Nuno Manuel Clemente de Oliveira

Prática de Computação

IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

COIMBRA 2006

Muitas das tarefas existentes em todas as vertentes da vida comum são actualmente realizadas com o auxílio de computadores. Estas notas visam o desenvolvimento de capacidades de resolução de alguns problemas computacionais simples, de natureza numérica, encontrados com relativa frequência na actividade de Engenharia.

Tratando-se de uma abordagem introdutória à resolução de problemas numéricos com o auxílio do computador, procura-se imprimir a esta formação uma índole vincadamente prática, limitando voluntariamente a extensão teórica das matérias tratadas, e enfatizando simultaneamente os aspectos práticos necessários à implementação das técnicas ensinadas.

Neste sentido, prevê-se que esta actividade seja naturalmente completada com um número considerável de sessões "em frente ao computador", onde o carácter prático da disciplina pode ser devidamente exercitado. A formação recebida é consolidada mais tarde, com vantagem, no âmbito da formação em Métodos Numéricos, já na posse de uma maior sensibilidade e com bases mais sólidas relativamente à utilização dos meios de cálculo. As aptidões desenvolvidas neste grupo de disciplinas revelam-se fundamentais para a resolução de problemas concretos em áreas avançadas dos currículos em Engenharia, em especial nos domínios do Projecto de Processos e Produtos, e na análise e síntese de Estratégias de Supervisão.

Nuno Manuel Clemente de Oliveira

Prática de Computação

IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

• COIMBRA 2006

Coordenação Editorial
Imprensa da Universidade de Coimbra

Concepção Gráfica
António Barros

Execução Gráfica
Inova — Artes Gráficas, Porto

ISBN
972-8704-74-7

Depósito Legal
246754/06

"A slow sort of country!" said the Queen. "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"

LEWIS CARROLL, *Through the Looking Glass (1871)*

Sumário

Lista de Figuras

Prefácio

Estes apontamentos descrevem a matéria abordada na disciplina de *Prática de Computação* da licenciatura em Engenharia Química da Faculdade de Ciências e Tecnologia da Universidade de Coimbra, em particular durante o ano lectivo de 1998/99. A sua organização e ênfase reflectem o conteúdo programático estabelecido para esta disciplina, bem como o ordenamento da formação nesta área pelas diferentes disciplinas do currículo.

Com a frequência deste curso pretende-se que os alunos desenvolvam capacidades de formulação e de resolução de alguns problemas computacionais simples, encontrados com relativa frequência na actividade de Engenharia. Esta formação é antecedida pela familiarização com uma (ou mais) linguagens de programação (tipicamente Fortran 77 ou 90) na disciplina de *Introdução aos Computadores e Programação*, sendo posteriormente completada pelo estudo da resolução de outros problemas, na disciplina de *Métodos Numéricos*. O domínio de alguns conceitos básicos de Álgebra Linear e de Análise Matemática, abordados nas disciplinas dos 1º e 2º anos da licenciatura, é também vital para a correcta assimilação dos assuntos aqui mencionados.

Tratando-se de uma abordagem introdutória à resolução de problemas numéricos com o auxílio do computador, procura-se imprimir a esta formação uma índole vincadamente prática, limitando voluntariamente a extensão teórica das matérias tratadas, e enfatizando simultaneamente os aspectos necessários à implementação das técnicas ensinadas. Neste sentido, faz parte do ensino desta disciplina um número considerável de sessões "em frente ao computador", onde o carácter prático da disciplina pode ser devidamente exercitado. A formação recebida é consolidada, com vantagem, na disciplina de Métodos Numéricos, já na posse de uma maior sensibilidade e com bases mais sólidas relativamente à utilização dos meios de cálculo. As aptidões desenvolvidas neste grupo de disciplinas revelam-se fundamentais, mais tarde, para a resolução de problemas concretos em áreas avançadas dos currículos em Engenharia, em especial nos domínios do *Projecto de Processos e Produtos*, e na análise e síntese de *Estratégias de Supervisão*.

Este tipo de abordagem tem também reflexo na organização destas notas. Sempre que se justifique são incluídos algoritmos (em pseudo-código) para os mé-

todos descritos. São também incluídas referências bibliográficas abundantes relativas a algoritmos mais sofisticados, à resolução de problemas relacionados com os descritos, ou ainda referentes à demonstração de resultados teóricos adicionais considerados importantes. Um conjunto de exercícios típicos de ilustração das matérias abordadas é incluído no final destas notas. É também efectuada uma breve resenha crítica dos principais grupos de ferramentas disponíveis (tais como sistemas, bibliotecas numéricas, conjuntos de informações) considerados úteis à resolução de problemas numéricos do tipo considerado.

Estas notas estão divididas em vários Capítulos, com a seguinte estrutura:

Capítulo 1 São descritas algumas das principais características dos computadores, e o modo como estas influenciam as tarefas numéricas mais comuns. É considerada a representação de quantidades numéricas, e efectuada uma introdução à análise de erros. Outros conceitos básicos necessários à análise dos algoritmos descritos nos Capítulos seguintes (tais como normas vectoriais e matriciais) são também tratados.

Capítulo 2 São descritos algoritmos para a resolução de equações e de sistemas de equações não-lineares.

Capítulo 3 É abordada a resolução de sistemas de equações lineares (através de métodos directos e iterativos), conjuntamente com a análise de erros e o melhoramento iterativo da solução obtida.

Apêndice A É efectuada uma breve descrição de algumas ferramentas e recursos adicionais disponíveis.

Apêndice B Neste apêndice é incluído um conjunto de exercícios típicos, que ilustram a aplicação da matéria abordada.

É natural que estas notas contenham várias gralhas e omissões, e que algumas passagens não tenham sido descritas com toda a clareza possível. Correcções e sugestões adicionais relativas a este manuscrito devem ser enviadas para:

Nuno Oliveira (`nuno@eq.uc.pt`)
Departamento de Engenharia Química
Universidade de Coimbra
Rua Sílvio Lima — Pólo II
3030–790 Coimbra
Portugal

Uma errata, conjuntamente com outros recursos adicionais relacionados com este documento, incluindo os ficheiros relativos aos programas usados nos exemplos considerados, podem ser consultados através da Internet, em `<http://www.eq.uc.pt/~nuno/pc/>`.

Agradecimentos

Apesar da extensão algo limitada destas notas, diversas pessoas contribuíram, de forma expressiva, para a sua elaboração e publicação. Na impossibilidade de identificar todo o apoio recebido, gostaria de mencionar alguns dos contributos mais significativos.

Em primeiro lugar gostaria de agradecer o apoio e incentivo prestados pelo Professor Doutor Lélio Quaresma Lobo, na sua qualidade de Editor da Imprensa da UC, e como professor do DEQ/FCTUC. Sem a sua visão e persistência, este trabalho não teria sido completado em tempo útil. A sua ajuda foi também decisiva na criação de condições excepcionais para a minha formação científica nesta área.

O Professor José Almiro e Castro foi responsável directo, em larga medida, pela minha iniciação neste domínio científico, e pelo estabelecimento de inúmeras pontes entre o mundo científico e empresarial, onde muitos deste conceitos desempenharam, e continuam a desempenhar, um papel decisivo no sucesso de projectos de colaboração bastante diversificados. A sua memória é aqui recordada, com especial carinho.

A revisão dos Professores Doutora Fernanda Aragão Oliveira, da Universidade de Coimbra, e Doutor Sebastião Feyo de Azevedo, da Universidade do Porto, permitiu também enriquecer consideravelmente o manuscrito, e eliminar muitos erros e gralhas presentes nas versões preliminares deste documento. À primeira agradeço especialmente o cuidado colocado na revisão dos detalhes e da linguagem matemática usada, aspectos importantíssimos num trabalho desta natureza. Ao Prof. Sebastião, para além das inúmeras sugestões quanto ao conteúdo, agradeço ainda a especial amizade e hospitalidade.

Diversas falhas e oportunidades de melhoria foram também identificados em versões preliminares pelo Engenheiro João Gândara, Doutora Dulce Silva e Professor Lino Santos. A troca de experiências relativa ao sistema LaTeX com o Professor Pedro Nuno Simões permitiu tornar o processo de formatação electrónica mais ameno.

Na Imprensa da Universidade, o apoio e compreensão por parte de diversas pessoas, em especial do Professor Doutor Fernando Regateiro e da Doutora Maria João Castro foi absolutamente indispensável durante o processo de revisão e publicação.

Finalmente, uma palavra especial de apreço à Marta, por dar sentido a muitas destas aventuras computacionais.

Capítulo 1

Introdução

Muitas das tarefas existentes em todas as vertentes da vida comum são actualmente realizadas com o auxílio de computadores. No domínio da Engenharia, e mais particularmente no âmbito da Engenharia Química, estes meios são frequentemente utilizados como auxiliares de cálculo, em problemas de *carácter numérico*, envolvendo tarefas de natureza repetitiva ou de grandes dimensões.

Um conjunto importante de problemas estudados não possui soluções analíticas "exactas", mesmo depois de formulados com rigor em linguagem matemática. As soluções podem não ter sido ainda descobertas ou, em alguns casos, ter sido demonstrada a não existência de métodos exactos para a sua determinação; noutras situações, é a complexidade dos procedimentos de resolução analítica rigorosa que dificulta (podendo mesmo impedir) a sua utilização em problemas reais. Nestes casos torna-se necessário o recurso a *métodos aproximados*, onde a qualidade da solução é tradicionalmente seleccionada através de um compromisso face ao esforço envolvido na sua obtenção, traduzido por exemplo no volume de cálculos necessário ou no tempo de espera correspondente. O computador revela-se um auxiliar precioso no tratamento destes problemas, permitindo a realização dos cálculos necessários para a obtenção de soluções precisas.

Noutras situações, embora existam métodos de resolução conhecidos e aplicáveis, a solução dos problemas é dificultada pelo elevado volume de dados a tratar. Estes problemas requerem o estudo de implementações computacionais *eficientes* para a sua resolução.

O estudo de *algoritmos* (ou procedimentos detalhados) para a resolução de problemas de natureza numérica faz parte do domínio da *Análise Numérica* ou dos *Métodos Numéricos*. A implementação prática destes algoritmos requer contudo a familiaridade com um conjunto adicional de conhecimentos relativos ao funcionamento e à utilização de meios de cálculo, habitualmente considerados

no âmbito das *Ciências da Computação*. Da associação destes dois ramos de conhecimento surge a disciplina aqui designada por *Prática de Computação*, visando a resolução prática de problemas numéricos comuns no domínio da Engenharia com o auxílio de computadores.

Antes de iniciar o estudo de métodos de resolução para alguns problemas de carácter numérico é conveniente analisar as particularidades das representações numéricas mais comuns, tendo em vista a avaliação das suas principais limitações. O uso de representações ou processos de cálculo aproximados pode originar o aparecimento de *erros*, que devem ser identificados e controlados na sua propagação, por forma a garantir a qualidade do resultado final. Estes assuntos, juntamente com a exposição de outros conceitos elementares, são os temas abordados neste Capítulo.

1.1 Representação de quantidades numéricas

Um sistema de numeração de base n é caracterizado pela utilização de n dígitos distintos. O sistema binário utiliza os dígitos $d_2 = \{0, 1\}$; no sistema decimal (de base 10), a que estamos acostumados, são usados os dígitos $d_{10} = \{0, 1, 2, \ldots, 9\}$[1].

O sistema binário (de base 2) é presentemente usado, quase exclusivamente, para representar todos os elementos de informação armazenados nos computadores. Esta utilização generalizada deve-se sobretudo ao facto dos elementos de informação assim representados poderem ser facilmente codificados em circuitos eléctricos, usando apenas dois valores distintos (ligado / desligado, voltagem elevada / baixa). Esta facilidade é aproveitada com vantagem para a construção de circuitos que efectuam milhares de milhões de operações por segundo, como os microprocessadores actuais. Dada a sua relativa imunidade ao ruído, o sistema de codificação binário é também cada vez mais utilizado em sistemas de comunicação, como por exemplo em circuitos telefónicos, discos compactos de áudio e vídeo, e muito brevemente também em sistemas de rádio e televisão totalmente digitais.

A quantidade elementar de informação que pode ser representada através de um dígito binário (elemento de d_2) é designada por *bit* (Figura 1.1). Como 1 bit é uma quantidade muito pequena, é utilizado habitualmente o múltiplo

$$1 \text{ byte } (1 \text{ B}) = 8 \text{ bits}$$

[1] Na vida comum, algumas quantidades são ainda medidas às dúzias (base 12), quarteirões (base 25), etc.; alguns sistemas de unidades arcaicos também usam uma base diferente da decimal (por exemplo, 1 pé = 12 polegadas). A principal razão apontada para a popularidade do sistema decimal é o facto de possuirmos 10 dedos nas duas mãos :).

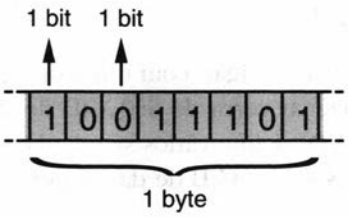


Figura 1.1 Organização típica de informação na memória dos computadores.

e definidos os múltiplos decimais comuns, na base decimal:

$$\begin{aligned} 1 \text{ kilobyte (kB)} &= 1\,000 \text{ bytes} &&= 10^3 \text{ B} \\ 1 \text{ megabyte (MB)} &= 1\,000 \text{ kilobytes} &&= 10^6 \text{ B} \\ 1 \text{ gigabyte (GB)} &= 1\,000 \text{ megabytes} &&= 10^9 \text{ B} \end{aligned}$$

Até à pouco tempo atrás, estas designações eram também usadas para indicar múltiplos binários de quantidades de informação; daqui resultava, por vezes, alguma ambiguidade nos valores representados. Para eliminar este problema foram recentemente criadas novas designações para os nomes e símbolos dos prefixos dos múltiplos binários a usar na expressão de quantidades de informação (Heldoorn, 2000). Assim, temos:

$$\begin{aligned} 1 \text{ kibibyte (KiB)} &= 1\,024 \text{ bytes} &&= 2^{10} \text{ bytes} &&= 1\,024 \text{ B} \\ 1 \text{ mebibyte (MiB)} &= 1\,024 \text{ kibibytes} &&= 2^{20} \text{ bytes} &&= 1\,048\,576 \text{ B} \\ 1 \text{ gibibyte (GiB)} &= 1\,024 \text{ mebibytes} &&= 2^{30} \text{ bytes} &&= 1\,073\,741\,824 \text{ B} \end{aligned}$$

Alguns exemplos de quantidades de informação correntes, expressos nestas unidades são: 3

- Um caracter alfa-numérico (letra do alfabeto romano, dígito decimal, ou sinal de pontuação), expresso no standard ASCII[2] ocupa geralmente 1 byte de memória num computador[3]. Uma página A4 com bastante texto em ASCII ocupa, em formato binário, cerca de 2 kB de memória. Assim, a quantidade de informação expressa num livro com cerca de 500 páginas (sem ilustrações) caberá, teoricamente, numa *disquete* de 3,5", com uma

[2] American Standard Code for Information Interchange.

