuadas.

6.2 Notas e referências

➠ **Funções pré-definidas**

Função	Objectivo
fminbnd	Minimizante de uma função (num certo intervalo)
fzero	Zero de uma função
roots	Raízes de um polinómio

➠ **Referências**

Os livros de Atkinson [Atk89], Householder [Hou70] e Murray e Wright [MW91] são especialmente indicados para este capítulo.

6.3 Exercícios

Exercício 6.1. Seja $I = [a, b] \subseteq \mathbb{R}$ e seja $g \in C^1[a, b]$ uma aplicação que satisfaz as condições de convergência do método do ponto fixo no intervalo I, isto é, suponhamos que $g(I) \subseteq I$ e $|g'(x)| < 1, \forall x \in (a, b)$. Mostre que, se $0 < g'(x) < 1,\ \forall x \in (a, b)$, então a convergência da sequência gerada pelo método do ponto fixo para α (ponto fixo de g em I) é monótona. Mais precisamente, mostre que:

- Se $x_0 > \alpha$, então $x_0 > x_1 > x_2 > x_3 > \ldots$
- Se $x_0 < \alpha$, então $x_0 < x_1 < x_2 < x_3 < \ldots$

O que acontece quando $-1 < g'(x) < 0\,,\ \forall x \in (a, b)$? Interprete geometricamente estes resultados.

Exercício 6.2. Para cada uma das equações seguintes, determine uma função iterativa g e um intervalo I, de tal modo que se verifiquem as condições de aplicação do Teorema do Ponto Fixo. Encontre uma aproximação para a menor raiz positiva dessas equações, usando o referido teorema.

a) $x^3 - x - 1 = 0$

b) $x - 2\text{sen}\, x = 0$

Nota: Comece por localizar graficamente as raízes das equações dadas.

Exercício 6.3. Seja a um inteiro positivo e considere a sucessão definida por

$$x_0 = \frac{1}{a},\ x_1 = \frac{1}{a + \frac{1}{a}},\ x_2 = \frac{1}{a + \frac{1}{a + \frac{1}{a}}}, \cdots,$$

ou seja, $\{x_k\}$ é a sucessão definida por $x_0 = \frac{1}{a}, \quad x_{k+1} = \dfrac{1}{a + x_k}$.

a) Mostre que essa sucessão converge para um valor $\alpha \in [\frac{1}{a+1}, \frac{1}{a}]$.

Nota: O valor de α definido pela sucessão anterior é normalmente apresentado sob a forma de uma fracção contínua:

$$\alpha = \cfrac{1}{a + \cfrac{1}{a + \cfrac{1}{a + \ldots}}}$$

b) Quando $a = 1$, que número "famoso"é $1 + \alpha$? Determine uma aproximação para esse número, usando 8 iterações do método do ponto fixo. Obtenha uma estimativa para o erro $|\alpha - x_8|$.

Exercício 6.4. Pretende-se resolver a equação $\dfrac{1}{x} - e^x = 0$, a qual admite uma raiz perto do ponto $x = 0.5$.

a) Quais das seguintes fórmulas iterativas

$$x_{k+1} = -\ln x_k \, ; \qquad x_{k+1} = e^{-x_k} \, ; \qquad x_{k+1} = \frac{x_k + e^{-x_k}}{2} \, ,$$

poderão ser usadas ? E qual deverá ser usada ?

b) Calcule uma aproximação para essa raiz, usando a fórmula mais eficiente, iterando até que $|x_k - x_{k-1}| \leq 0.5 \times 10^{-3}$. Estime, então, o valor de $|\alpha - x_k|$.

c) Indique uma estimativa para o número de iterações que deveria efectuar para, usando a outra fórmula possível, garantir uma aproximação para a raiz com o mesmo número de casas decimais da aproximação obtida na alínea anterior.

Exercício 6.5. Considere a sucessão de números reais definida por

$$x_0 = 1, \quad x_{k+1} = 1 - \frac{4}{25 x_k}; k = 0, 1, 2, \ldots$$

a) Mostre que os termos desta sucessão estão todos no intervalo $[\frac{4}{5}, 1]$ e que a sucessão converge.

b) Diga, justificando, qual o limite α dessa sucessão.

c) Mostre que se tem

$$|\alpha - x_k| \leq \frac{16}{75}(\frac{1}{4})^k.$$

Exercício 6.6. Seja $g : [a, b] \longrightarrow [a, b]$ uma aplicação contractiva, com constante de Lipschitz L e seja α o ponto fixo de g em $I = [a, b]$. Seja $\{x_k\}$ a sequência definida por iteração do ponto fixo: $x_k = g(x_{k-1});\ x_0 \in I$. Mostre que, se $L < \frac{1}{2}$, então, ao utilizarmos o critério de paragem $|x_k - x_{k-1}| < \delta$, teremos garantia de que a iteração x_k satisfará também $|\alpha - x_k| < \delta$. Que poderá acontecer se $\frac{1}{2} < L < 1$?

✔ Exercício 6.7. Escreva uma função $[alfa, erro] = \mathbf{metPontoFixo}(g, x_0, tol, maxit)$ para determinar uma aproximação para um ponto fixo de uma determinada função, usando o esquema iterativo do ponto fixo. Essa função deve

- aceitar como argumentos:
  - uma certa função g (cujo ponto fixo se pretende);
  - um número real x_0 (aproximação inicial para o ponto fixo);
  - um número real positivo tol (tolerância para o erro);
  - um inteiro positivo $maxit$ (número máximo de iterações a efectuar);
- usar a fórmula iterativa do método do ponto fixo

  $$x_k = g(x_{k-1}); k = 1, 2, \ldots,$$

  para gerar uma sequência de aproximações para o ponto fixo de g;
- em cada passo, calcular o valor $erro = |x_k - x_{k-1}|$, devendo o processo iterativo parar se esse valor for inferior à tolerância tol; nesse caso, $alfa$ será a última iteração calculada;
- parar o processo iterativo se forem atingidas $maxit$ iterações sem que $erro < tol$, devendo, nesse caso, ser dada uma mensagem indicando que o número de iterações permitidas não foi suficiente para determinar o ponto fixo com a tolerância desejada.

Exercício 6.8. Considere a equação $x^2 - 5 = 0$. Use a função **metPontoFixo** para procurar uma aproximação para a raiz dessa equação, considerando como função iterativa cada uma das funções seguintes e tomando $x_0 = 2.5$ para aproximação inicial; comente os resultados obtidos.

a) $g_1(x) = x^2 + x - 5$

b) $g_2(x) = 5/x$

c) $g_3(x) = \frac{1}{2}(x + \frac{5}{x})$

Exercício 6.9. Considere o problema da determinação da menor raiz positiva da equação não linear

$$\cos x + \frac{1}{1 + e^{-2x}} = 0.$$

Mostre que os esquemas iterativos seguintes podem ser considerados formulações do método de ponto fixo para a determinação dessa raiz; considerando $x_0 = 3$, use a função **metPontoFixo** para procurar a raiz com cada um dos esquemas; justifique os resultados obtidos.

a) $x_{k+1} = \arccos\left(\frac{-1}{1 + e^{-2x_k}}\right)$

b) $x_{k+1} = \frac{1}{2}\ln\left(-\frac{\cos x_k}{1 + \cos x_k}\right)$

Exercício 6.10. Mostre que a equação $xe^{-x} - 0.25 = 0$ tem uma raiz no intervalo $[0, 1]$ e outra no intervalo $[1, 3]$. Determine, usando o método da bissecção, um valor aproximado para cada uma dessas raízes, com erro inferior a 0.05.

Exercício 6.11.

a) Determine um intervalo de amplitude 1 que contenha a raiz da equação $x \ln x - 1 = 0$.

b) Estime, "a priori", o número de iterações a efectuar para, usando o método da bissecção, determinar uma aproximação para essa raiz com 3 algarismos significativos e calcule essa aproximação.

Exercício 6.12. Como se comporta o método da bissecção no caso de zeros múltiplos da função f? Considere os casos de zeros de ordem par e de ordem ímpar.