[3] Este standard não considera contudo a representação de caracteres acentuados existentes nas linguagens europeias, ou dos caracteres especiais das línguas orientais. Para ultrapassar esta dificuldade foi desenvolvido mais recentemente o standard de codificação Unicode (Unicode, 2003). Este standard adoptou uma representação de 2 bytes por caracter, embora seja possível a utilização de formatos abreviados como o UTF-8, onde alguns caracteres são representados apenas por um byte.

capacidade de 1,4 MiB.

- Um disco compacto (CD) vulgar, com uma capacidade de 74 minutos de áudio, permite armazenar cerca de 650 MiB de dados. O novo formato de discos digitais (DVD) define vários standards de armazenamento, com capacidades próximas de 4,4 GiB de dados por camada do disco.
- As conversações telefónicas actuais são digitalizadas, por forma a tirar partido da maior imunidade ao ruído deste formato de transmissão, e a simplificar o seu encaminhamento. Na Europa, a velocidade de transmissão habitual nestes sistemas é de 8 kB por segundo (64 kbit/s) por canal de áudio, na rede fixa. Nas redes móveis e na telefonia através da Internet (VOIP) são definidos vários standards de codificação de som (*codecs*), usando frequentemente técnicas de compressão de dados. Isto permite uma redução típica no fluxo de dados para cerca de metade, sem perda de qualidade aparente.

1.1.1 Quantidades inteiras

As quantidades inteiras positivas $i \in \mathbb{N}$ são representadas habitualmente num computador usando uma notação semelhante à notação da base decimal. Isto permite a sua conversão para o sistema decimal, através de uma simples analogia:

$$1998_{10} = 1 \times \mathbf{10}^{3} + 9 \times \mathbf{10}^{2} + 9 \times \mathbf{10}^{1} + 8 \times \mathbf{10}^{0}$$
$$10011101_2 = 1 \times \mathbf{2}^{7} + 1 \times \mathbf{2}^{4} + 1 \times \mathbf{2}^{3} + 1 \times \mathbf{2}^{2} + 1 \times \mathbf{2}^{0} = 157_{10}$$

Como se verifica:

- a base da potência usada coincide com a base do sistema de numeração.
- os dígitos da representação vão sendo multiplicados por potências sucessivamente maiores da base, à medida que estão situados mais à esquerda no número.

As linguagens de programação recorrem por vezes a representações noutras bases, como a *octal*, com $d_8 = \{0, 1, 2, \ldots, 7\}$, e a *hexadecimal*, onde $d_{16} = \{0, 1, 2, \ldots, 9, \mathrm{A}, \mathrm{B}, \ldots, \mathrm{F}\}$. Nesta última base tem-se por exemplo:

$$\mathrm{FC9}_{16} = 15 \times \mathbf{16}^{2} + 12 \times \mathbf{16}^{1} + 9 \times \mathbf{16}^{0} = 4041_{10}$$

Como se pode ver, quanto maior for a base, maior será a economia de dígitos para representar a mesma quantidade, embora à custa de um "alfabeto" mais

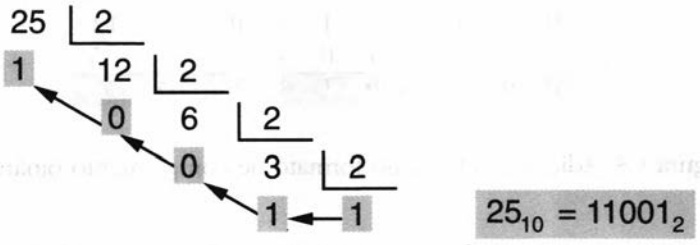


Figura 1.2 Exemplo de conversão decimal / binário.

extenso[4].

A conversão de números no sentido oposto (decimal → binário, por exemplo) pode ser obtida facilmente, por divisão sucessiva do número original pela base para a qual se pretende converter o número, retendo depois os restos obtidos (por ordem inversa), como se pode observar com o exemplo da Figura 1.2.

Problema 1.1 *Obtenha as representações de* 2001_{10} *nas bases binárias e hexadecimal.*

Esta codificação carece contudo de alterações por forma a possibilitar a representação de quantidades inteiras que podem assumir valores positivos ou *negativos*. As modificações mais usadas actualmente requerem que seja conhecido o *tamanho* da representação (i.e., o número total de dígitos das maiores quantidades representáveis), sendo comum o uso dos seguintes sistemas:

- Reserva de um *bit* adicional para representar o sinal do número através de uma convenção simples (por exemplo, 0 → positivo, 1 → negativo). Neste caso ter-se-ia, por exemplo numa representação com tamanho de 4 bits

  $$5_{10} = 0101_2 \quad \text{e} \quad -5_{10} = 1101_2$$

  usando o bit mais à esquerda para expressar o sinal do número.

- Formato de *complemento binário*: para obter um número negativo nesta representação escreve-se a quantidade positiva correspondente, complementam-se todos os dígitos binários[5], e soma-se 1 unidade ao resultado obtido.

  Por exemplo, usando um formato de 8 bits, a quantidade -1 pode ser obtida começando com $+1$ (00000001_2), complementando os bits

[4]Esta é a principal razão para o uso da base hexadecimal em algumas linguagens próximas da linguagem máquina dos computadores (por exemplo, nas linguagens *Assembler*), em vez do uso directo da numeração binária.

[5]I.e., substituem-se todos os dígitos 0 por 1, e vice-versa.

	0	0	0	0	1	1	0	0	$\longrightarrow$	12
+	1	1	1	1	1	0	1	1	$\longrightarrow$	−5
	0	0	0	0	0	1	1	1	$\longrightarrow$	7

Figura 1.3 Adição algébrica no formato de complemento binário.

(11111110_2), e somando uma unidade ao resultado, obtendo-se finalmente:

$$11111111_2 = -1$$

Esta representação permite a realização de operações de adição e subtracção com o mesmo *hardware*; a subtracção corresponde, neste caso, à adição do número complementado (Figura 1.3).

É importante notar que a mesma sequência binária 11111111_2 pode ser interpretada quer como 255_{10} (no caso da representação de *apenas* quantidades positivas), ou como -1_{10} no caso da representação em complemento binário. Mais genericamente, a utilização de 8 dígitos binários permite representar os inteiros $(0 \cdots 255)$ ou $(-128 \cdots +127)$ usando cada um dos sistemas anteriores. Devido a este facto, a interpretação correcta das quantidades em causa requer o conhecimento prévio do método usado na sua representação.

1.1.2 Números reais

Os números reais $x \in \mathbb{R}$ são representados habitualmente nos computadores usando o formato de *vírgula flutuante*, em base binária. Esta representação é actualmente aceite como um compromisso eficaz entre o espaço de memória usado, a sua precisão, e a velocidade de manipulação que esta forma permite.

Uma quantidade deste tipo, expressa na base decimal, é por exemplo:

$$-2{,}321 \times 10^{25} = -2\underbrace{321000\cdots0}_{\text{25 algarismos}}$$

Nesta representação, a quantidade 2,321 é designada por *mantissa*, sendo 25 o *expoente*. Uma quantidade semelhante expressa em base binária será por exemplo $-1{,}001_2 \times 2^{110_2}$. Esta quantidade pode ser facilmente convertida para decimal, atendendo a que

$$\text{mantissa:} \quad -1{,}001_2 = 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1{,}125_{10}$$

$$\text{expoente:} \quad 110_2 = 6_{10}$$

$$
\begin{aligned}
\mathbf{0,3} \times 2 &= \mathbf{0},6 \\
0,6 \times 2 &= \mathbf{1},2 \\
0,2 \times 2 &= \mathbf{0},4 \\
0,4 \times 2 &= \mathbf{0},8 \\
0,8 \times 2 &= \mathbf{1},6 \\
0,6 \times 2 &= \mathbf{1},2 \\
&\vdots
\end{aligned}
\quad \Rightarrow \quad \mathbf{0,3}_{10} = \mathbf{0,010011\cdots}_2
$$

Figura 1.4 Conversão de valores fraccionários para a base binária.

e logo:

$$(-1{,}001 \times 10^{110})_2 = (-1{,}125 \times 2^6)_{10} = -72_{10}$$

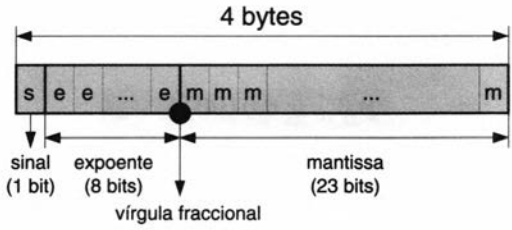
A conversão inversa (decimal → binária) pode ser efectuada separando previamente as partes inteira e fraccional da quantidade a converter, e juntando-as novamente depois de convertidas. Uma vez que a conversão de quantidades inteiras foi já descrita na Secção anterior, resta apenas ilustrar aqui a conversão da parte fraccional de quantidades reais. Isto é conseguido através de multiplicações sucessivas da quantidade a converter (ou da sua parte fraccional) por 2, *retendo o algarismo das unidades obtido em cada passo*, conforme ilustrado na Figura 1.4. Este exemplo permite também constatar que os números fraccionais expressos numa base através de um número finito de dígitos não o são necessariamente noutra base. Este facto origina *erros de arredondamento*, analisados na Secção 1.2.

O standard IEEE 754

Praticamente todos os computadores actuais usam o standard ISO / IEEE 754 (IEEE, 1985), na representação de quantidades reais em memória. Este standard é bastante exaustivo, uniformizando não só as representações usadas como também o seu tratamento, inclusivamente em situações excepcionais. O uso deste standard permite o desenvolvimento de programas científicos *portáteis* entre diferentes sistemas (i.e., que podem ser executados, de forma fiável, em computadores com arquitecturas bastante diferentes)[6].

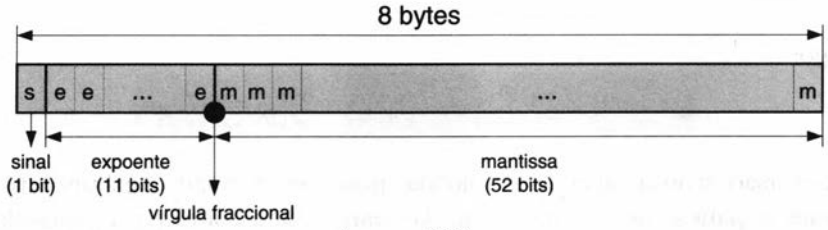
[6]Isto não significa, no entanto, que um programa executado em dois sistemas que obedecem estritamente a esta norma, terá que produzir necessariamente os mesmos resultados. Por exemplo, com a introdução crescente das capacidades de paralelismo nos microprocessadores actuais, a probabilidade de os cálculos serem efectuados por ordem diferente em arquitecturas distintas, produzindo erros de aproximação diferentes, é cada vez maior. Este comportamento é analisado

Precisão simples:



$$x = (-1)^s \times 1{,}\mathrm{mm}\cdots\mathrm{m}_2 \times 2^{(\mathrm{ee}\cdots\mathrm{e}_2 - 127)}$$

Precisão dupla:



$$x = (-1)^s \times 1{,}\mathrm{mm}\cdots\mathrm{m}_2 \times 2^{(\mathrm{ee}\cdots\mathrm{e}_2 - 1023)}$$

Figura 1.5 Formatos de representação no standard IEEE 754. O sinal do número é codificado especificando o primeiro bit igual a 0 ou 1, consoante se tratem de quantidades positivas ou negativas. O *offset* no expoente (127 ou 1023) permite representar quantidades positivas e negativas de magnitude aproximadamente idênticas.

O standard define várias precisões de representação, denominadas *simples*, *dupla* e *dupla estendida* (Kahan, 1996). As duas primeiras categorias são as mais usadas, correspondendo ao modelo de representação disponível nos standards das linguagens Fortran 77 e 90; estas duas categorias de representação encontram-se disponíveis na maioria dos sistemas actuais, mesmo noutras linguagens de programação. O standard IEEE 754 especifica os *tamanhos* e a *organização* de cada um dos tipos anteriores de dados na memória do computador, de acordo com a informação contida na Figura 1.5.

Em todos estas representações é usado um formato de representação *normalizado*, com um único dígito (não-nulo) à esquerda da vírgula (Figura 1.6). Representações *desnormalizadas* (também designadas como *subnormais*), com um dígito nulo à esquerda da vírgula, apenas são permitidas em casos excepcionais, evitando o arredondamento para zero de quantidades demasiado

no Exemplo 1.3, mais à frente.

$10{,}11 \times 2^{011} \rightarrow$ não-normalizado
$1{,}011 \times 2^{100} \rightarrow$ normalizado
$0{,}1011 \times 2^{101} \rightarrow$ não-normalizado

Figura 1.6 Representações alternativas da mesma quantidade no formato de vírgula flutuante.