Exercício 6.13. Escreva uma função $r = \mathbf{metBissec}(f, a, b, tol)$ para determinar uma aproximação para um zero de determinada função, usando o método da bissecção, seguindo o Algoritmo 6.1.

função $r = \mathbf{metBissec}(f, a, b, tol)$

% *Algoritmo do método da bissecção*

% **Parâmetros de entrada:** f função de uma variável
% a, b reais, $a < b$
% (e tais que $f(a)f(b) < 0$)
% tol real positivo (tolerância para o erro)
% **Parâmetro da saída:** r aproximação para uma raiz de $f(x) = 0$

$fa = f(a)$
$fb = f(b)$
% Determinar número de iterações para que o erro não seja superior a tol; ver (6.14)
$n = ceil((log(b - a) - log(tol))/log(2))$
para $k = 1 : n$
    $r = a + (b - a)/2$
    $fr = f(r)$
    se $fa * fr < 0$
        $b = r$
        $fb = fr$
    de outro modo se $fa * fr > 0$
        $a = r$
        $fa = fr$
    de outro modo
        sair do ciclo para
    fim
fim

Algoritmo 6.1. Método da bissecção

Exercício 6.14. Use a função **metBissec** para calcular as raízes das seguintes funções, nos intervalos indicados (considere $tol = 10^{-4}$).

a) $f(x) = x^3 - 2x - 5$; $[a, b] = [0, 3]$

b) $f(x) = e^x - 2$; $[a, b] = [0, 1]$

c) $f(x) = x^2 - \text{sen}x$; $[a, b] = [0.1, \pi]$

d) $f(x) = (x + 2)(x + 1)^2 x(x - 1)^3(x - 2)$:

(i) $[a, b] = [-1.5, 2.5]$;

(ii) $[a, b] = [-0.5, 2.4]$;

(iii) $[a, b] = [-0.5, 3]$;

(iv) $[a, b] = [-3, -0.5]$.

Exercício 6.15. Suponha que f admite uma raiz simples r e que usamos o método de Newton para calcular essa raiz, com x_0 suficientemente próximo de r. Obtenha, então, a seguinte estimativa para o erro na iteração k: $|r - x_k| \approx |x_{k+1} - x_k|$.

Exercício 6.16. Considere a equação $x^3 - 2x - 5 = 0$.

a) Mostre que a equação tem uma raiz real no intervalo $[2, 3]$.

b) Mostre que, para qualquer aproximação inicial $x_0 \in [2, 3]$, o método de Newton converge para essa raiz.

c) Efectue 5 iterações do método de Newton e estime o erro com que x_4 aproxima a raiz.

Exercício 6.17. Seja $f(x) = e^x + x - 7$. Prove que f tem um único zero e que o método de Newton converge para esse zero para qualquer aproximação inicial $x_0 \in \mathbb{R}$. Efectue 4 iterações do método de Newton começando a iterar em $x_0 = 0$.

Exercício 6.18. Modificando de forma conveniente a função **metPontoFixo**, escreva uma função $[r, erro] = \mathbf{metNewton}(f, fder, x_0, tol, maxit)$ para determinar uma aproximação para um zero de uma dada função, usando o método de Newton.

Nota: Neste caso, o parâmetro de entrada $fder$ destina-se a introduzir a derivada da função cujo zero se pretende (necessária ao uso da fórmula iterativa do método de Newton). Os critérios de paragem devem ser idênticos aos usados na função **metPontoFixo**.

Exercício 6.19. Utilize a função **metNewton** para obter aproximações para raízes das equações seguintes, com as aproximações iniciais indicadas, e comente os resultados obtidos.

a) $4\text{sen}\, x - e^x = 0; \quad x_0 = 0.5$

b) $x \ln x - 1 = 0; \quad x_0 = 2.5$

c) $x^3 - x - 1; \quad x_0 = 0$

d) $x^3 - 3.5x^2 + 4x - 1.5 = (x-1)^2(x-1.5) = 0$:

(i) $x_0 = 0.5$;

(ii) $x_0 = 1.3333$;

(iii) $x_0 = 1.4$

e) $-0.5x^3 + 2.5x = 0$:

(i) $x_0 = 1$;

(ii) $x_0 = -1$;

(iii) $x_0 = 2$

Exercício 6.20. Seja $f(x) = x + \exp(-20x^2)\cos x$.

a) Utilize a função **metNewton** para tentar obter uma aproximação para um zero de f, tomando $x_0 = 0$ e $tol = 10^{-10}$.

b) Esboce o gráfico de f relativo ao intervalo $[-1, 1]$ e justifique, então, os resultados da alínea anterior.

c) Faça 5 iterações do método da bissecção, com intervalo inicial $I = [-1, 1]$ para encontrar uma aproximação inicial mais conveniente para usar o método de Newton. Repita a alínea *a)* com esse novo valor de x_0 e comente os resultados.

Exercício 6.21. A curva definida por $y = x^3 - e^x$ tem dois pontos de inflexão. Use o método de Newton para os determinar.

Nota: Comece por esboçar um gráfico da função $f(x) = x^3 - e^x$.

Exercício 6.22. Modifique convenientemente a função $\mathbf{metNewton}$ para escrever uma função $[r, erro] = \mathbf{metSecante}(f, x_0, x_1, tol, maxit)$ destinada a implementar o método da secante; neste caso, devem ser fornecidas duas aproximações iniciais x_0 e x_1 para a raiz e não há, naturalmente, que introduzir a função derivada de f. Use essa função na resolução das equações de alguns exercícios anteriores, com aproximações iniciais x_0 e x_1 por si escolhidas.

Exercício 6.23. Use o método da secante para obter uma aproximação para a menor raiz positiva da equação $e^{-x} - \operatorname{sen} x = 0$.

Exercício 6.24. Use os métodos da bissecção, secante e Newton para encontrar (pelo menos) uma raiz das equações a seguir indicadas; em cada caso indique qual será a ordem do método usado.

a) $e^{-x} = x$

b) $x \operatorname{sen} x = 1$

c) $x^3 - 3x^2 + 3x - 1 = 0$

Exercício 6.25. Use a função pré-definida $\mathbf{fzero}$ para resolver as equações de alguns dos exercícios anteriores.

Exercício 6.26. Use a função pré-definida $\mathbf{fminbnd}$ para determinar o mínimo da função $f(x) = x^2 + 1 - \operatorname{sen} x$.

6.4 Trabalhos

Trabalho 6.1.

a) Considere a aplicação do método de Newton para resolver a equação

$$2 - e^x = 0,$$

a qual admite $r = \ln 2$ como solução, tomando como aproximação inicial $x_0 = 1$.

(i) Calcule os quocientes

$$R_k^{(2)} = \frac{|r - x_{k+1}|}{|r - x_k|^2}$$

e verifique se $R_k^{(2)}$ tende para o valor esperado (qual?) quando $k \to \infty$.

(ii) Calcule os quocientes

$$R_k^{(p)} = \frac{|r - x_{k+1}|}{|r - x_k|^p}$$

para valores de $p \neq 2$, mas próximos de 2, por exemplo, $p = 1.9, 1.8, 2.1$. Que verifica?

b) Considere novamente a aplicação do método de Newton, agora para a equação

$$1 - xe^{1-x} = 0,$$

a qual admite $r = 1$ como raiz dupla (tome $x_0 = 0$). Determine os quocientes $R_k^{(p)}$ para $p = 2$ e $p = 1$.

c) Mostre que se $f(r) = f'(r) = \cdots = f^{(m-1)}(r) = 0$ e $f^{(m)}(r) \neq 0$, então o seguinte *método de Newton modificado*

$$x_{k+1} = x_k - m\frac{f(x_k)}{f'(x_k)} \tag{6.20}$$

converge quadraticamente para r (supondo x_0 suficientemente próximo de r).

d) Modifique a função **metNewton** para implementar o método de Newton modificado (6.20) com $m = 2$ e repita a alínea b), usando esse método.

Comente devidamente os resultados obtidos.

Trabalho 6.2. A *equação de Kepler* relativa ao movimento dos planetas tem a forma

$$\omega t = \phi - \epsilon \operatorname{sen} \phi, \tag{6.21}$$

onde t é o tempo, ω é a frequência angular, ϵ é a excentricidade da órbita do planeta (elipse) e ϕ é o ângulo. Para determinar a localização do planeta no instante t, é necessário determinar o valor de ϕ da equação (6.21), sendo, então, as coordenadas x e y dadas por

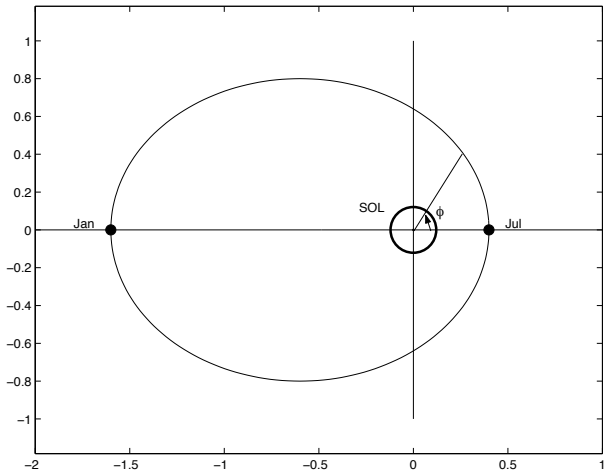
$$x = a(\cos \phi - \epsilon) \tag{6.22}$$

$$y = a\sqrt{1-\epsilon^2} \operatorname{sen} \phi, \tag{6.23}$$

onde a é o semi-eixo maior da elipse.