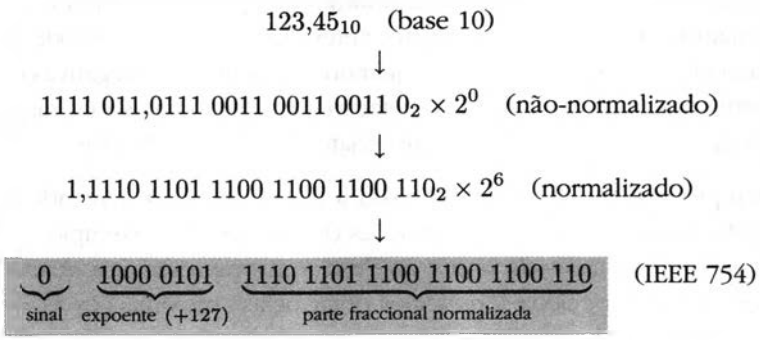


Figura 1.7 Exemplo de utilização do formato IEEE 754 em precisão simples. É notório que a quantidade $123{,}45_{10}$ não é representável de forma exacta no sistema binário, produzindo o padrão infinito de dígitos $00110011\cdots$ na sua parte fraccional.

pequenas para serem representadas no formato normalizado[7].

Uma vez que no formato normalizado o primeiro bit é sempre unitário, este pode ser implicitamente assumido (e não representado), usando-se a posição livre para representar um algarismo adicional na mantissa. Desta forma torna-se possível trabalhar com quantidades com 24 dígitos binários na mantissa, em precisão simples, e 53 dígitos em precisão dupla. A Figura 1.7 ilustra a utilização desta representação em precisão simples. O programa `REPIEE` (Apêndice A) permite examinar facilmente a representação em memória de quantidades segundo esta norma.

Para representar quantidades complexas, este standard utiliza um par de números do tipo anterior, guardando separadamente as componentes real e imaginária do número. Posteriormente, o standard IEEE 854, da mesma origem, veio

[7] A implementação da representação desnormalizada pode ser por vezes ineficiente, e efectuada apenas por software. Isto torna a manipulação destas quantidades mais lenta do que o uso do formato normalizado. Por esta razão, alguns programas podem requerer maior tempo de execução com conjuntos de dados onde a geração de quantidades desnormalizadas seja mais frequente (por exemplo, na resolução de equações diferenciais através de diferenças finitas). A utilização destas quantidades evita a divisão por zero em instruções do tipo `if (a ≠ b) then x = a/(a-b)`.

permitir uma representação mais liberal de quantidades reais. Esta norma possibilita a utilização das bases binárias *ou* decimal, não especificando a forma exacta como os números são codificados em memória, e caracterizando as precisões simples e dupla apenas em termos de gamas de precisão possíveis (Goldberg, 1991).

Em qualquer dos casos, o uso das normas anteriores tem implicações profundas em termos das possibilidades de representação. Assim:

1. Existe um número de *magnitude máxima* (e mínima) que pode ser representado em cada um dos formatos anteriores. Este número pode ser ligeiramente diferente nos casos de quantidades positivas e negativas devido a pequenas assimetrias da representação usada, embora as suas magnitudes (o aspecto mais importante aqui) sejam sempre semelhantes.

2. A representação da recta real passa a ser descontínua, havendo necessidade de introdução de aproximações (Figura 1.8). Por exemplo, com uma precisão de 4 casas *decimais*, ao desejarmos representar a quantidade π (um número irracional), podemos optar pelas aproximações $\pi_1 = 3{,}1415$ e $\pi_2 = 3{,}1416$. Qualquer destas quantidades constitui uma aproximação de $\pi = 3{,}1415927\cdots$, sendo o erro de aproximação menor (embora finito) no caso de π_2.

   Convém também esclarecer que estes erros não são apenas cometidos com quantidades irracionais, mas com todos os números que não podem ser representados exactamente na base escolhida. Por exemplo, a quantidade $1/10 = 0{,}1_{10} = 0{,}0001\ 1001\ 1001\cdots_2$ pode ser representada exactamente na base decimal, mas não na base 2. A quantidade $1/3 = 0{,}333\cdots_{10} = 0{,}0101\ 0101\cdots_2$ não pode ser representada exactamente em qualquer destas bases, com um número finito de dígitos.

10 Ao substituirmos quantidades reais pelas aproximações de vírgula flutuante mais próximas, de acordo com as limitações da representação usada, estamos a cometer *erros de arredondamento*. Estes erros são cometidos na aproximação de infinitas quantidades, uma vez que a recta real (um conjunto contínuo) é aproximado, neste modelo de representação, por um conjunto discreto de pontos. A implementação de um *algoritmo* numérico (que pode envolver, por exemplo, 10^{10} operações elementares), tendo por base esta representação inexacta (usada tanto para os dados de entrada como para guardar os valores intermédios produzidos), deverá garantir que estes erros não se acumulam de forma perigosa, retirando toda a precisão aos resultados finais obtidos[8].

[8]Como notado por Dowd e Severance (1998), o cálculo científico é muitas vezes usado para criar modelos do universo que nos rodeia, sendo irónica a utilização de números reais simulados nestas simulações do mundo real.

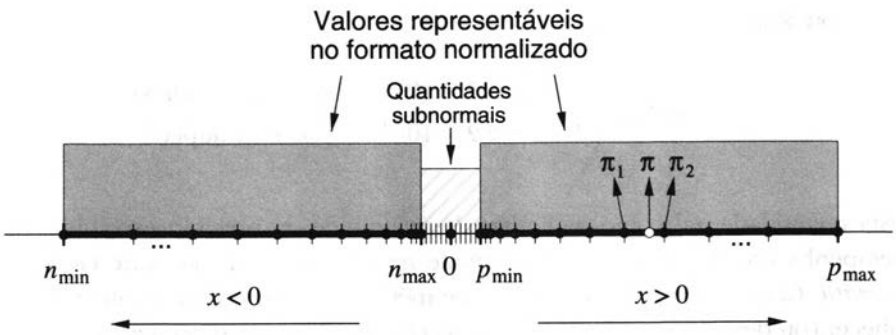


Figura 1.8 Aproximação da recta dos números reais usando aritmética de vírgula flutuante. Os pontos nesta recta correspondentes ao formato normalizado não estão regularmente espaçados, tendo antes uma distribuição semi-exponencial; na zona subnormal o espaçamento é uniforme.

Os limites das representações anteriores podem ser determinados de forma aproximada, tendo em consideração o espaço ocupado por cada um dos seus componentes. Assim, atendendo à Figura 1.8:

Magnitude máxima: Usando um formato normalizado, o número de maior magnitude que pode ser representado será $\pm 1{,}11\cdots 1 \times 2^{1\cdots 1}$, ou seja aproximadamente $2^{(2^{\text{tamanho expoente}-1})}$, medindo o tamanho do expoente através do seu número de bits. No caso da representação IEEE 754, isto significa que:

$$|n_{\min}|, p_{\max} = \begin{cases} 2^{128} \simeq 10^{38} & \text{(precisão simples)} \\ 2^{1024} \simeq 10^{308} & \text{(precisão dupla)} \end{cases}$$

Como se pode ver a magnitude máxima representável depende sobretudo do tamanho do expoente usado.

Precisão: A precisão da representação quantifica as distâncias entre pontos vizinhos na Figura 1.8. Se o arredondamento for efectuado correctamente, o seu erro máximo corresponderá a metade desta distância. Como o espaçamento da representação não é uniforme, é definida a quantidade ϵ_{mach} (precisão da máquina, numa dada representação) como a *menor* quantidade que pode ser somada à unidade, e produzir, depois de correctamente arredondada, um valor distinto de 1:

$$\epsilon_{\text{mach}} = \inf\{x > 0 : 1 + x \neq 1\}$$

Este valor depende do tamanho da mantissa usado. Designando por n.c.b. o número de casas binárias usadas, tem-se

$$\epsilon_{\text{mach}} = 2^{-\text{n.c.b.}}$$

ou seja:

$$\epsilon_{\text{mach}} = \begin{cases} 2^{-23} \simeq 1{,}2 \times 10^{-7} & \text{(precisão simples)} \\ 2^{-52} \simeq 2{,}2 \times 10^{-16} & \text{(precisão dupla)} \end{cases}$$

Esta quantidade, tal como os limites de magnitude da precisão escolhida, desempenha um papel bastante importante na concepção de software científico *portátil*. Como tal, muitas bibliotecas de métodos numéricos necessitam de conhecer (ou determinam automaticamente) estes valores, para garantir que a sua função é executada adequadamente. A quantidade ϵ_{mach} pode ser facilmente determinada através do seguinte algoritmo:

Algoritmo 1.1 (Determinação da precisão da máquina)

$\epsilon = 1$
Enquanto $(1 + \epsilon) \neq 1$ **fazer**
    $\epsilon = \epsilon/2$
Fim_Enquanto
Escrever $\epsilon_{\text{mach}} = 2\epsilon$

Problema 1.2 *Usando o algoritmo anterior, determine a quantidade ϵ_{mach} da sua calculadora. O que pode dizer acerca da precisão da representação usada? (NOTA: As calculadoras não utilizam necessariamente o sistema binário para a representação interna de quantidades numéricas).*

Problema 1.3 *Preveja o resultado de execução do seguinte programa num sistema que implemente a norma IEEE 754, em precisão simples:*


```
PROGRAM TESTPS

IMPLICIT NONE
REAL x, y

x = 1.23E6
y = x + 4.56E-3
IF (x .EQ. y) THEN
   PRINT *, "x = y. Estranho, não?"
ELSE
   PRINT *, "OK"
END IF
END
```

Problema 1.4 *Preveja o resultado de execução do seguinte programa num sistema que implemente a norma IEEE 754, em precisão simples:*

```
PROGRAM TESTRD

IMPLICIT NONE
DOUBLE PRECISION x1, x2
REAL y1, y2

x1 = 2453957.09236
y1 = 2453957.09236
x2 = 2453957.09236D0
y2 = 2453957.09236D0
WRITE (*,'(2f12.3)') x1, y1
WRITE (*,'(2f12.3)') x2, y2
END
```

O formato das instruções `WRITE` *indica que os resultados devem ser apresentados com 3 casas decimais, sem usar uma notação de vírgula flutuante.*

Os limites $|n_{\min}|$ e $p_{\max}$ são mais difíceis de determinar directamente. A aplicação directa de um método semelhante ao usado no Algoritmo 1.1 produziria um número tão grande cuja representação deixaria de ser possível. Em vez destas quantidades, é mais fácil calcular primeiro os valores correspondentes $|n_{\max}|$ e $p_{\min}$ através de um algoritmo semelhante ao anterior, detectando quando quantidades muito pequenas são arredondadas finalmente para zero[9]. Estes valores podem depois ser usados na determinação de $p_{\max}$ e $|n_{\min}|$, efectuando apenas algumas suposições adicionais (Cody, 1988).

Aritmética com precisão infinita e o standard IEEE 754 Uma característica extremamente importante desta norma é a especificação do uso de *3 dígitos binários adicionais* na mantissa (para além dos tamanhos indicados na Figura 1.5), durante a realização das operações aritméticas previstas pelo standard. Assim, as quantidades a processar são inicialmente copiadas da memória para os registos internos do processador onde são efectuados os cálculos (estes registos necessitam portanto de possuir um tamanho adequado). Quando necessário (por exemplo, no final dos cálculos) as quantidades produzidas como resultado das operações efectuadas são de novo guardadas em memória, arredondando o seu valor para os formatos da Figura 1.5.

Os dois primeiros dígitos adicionais existentes nos registos (denominados *de guarda*) são usados de forma idêntica aos restantes dígitos da mantissa previstos na representação. Estes são úteis, por exemplo, durante o alinhamento da vírgula fraccional dos operandos para as operações de soma e subtracção, evitando a perda de precisão do resultado. O 3º dígito adicional (denominado por "sticky bit") assume o valor 1 sempre que um dado operando possuir dígitos não nulos para além das posições disponíveis na mantissa (Figura 1.9). Este

[9]Será necessário, no entanto, lidar ainda com a possibilidade de serem geradas quantidades representadas num formato *subnormal*.

	Mantissa						Bits extra g	g	s	Parte não armazenável	
	1 , ···	0	1	0	0		0	0	0	0 0 0 ···	1º operando
+	0 , ···	0	1	1	0		0	1	0	0 1 0 ···	2º operando
	1 , ···	1	0	1	0		0	1	0	0 1 0 ···	soma c/precisão infinita
	1 , ···	1	0	1	0		0	1	1	→ "sticky bit" usado	
	1 , ···	**1**	**0**	**1**	**0**		→ valor final arredondado				

Figura 1.9 Exemplo de utilização dos dígitos adicionais no standard IEEE 754.

Tabela 1.1 Regras de arredondamento no standard IEEE 754 (Dowd e Severance, 1998).

Resultado da operação	Valor armazenado	Justificação para o valor guardado
1,0100 **000**	1,0100	Truncado com base nos dígitos de guarda
1,0100 **001**	1,0100	Truncado com base nos dígitos de guarda
1,0100 **010**	1,0100	Arredondado para baixo com base nos dígitos de guarda
1,0100 **011**	1,0100	Arredondado para baixo com base nos dígitos de guarda
1,0100 **100**	1,0100	Arredondado para baixo com base no "sticky bit"
1,0100 **101**	1,0101	Arredondado para cima com base no "sticky bit"
1,0100 **110**	1,0101	Arredondado para cima com base nos dígitos de guarda
1,0100 **111**	1,0101	Arredondado para cima com base nos dígitos de guarda

bit é consultado para desempatar as indicações de arredondamento fornecidas pelos 2 dígitos de guarda, de acordo com as regras indicadas na Tabela 1.1[10].

É possível demonstrar que os resultados obtidos pelo modelo de aritmética IEEE 754 coincidem com os resultados equivalentes obtidos num sistema com *precisão infinita*, arredondados correctamente no final das operações para o tamanho da representação utilizada (Goldberg, 1991).

Problema 1.5 *Elabore um programa numa linguagem à sua escolha para determinar* ϵ_{mach} *num computador que implemente o standard IEEE 754. Quais as*

[10]Para efeitos de ilustração foram apenas considerados 5 dígitos significativos nesta Tabela.

implicações da utilização de dígitos de guarda na determinação desta quantidade?