Considere um planeta com $a = 1$ AU [3] e excentricidade $\epsilon = 0.6$. [4]

A figura seguinte mostra a órbita desse planeta e a sua posição em "Janeiro"($\omega t = \pi \Leftrightarrow \phi = \pi$) e em "Julho"($\omega t = 0 \Leftrightarrow \phi = 0$).



O objectivo deste exercício é determinar a posição do planeta nos restantes dez meses do ano, assumindo que cada mês tem a duração de 1/12 do ano. Para isso, deverá

[3] AU (unidade astronómica)

[4] Este valor de ϵ é, na realidade, muito maior do que a excentricidade da órbita da Terra ...

resolver a equação de Kepler para $\omega t = \frac{k\pi}{6}; k = 1, \ldots, 11$, calculando depois, para cada um dos valores de ϕ obtidos, as correspondentes coordenadas x e y. Deverá, assim, acabar de preencher a tabela seguinte e esboçar a figura correspondente com as posições do planeta (semelhante à figura anterior, mas mais completa).

ωt	ϕ	x	y
0	0.000000	0.400000	0.000000
$\vdots$	$\vdots$	$\vdots$	$\vdots$
π	0.3141593	-1.600000	0.000000
$\vdots$	$\vdots$	$\vdots$	$\vdots$

6.5 Resoluções

Resolução do Exercício 6.7.

```
function [alfa,erro]=metPontoFixo(g,x0,tol,maxit)
%METPONTOFIXO  Calcula o ponto fixo de uma funcao e estima o erro
%
% [ALFA,ERRO]=METPONTOFIXO(G,X0,TOL,MAXIT)
%   calcula uma aproximacao para um ponto fixo de G e uma estimativa
%   para o respectivo erro, partindo de uma aproximacao inicial X0;
%   usa m'etodo do ponto fixo
%
% PARAMETROS DE ENTRADA:
%   G: uma funcao
%   X0: real (aproximacao inicial)
%   TOL: real positivo (tolerancia para o erro)
%   MAXIT: inteiro positivo (numero maximo de iteracoes a efectuar)
%
% PARAMETROS DE SAIDA:
%   ALFA: aproximacao para o ponto fixo de G (G(ALFA)=ALFA)
%   ERRO: estimativa para o erro
%
%  A funcao deve ser especificada como uma funcao anonima ou atraves de
%  uma function_handle

%-------------------------------------------------------------------------
%             PARAMETROS MAXIT e TOL POR DEFEITO
%-------------------------------------------------------------------------
if nargin==3
   maxit=30;
elseif nargin==2
    maxit=30; tol=1e-6;
end
%-------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%-------------------------------------------------------------------------
% Verificar se MAXIT 'e inteiro positivo e se TOL 'e positivo
if (length(maxit)~=1) | (fix(maxit) ~= maxit) | (maxit < 1)
  error('MAXIT deve ser inteiro positivo');
elseif (length(tol)~=1) | (tol<=0)
    error('a tolerancia para o erro deve ser positiva')
end
```

(continua)

```
%-------------------------------------------------------------------------
%               CABECALHO PARA TABELA DE RESULTADOS
%-------------------------------------------------------------------------
disp([' k ','  Aproximacao   ','  Erro '])
disp('---------------------------')
%-------------------------------------------------------------------------
%                       PROCESSO ITERATIVO
%-------------------------------------------------------------------------
iter=0;      % Inicializar o contador de iteracoes
erro=tol+1;  % Erro inicial superior a tolerancia
while ( erro>=tol & niter < maxit )
    alfa=g(x0);  % Calcular a nova iteracao
    erro=abs(alfa-x0); % Calcular o erro
    iter=iter+1;       % Incrementar o contador de iteracoes
    %
    % Imprimir resultados intermedios
    fprintf('%3.0f    %12.8g   %4.2e\n',iter,alfa,erro)
    %
    % Armazenar em x0 a ultima iteracao
    x0=alfa;
end
%-------------------------------------------------------------------------
%                    MENSAGEM (NAO CONVERGENCIA)
%-------------------------------------------------------------------------
if niter==maxit
        disp(' ')
        disp('Foi atingido o numero maximo de iteracoes !! ')
end
```

Listagem 6.1. Função **metPontoFixo**

7. Equações Diferenciais Ordinárias

Método de Euler
Métodos de Runge-Kutta
Métodos preditores-correctores

7.1 Notações, definições e resultados básicos

Neste capítulo, consideramos métodos para a resolução de problemas da forma

$$y'(x) = f(x, y(x)), \quad \text{para } x \in [a, b], \tag{7.1}$$

sujeitos a uma condição inicial

$$y(a) = \alpha. \tag{7.2}$$

Tais problemas são chamados problemas de valores iniciais (PVIs) (para equações de primeira ordem).

Naturalmente, muitos problemas físicos envolvem equações diferenciais de ordem superior à primeira; note-se, todavia, que uma tal equação diferencial pode ser reformulada como um sistema de equações diferenciais de primeira ordem, através do "artifício" de introduzir novas variáveis iguais às derivadas de y. Por exemplo, a equação diferencial de segunda ordem

$$y'' + f(x)y' + g(x)y = h(x)$$

pode ser transformada no seguinte sistema de duas equações de primeira ordem:

$$\begin{cases} y' = u \\ u' = h(x) - g(x)y - f(x)u \end{cases}$$

Os métodos numéricos que descreveremos para uma equação de primeira ordem generalizam-se de maneira simples para sistemas de tais equações, pelo que o caso aqui abordado não é tão restritivo quanto parece.

No que se segue admitimos que o problema dado tem uma e uma só solução $y(x)$, solução essa que pretendemos aproximar numericamente.

Os métodos numéricos considerados são ditos *métodos de variável discreta*, porque apenas fornecem aproximações y_k para y num conjunto discreto de pontos x_k no intervalo $[a, b]$. Para simplificar, suporemos que esses pontos são equidistantes, isto é, que

$$x_k = a + (k-1)h;\ k = 1, \ldots, N+1, \tag{7.3}$$

onde $h = (b-a)/N$ é o chamado *passo*. O valor inicial dá-nos

$$y_1 = y(x_1) = y(a) = \alpha.$$

7.1.1 Método de Euler

O mais simples desse métodos é o chamado *método de Euler*, definido por:

$$\left.\begin{array}{l} y_1 = \alpha \\ \text{Para } k = 1, \ldots, N: \\ \quad y_{k+1} = y_k + hf(x_k, y_k) \end{array}\right\} \tag{7.4}$$

A solução calculada é, naturalmente, afectada de erro. O erro nas aproximações obtidas vem de duas fontes: acumulação de erros de arredondamento cometidos ao efectuar as operações aritméticas envolvidas na utilização do método e erro de truncatura (ou discretização), inerente ao próprio método. Relativamente ao erro de discretização, distinguimos o chamado *erro de discretização local* do chamado *erro de discretização global*. Chama-se *erro de discretização local* no ponto x_k e denota-se por $\mathcal{L}_h(x_k)$, ao erro produzido na aplicação do passo $k-1$ do método (isto é, ao erro cometido ao passar de y_{k-1} para y_k), supondo que partimos da solução exacta, isto é, supondo que $y_{k-1} = y(x_{k-1})$. No caso do método de Euler, é fácil estabelecer o seguinte resultado.

Teorema 7.1 *Se $y \in C^2[a, b]$ e se for $M = \max_{a \le x \le b} |y''(x)|$, tem-se*

$$|\mathcal{L}_h(x_k)| \le \frac{h^2}{2} M. \tag{7.5}$$

Assim, nessas condições, podemos concluir que o erro de discretização local do método de Euler é $\mathcal{O}(h^2)$. Na prática, ao calcularmos o valor y_k, usaremos o valor de y_{k-1} (aproximação para $y(x_{k-1})$), o qual já foi calculado usando y_{k-2}, etc. Há, assim, que ter em conta a acumulação dos erros de discretização até chegar ao valor de y_k. Mais precisamente, chama-se *erro de discretização global* no ponto x_k e denota-se por $\epsilon_h(x_k)$ ao erro acumulado nos diversos passos até se chegar a y_k, isto é

$$\epsilon_h(x_k) = y(x_k) - y_k. \tag{7.6}$$

O teorema seguinte estabelece um limite superior para o valor absoluto do erro de discretização global do método de Euler.

Teorema 7.2 (Erro global do método de Euler) *Suponhamos que $y \in C^2[a,b]$, seja $M = \max_{a\leq x\leq b} |y''(x)|$ e suponhamos além disso que existe $L > 0$ tal que $|\frac{\partial f}{\partial y}| \leq L$, $\forall x \in [a,b]$. Então, tem-se*

$$E(h) := \max_{1\leq k\leq N} |\epsilon_h(x_k)| \leq \frac{hM}{2L}(e^{(b-a)L} - 1). \tag{7.7}$$

Podemos assim, concluir que, nas condições do teorema anterior, o erro de discretização global do método de Euler é $\mathcal{O}(h)$. Dizemos, por isso, que o método de Euler é um método de *primeira ordem*.

Nota: De um modo geral, se o erro de discretização local de um método é $\mathcal{O}(h^{p+1})$, então o erro de discretização global é $\mathcal{O}(h^p)$; dizemos, nesse caso, que o método tem *ordem* p.

7.1.2 Métodos baseados na série de Taylor

O método de Euler pode ser deduzido truncando a expansão em série de Taylor de $y(x_{k+1})$ em torno de x_k antes do termo $\mathcal{O}(h^2)$. Métodos de ordem superior podem ser obtidos de modo análogo, retendo mais termos da série de Taylor. Por exemplo, tem-se

$$\begin{aligned} y(x_{k+1}) &= y(x_k) + hy'(x_k) + \frac{h^2}{2}y''(x_k) + \frac{h^3}{6}y'''(\xi_k) \\ &= y(x_k) + hf(x_k, y(x_k)) + \frac{h^2}{2}(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}f)(x_k, y(x_k)) \\ &\qquad + \frac{h^3}{6}y'''(\xi_k) \end{aligned}$$

donde se poderia obter o método

$$y_{k+1} = y_k + hf(x_k, y_k) + \frac{h^2}{2}\left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} f\right)(x_k, y_k), \tag{7.8}$$

o qual teria ordem de convergência local $\mathcal{O}(h^3)$ e ordem de convergência global $\mathcal{O}(h^2)$. Este tipo de métodos baseados na expansão em série de Taylor têm, no entanto, a desvantagem de necessitarem do cálculo das derivadas parciais da função f, o que os pode tornar bastante trabalhosos.

7.1.3 Métodos de Runge-Kutta

Os *métodos de Runge-Kutta* (RK) foram desenvolvidos com o objectivo de produzir resultados com o mesmo grau de precisão dos métodos obtidos pela expansão em série de Taylor, mas evitando o cálculo das diversas derivadas da função f. Como casos particulares especialmente importantes, consideramos os seguintes:

Método de Runge-Kutta de 2ª ordem (RK2):

$$\left.\begin{array}{l} y_1 = \alpha \\ \text{Para } k = 1, \ldots, N: \\ \quad K_1 = hf(x_k, y_k), \\ \quad K_2 = hf(x_k + h, y_k + K_1), \\ \quad y_{k+1} = y_k + \frac{1}{2}(K_1 + K_2) \end{array}\right\} \tag{7.9}$$

Método de Runge-Kutta de 4ª ordem (RK4):

$$\left.\begin{array}{l} y_1 = \alpha \\ \text{Para } k = 1, \ldots, N: \\ \quad K_1 = hf(x_k, y_k), \\ \quad K_2 = hf(x_k + \frac{h}{2}, y_k + \frac{K_1}{2}), \\ \quad K_3 = hf(x_k + \frac{h}{2}, y_k + \frac{K_2}{2}), \\ \quad K_4 = hf(x_k + h, y_k + K_3), \\ \quad y_{k+1} = y_k + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{array}\right\} \tag{7.10}$$

7.1.4 Métodos de passo múltiplo

O método de Euler e os métodos de Runge-Kutta são exemplos dos chamados *métodos de passo único*, porque calculam o valor aproximado y_{k+1}, usando apenas o valor da aproximação no ponto anterior x_k. Nos chamados *métodos de passo múltiplo*, a aproximação da solução num certo ponto é calculada usando informação acerca do valor aproximado da solução em vários pontos. Uma classe importante desses métodos é baseada no princípio de integração numérica. Se integrarmos a equação diferencial

$$y'(x) = f(x, y(x))$$

entre o ponto x_k e o ponto x_{k+1}, vem

$$\int_{x_k}^{x_{k+1}} y'(x)dx = \int_{x_k}^{x_{k+1}} f(x, y(x))dx,$$

ou seja, vem

$$y(x_{k+1}) = y(x_k) + \int_{x_k}^{x_{k+1}} f(x, y(x))dx.$$

Substituindo $f(x, y(x))$ por um polinómio p que aproxime f, obter-se-á

$$y(x_{k+1}) \approx y(x_k) + \int_{x_k}^{x_{k+1}} p(x)dx$$

o que dará, portanto origem ao método

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} p(x)dx.$$

Métodos de Adams-Bashforth

No caso dos chamados *métodos de Adams-Bashforth* (AB), tomamos para p o polinómio p_m de grau não superior a m ($m \geq 1$), interpolador dos valores $f_i := f(x_i, y_i); i = k, k-1, \ldots, k-m$, onde $y_k, y_{k-1}, \ldots, y_{k-m}$ são aproximações para a solução do problema nos pontos $x_k, x_{k-1}, \ldots, x_{k-m}$, que admitimos conhecer. Apresentam-se de seguida as fórmulas dos métodos de Adams-Bashforth correspondendo aos casos $m = 1, 2, 3$, com a indicação da respectiva ordem.

Método de Adams-Bashforth de 2ª ordem (AB2)

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2 \text{ dado} \\ \text{Para } k = 2, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{2}(3f_k - f_{k-1}) \end{array}\right\} \tag{7.11}$$

Método de Adams-Bashforth de 3ª ordem (AB3)

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2, y_3 \text{ dados} \\ \text{Para } k = 3, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{12}(23f_k - 16f_{k-1} + 5f_{k-2}) \end{array}\right\} \tag{7.12}$$

Método de Adams-Bashforth de 4ª ordem (AB4)

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2, y_3, y_4 \text{ dados} \\ \text{Para } k = 4, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}) \end{array}\right\} \tag{7.13}$$

Métodos de Adams-Moulton

Os métodos de Adams-Moulton (AM) são obtidos tomando para p o polinómio interpolador dos valores $f_k, f_{k-1}, \ldots, f_{k-m}$ previamente calculados, mas também o valor desconhecido f_{k+1}, isto é, p é o polinómio de grau não superior a $m+1$ satisfazendo as condições de interpolação $p_{m+1}(x_i) = f_i; i = k-m, \ldots, k, k+1$. Seguem-se as fórmulas dos métodos de Adams-Moulton correspondendo aos casos $m = 0, m = 1$ e $m = 2$, com a indicação da respectiva ordem.

Método de Adams-Moulton de 2ª ordem (AM2)

$$\left.\begin{array}{l} y_1 = \alpha \\ \text{Para } k = 1, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{2}(f_{k+1} + f_k) \end{array}\right\} \tag{7.14}$$

Método de Adams-Moulton de 3ª ordem (AM3)

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2 \text{ dado} \\ \text{Para } k = 2, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{12}(5f_{k+1} + 8f_k - f_{k-1}) \end{array}\right\} \tag{7.15}$$

Método de Adams-Moulton de 4ª ordem (AM4)

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2, y_3 \text{ dados} \\ \text{Para } k = 3, \ldots, N: \\ \quad y_{k+1} = y_k + \frac{h}{24}(9f_{k+1} + 19f_k - 5f_{k-1} + f_{k-2}) \end{array}\right\} \tag{7.16}$$

Note-se que, contrariamente aos métodos de Adams-Bashforth, que são *explícitos*, todos os métodos de Adams-Moulton são métodos *implícitos*, uma vez que o valor de y_{k+1} aparece em ambos os lados da equação que define o método (relembre que $f_{k+1} = f(x_{k+1}, y_{k+1})$), isto é, está definido implicitamente.

7.1.5 Métodos preditores-correctores

Na prática, geralmente, os métodos implícitos são usados apenas para corrigir aproximações obtidas por fórmulas explícitas. Deste modo obtêm-se métodos conhecidos por *métodos preditores-correctores*.