1.1.3 Condições de excepção

O standard IEEE 754 define as *condições de excepção* como situações anómalas, quando as operações especificadas geram valores especiais ou resultados indefinidos (tal como a que acontece quando se divide uma quantidade finita por zero), ou quando a representação do resultado da operação deixa de ser possível no formato utilizado. Nos sistemas actuais, estas situações podem ser tratadas de duas formas distintas:

1. O programa que está a ser executado é interrompido imediatamente, com a indicação da condição especial que motivou a paragem.

2. O programa assinala a ocorrência de uma condição excepcional, numa variável interna apropriada, prosseguindo os cálculos como habitualmente. O resultado da operação onde foi gerada a excepção é substituído por uma quantidade especial (pré-definida). Cabe, neste caso, ao utilizador, a responsabilidade de detectar a ocorrência destes erros, examinando as variáveis apropriadas.

A maioria dos sistemas actuais que implementam a norma IEEE 754 possui um comportamento (por defeito) correspondente ao segundo caso. Algumas implementações permitem a escolha da estratégia a adoptar, tendo em consideração a natureza da aplicação pretendida. Como a verificação das condições de falha em cada passo acarreta habitualmente uma penalidade importante no desempenho, é frequente a adopção da seguinte estratégia:

- É usado um algoritmo rápido, mas com a possibilidade de produzir em alguns casos resultados pouco precisos, ou gerar condições de excepção. Caso estas ocorram, o programa prossegue a sua execução da forma habitual.

- Efectua-se um teste rápido à exactidão da solução obtida, verificando-se também se foram geradas condições de excepção durante a sua execução.

- Se necessário (supostamente apenas em casos raros), a tarefa numérica é repetida, usando algoritmos mais seguros embora geralmente mais lentos.

O exemplo seguinte ilustra as consequências possíveis do não tratamento adequado destas situações.

Exemplo 1.1 (Tratamento incorrecto de condições de excepção)

São conhecidos vários casos onde o tratamento incorrecto de condições de excepção deu origem a consequências desastrosas. Dois incidentes recentes foram:

Ariane 5 (ESA, 1996): No voo inaugural, em 4 de Junho de 1996, o foguete Ariane 5, da Agência Espacial Europeia, explodiu apenas cerca de 40 segundos depois do início do voo, após um desvio violento da sua trajectória prevista. A análise dos dados de voo revelou que o veículo tinha perdido o controlo de orientação, apesar da redundância dos seus sistemas.

Uma análise mais detalhada permitiu compreender que os sistemas de orientação inerciais, reaproveitados do veículo Ariane 4, tinham produzido *condições de excepção* ao tentarem converter um dado (a velocidade lateral do veículo) de um formato de 64 bits (em precisão dupla) para um formato inteiro de 16 bits. O resultado produzido era demasiado grande, tendo sido produzido um erro de *overflow*. Como o software de controlo não foi concebido para lidar com estas situações de forma apropriada, o sistema de cálculo parou, tendo como consequência a destruição do veículo e da sua carga. A parte mais curiosa deste exemplo é que a informação gerada pelos sistemas que falharam apenas era usada antes da descolagem. Caso estes componentes tivessem sido desligados após o lançamento, o acidente não teria ocorrido.

USS Yorktown (Hayashi, 1998): Em Setembro de 1998, um navio da marinha de guerra norte-americana equipado com novas tecnologias de controlo, ficou totalmente imobilizado durante várias horas em alto mar, depois de um membro da tripulação ter introduzido, por engano, o valor zero num dado de entrada. O sistema dividiu outra quantidade pelo dado nulo introduzido, provocando um *overflow* e a corrupção de uma zona de memória contínua. Isto causou a desactivação completa do sistema de propulsão do navio, que acabou por ter de ser rebocado para o porto. Como é óbvio, acontecimentos deste tipo não abonam de forma positiva a fiabilidade do sistema operativo usado (Windows NT).

No standard IEEE 754 são definidas as seguintes excepções numéricas:

1. *Overflow*, quando uma quantidade *finita* se torna tão grande que a sua representação no formato em causa (também finito) se torna impossível.
2. *Underflow*, quando uma quantidade se torna demasiado pequena para ser representada (em formato normalizado), sendo consequentemente aproximada por zero. A possibilidade de utilização de quantidades *subnormais* evita a existência de um espaço vazio significativo entre o menor

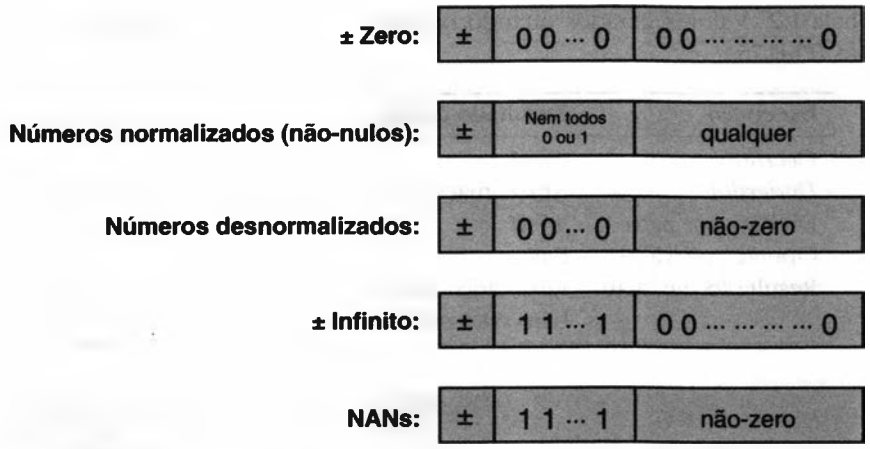


Figura 1.10 Representações possíveis no formato IEEE 754. Os campos dos expoentes com dígitos todos iguais a 1 são usados para representar quantidades especiais.

inteiro positivo e zero, permitindo o *underflow gradual* para zero (Figura 1.8).

3. *Divisão por zero* (dado que o seu resultado é indefinido).
4. *Operação inválida* (por exemplo, ao calcular $\sqrt{-1}$, $0/0$, $0 \times \infty$, ou na conversão de números de vírgula flutuante para outros formatos, quando os seus limites são ultrapassados).
5. *Resultado inexacto*, quando a operação pretendida não pode ser calculada com a precisão apropriada.

Na sequência de algumas destas ocorrências podem ser gerados os seguintes valores especiais: 17

- `NaN`, uma abreviatura de "Not a Number". Esta quantidade é produzida, por exemplo, nas operações $0/0$, $\infty + (-\infty)$, $0 \times \infty$, ∞/∞, e $\sqrt{-1}$.
- $\pm$`Inf` ($\pm\infty$): estas quantidades são produzidas, por exemplo, ao tentarmos calcular $1/0$, ou em situações de *overflow*.

Por curiosidade, as representações usadas no standard IEEE 754 para estas quantidades especiais, juntamente com as outras formas possíveis são apresentadas na Figura 1.10.

A Tabela 1.2 descreve os valores gerados quando ocorre cada uma das condições de excepção anteriores. Em geral, estes valores especiais propagam-se através das operações aritméticas seguintes (por exemplo, $0 +$ `NaN` $=$ `NaN`, $0 +$ `Inf` $=$ `Inf`), gerando resultados consistentes.

Tabela 1.2 Valores gerados quando ocorrem condições de excepção no standard IEEE 754.

Excepção	Resultado gerado
Overflow	±`Inf`
Underflow	Aproximação por uma quantidade subnormal
Divisão por zero	±`Inf`
Operação inválida	`NaN`
Resultado inexacto	Resultado aproximado, eventualmente também com *over* ou *underflow*

Detecção de `Nan`:

```
IF (x .NE. x) THEN
  PRINT *, 'Detectado Nan!'
  STOP
END IF
```

Detecção de `Nan` ou `Inf`:

```
IF (x - x .NE. 0) THEN
  PRINT *, 'Nan ou Inf!'
  STOP
END IF
```

Detecção de `Inf`:

```
IF (1e37/(1 + ABS(X)) .EQ. 0) THEN
  PRINT *, 'Detectado Inf!'
  STOP
END IF
```

Figura 1.11 Detecção dos valores especiais `NaN` e `Inf` no standard IEEE 754.

Apesar destes mecanismos estarem disponíveis actualmente na generalidade dos microprocessadores, os standards de muitas linguagens de programação (por exemplo, Fortran 77, Fortran 90/95 e ISO/ANSI C 1989–90) ainda não especificam mecanismos para detectar a ocorrência destas excepções numéricas (excepto quando o programa é interrompido abruptamente), sendo necessário recorrer a soluções específicas do compilador ou sistema operativo usado[11]. Nos casos em que não são viáveis melhores soluções, a ocorrência de excepções numéricas com geração das quantidades especiais `NaN` e `Inf` pode ser detectada através de fragmentos de código tais como os indicados na Figura 1.11. O funcionamento destes testes apenas é possível devido às particularidades da definição do próprio standard[12].

[11] O segundo standard ISO/ANSI para a linguagem C (C99, oficialmente publicado em 1999) já inclui mecanismos detalhados para a detecção e tratamento destas excepções numéricas. Algumas versões recentes dos sistemas UNIX incorporam estes desenvolvimentos, acessíveis também a partir de outras linguagens. Capacidades semelhantes foram também adicionadas à linguagem Fortran, no seu standard ISO/J3 de 2003.

[12] Por exemplo, as quantidades $+0$ e -0, com representações distintas possíveis, devem verificar o primeiro teste da igualdade da Figura 1.11.

1.1.4 Outras representações

Algumas linguagens de programação permitem outras representações (por exemplo, a precisão *dupla estendida* do standard IEEE 754, ou mesmo *precisão arbitrária*, disponível através de algumas bibliotecas especializadas, ou directamente em linguagens de manipulação simbólica como os sistemas *Mathematica* ou *Maple*[13]). Contudo, como estas formas não são habitualmente suportadas pelo *hardware* de forma nativa, a sua manipulação (por software) torna-se significativamente mais lenta. De facto, a diferença de velocidades de processamento entre quantidades manipuláveis directamente pela máquina e quantidades de precisão múltipla pode facilmente atingir várias ordens de magnitude. Este facto torna proibitivo o uso destas representações especiais, excepto em situações que exijam uma grande precisão nos cálculos efectuados, ou em problemas de pequena dimensão.

1.2 Erros numéricos e sua propagação

Pela sua natureza, e também devido à representação numérica usada, todos os métodos para a resolução de problemas numéricos com o auxílio de computadores são susceptíveis de gerarem erros. Os erros cometidos podem ter uma expressão individual diminuta, como por exemplo no caso da necessidade de arredondamento. No entanto, dado o elevado número de operações aritméticas elementares efectuadas[14], torna-se necessário garantir que os erros individuais não se acumulam, conduzindo a resultados finais sem qualquer significado para o problema que se pretendia resolver inicialmente. O estudo da estabilidade dos métodos de cálculo constitui pois um dos principais objectivos da *Análise Numérica*. Para uma melhor compreensão do fenómeno da propagação dos erros, é conveniente começar por enumerar as suas fontes, tentando simultaneamente perceber a sua natureza. Assim, temos (Dahlquist e Björck, 1974):

Erros nos dados de entrada: Quando os dados usados, provenientes por exemplo de um processo de medição, estão sujeitos a incertezas. Estes desvios podem ter uma natureza *sistemática* ou *não determinística*.

Erros de arredondamento: Estes erros são cometidos quando as quantidades a manipular não podem ser representadas directamente na máquina, sendo

[13] Descritos no Apêndice A.

[14] As unidades aritméticas dos microprocessadores usados nos computadores pessoais actuais utilizam uma arquitectura com diversas etapas em série, tendo também a possibilidade de execução de várias operações em paralelo. Isto permite-lhes efectuar uma ou mais operações de vírgula flutuante por ciclo de relógio. Para processadores com frequências de relógio da ordem dos GHz, isto pode significar centenas ou milhares de milhões de operações com números de vírgula flutuante por segundo.

aproximados por representações com um número finito de algarismos fraccionais. Por exemplo, a quantidade $1/10 = 0{,}1_{10}$ não pode ser representada exactamente na base 2, usando um número finito de dígitos fraccionais, tal como o número 1/3 não pode ser representado exactamente na base decimal.

Erros de truncatura: Quando a resolução de um problema inicial, envolvendo um número infinito de operações aritméticas, é aproximada pela resolução de um outro problema, a que corresponde um número finito de operações (por exemplo, truncando a soma de uma série infinita).

Simplificações no modelo matemático: Quando são cometidas idealizações na modelação dos fenómenos físicos que se pretendem descrever (por exemplo, no caso de modelos físicos sem atrito, ou supondo tanques de mistura onde é atingida uma homogeneização perfeita).

Erros humanos ou do equipamento: Nesta categoria podem ser incluídos erros de natureza bastante diversa, tais como erros de programação (por exemplo ao escrever `x = a x 2` em vez de `x = a * 2`), ou avarias do *hardware* (equipamento físico[15]).

Infelizmente, são conhecidas bastantes situações recentes, onde erros de várias categorias anteriores tiveram consequências graves importantes[16].

Exemplo 1.2 (Estimativa dos erros de natureza humana)

Relativamente à componente humana, algumas fontes apontam para as seguintes estimativas relativas à presença de erros deste tipo em *software* (Huckle, 1999), também conhecidos como "bugs":

- programas de *qualidade média* contêm até 25 erros por 1000 linhas de código.
- programas de *boa qualidade* contêm até 2 erros por 1000 linhas de código.
- programas onde a fiabilidade é primordial (por exemplo, na indústria aeroespacial) podem conter menos de 1 erro por 10 000 linhas de código.