Um método preditor corrector frequentemente utilizado combina os métodos de Adams-Bashforth e Adams-Moulton de 4ª ordem, do seguinte modo:

$$\left.\begin{array}{l} y_1 = \alpha,\ y_2, y_3, y_4 \text{ dados} \\ \text{Para } k = 4, \ldots, N: \\ \quad y_{k+1}^{(p)} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}) \\ \quad f_{k+1}^{(p)} = f(x_{k+1}, y_{k+1}^{(p)}) \\ \quad y_{k+1} = y_k + \frac{h}{24}(9f_{k+1}^{(p)} + 19f_k - 5f_{k-1} + f_{k-2}) \end{array}\right\} \tag{7.17}$$

Note-se que este método é totalmente explícito; primeiro utiliza-se o método explícito de Adams-Bashforth para "predizer" uma aproximação $y_{k+1}^{(p)}$; esta aproximação é utilizada

para obter um valor aproximado de f_{k+1}, $f_{k+1}^{(p)}$, o qual é então usado na fórmula do método de Adams-Moulton para corrigir o valor de $y_{k+1}^{(p)}$.

7.2 Notas e referências

➠ Funções pré-definidas

Função	Objectivo
ode15i	Resolução de EDO's definidas implicitamente
ode23	Resolução de EDO's (método ordem baixa)
ode45	Resolução de EDO's (método ordem média)

➠ Referências

Existe uma vasta bibliografia especializada na resolução numérica de problemas de valores iniciais, da qual destacamos, [Hen62], [Gea71], [Lam73] , [EEHJ96] e [Ise96]. Em [Sha97] encontrará referências a *softawre* facilmente acessível. Se procurar em

`http://www.netlib.org.`

encontrará códigos de diversos programas para reolver EDO's.

7.3 Exercícios

Exercício 7.1. Considere o seguinte problema de valores iniciais

$$\begin{cases} y' = 2x - y, \\ y(0) = -1. \end{cases}$$

a) Mostre que $y(x) = e^{-x} + 2x - 2$ é a solução exacta do problema dado.

b) Determine uma aproximação para a solução nos pontos $x = 0.1, 0.2, \ldots, 0.5$, usando o método de Euler, com passo $h = 0.1$.

c) Represente graficamente a solução exacta e a solução aproximada obtida na alínea anterior.

d) Calcule uma nova aproximação para $y(0.2)$, usando agora o passo $h = 0.05$.

✔ Exercício 7.2. Escreva uma função $y = \mathbf{metEuler}(f, a, b, \alpha, N)$, para determinar a solução de um problema de valores iniciais

$$\begin{cases} y'(x) = f(x, y), \quad x \in [a, b], \\ y(a) = \alpha, \end{cases} \tag{7.18}$$

usando o método de Euler. A sua função deve

- aceitar como argumentos:
  - uma função f (de duas variáveis x e y);
  - dois números reais a, b com $a < b$ (pontos inicial e final do intervalo onde se pretende a solução);
  - um real α (valor inicial da solução do problema);
  - um inteiro positivo N (número de subintervalos em que se pretende dividir o intervalo $[a, b]$).
- calcular o passo $h = (b-a)/N$ e obter aproximações para a solução do problema nos pontos $x_{k+1} = a + kh; k = 1, \ldots, N$, pelo método de Euler, apresentando os resultados num vector y.

Exercício 7.3. Use a função $\mathbf{metEuler}$ para resolver os PVI's abaixo indicados. Em cada caso, compare os valores aproximados com os valores da solução analítica; use $N = 10$ e repita para $N = 20$.

a) $y'(x) = xe^{3x} - 2y, \ \ x \in [0, 1]; \quad y(0) = 0$

Solução analítica: $y(x) = \frac{1}{5}xe^{3x} - \frac{1}{25}e^{3x} + \frac{1}{25}e^{-2x}$

b) $y'(x) = 1 + \frac{y}{x}, \ \ x \in [1, 2]; \quad y(1) = 2$

Solução analítica: $y(x) = x \ln x + 2x$

c) $y'(x) = 1 + \frac{y}{x} + \left(\frac{y}{x}\right)^2, \ x \in [1, 3]; \quad y(1) = 0$

Solução analítica: $y(x) = x \tan(\ln x)$

d) $y'(x) = \frac{1}{x^2} - \frac{y}{x} - y^2, \ \ x \in [1, 2]; \quad y(1) = -1$

Solução analítica: $y(x) = -1/x$

e) $y'(x) = x - 2\,x\,y, \;\; x \in [0,1]; \quad y(0) = 0$

Solução analítica: $y(x) = \frac{1}{2}(1 - e^{-x^2})$.

Exercício 7.4. Considere o seguinte problema de valores iniciais

$$\begin{cases} y' = xy^2, \; x \in [0,1]\,, \\ y(0) = 1. \end{cases}$$

a) Usando a função $\mathbf{metEuler}$, calcule aproximações para y(1.0), usando, sucessivamente, $N = 10, 20, 40, 80$.

b) Sabendo que a solução exacta do problema é dada por $y(x) = \frac{2}{2-x^2}$, diga se os seus resultados ilustram a ordem de convergência $\mathcal{O}(h)$ $(h = 1/N)$ do método.

Exercício 7.5. Use o método de Euler com passo $h = 0.1$ para encontrar uma solução aproximada, no ponto $x = 0.2$, do seguinte problema de valores iniciais:

$$\begin{cases} y''(x) = x\,y, \\ y(0) = 1\,, \quad y'(0) = 1\,. \end{cases}$$

Sugestão: Reescreva o problema como um sistema de duas equações diferenciais de 1ª ordem e adapte o método de Euler, que descrevemos para resolver uma equação, à resolução de um sistema de equações.

Exercício 7.6. Considere o problema de valores iniciais

$$\begin{cases} y'(x) = \text{sen}\,(x\,y), \\ y(0) = \pi. \end{cases}$$

a) Encontre uma solução aproximada no ponto $x = 0.4$, usando o método de Euler, com passo $h = 0.1$.

b) Escreva a expressão do método de Taylor de 2ª ordem para o problema dado.

c) Encontre uma solução aproximada para o mesmo ponto $x = 0.4$, usando o método de Taylor de 2ª ordem, com passo $h = 0.1$.

Exercício 7.7. Use o método de Runge-Kutta de 2ª ordem, com passo $h = 0.1$, para determinar um valor aproximado, nos pontos 0.1 e 0.2, da solução do seguinte problema de valores iniciais:

$$y'(x) = -\frac{y^2}{1+x}\,; \quad y(0) = 1\,.$$

Exercício 7.8. Use o método de Runge-Kutta de 4ª ordem, com passo $h = 0.1$, para determinar um valor aproximado, nos pontos 0.1 e 0.2, da solução do seguinte problema de valores iniciais:

$$y'(x) = -2xy\,; \quad y(0) = 1\,.$$

Exercício 7.9.

a) Modifique a função $\mathbf{metEuler}$ para obter uma função $y = \mathbf{metRK2}(f, a, b, \alpha, N)$ destinada a determinar a solução de um problema de valores iniciais do tipo (7.18), usando o método de Runge-Kutta de 2ª ordem.

b) Use essa função para resolver novamente os PVI's do Exercício 7.3.

Exercício 7.10.

a) Escreva uma função $y = \mathbf{metRK4}(f, a, b, \alpha, N)$ semelhante à função $\mathbf{metRK2}$, mas destinada a implementar o método de Runge-Kutta de 4ª ordem.

b) Use essa função para resolver os PVI's do Exercício 7.3.

Exercício 7.11. Considere um problema de valores iniciais

$$\begin{cases} y'(x) = f(x, y), \quad x \in [a, b], \\ y(a) = \alpha, \end{cases}$$

e suponha que $f(x, y)$ é apenas função de x. Mostre que, neste caso, o método de Runge-Kutta de 4ª ordem se reduz à aplicação da regra de Simpson:

$$\int_{x_k}^{x_{k+1}} f(x)dx = y(x_{k+1}) - y(x_k) \approx \frac{h}{6}[f(x_k) + 4f(x_k + \frac{h}{2}) + f(x_k + h)].$$

Exercício 7.12. Use o método preditor-corrector de 2ª ordem (Adams-Bashforth de 2ª ordem e Adams-Moulton de 2ª ordem) para determinar uma aproximação, no ponto $x = 0.75$, da solução do problema

$$y'(x) = -2xy^2(x), \qquad y(0) = 1.$$

Considere $h = 0.25$ e tome como aproximação para y em $x = 0.25$ o valor $y_2 = 0.9375$.