[15]Por vezes, podem surgir erros imprevisíveis, em equipamentos em perfeito estado de funcionamento. Com a crescente miniaturização dos componentes, a energia necessária para alterar o valor guardado num registo ou numa posição de memória de um computador é cada vez menor. Este facto, conjugado com o aumento das capacidades de armazenamento, torna maior a probabilidade de ocorrência de alterações imprevisíveis, devido a causas como a radiação exterior, ou erros durante a transferência de valores entre subsistemas de memória. Por esta razão alguns sistemas, onde é importante uma fiabilidade elevada, recorrem a mecanismos de ***paridade*** ou outros esquemas para a detecção e correcção destes erros.

[16]Algumas compilações deste género podem ser encontradas em Huckle (1999) e Dershowitz (1998).

Esta qualidade é conseguida prolongando consideravelmente os períodos iniciais de teste e certificação, e resistindo à introdução de modificações posteriores no sistema.

Usando estas estimativas é possível efectuar as seguintes previsões:

- A versão inicial do sistema de gestão de um telefone móvel digital de 1ª geração (cerca de 200 000 linhas de código) poderá conter até 400 *bugs*.
- A versão inicial de um sistema operativo como o Windows 95 (cerca de 10^6 linhas de código) poderá conter entre 20 000 e 200 000 *bugs*.

Obviamente, estes números devem ser considerados apenas como *estimativas*. Na prática o número de erros depende de um grande número de outros factores, tais como a natureza da tarefa (e o seu "grau de dificuldade"), da experiência do(s) programador(es), da linguagem escolhida, da reutilização de código previamente testado, bem como até do próprio compilador ou do sistema de implementação da linguagem usada, e do uso de ferramentas sistemáticas de depuração de código (Apêndice A).

Exemplo 1.3 (Erros de arredondamento)

Considere-se o problema de determinar a soma de um conjunto de termos consecutivos da *série harmónica*:

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots=\sum_{i=1}^{\infty}\frac{1}{i} \tag{1.1}$$

Como é conhecido, esta série é divergente. Para n finito mas elevado, não é portanto possível aproximar a sua soma de ordem n

$$S_n=\sum_{i=1}^{n}\frac{1}{i} \tag{1.2}$$

por uma quantidade finita, conhecida à priori. O programa da Figura 1.12 tenta obter este resultado para $n = 10^7$, directamente, ordenando os termos por ordem *crescente* e *decrescente* do índice i[17], e comparando os resultados com a aplicação do *método de Kahan* (Goldberg, 1991) para a soma de um conjunto de termos (ordenando também os termos por ordem crescente e decrescente).

Os cálculos são repetidos em precisão simples e em precisão dupla, obtendo-se os seguintes resultados num computador que respeita a norma IEEE 754:

```
Termos em ordem crescente:
 15.4036827 (p. simples),   16.6953114 (p. dupla)
```

[17] I.e., usando as fórmulas $1/1 + 1/2 + 1/3 + \cdots$ e $1/n + 1/(n-1) + 1/(n-2) + \cdots$.

```
      PROGRAM CALSUM

c     --- Cálculo de uma soma através de vários métodos ---

      IMPLICIT NONE
      INTEGER i, N
      PARAMETER (N=10000000)
      REAL rsum, rc, ry, rt
      DOUBLE PRECISION dsum, dc, dy, dt

c     --- Cálculo directo (ordem crescente) ---
      rsum = 0E0
      dsum = 0D0
      DO 100 i = 1, N
         rsum = rsum + 1E0/REAL(i)
         dsum = dsum + 1D0/DBLE(i)
 100  CONTINUE
      WRITE (*, *) 'Termos em ordem crescente:'
      WRITE (*, *) rsum, ' (p. simples), ', dsum, ' (p. dupla)'

c     --- Cálculo directo (ordem decrescente) ---
      rsum = 0E0
      dsum = 0D0
      DO 200 i = N, 1, -1
         rsum = rsum + 1E0/REAL(i)
         dsum = dsum + 1D0/DBLE(i)
 200  CONTINUE
      WRITE (*, *) 'Termos em ordem decrescente:'
      WRITE (*, *) rsum, ' (p. simples), ', dsum, ' (p. dupla)'

c     --- Fórmula de Kahan (ordem crescente) ---
      rsum = 1E0
      dsum = 1D0
      rc = 0E0
      dc = 0D0
      DO 300 i = 2, N
         ry = 1E0/REAL(i) - rc
         dy = 1D0/DBLE(i) - dc
         rt = rsum + ry
         dt = dsum + dy
         rc = (rt-rsum) - ry
         dc = (dt-dsum) - dy
         rsum = rt
         dsum = dt
 300  CONTINUE
      WRITE (*, *) 'Fórmula de Kahan (ordem crescente):'
      WRITE (*, *) rsum, ' (p. simples), ', dsum, ' (p. dupla)'

c     --- Fórmula de Kahan (ordem decrescente) ---
      rsum = 1E0/REAL(N)
      dsum = 1D0/DBLE(N)
      rc = 0E0
      dc = 0D0
      DO 400 i = N - 1, 1, -1
         ry = 1E0/REAL(i) - rc
         dy = 1D0/DBLE(i) - dc
         rt = rsum + ry
         dt = dsum + dy
         rc = (rt-rsum) - ry
         dc = (dt-dsum) - dy
         rsum = rt
         dsum = dt
 400  CONTINUE
      WRITE (*, *) 'Fórmula de Kahan (ordem decrescente):'
      WRITE (*, *) rsum, ' (p. simples), ', dsum, ' (p. dupla)'
      END
```

Figura 1.12 Cálculo da soma de ordem n da série harmónica, em Fortran 77.

```
Termos em ordem decrescente:
 16.6860313 (p. simples),   16.6953114 (p. dupla)
Fórmula de Kahan (ordem crescente):
 16.6953106 (p. simples),   16.6953114 (p. dupla)
Fórmula de Kahan (ordem decrescente):
 16.6953106 (p. simples),   16.6953114 (p. dupla)
```

Como se pode verificar, estes resultados diferem entre si. Especialmente quando o índice i varia de forma crescente, a tradução directa de (1.2) torna-se o método mais impreciso, devido a uma maior acumulação de erros de arredondamento. O procedimento recomendado para este caso consiste em ordenar os termos por por *ordem crescente de magnitude* antes de proceder à sua soma.

Estas considerações originam as seguintes implicações:

- Com as crescentes capacidades de paralelismo dos computadores e microprocessadores actuais, os resultados obtidos num determinado cálculo podem depender da ordem de execução dos seus passos intermédios, que pode diferir entre compiladores ou outros sistemas de sequenciamento das instruções de cálculo usados.

- Conforme ilustrado no problema 1.3, as expressões `(A + B) + C` e `A + (B + C)`, embora equivalentes do ponto de vista algébrico, não são equivalentes do ponto de vista computacional quando é usando o modelo de aritmética de vírgula flutuante, podendo conduzir a resultados bastante distintos. Este facto deve ser considerado quando um dado programa é compilado de forma a *optimizar* o seu tempo de execução.

Em particular quando são usadas opções de optimização dos programas, com o objectivo de reduzir o seu tempo de execução, alguns compiladores podem introduzir alterações significativas no código produzido. Um exemplo é a remoção de parêntesis "supérfluos" entre operandos, respeitando apenas as prioridades de aplicação dos diversos operadores matemáticos. Por exemplo, as instruções `x = a + (b + c)` e `x = a + b + c` poderão produzir exactamente o mesmo código, neste caso. Outra possibilidade (mais drástica, embora também realista) é a eliminação de código considerado *redundante* para os resultados finais pretendidos.

Problema 1.6 *Explique as principais diferenças entre o uso das linhas de código*

```
x = x0 + j*h
```

para $j = 0, 1, 2, \ldots, n$, *com* n *dado, e*

```
x = x + h
```

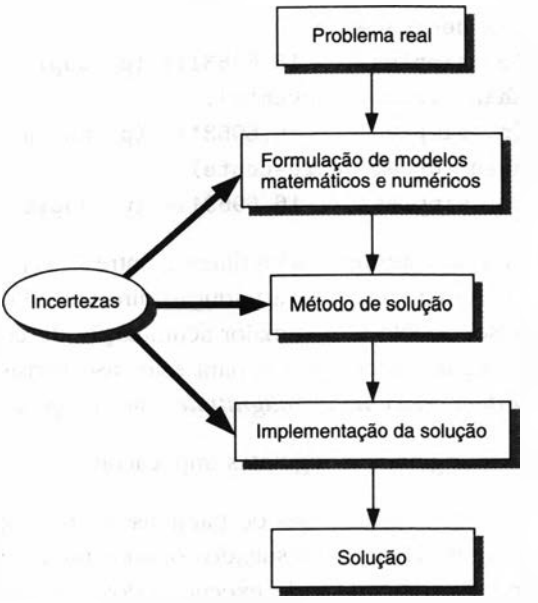


Figura 1.13 Fontes de erro na resolução de problemas numéricos (Rice, 1993).

inicializando previamente `x = x0`. *Embora estas instruções correspondam a expressões matemáticas equivalentes, indique qual é o procedimento:*

- *Mais demorado.*
- *Mais preciso.*

Tenha em atenção que h pode não poder ser representado exactamente no sistema de vírgula flutuante do computador (por exemplo, para $h = 0{,}1$*).*

A Figura 1.13 ilustra a dependência do resultado final relativamente a todas as fontes de incerteza discutidas anteriormente. Face a este cenário, torna-se praticamente impossível garantir que um valor produzido como resultado de um processo de cálculo, mesmo bastante simples, esteja isento de erro. Importa portanto conhecer a forma como os diferentes erros cometidos se propagam, estabelecendo limites superiores para a precisão dos resultados obtidos. Isso poderá ser conseguido depois da quantificação dos erros respectivos.

1.2.1 Erro absoluto e relativo

Dada uma quantidade $x \in \mathbb{R}$, e uma aproximação desta $\bar{x} \in \mathbb{R}$, define-se o *erro absoluto* da aproximação como:

$$e \equiv e(\bar{x}) = |x - \bar{x}| \tag{1.3}$$

Por exemplo:

$$\begin{aligned} x = 1/3, \quad \bar{x} = 0{,}333 \quad &\Rightarrow \quad e = |1/3 - 0{,}333| = 0{,}000333\cdots \\ x = \pi, \quad \bar{x} = 3{,}14 \quad &\Rightarrow \quad e = |\pi - 3{,}14| = 0{,}001592\cdots \\ x = 1/3000, \quad \bar{x} = 0 \quad &\Rightarrow \quad e = |1/3000 - 0| = 0{,}000333\cdots \end{aligned}$$

Deve notar-se que:

- a quantidade e nem sempre é calculável, uma vez que pressupõe o conhecimento da quantidade exacta x, nem sempre disponível.
- se x não pode ser representado exactamente na máquina, e também não o poderá ser, em geral.

O *erro relativo* é definido como

$$r \equiv \frac{e}{|x|} = \frac{|x - \bar{x}|}{|x|} \tag{1.4}$$

para $x \neq 0$. Nos exemplos anteriores, tem-se:

$$\begin{aligned} x = 1/3, \quad \bar{x} = 0{,}333 \quad &\Rightarrow \quad r \simeq 0{,}001 \\ x = 1/3000, \quad \bar{x} = 0 \quad &\Rightarrow \quad r \simeq 1 \end{aligned}$$

Como se pode verificar, embora as aproximações anteriores tenham erros absolutos idênticos, os erros relativos correspondentes são bastante diferentes.

Com base na definição de r, é usada também a quantidade *percentagem de erro*, igual a $100r$. O cálculo de r é por vezes difícil, uma vez que a quantidade exacta x aparece nesta expressão. É no entanto possível usar a aproximação:

$$r \simeq \frac{e}{|\bar{x}|}$$

Quando $x \simeq 0$, a definição de erro relativo perde a sua utilidade, e a definição de erro absoluto é mais usada. Para prevenir esta dificuldade no uso do erro relativo, é por vezes utilizada em implementações computacionais uma definição

mista:

$$\epsilon = \frac{|x - \bar{x}|}{1 + |\bar{x}|} \tag{1.5}$$

Esta definição tem as propriedades

$$\begin{aligned} |x| \to \infty \quad &\Rightarrow \quad \epsilon \to r \\ x \to 0 \quad &\Rightarrow \quad \epsilon \to e \end{aligned}$$

assumindo portanto o papel de cada uma das definições anteriores, consoante a magnitude da quantidade em causa.

Problema 1.7 *Complete a seguinte tabela:*

x	$\bar{x}$	e	r	ϵ
e	2,7183			
$1/13$	0,076923			
$\sqrt{200}$	14,142			

1.2.2 Algarismos significativos e arredondamento

Como o próprio nome indica, na representação aproximada de uma quantidade numérica, são designados *significativos* todos os algarismos que expressam informação concreta relativamente a essa quantidade. Em termos práticos são considerados significativos todos os dígitos com potencial excepção do 0, sobretudo quando este é usado à esquerda da quantidade a representar, ou apenas para posicionar a vírgula fraccional. Assim:

- Um zero à esquerda do 1º dígito não nulo não é considerado algarismo significativo. Por exemplo a quantidade 00021 possui 2 algarismos significativos, podendo ser representada sem perda de informação apenas como 21.
- Os zeros usados à esquerda para posicionamento da vírgula fraccional também não são considerados algarismos significativos. Por exemplo $0{,}0271 = 2{,}71 \times 10^{-2}$ possui apenas 3 algarismos significativos; no entanto $0{,}027100 = 2{,}7100 \times 10^{-2}$ já possui 5 algarismos significativos[18].
- Quando subsistir alguma ambiguidade na caracterização do número de algarismos significativos de uma dada quantidade (por exemplo, 1000)

[18]Deve notar-se que estas quantidades apenas podem ser consideradas numericamente equivalentes em contextos muito restritos, por exemplo através da sua caracterização exacta (i.e., com infinita precisão).