Exercício 7.13. Use o método preditor-corrector de 4ª ordem (Adams-Bashforth de 4ª ordem e Adams-Moulton de 4ª ordem) para determinar uma aproximação, no ponto $x = 0.5$, da solução do problema

$$y'(x) = y(x), \qquad y(0) = 1.$$

Tome $h = 0.1$ e considere as seguintes aproximações iniciais; $y_2 = 1.1051708$, $y_3 = 1.2214026$ e $y_4 = 1.3498585$. Compare a aproximação obtida com o valor exacto e comente.

Exercício 7.14.

a) Obtenha informação sobre as funções pré-definidas **ode23** e **ode45**.

b) Use a função **ode23** para resolver o seguinte problema de valores iniciais:

$$y'(x) = -y(x) - 5e^{-x}\text{sen}5x;\ y(0) = 1.$$

Determine a solução aproximada do problema no intervalo $[0, 3]$.

c) A solução exacta do problema considerado na alínea anterior é:

$$y(x) = e^{-x}\cos 5x.$$

Esboce o gráfico de $y(x)$ e da solução aproximada.

Exercício 7.15. Use a função **ode45** para encontrar aproximações para a solução do problema seguinte (*equação do pêndulo*):

$$\begin{cases} \frac{d^2}{dt^2}\theta(t) + \text{sen}\theta(t) = 0, \quad t \in [0, 10], \\ \theta(0) = 1;\ \frac{d\theta}{dt}(0) = 1\ . \end{cases}$$

Esboce o gráfico da solução. Repita o exercício para as condições iniciais

$$\theta(0) = -5;\ \frac{d\theta}{dt}(0) = 2 \qquad \text{e} \qquad \theta(0) = 5;\ \frac{d\theta}{dt}(0) = -2.$$

Sugestão: Comece por escrever a equação dada na forma

$$\begin{aligned} \frac{d}{dt}y_1(t) &= y_2(t) \\ \frac{d}{dt}y_2(t) &= -\text{sen}\, y_1(t), \end{aligned}$$

onde $y_1(t) = \theta(t)$ e $y_2(t) = \dfrac{d\theta(t)}{dt}$.

7.4 Trabalhos

Trabalho 7.1. **Método preditor-corrector AB4-AM4**

a) Escreva uma função $y = \mathbf{metPC4}(f, a, b, alfa, N)$ destinada a implementar o método preditor-corrector AB4-AM4 definido por (7.17). A sua função deve aceitar o mesmo tipo de argumentos de qualquer das funções escritas neste capítulo e

- usar o método de Runge-Kutta de 4ª ordem para obter os três primeiros valores y_2, y_3 e y_4;
- Para $k = 4, 5, \ldots, N$:
  - usar a fórmula do método AB4 para calcular uma primeira aproximação para y_{k+1};
  - estimar f_{k+1}, usando essa aproximação;
  - gerar uma nova aproximação y_{k+1}, usando a fórmula do método AM4.

b) Considere o PVI

$$\begin{cases} y'(x) = y - \frac{y}{x},\ x \in [1, 2], \\ y(0) = \frac{1}{2} \end{cases}$$

cuja solução exacta é: $y(x) = \frac{e^{x-1}}{2x}$. Use a função $\mathbf{metPC4}$ para resolver o problema, tomando $N = 10$. Faça uma tabela com os valores exactos, aproximados e respectivo erro, nos pontos x_k. Esboce o gráfico da função $y(x) = \frac{e^{x-1}}{2x}$, marcando também os valores aproximados obtidos.

Trabalho 7.2.

a) Modifique convenientemente a função $\mathbf{metRK4}$ para que ela possa calcular a solução de um sistema de m equações de primeira ordem:

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \ x \in [a, b], \\ \mathbf{y}(a) = \mathbf{a}, \end{cases} \tag{7.19}$$

onde

$$\begin{aligned} \mathbf{y}(x) &= (y_1(x), \ldots, y_m(x)) \\ \mathbf{f} &= (f_1, \ldots, f_m) \\ \mathbf{a} &= (a_1, \ldots, a_m). \end{aligned}$$

b) Considere um problema de valores iniciais associado a uma equação diferencial de 2ª ordem

$$\begin{cases} y''(x) = f(x, y(x), y'(x)), \ x \in [a, b], \\ y(a) = \alpha, \ y'(a) = \beta. \end{cases} \tag{7.20}$$

Reescreva este problema como um sistema de duas equações de primeira ordem, fazendo $y_1(x) = y(x)$ e $y_2(x) = y'(x)$.

c) Use a função construída na alínea *a)* para resolver o seguinte problema de valores iniciais

$$\begin{cases} y''(x) - 3y'(x) + 2y(x) = 6e^{3x}, \ x \in [0, 1], \\ y(0) = 1, \quad y'(0) = -1, \end{cases}$$

tomando $N = 10$. Apresente uma tabela com os valores aproximados para y e y', valores exactos e erros respectivos.

Nota: A solução exacta do problema é $y(x) = -8e^{2x} + 6e^{x} + 3e^{3x}$.

7.5 Resoluções

Resolução do Exercício 7.2.

```
function y=metEuler(f,a,b,alfa,N)
%METEULER  Calcula solucao de PVI pelo metodo de Euler
%
% Y=METEULER(F,A,B,ALFA,N)
%   calcula aproximacoes  para a solucao de um PVI
%   Y'=F(X,Y), Y(A)=ALFA, usando o metodo de Euler
%
% PARAMETROS DE ENTRADA:
%   F: uma funcao de duas variaveis;
%   A,B: reais, com A<B (extremos do intervalo onde se procura solucao);
%   ALFA: ral, valor inicial da solucao
%   N: numero inteiro positivo (numero de subintervalos).
%
% PARAMETRO DE SAIDA:
%   Y: vector com valores aproximados da solucao nos pontos
%       X_K=A+KH; H=(B-A)/N ; K=1,...,N+1,
%       otidas usando o metodo de Euler.
%
%  A funcao deve ser especificada como uma  funcao anonima ou atraves de
% uma function_handle.

%---------------------------------------------------------------------------
%             VERIFICACOES SOBRE PARAMETROS DE ENTRADA
%---------------------------------------------------------------------------
% Verificar se N e inteiro e se A<B
if (length(N)~=1) | (fix(N) ~= N) | (N < 1)
  error('N deve ser inteiro positivo');
elseif (a>=b)
    error('a deve ser menor que b')
end
%---------------------------------------------------------------------------
%          DETERMINACAO DE H E CRIACAO DE VECTOR DOS X_K
%---------------------------------------------------------------------------
h=(b-a)/N;
x=a:h:b;
%---------------------------------------------------------------------------
%                         METODO DE EULER
%---------------------------------------------------------------------------
y=zeros(1,N+1);  % Inicializacao do vector solucao.
y(1)=alfa;       % Primeira componente de y 'e o valor inicial ALFA.
for k=1:N
   y(k+1)=y(k)+h*f(x(k),y(k));
end
```


(continua)

```
%--------------------------------------------------------------------------
%               IMPRESSAO DE TABELA DE RESULTADOS
%--------------------------------------------------------------------------
disp('____________________')
disp([' x_k   ','     y_k    '])
disp('____________________')
tabela=[x;y];
fprintf('%5.2f %12.6f\n',tabela)
```

Listagem 7.1. Função **metEuler**

Bibliografia

[ABB+95] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User' s Guide*. Phildelphia, 1995.

[AS66] M. Abramowitz and C. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*. Dover, New York, 1966.

[Atk] K. E. Atkinson. Numerical Analysis.
disponível em `http://www.math.uiowa.edu/~atkinson/NA_Overview.pdf`.

[Atk89] K. E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons, New York, 1989.

[BF98] R. Burden and J. D. Faires. *Numerical Methods*. Brooks/Cole Publishing Company, Pacific Grove, USA, 1998.

[Cd80] S. D. Conte and C. de Boor. *Elementary Numerical Analysis: an Agorithmic Approach*. McGraw-Hill, Tokyo, 1980.

[Dav75] P. J. Davis. *Interpolation and Approximation*. Dover, New York, 1975.

[dB78] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[DBMS79] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Phildelphia, 1979.

[Dem97] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[DR84] P. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Academic Press, New York, 1984.

[EEHJ96] K. Eriksson, D. Estep, P. Hanso, and C. Johnson. *Computational Differential Equations*. Cambridge University Press, Cambridge, 1996.

[Eng80] H. Engels. *Numerical Quadrature and Cubature*. Academic Press, New York, 1980.

[Eva93] G. Evans. *Practical Numerical Integration*. John Wiley & Sons, New York, 1993.

[Gea71] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[GV96] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 1996.

[Hen62] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations*. John Wiley & Sons, New York, 1962.