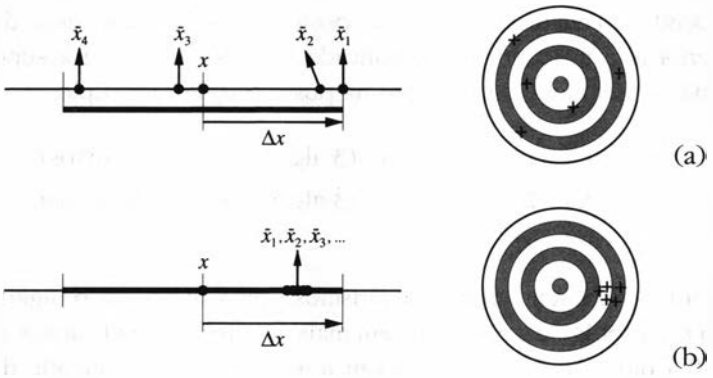


Figura 1.14 Ilustração dos conceitos de exactidão e precisão numéricas. À esquerda, na recta real, x – quantidade exacta; $\bar{x}_i$ – estimativas. À direita é efectuada uma analogia bidimensional, usando um alvo centrado em x: (a) Exactidão comparável com a precisão dos resultados. (b) Precisão superior à exactidão dos mesmos.

pode recorrer-se à notação de vírgula flutuante para suprimir esta ambiguidade. No caso anterior, a quantidade pode escrever-se como $1{,}000 \times 10^3$ no caso de pretendermos denotar 4 algarismos algarismos significativos, ou 1×10^3 no caso de apenas ser usado um algarismo significativo.

A representação de quantidades numéricas incertas pode ser efectuada usando os algarismos significativos disponíveis (ou um subconjunto considerado adequado), caracterizando simultaneamente a incerteza do valor indicado, por exemplo através da forma:

$$x = 4{,}5312 \pm 0{,}0082 \qquad \text{e} \qquad y = 2{,}7 \times 10^3 \pm 1 \tag{1.6}$$

Esta descrição permite individualizar (pelo menos implicitamente) as componentes de *precisão* e de *exactidão* do valor apresentado (Figura 1.14). Claramente o número de algarismos usado pode ser relacionado com a precisão da representação usada (e com a resolução última atingível na caracterização de uma dada quantidade, na ausência de outros erros), enquanto que a incerteza apresentada, ao fixar o raio de um intervalo que contém seguramente o valor numérico em causa, poderá servir como medida da exactidão do valor proposto.

Quando se torna conveniente reduzir o número de algarismos significativos de uma representação, recorre-se ao *arredondamento* desta quantidade. Este processo obedece ao seguinte conjunto de regras, com vista a minimizar o erro cometido:

- O arredondamento de uma quantidade para n algarismos significativos

depende do valor do dígito na posição $n+1$. Caso esse dígito seja inferior a 5, a quantidade é truncada na posição n; se for superior a 5, soma-se uma unidade ao dígito na posição n. Por exemplo:

$$4{,}5312 \rightarrow 4{,}53 \quad \text{(3 algarismos significativos)}$$
$$2{,}5362 \rightarrow 2{,}54 \quad \text{(3 algarismos significativos)}$$

- No arredondamento para n algarismos significativos, se o dígito na posição $n+1$ for igual a 5, e existirem mais algarismos significativos diferentes de 0 à direita do dígito de ordem $n+1$, então a quantidade deverá ser arredondada somando 1 unidade ao dígito na posição n[19]. Assim:

$$4{,}25003 \rightarrow 4{,}3 \quad \text{(2 algarismos significativos)}$$

Caso o dígito na posição $n+1$ seja igual a 5, sendo simultaneamente o último dígito diferente de zero, então a quantidade poderia ser arredondada na posição n tanto para cima como para baixo, com erro idêntico. Por exemplo:

$$4{,}15 \rightarrow 4{,}1 \text{ (ou } 4{,}2)$$

Nesta situação opta-se por arredondar as quantidades terminadas em 5 em metade dos casos para cima (estatisticamente), e nas restantes vezes para baixo. Isto pode ser conseguido, por exemplo, usando a *regra par*: se o dígito na posição n for par o arredondamento é efectuado para baixo, e caso seja ímpar o arredondamento é efectuado para cima[20]. Assim:

$$1{,}235 \rightarrow 1{,}24 \quad \text{(3 algarismos significativos)}$$
$$2{,}305 \rightarrow 2{,}30 \quad \text{(3 algarismos significativos)}$$

Exemplos adicionais de aplicação destes conceitos podem ser encontrados em Lemos Ferreira (1999).

É importante notar que, para além da sua verbosidade, o processo ilustrado por (1.6) pode conduzir, por si só, a uma escolha pouco adequada do número de algarismos usados. No 1º caso anterior tem-se

$$x = 4{,}5312 \pm 0{,}0082 \quad \Rightarrow \quad x \in [4{,}5230; 4{,}5394]$$

[19]De forma análoga à utilização do "sticky bit" na norma IEEE 754.

[20]Um outro método mais simples consiste em truncar a quantidade na posição n se o dígito na posição seguinte for 0, 1, 2, 3 ou 4, e adicionar uma unidade ao dígito na posição n caso o dígito da posição seguinte seja 5, 6, 7, 8 ou 9. As vantagens do sistema da *regra par* sobre esta solução são analisadas por Goldberg (1991).

subsistindo dúvidas quando à veracidade dos algarismos (significativos) usados a partir da 2ª casa decimal (inclusive). No 2º caso fica

$$y = 2{,}7 \times 10^3 \pm 1 \quad \Rightarrow \quad y \in [2709; 2711]$$

sendo possível inferir que a quantidade representada deveria ter sido expressa com uma precisão superior, uma vez que pode ser correctamente arredondada para $y = 2{,}71 \times 10^3$ (portanto com mais um algarismo significativo do que os usados).

Para evitar estas dificuldades, usando simultaneamente uma notação mais compacta, as quantidades numéricas são habitualmente expressas de forma a apresentarem *todos os seus algarismos significativos correctos, com potencial excepção do último* que, embora correspondendo à melhor estimativa disponível, é usualmente incerto.

Deste modo, as quantidades mencionadas em (1.6) poderão ser expressas apenas como:

$$x = 4{,}53 \qquad \text{e} \qquad y = 2{,}71 \times 10^3$$

1.2.3 Algarismos significativos e casas decimais correctos

Os conceitos anteriores de *erros* podem ser usados para a caracterização de algarismos significativos e casas decimais *correctos* numa dada aproximação numérica. Assim, temos:

Casas decimais correctas: Uma aproximação $\bar{x}$ de x tem k casas decimais correctas se $|x - \bar{x}| \leq 0{,}5 \times 10^{-k}$.

Algarismos significativos correctos: Se

$$|x - \bar{x}|/|x| \leq 0{,}5 \times 10^{-k}$$

então $\bar{x}$ é uma aproximação de x com k algarismos significativos correctos. Portanto, um algarismo significativo é *correcto* se o arredondamento do número aproximado depois desse dígito corresponder a um erro absoluto inferior a $1/2$ na posição daquele dígito.

Por exemplo

$$|\pi - 22/7| = 0{,}00126\cdots \leq 0{,}5 \times 10^{-2}$$

e logo $22/7 = 3{,}14286\cdots$ constitui uma aproximação de $\pi = 3{,}14159\cdots$ com 2 casas decimais correctas. Também

$$|\pi - 22/7|/|\pi| = 4{,}025\cdots \times 10^{-4} \leq 0{,}5 \times 10^{-3}$$

e portanto a aproximação $22/7$ possui 3 algarismos significativos correctos. Estas definições são úteis no arredondamento dos resultados obtidos, tendo em atenção a precisão dos dados de entrada, e as formas possíveis de propagação de erros durante o processo de cálculo.

Fórmulas alternativas que relacionam os vários tipos de erros com o número de algarismos significativos e de casas decimais correctos podem ser encontradas em Conte e de Boor (1981) e Rosa (1988).

1.2.4 Fórmula de Taylor

A ferramenta básica na análise de propagação de erros é o desenvolvimento em série de Taylor. Para uma função $f(x)$ univariável tem-se, caso $f(x)$ possua derivadas contínuas até à ordem $n+1$, num intervalo $[a,b]$

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x) \qquad (1.7)$$

para $x, x_0 \in [a,b]$. O termo $R_n(x)$ é designado por *resto de ordem* n, podendo ser expresso de vários modos. Para as aplicações aqui consideradas é suficiente usar a forma de Lagrange:

$$R_n(x) = \frac{f^{(n+1)}(\eta)}{(n+1)!}(x - x_0)^{n+1}, \qquad \eta \in [\min\{x, x_0\}, \max\{x, x_0\}] \qquad (1.8)$$

Esta expressão é aplicável a quaisquer pontos $x, x_0 \in [a,b]$, desde que a derivada $f^{(n+1)}(\eta)$ se encontre definida no intervalo correspondente. $R_n(x)$ poderá ser exactamente zero para funções polinomiais a partir de uma determinada ordem. No caso de funções "bem comportadas" (i.e., com comportamento aproximável por funções polinomiais), este termo pode também assumir magnitudes bastante pequenas para n suficientemente elevado, passando o segundo membro da equação (1.7) sem o termo $R_n(x)$

$$f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

a constituir uma aproximação razoável de $f(x)$. Frequentemente, é também usada a notação $R_n(x) = O(h^{n+1})$, com $h = x - x_0$, pretendendo-se evidenciar o facto de que a magnitude deste termo depende da distância entre estes dois

pontos, elevada à potência apropriada[21]. A fórmula de Taylor com resto de 1ª ordem

$$f(x) = f(x_0) + f'(\eta)(x - x_0), \qquad \eta \in [\min\{x, x_0\}, \max\{x, x_0\}] \tag{1.9}$$

é bastante utilizada, sendo este resultado também conhecido como *teorema do valor médio*.

1.2.5 Fórmula básica de propagação de erros

A fórmula geral de propagação de erros pode ser deduzida com relativa facilidade para o caso de uma função de duas variáveis $f(x,y)$. O método usado pode depois ser generalizado sem dificuldade a qualquer outra função envolvendo um número arbitrário de variáveis.

Supondo que estão disponíveis duas aproximações $\bar{x}$ e $\bar{y}$ para as quantidades x e y, o objectivo deste exercício consiste em verificar qual o erro correspondente induzido em $f(\bar{x},\bar{y})$, relativamente a $f(x,y)$. Usando uma generalização de (1.7) para funções de duas variáveis, e admitindo que $f(x,y)$ e as suas primeiras derivadas são contínuas, tem-se

$$f(x,y) = f(\bar{x},\bar{y}) + \left.\frac{\partial f(x,y)}{\partial x}\right|_{(\eta_1,\eta_2)} \cdot (x - \bar{x}) + \left.\frac{\partial f(x,y)}{\partial y}\right|_{(\eta_1,\eta_2)} \cdot (y - \bar{y})$$

com:

$$\eta_1 \in [\min\{x,\bar{x}\}, \max\{x,\bar{x}\}], \qquad \eta_2 \in [\min\{y,\bar{y}\}, \max\{y,\bar{y}\}]$$

Tendo em atenção a *desigualdade triangular*

$$|a + b| \leq |a| + |b| \tag{1.10}$$

e a identidade

$$|a \cdot b| = |a| \cdot |b| \tag{1.11}$$

para $a, b \in \mathbb{R}$, vem:

$$\underbrace{|f(x,y) - f(\bar{x},\bar{y})|}_{e_f} \leq \left|\frac{\partial f(\eta_1,\eta_2)}{\partial x}\right| \cdot \underbrace{|x - \bar{x}|}_{e_x} + \left|\frac{\partial f(\eta_1,\eta_2)}{\partial y}\right| \cdot \underbrace{|y - \bar{y}|}_{e_y}$$

A utilização desta equação está limitada pelo facto de, em geral, o ponto (η_1,η_2) não ser conhecido, e portanto as suas derivadas não poderem ser calculadas.

[21] A notação $O(\cdot)$ — "ordem de" — identifica os termos assimptoticamente dominantes numa expressão matemática.

No entanto, em muitas situações concretas, é possível avaliar os limites M_x, M_y de variação das derivadas de f neste intervalo:

$$M_x = \sup_{\eta_1,\eta_2} \left| \frac{\partial f(\eta_1,\eta_2)}{\partial x} \right|, \qquad M_y = \sup_{\eta_1,\eta_2} \left| \frac{\partial f(\eta_1,\eta_2)}{\partial y} \right| \tag{1.12}$$
$$\eta_1 \in [\min\{x,\bar{x}\}, \max\{x,\bar{x}\}], \quad \eta_2 \in [\min\{y,\bar{y}\}, \max\{y,\bar{y}\}]$$

Substituindo, por majoração, estas quantidades na equação prévia, resulta:

$$e_f \leq M_x \cdot e_x + M_y \cdot e_y \tag{1.13}$$

Em alternativa à utilização de (1.13), para $\bar{x}$ e $\bar{y}$ suficientemente próximos de x e y, é possível por vezes aproximar:

$$M_x \simeq \frac{\partial f(\bar{x},\bar{y})}{\partial x} \qquad \text{e} \qquad M_y \simeq \frac{\partial f(\bar{x},\bar{y})}{\partial y}$$

Neste caso, a equação (1.13) pode ser *aproximada* por

$$e_f \leq \left| \frac{\partial f(\bar{x},\bar{y})}{\partial x} \right| \cdot e_x + \left| \frac{\partial f(\bar{x},\bar{y})}{\partial y} \right| \cdot e_y \tag{1.14}$$

requerendo apenas o conhecimento das derivadas no ponto base $(\bar{x},\bar{y})$. Esta expressão indica que no caso mais desfavorável os erros cometidos na especificação dos valores de entrada de $f(x,y)$ surgem, *de forma aditiva*, na saída. Os erros em cada um dos termos são "pesados" através de uma constante de proporcionalidade, que corresponde à derivada parcial (ou *sensibilidade*) da função em relação a essa variável. Isto significa que quanto mais sensível for uma função em relação a uma sua variável, maior será a propagação do erro cometido na especificação dessa variável para a saída.