[HH91] G. Hämmerlin and K.-H. Hoffman. *Numerical Mathematics*. Springer-Verlag, New York, 1991.

[HH00] D. J. Higham and N. J. Higham. *MATLAB guide*. SIAM, 2000.

[Hig96] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[Hou70] A. Householder. *The Numerical Treatment of a Single Nonlinear Equation*. McGraw-Hill, New York, 1970.

[IEE85] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*. Institute for Electrical and Electronics Engineers, New York, 1985.

[Ise96] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge, 1996.

[Lam73] J. D. Lambert. *Computational Methods for Ordinary Differential Equations*. John Wiley & Sons, London, 1973.

[Mat] The Math Works, Inc., Natwick,MA. *Using MATLAB*.

[MW91] P. G. Murray and M. H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Addison-Wesley, Reading, MA, 1991.

[Ove01] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, New York, 2001.

[PdUK83] R. Piessens, E. de Doncker-Kapenga, C. Uberhuber, and D. Kahaner. *QUADPACK: A Subroutine Package for Automatic Integration*. Springer-Verlag, New York, 1983.

[Pin95] H. Pina. *Métodos Numéricos*. McGraw-Hill, Lisboa, 1995.

[Pow81] M. J. D. Powell. *Approximation Theory and Methods*. Cambridge University Press, Cambridge, 1981.

[SB80] J. Stöer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New-York, 1980.

[Sch81] L. Schumaker. *Spline Functions: Basic Theory*. John Wiley, New York, 1981.

[Sha97] L. F. Shampine. The MATLAB ODE suite. *SIAM Journal of Scientific Computation*, 18:1–22, 1997.

[SS96] A. H. Stroud and D. Secrest. *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, New Jersey, 1996.

[Tre92] L. N. Trefethen. The definition of Numerical Analysis. *SIAM News*, 1992. disponível em `http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/`.

[Tre00] L. N. Trefethen. Predictions for scientific computing fifty years from now. *Mathematics Today*, (36):53–57, 2000.
disponível em `http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/`.

[Val96] M. R. Valença. *Análise Numérica*. Universidade Aberta, Lisboa, 1996.

[Van97] C. F. Van Loan. *Introduction to Scientific Computing*. Prentice Hall, Upper Saddle River, NJ, 1997.

[Wil63] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice Hall, Englewood Cliffs, NJ, 1963.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, New York, 1965.

A. Lista de Funções

Segue-se uma lista com todas as funções em MATLAB propostas ao longo do texto, separadas por capítulos.

	Nome	Objectivo	Pág.
Aritmética Computacional	**fl**	arredondamento	60
	fracDec2Bin ✔	conversão de fraccionário (decimal→ binário)	58
	mediaDesvios	média e desvios de uma amostra	67
Interpolação	**plotPolInt**	gráfico do polinómio interpolador	84
	polDifDes	polinómio interpolador (dif. descendentes)	82
	polDifDiv ✔	polinómio interpolador (dif. divididas)	80
	tabDifFin	tabela de diferenças finitas	81
	tabDifDiv ✔	tabela de diferenças divididas	79
Quadratura	**erroSimpson**	regra de Simpson e erro	99
	erroTrapezio	regra do trapézio e erro	98
	pesosAbcissasGL	pesos e abcissas de Gauss-Legendre	100
	regQuadGL ✔	quadratura de Gauss-Legendre	100
	regSimpson	regra de Simpson	98
	regSimpsonAdapt	regra de Simpson adaptativa	105
	regTrapezio ✔	regra do trapézio	97
	regTrapezioCorr	regra do trapézio corrigida	102
	tabRomberg	tabela de Romberg	104

(continua)

	Nome	Objectivo	Pág.
Sistemas	**decCholesky**	decomposição de Cholesky	128
Lineares	**luTrid**	decomposição LU (para matrizes tridiagonais)	127
	metGauss	método de Gauss (com escolha parcial de pivot)	126
	metGaussMod	método de Gauss (vários sistemas, determinante)	134
	metGSeidel	método de Gauss-Seidel	131
	metJacobi ✔	método de Jacobi	129
	metSR	método SR	135
	raioJacobi	raio espectral matriz de Jacobi	131
	raioGSeidel	raio espectral matriz de Gauss-Seidel	131
	resolveSistema	resolução sistemas lineares	134
	subDirecta ✔	substituição directa (sistema triangular inferior)	124
	subInversa	substituição inversa (sistema triangular superior)	125
Equações	**metBissec**	método da bissecção	152
Não Lineares	**metNewton**	método de Newton	153
	metPontoFixo ✔	método do ponto fixo	150
	metSecante	método da secante	155
EDO's	**metEuler** ✔	método de Euler	169
	metPC4	método preditor-corrector AB4-AM4	173
	metRK2	método Runge-Kutta de 2ª ordem	171
	metRK4	método Runge-Kutta de 4ª ordem	171

B. Revisões de Análise

Neste apêndice, apresenta-se um breve resumo de alguns resultados básicos de Análise particularmente relevantes para a disciplina de Análise Numérica.

Funções

Teorema B.1 (do valor intermédio ou de Bolzano) *Seja $f \in C[a,b]$ e seja L tal que $f(a) < L < f(b)$. Então existe $\xi \in (a,b)$ tal que $f(\xi) = L$. Em particular, se $f(a)f(b) < 0$, então existe (pelo menos) uma raiz da equação $f(x) = 0$ no interior do intervalo $[a,b]$.*

Teorema B.2 (de Weierstrass) *Seja $f \in C[a,b]$. Então f admite, em $[a,b]$, um máximo e um mínimo absolutos, isto é, existem valores m e M e $x_m, x_M \in [a,b]$:*

$$m = f(x_m) \leq f(x) \leq f(x_M) = M, \qquad \forall x \in [a,b].$$

Escrevemos $m = \min_{x\in[a,b]} f(x)$ e $M = \max_{x\in[a,b]} f(x)$.

Nota:
• Se, para além das condições do teorema anterior, tivermos que $f'(x)$ existe para todo o x em (a,b), então o máximo e mínimo absolutos só podem ser atingidos nos extremos do intervalo ou em pontos onde a derivada se anule.
• Na disciplina de Análise Numérica, é frequente termos necessidade de maximizar (ou, por vezes, minimizar) o valor absoluto de uma função f definida num certo intervalo $[a,b]$. Mesmo sendo f diferenciável em (a,b), o mesmo não se passa, em geral, com a a função $|f|$ (a não ser que f seja sempre positiva ou negativa). Assim, não podemos aplicar a $|f|$ o estabelecido no ponto anterior. Neste caso, o melhor processo de procurar os extremos de $|f|$ consiste em determinar os extremos de f, isto é, determinar $m := \min\limits_{x\in[a,b]} f(x)$ e $M := \max\limits_{x\in[a,b]} f(x)$, e ter em atenção que:

(i) $\max\limits_{x\in[a,b]} |f(x)| = \max\{|m|, |M|\}$.

(ii) $\min\limits_{x\in[a,b]} |f(x)| = \min\{|m|, |M|\}$, excepto se m for negativo e M for positivo, caso em que $\min\limits_{x\in[a,b]} |f(x)| = 0$.

Teorema B.3 (de Rolle) *Seja $f \in C[a,b]$ com derivada (finita ou infinita de sinal determinado) em todos os pontos de (a,b). Se $f(a) = f(b)$, então existe (pelo menos) um ponto $\xi \in (a,b)$ tal que $f'(\xi) = 0$.*

Teorema B.4 (de Rolle generalizado) *Seja $f \in C[a,b]$ e tal que $f'(x), \ldots, f^{(n)}(x)$ existam para todo o x em (a,b). Sejam $x_0, \ldots, x_n \in [a,b]$ tais que $f(x_0) = f(x_1) = \ldots = f(x_n)$. Então, existe $\xi \in (a,b)$ tal que $f^{(n)}(\xi) = 0$.*

Teorema B.5 (do valor médio de Lagrange ou dos acréscimos finitos) *Seja $f \in C[a,b]$ e tal que $f'(x)$ existe para todo o ponto x em (a,b). Então, existe $\xi \in (a,b)$ tal que*

$$f(b) - f(a) = f'(\xi)(b-a)$$

Teorema B.6 (do valor médio para somas) *Sejam $f \in C[a,b]$, $x_1, \ldots, x_n \in [a,b]$ e $\mu_1, \ldots, \mu_n$ números reais todos do mesmo sinal. Então, existe $\xi \in [a,b]$ tal que*

$$\sum_{i=1}^{n} \mu_i f(x_i) = f(\xi) \sum_{i=1}^{n} \mu_i$$

Teorema B.7 (do valor médio para integrais pesados) *Sejam $f, g \in C[a,b]$ e g de sinal constante em $[a,b]$. Então, existe $\xi \in [a,b]$ tal que*

$$\int_a^b f(x)g(x)\,dx = f(\xi) \int_a^b g(x)\,dx.$$

Nota: Em particular, tomando $g(x) \equiv 1$ no Teorema anterior, obtém-se o seguinte corolário.