Uma expressão semelhante pode ser obtida para o erro relativo de f, obtendo-se neste caso:

$$r_f \simeq \frac{e_f}{|f(\bar{x},\bar{y})|} \leq \frac{\left|\frac{\partial f(\bar{x},\bar{y})}{\partial x}\right|}{|f(\bar{x},\bar{y})|} e_x + \frac{\left|\frac{\partial f(\bar{x},\bar{y})}{\partial y}\right|}{|f(\bar{x},\bar{y})|} e_y$$

Problema 1.8 *Usando uma análise semelhante à anterior, deduza a fórmula básica de propagação de erros relativa à determinação de uma quantidade escalar z a partir das variáveis x e y também escalares, quando estas estão relacionadas através de uma função* implícita $g(x,y,z) = 0$.

Problema 1.9 *Considere o problema inverso na propagação dos erros, no cálculo do volume de um reservatório cilíndrico: Usando a fórmula* $V = \pi D^2 L/4$,

determine as incertezas máximas em D e L (em termos dos seus erros absolutos), que permitem garantir que V é calculado com um erro relativo máximo de $0{,}5 \times 10^{-3}$. *Utilize* $D = 0{,}5$ m *e* $L = 1{,}7$ m, *e suponha comparáveis as incertezas* absolutas *na especificação destas 2 variáveis.*

1.2.6 Condicionamento e estabilidade

Quando se pretende avaliar a origem dos erros observados na resolução numérica de um dado problema, é também importante distinguir entre a *sensibilidade* do próprio problema (por exemplo, face a pequenas variações nos dados de entrada), e a *estabilidade* do algoritmo numérico usado para o resolver.

Condicionamento de problemas numéricos

O condicionamento numérico descreve a *sensibilidade de um problema face a variações nos valores dos seus dados.* Este conceito pode ser ilustrado através do seguinte exemplo:

Exemplo 1.4 (Pontos de intersecção de duas rectas)

Uma interpretação possível para a resolução do sistema de equações

$$\begin{cases} x_1 + 2x_2 = 2 \\ x_1/2 + (1+\epsilon)x_2 = 4{,}5 \end{cases} \tag{1.15}$$

corresponde à localização do ponto de intersecção de duas rectas, em $\mathbb{R}^2$. O determinante deste sistema é ϵ, sendo fácil observar que se trata de um problema *mal condicionado* (i.e., extremamente sensível a variações em ϵ), à medida que este parâmetro tende para zero (Figura 1.15). Por exemplo, quando ϵ varia entre 0,01 e 0,02 a localização deste ponto sofre uma variação de cerca de 50% no valor das suas coordenadas. O mau condicionamento deste problema linear reflecte-se também na matriz de coeficientes correspondentes, que se torna singular para $\epsilon = 0$.

É importante podermos dispor de uma medida quantitativa da condição de problemas numéricos, sobretudo tendo em vista o diagnóstico de problemas mal-condicionados, como o exemplo anterior. Isto pode ser conseguido equacionando o problema em causa à avaliação de uma função escalar $f(x)$ num ponto x_0 pré-determinado. Se $f(x)$ satisfizer as condições de aplicabilidade de (1.7), ao considerarmos ligeiras perturbações no valor de entrada x_0, é possível

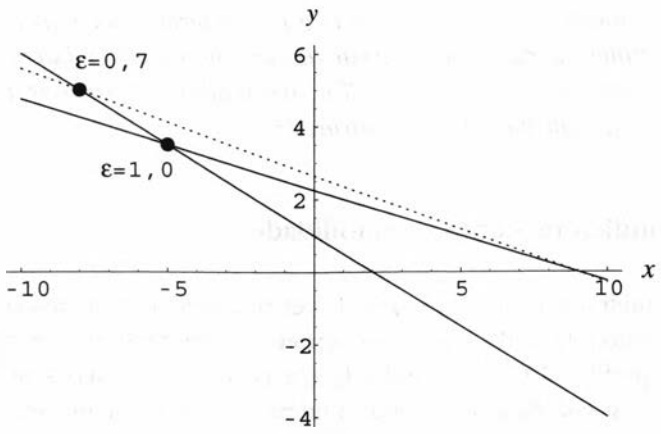


Figura 1.15 Pontos de intersecção das rectas (1.15), para vários valores de ϵ.

escrever a relação

$$f(x_0 + \Delta x) - f(x_0) = f'(\xi)\Delta x$$

usando o teorema do valor médio, com $\xi \in [\min\{x_0, x_0 + \Delta x\}, \max\{x_0, x_0 + \Delta x\}]$. Para Δx pequeno pode considerar-se

$$f(x_0 + \Delta x) - f(x_0) \simeq f'(x_0)\Delta x \tag{1.16}$$

de onde se obtém, evidenciando os termos dos erros relativos:

$$\underbrace{\left(\frac{f(x_0 + \Delta x) - f(x_0)}{f(x_0)}\right)}_{r_{f_0}} \simeq \frac{\Delta x f'(x_0)}{f(x_0)} = \underbrace{\left(\frac{x_0 f'(x_0)}{f(x_0)}\right)}_{\kappa(x_0)} \underbrace{\left(\frac{\Delta x}{x_0}\right)}_{r_{x_0}}$$

O factor da equação anterior

$$\kappa(x) \equiv \frac{x f'(x)}{f(x)} \tag{1.17}$$

pode ser interpretado como o *índice de condição* do problema correspondente à avaliação de $f(x)$, uma vez que permite quantificar a propagação das incertezas nos dados da entrada para o resultado, na ausência de outros erros durante o processo de cálculo. Problemas *mal-condicionados* estão associados a valores elevados deste índice.

O conceito de condicionamento descreve *apenas* a sensibilidade da formulação, não dependendo por isso do método de resolução usado. Contudo, para resolver um problema bem-condicionado podem ser propostos vários algoritmos

que, embora equivalentes do ponto de vista matemático (quando as operações envolvidas são efectuadas com precisão infinita), possuem diferentes graus de *estabilidade numérica*, depois de implementados computacionalmente.

Problema 1.10 *Usando a expressão* (1.17)*, avalie a condição do problema descrito no exemplo anterior em função de* ϵ *(efectue separadamente os cálculos para cada uma das coordenadas* x *e* y *da solução).*

Estabilidade de algoritmos

Este conceito descreve a sensibilidade dos algoritmos relativamente à acumulação de erros ocorridos durante o processo de cálculo. Consoante a sua sensibilidade, os métodos numéricos podem ser classificados como *estáveis* ou *instáveis*. Uma propriedade desejável para a maioria dos algoritmos é a sua *estabilidade regressiva* (Stewart, 1973):

> Designando por $f_{\text{calc}}(x)$ uma implementação computacional para o cálculo de $f(x)$, então $f_{\text{calc}}(x)$ é descrito como um algoritmo com *estabilidade regressiva* se, para todos os argumentos x, existir um valor "pequeno" Δx tal que:
>
> $$f_{\text{calc}}(x) = f(x + \Delta x)$$

Isto corresponde a dizer que $f_{\text{calc}}(x)$ corresponderá sempre à solução *exacta* (devido à igualdade na equação anterior) de um problema com características ligeiramente diferentes das pretendidas (atendendo a $x + \Delta x$).

A propriedade anterior permite estabelecer facilmente limites para o erro das quantidades pretendidas, uma vez que

$$|f_{\text{calc}}(x) - f(x)| = |f(x + \Delta x) - f(x)| \simeq |f'(x)||\Delta x|$$

usando um raciocínio semelhante a (1.16). Desta forma, se $f_{\text{calc}}(x)$ tiver estabilidade regressiva, $|\Delta x|$ na equação anterior terá sempre um valor pequeno. O erro final correspondente será também pequeno, a menos que a condição do problema (relacionada, como se viu, com $|f'(x)|$) seja *elevada*. É possível portanto afirmar que *um método dito estável deverá produzir sempre bons resultados* (com erros reduzidos), *em problemas razoavelmente bem condicionados*.

Devido à forma de propagação dos erros numéricos, métodos equivalentes do ponto de vista teórico podem produzir, na prática, soluções bastante diferentes, como ilustra o exemplo seguinte.

Exemplo 1.5 (Raízes de polinómios de segundo grau)

As operações de subtracção de quantidades de magnitude quase idênticas constituem uma fonte especial de erros, devido à perda de precisão envolvida. Por exemplo, numa calculadora com 7 dígitos, ao subtrairmos 1,000007 – 1,000002 obtemos o resultado de 5×10^{-5}. Deve ser notado que embora os valores iniciais possuam 7 dígitos significativos, o resultado só possui um único dígito significativo! Sempre que possível, estas ocorrências devem ser evitadas, devido à perda de precisão envolvida. Muitas vezes é possível rearranjar o método de cálculo por forma a evitar estas situações.

Um exemplo prático deste comportamento ocorre na fórmula resolvente para equações polinomiais de 2ª ordem. Se $ax^2+bx+c=0$ for a equação a resolver (com $c \neq 0$), a fórmula usada para determinar as soluções tem a forma habitual:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.18}$$

No entanto, se b^2 for muito superior em magnitude a $4ac$, $\sqrt{b^2 - 4ac} \simeq |b|$, e a fórmula anterior fica, num dos casos

$$x_2 = \frac{-b+b}{2a} \simeq 0$$

com a consequente perda de precisão. Torna-se preferível neste caso determinar primeiro uma raiz usando

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

no caso de b ser positivo, ou

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

se $b < 0$. A determinação da raiz restante pode ser efectuada reconhecendo que $c/a = x_1 x_2$, de onde resulta:

$$x_2 = \frac{c}{ax_1} \qquad \text{ou} \qquad x_1 = \frac{c}{ax_2}$$

Este método alternativo pode classificar-se como mais estável que a fórmula clássica (1.18).

Problema 1.11 *Determine as raízes de uma equação de 2º grau, com* $a = 1$, $b = 100$, *e* $c = 2$, *usando cada um dos métodos anteriores. Compare a exactidão das soluções obtidas, substituindo os seus valores na equação original.*

Problema 1.12 *Efectue a análise da propagação de erros na fórmula resolvente de equações de segunda ordem, para ambos os casos, com* $a = 1$, $b = 100$ *e* $c = 2$. *A sensibilidade é idêntica para todos os coeficientes?*

1.2.7 Exemplo de aplicação: aproximação de derivadas

Uma tarefa frequente em problemas de engenharia consiste na aproximação de derivadas de funções. Em muitos casos as funções cujas derivadas se tornam necessárias são complexas, envolvendo cálculos complicados ou morosos. Neste casos é possível o recurso a sistemas de diferenciação automática, ou ainda à aproximação das derivadas, sobretudo em situações que requeiram apenas uma estimativa da mesma, como é o caso de muitos problemas de solução de equações ou de optimização.

Usando a definição, a derivada de uma função escalar $f(x)$ em $x = a$ pode, por exemplo, ser aproximada por uma diferença finita *progressiva*:

$$f'(a) \simeq \frac{f(a+h) - f(a)}{h} \tag{1.19}$$

Para $|h|$ arbitrariamente pequeno, o erro cometido será, em princípio, diminuto, uma vez que esta expressão coincide com a própria definição de derivada. No entanto, isto só é rigorosamente verdade quando é usada aritmética de precisão infinita. Em aritmética finita, se a perturbação h for demasiado pequena, corre-se o risco de termos $|f(a+h) - f(a)| < \epsilon_{\text{mach}}|f(a)|$, obtendo-se neste caso $f'(a) = 0$, para qualquer função f. Mesmo que esta situação não se verifique, podemos incorrer facilmente em dificuldades semelhantes às descritas no Exemplo 1.5, ao subtrairmos duas quantidades de magnitudes semelhantes.

Por outro lado, quanto maior for $|h|$ mais grosseira será a aproximação (1.19). Coloca-se portanto a questão de saber se existe um incremento h^* óptimo a ser usado em expressões deste tipo, atendendo a estes dois factores.

Torna-se necessário reconhecer que na prática é impossível garantir que o valor exacto de $f(a)$ seja obtido no decurso do processo de cálculo. Mesmo que isso aconteça, não é possível garantir que este valor seja exactamente representável na máquina usada. Assim, em vez desta quantidade exacta, poderemos estar a trabalhar com uma quantidade próxima $f_{\text{calc}}(a)$, relacionada com a quantidade anterior através de:

$$f_{\text{calc}}(a) = f(a) \pm e_a = f(a)(1 \pm r_a) \tag{1.20}$$

Nesta expressão, e_a representa o erro *absoluto* na determinação de $f(a)$ e r_a o respectivo erro *relativo*, calculado de acordo com (1.4). O termo r_a depende

fundamentalmente da natureza de $f(x)$. Para funções simples, este pode ser associado sobretudo à propagação dos erros de arredondamento cometidos no cálculo dos vários termos de $f(x)$, sendo consequentemente diminuto. Um limite inferior para esta quantidade é o *erro máximo de arredondamento* ϵ_{mach}, característico da precisão usada, independente portanto do ponto $x = a$ em que é avaliado. No entanto, para funções $f(x)$ mais complexas (envolvendo nomeadamente a aplicação de outros métodos numéricos) este termo pode ser considerável, dependendo por exemplo das tolerâncias impostas nas diferentes etapas do cálculo de $f(x)$.

O erro correspondente à aproximação de $f'(a)$ por (1.19) representa um *erro de truncatura*, uma vez que esta expressão pode ser obtida a partir de uma série de Taylor truncada após o termo de 1ª ordem. O erro correspondente pode ser estimado a partir de

$$f(a+h) = f(a) + f'(a)h + \frac{f''(\eta)}{2}h^2 \tag{1.21}$$

onde $\eta \in [a, a+h]$ (supondo h positivo). De (1.21) vem

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \underbrace{\frac{h}{2}f''(\eta)}_{e_t}$$

em que e_t representa o erro de truncatura de (1.19).

Ao implementarmos a aproximação (1.19) num computador, estamos de facto a lidar com a quantidade

$$\frac{f_{\text{calc}}(a+h) - f_{\text{calc}}(a)}{h} = \frac{f(a+h) \pm e_a - f(a) \pm e_a}{h} =$$

$$= \frac{f(a+h) - f(a)}{h} \pm \frac{2e_a}{h}$$

supondo que h é escolhido de forma a poder ser representável exactamente no computador (recordando as considerações da Secção 1.1.2). Com este objectivo é comum encontrar fragmentos de código semelhantes a

```
xh = x + h
h = xh - x
```

na aproximação de derivadas, embora este tipo de soluções nem sempre produza os resultados desejados (Press et al., 1992). Deste modo fica

$$f'(a) = \frac{f(a+h) - f(a)}{h} \underbrace{\pm \frac{2e_a}{h} - \frac{h}{2}f''(\eta)}_{E_T(h)} \tag{1.22}$$

onde $E_T(h)$ designa o erro total da aproximação (1.19). O objectivo restante deste exercício consiste na escolha de um valor óptimo de h que minimize o valor absoluto deste erro.