Corolário B.1 (Teorema do valor médio para integrais) *Seja $f \in C[a,b]$. Então, existe $\xi \in [a,b]$ tal que*

$$\int_a^b f(x)\,dx = (b-a)f(\xi).$$

Teorema B.8 (de Taylor com resto de Lagrange) *Seja $f \in C^{n+1}[a,b]$ e seja x_0 um valor fixo no intervalo $[a,b]$. Então, para cada $x \in [a,b]$, existe um ponto ξ_x situado entre x_0 e x (ou seja, $\xi_x = x_0 + \theta(x - x_0),\ 0 < \theta < 1$) tal que*

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2}(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x-x_0)^{n+1}.$$

É mais frequente escrever a série de Taylor na seguinte forma

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \cdots + \frac{h^n}{n!}f^{(n)}(x_0) + \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(\xi_x). \qquad \text{(B.1)}$$

Teorema B.9 (Série de Taylor para funções de duas variáveis) *Sejam (x_0, y_0) e $(x_0+\xi, y_0+\eta)$ dois pontos dados e suponhamos que $f(x, y)$ é uma função $n+1$-vezes continuamente diferenciável num aberto que contém o segmento de recta que une (x_0, y_0) a $(x_0 + \xi, y_0 + \eta)$. Então,*

$$f(x_0+\xi, y_0+\eta) = f(x_0, y_0) + \left(\sum_{j=1}^{n} \frac{1}{j!} \left(\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right)^j f(x, y) \right) \Big|_{\substack{x=x_0 \\ y=y_0}} + \frac{1}{(n+1)!} \left(\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right)^{n+1} f(x, y) \Big|_{\substack{x=x_0+\theta\xi \\ y=y_0+\theta\eta}},$$

para algum $\theta \in (0, 1)$.

Séries

Teorema B.10 (Série geométrica) *Considere-se a série geométrica $\sum_{n=0}^{\infty} cr^n$. Então,*

- *Se $|r| < 1$, a série converge e a sua soma é $\frac{c}{1-r}$.*
- *Se $|r| \geq 1$, a série diverge.*

Teorema B.11 (Série alternada) *Considere-se a seguinte série alternada $\sum_{n=1}^{\infty}(-1)^{n-1}a_n$ onde $a_n \geq 0$, $\forall n$. Então, se a sucessão (a_n) decresce (em sentido lato) e tende para zero, a série converge e, designando por R_p o resto de ordem p dessa série, isto é, $R_p = \sum_{n=p+1}^{\infty} a_n$, tem-se*

$$|R_p| \leq a_{p+1}.$$

Nota: Os resultados sobre séries do tipo $\sum_{n=1}^{\infty}(-1)^{n-1}a_n$, com $a_n \leq 0$, $\forall n$ podem deduzir-se de imediato notando que, após multiplicação por -1, elas se convertem em séries do tipo anterior.

Símbolos de Landau $\mathcal{O}(\,.\,)$ e $o(\,.\,)$

Definição B.1 *Considerem-se duas funções $f, g : D \longrightarrow \mathbb{R}$, $D \subset \mathbb{R}$, tais que $g(x) \neq 0,\ x \in D$.*

- *Dizemos que f é de ordem $\mathcal{O}$ ("o grande") a respeito de g quando x tende para x_0 (finito ou $\pm\infty$), se existir uma constante $K > 0$ e um $\delta > 0$ tais que*

$$\left| \frac{f(x)}{g(x)} \right| \leq K,$$

para todo o $x \in D$, tal que $x \in B(x_0, \delta)$[1] e $x \neq x_0$. Escrevemos, então,

$$f(x) = \mathcal{O}(g(x)), \quad \text{quando} \quad x \to x_0.$$

[1] Entende-se por $B(+\infty, \delta) := \{x : x > \frac{1}{\delta}\}$, definindo-se, de modo análogo, $B(-\infty, \delta) := \{x : x < -\frac{1}{\delta}\}$.

- *Dizemos que f é de ordem o ("o pequeno") a respeito de g quando x tende para x_0 (finito ou $\pm\infty$), se a desigualdade anterior for válida para qualquer constante positiva K, ou seja, se*

$$\lim_{x\to x_o} \frac{f(x)}{g(x)} = 0.$$

Escrevemos, então,

$$f(x) = o(g(x)), \quad \text{quando} \quad x \to x_0.$$

Nota: Por vezes, escrevemos apenas $f(x) = \mathcal{O}(g(x))$ ou $f(x) = o(g(x))$, se for claro, pelo contexto, qual o ponto x_0 a que nos estamos a referir.
Exemplos:

1. Seja $f : [0,1] \longrightarrow \mathbb{R}$ tal que $f(0) = 0$. Se f é contínua em $[0,1]$, então $f(x) = o(1)$ (quando $x \to 0$); se f é continuamente diferenciável, então $f(x) = \mathcal{O}(x)$.
2. Seja (a_n) uma sucessão de números reais convergente para um certo número α. Então, $a_n = \alpha + o(1)$ (quando $n \to \infty$).
3. Considere-se a série geométrica $\sum_{i=0}^{\infty} r^i$, com $|r| < 1$ e seja s_n a respectiva sucessão das somas parciais $s_n = \sum_{i=0}^{n} r^i$. Então, como sabemos

$$s_n = \frac{1}{1-r} - \frac{r^{n+1}}{1-r}.$$

Então, é fácil de concluir que

$$s_n = \frac{1}{1-r} + \mathcal{O}(r^n).$$

Se tivermos uma sucessão (a_n) que verifique $a_n = \alpha + \mathcal{O}(r^n)$, para $|r| < 1$, dizemos que ela converge para α com convergência (pelo menos) geométrica, já que essa sequência se "comporta" (pelo menos) como a série geométrica.

4. Usando o símbolo $\mathcal{O}$ é usual escrever a expansão em série de Taylor referida em (B.1), como

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \cdots + \frac{h^n}{n!}f^{(n)}(x_0) + \mathcal{O}(h^{n+1}).$$

Seguem-se algumas propriedades que nos permitem operar com os símbolos $\mathcal{O}$ e o:

1. $f(x) = \mathcal{O}(f(x))$
2. $f(x) = \mathcal{O}(g(x))$ e $g(x) = \mathcal{O}(h(x)) \Longrightarrow f(x) = \mathcal{O}(h(x))$ (em notação mais simples, $\mathcal{O}(\mathcal{O}(h(x))) = \mathcal{O}(h(x))$.
3. $k\mathcal{O}(g(x)) = \mathcal{O}(g(x))$
4. $\mathcal{O}(g(x)) + \mathcal{O}(g(x)) = \mathcal{O}(g(x))$
5. $\mathcal{O}(g_1(x)).\mathcal{O}(g_2(x)) = \mathcal{O}((g_1.g_2)(x))$

6. $f(x) = o(g(x)) \Longrightarrow f(x) = \mathcal{O}(g(x))$.

Nota As propriedades 1.-5. são válidas com o símbolo $\mathcal{O}$ substituído por o.
Exemplo: Sejam $f(h) = \mathcal{O}(h^p)$ e $g(h) = \mathcal{O}(h^q)$ quando $h \to 0$, com $p, q \in \mathbb{N}$. Então, tem-se:

1. $f(h) = o(h^{p-1})$.
2. $f(h) + g(h) = \mathcal{O}(h^m),\ m = \min\{p, q\}$.
3. $f(h).g(h) = \mathcal{O}(h^{p+q})$.

Índice

UM introdução à álgebra linear
maria raquel valença

DOIS teoria das ôndulas
maria joana soares

TRÊS transformações conformes
maria irene falcão

QUATRO tópicos de análise real
lisa santos

CINCO equações diferenciais e integração múltipla
assis azevedo

SEIS funções de várias variáveis
assis azevedo

SETE introdução à análise real
lisa santos ⋆ fernando miranda

OITO autómatos e máquinas de turing
josé carlos costa

NOVE análise numérica: um curso prático com o MATLAB
maria irene falcão ⋆ maria joana soares

DEZ um curso de álgebra linear: um
paula mendes martins

ONZE um curso de álgebra linear: dois
paula mendes martins

DOZE um curso de matemática aplicada
ricardo severino