Quando $h \to 0$, o termo $2e_a/h$ cresce bastante em magnitude. Quando $|h| \to \infty$, o termo $hf''(\eta)/2$ torna-se em geral dominante. Para isto é apenas necessário admitir que $f''(\eta) \neq 0$, e que $|f''(\eta)|$ não decresce mais rapidamente do que $1/h^2$, com o aumento da amplitude do intervalo[22]. O comportamento observado neste caso é exemplificado na Figura 1.16. Deverá portanto existir um valor intermédio de h, denominado h^*, que minimize E_T. Este valor é obtido calculando o seu ponto de estacionaridade, onde a primeira derivada desta função em ordem a h se anula. Assim, considerando a soma dos valores absolutos dos dois termos[23]:

$$\min_{|h|} E_T = \frac{2e_a}{|h|} + \frac{|h|}{2}|f''(\eta)| \quad \Rightarrow \quad \boxed{\left.\frac{dE_T}{d|h|}\right|_{|h|^*} = 0}$$

Para simplificar a derivada da expressão anterior, é conveniente supor que a magnitude da 2ª derivada não se altera significativamente com o tamanho do intervalo escolhido, e portanto com o ponto η onde é calculada. Neste caso, a estimativa de $|h|^*$ vem:

$$-\frac{2e_a}{h^{*2}} + \frac{|f''(\eta)|}{2} = 0 \quad \Rightarrow \quad |h|^* = \sqrt{\frac{4e_a}{|f''(\eta)|}} \tag{1.23}$$

No ponto $|h|^*$, a 2ª derivada de $E_T(h)$ é positiva, confirmando tratar-se de um mínimo da função.

Em muitas situações, a quantidade $f''(\eta)$ é difícil de avaliar, dado tratar-se de uma 2ª derivada, e o problema em causa ser estimar a 1ª derivada. Várias soluções podem ser adoptadas para tornar o resultado anterior directamente utilizável:

- No caso de funções "bem comportadas", e na ausência de informação adicional sobre a 2ª derivada, é possível admitir magnitudes semelhantes

$$|f''(\eta)| \simeq |f(a)| \tag{1.24}$$

no intervalo $x \in [a, a+h]$[24]. Neste caso, devido a (1.20), a equação

[22] Estas condições são pouco restritivas, sendo verificadas num conjunto muito vasto de situações de natureza prática.

[23] Por conveniência é usado o valor de $|h|$ em vez de h, permitindo que esta expressão seja também válida para $h < 0$. Neste caso ter-se-ia $\eta \in [a+h, a]$.

[24] Uma forma simples de procurar atingir este comportamento, embora sem garantias formais, é *normalizar* previamente $f(x)$, de modo a que tanto os valores de x como de f sejam próximos da unidade, no domínio do problema.

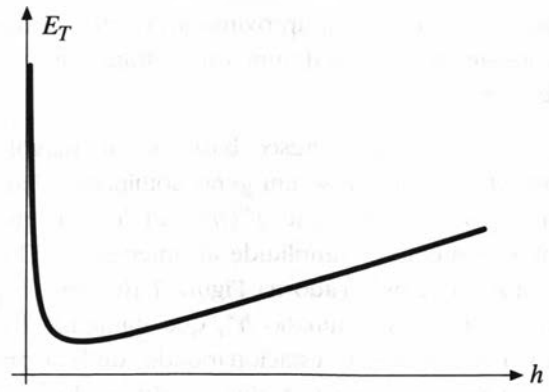


Figura 1.16 Comportamento típico de E_T em (1.22).

anterior pode ser simplificada para:

$$|h|^* \simeq 2\sqrt{r_a} \tag{1.25}$$

Este resultado indica que para um erro mínimo na estimativa da derivada, a amplitude da perturbação em (1.19) deve abranger cerca de *metade dos algarismos significativos de* $f(a)$, sendo a precisão da estimativa obtida da mesma ordem de magnitude deste valor. No caso mais favorável correspondente ao modelo de representação IEEE 754, quando $r_a \simeq \epsilon_{\text{mach}}$ (i.e., o erro máximo no cálculo de $f(x)$ é semelhante ao erro de arredondamento da própria representação), a expressão anterior aponta como valores indicados para o tamanho da perturbação:

$$|h|^* \simeq \begin{cases} 10^{-4} & \text{(precisão simples)} \\ 10^{-8} & \text{(precisão dupla)} \end{cases}$$

Deve ser realçado que os valores obtidos dizem apenas respeito a estimativas da 1ª derivada calculadas através de (1.19), quando a aproximação (1.24) for razoável. Se for usada outra expressão para aproximar a 1ª derivada, a dedução anterior deve ser repetida, podendo conduzir a valores de $|h|^*$ significativamente diferentes.

- Se a aproximação (1.24) não puder ser efectuada, então a equação (1.23) pode ser escrita na forma

$$|h|^* \simeq 2x_c\sqrt{r_a}$$

onde $x_c = \sqrt{|f(a)/f''(\eta)|}$, com as mesmas unidades físicas que a, representa um "comprimento característico" ou uma "escala de curvatura característica" da função $f(x)$ (Press et al., 1992). Este facto justifica a sugestão, encontrada por vezes, de escolher $h^* \simeq a\sqrt{r_a}$ para valor da amplitude da perturbação, na ausência de outra informação sobre $f(x)$. Em termos práticos, esta abordagem pode ser implementada estabelecendo (Kelley, 2003):

$$h^* = \max\{|a|, 1\}\sqrt{r_a}\,\mathrm{sig}(a) \tag{1.26}$$

Nesta expressão, o operador sig assume os valores $+1$ ou -1 consoante o sinal algébrico de a, sendo este definido a partir de:

$$\mathrm{sig}(x) = \begin{cases} x/|x|, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

A sua inclusão na equação anterior permite que sejam considerados valores superiores em magnitude (i.e., mais negativos) para $x + h$, com argumentos $x < 0$, o que faz parte da definição de diferenças finitas progressivas[25]. O operador max em (1.26) é usado neste caso com um objectivo semelhante à abordagem anterior: perturbar, na maioria dos casos, cerca de metade dos algarismos significativos de $f(x)$ no numerador de (1.19), com vista a obter a máxima precisão na estimativa efectuada. No entanto, isto apenas ocorrerá se $x_c \simeq |a|$, o que não é garantido na generalidade dos casos.

- Finalmente é possível adoptar procedimentos sistemáticos de estimativa da amplitude óptima da perturbação, tentando quantificar directamente o erro total da aproximação (1.19). Isto requer geralmente o cálculo de $f(x)$ em mais pontos na vizinhança de $x = a$. Abordagens deste tipo podem ser encontradas em Gill et al. (1981, Secção 8.6.2; 1983); Press et al. (1992); Herstine (1998).

Problema 1.13 *Considere a aproximação numérica da 1ª derivada do polinómio $f(x) = x^5 - x - 500$, no ponto $x = 3$. A Figura 1.17 representa o erro relativo r obtido na aproximação de $f'(x)$ pela aplicação de* (1.19)*, supondo que os valores de $f(x)$ são obtidos com 5 algarismos significativos, e que os restantes cálculos são efectuados com precisão dupla, de acordo com a norma IEEE 754. Com base nesta informação, compare as estimativas correspondentes a este caso:*

1. *Para os valores de h^* produzidos pelos diferentes métodos considerados*

[25] A influência da sua presença em (1.26) é portanto diminuta; a análise anterior permanece válida se o operador sig não fosse usado, embora neste caso, tecnicamente, pudessem estar a ser usadas diferenças finitas *regressivas* para alguns valores de x.

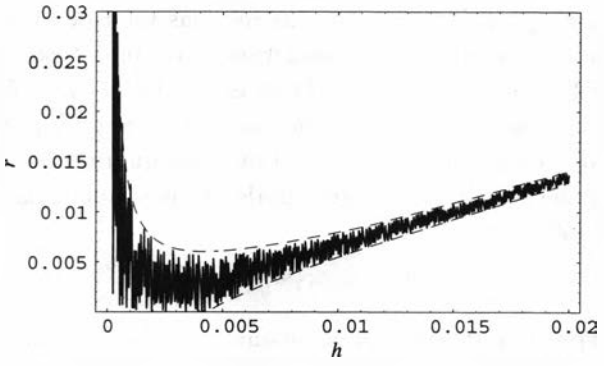


Figura 1.17 Erro relativo na aproximação da primeira derivada, no Problema 1.13. A curva representada a cheio foi obtida por amostragem desta quantidade para valores discretos de h, o que explica a sua natureza fortemente oscilatória. As curvas a tracejado representam duas possíveis envolventes para os valores máximo e mínimo deste erro.

anteriormente.

2. *Para o erro total mínimo de aproximação de $f'(x)$ através de* (1.19). *Quantos algarismos significativos podem ser esperados neste valor, no caso mais favorável?*

NOTA: Detalhes adicionais relativos à minimização de erros numéricos na manipulação de funções polinomiais são considerados na Secção 2.11.3.

Neste exemplo de aplicação, foi apenas usado um valor adicional da função para aproximar a sua derivada. Quando diversos valores de $f(x)$ estão disponíveis na vizinhança de $x = a$, ou quando o cálculo de $f(x)$ não é moroso, encontram-se disponíveis outros métodos, mais precisos, para aproximar as derivadas desta função (usando, por exemplo, outros tipos de diferenças finitas).

Problemas semelhantes

Dificuldades semelhantes às analisadas nesta Secção pode ocorrer também no cálculo da soma de séries convergentes. Por exemplo, na ausência de bibliotecas que implementem funções especiais, a série de Taylor (1.7) pode ser usada para aproximar funções transcendentes, como as funções $\exp(x)$ ou $\sin(x)$. No entanto, para além de eventuais erros de cancelamento, que podem ocorrer na soma de termos de magnitudes semelhantes mas de sinais opostos (como considerado no Exemplo 1.5), a precisão desta metodologia pode ser limitada pela acumulação de erros de arredondamento.

Quando são considerados poucos termos na soma da série, o erro total da aproximação é dominado pelo erro de truncatura cometido. A inclusão de mais termos na soma pode no entanto conduzir à acumulação de erros de arredondamento, não se tornando o resultado necessariamente mais preciso à medida que são adicionados novos termos. Estas observações conduzem nalguns casos à existência de um número "óptimo" de termos que devem ser usados na aproximação destas séries (Rice, 1993; Overton, 2001), à semelhança da escolha do incremento h no exemplo anterior.

Problema 1.14 *Considere novamente a* série harmónica, *analisada no Exemplo 1.3. Admita neste exemplo ignorar que esta série é* divergente, *i.e.*

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} = \infty$$

e experimente escrever um programa, numa linguagem à sua escolha que implemente o standard IEEE 754, para calcular a soma desta série. Utilize como critério de paragem o teste da soma não ser alterada pela adição de novos termos, somando os termos por ordem crescente do índice i (ignorando portanto os problemas de precisão deste método, analisados no Exemplo 1.3).

Para tornar a análise mais simples, implemente todos os cálculos em precisão simples. Acha que a execução deste programa vai terminar num intervalo de tempo finito? Se terminar sem overflow, *que relação existirá entre o valor calculado para a soma e o primeiro termo desprezado?*

1.3 Normas vectoriais e matriciais

A análise do comportamento dos métodos numéricos estudados nos Capítulos seguintes requer a definição do conceito de normas de vectores e matrizes, brevemente abordado nesta Secção.

1.3.1 Normas vectoriais

No espaço vectorial $\mathbb{R}^n$, uma *norma* é qualquer função $f : \mathbb{R}^n \to \mathbb{R}$ que verifica as seguintes condições:

1. $f(x) > 0, \quad \forall x \in \mathbb{R}^n \neq 0.$
2. $f(x) = 0 \Leftrightarrow x = 0.$
3. $f(\alpha x) = |\alpha| f(x), \quad \forall \alpha \in R, \ \forall x \in \mathbb{R}^n.$
4. $f(x + y) \leq f(x) + f(y), \quad \forall x, y \in \mathbb{R}^n$ (desigualdade triangular).

O significado físico de uma norma $f(x)$ coincide pois com o *tamanho* ou *comprimento* do vector x. As infinitas normas que se podem definir, respeitando as condições anteriores, correspondem aos vários "sistemas de medida" possíveis desse tamanho. A desigualdade triangular (propriedade 4) é idêntica à propriedade escalar usada em (1.10).

A norma mais comum é a *norma Euclidiana*:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Esta norma é um caso particular da família de *normas-p*:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Como casos especiais nesta família temos:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \qquad \text{e} \qquad \|x\|_\infty = \max_{1 \le i \le n} |x_i|$$

Um vector *unitário* relativamente à norma $\|\cdot\|$ possui $\|x\| = 1$. Na Figura 1.18 são ilustrados os contornos delimitados pelas extremidades de todos os vectores unitários em várias normas-p no espaço $\mathbb{R}^2$.

1.3.2 Normas matriciais

As normas matriciais podem ser definidas por analogia às normas vectoriais. É no entanto preferível introduzir as normas matriciais *induzidas* ou *subordinadas* através da definição

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

para qualquer norma vectorial $\|\cdot\|$. Neste caso, a matriz $A \in \mathbb{R}^{m \times n}$ pode ser vista como um operador linear, cuja aplicação através de $y = Ax$ produz valores $y \in \mathbb{R}^m$ a partir de quantidades $x \in \mathbb{R}^n$. De acordo com a definição anterior, a norma de A corresponderá à *norma máxima* dos valores possíveis de saída y, quando este operador é aplicado a vectores x unitários.

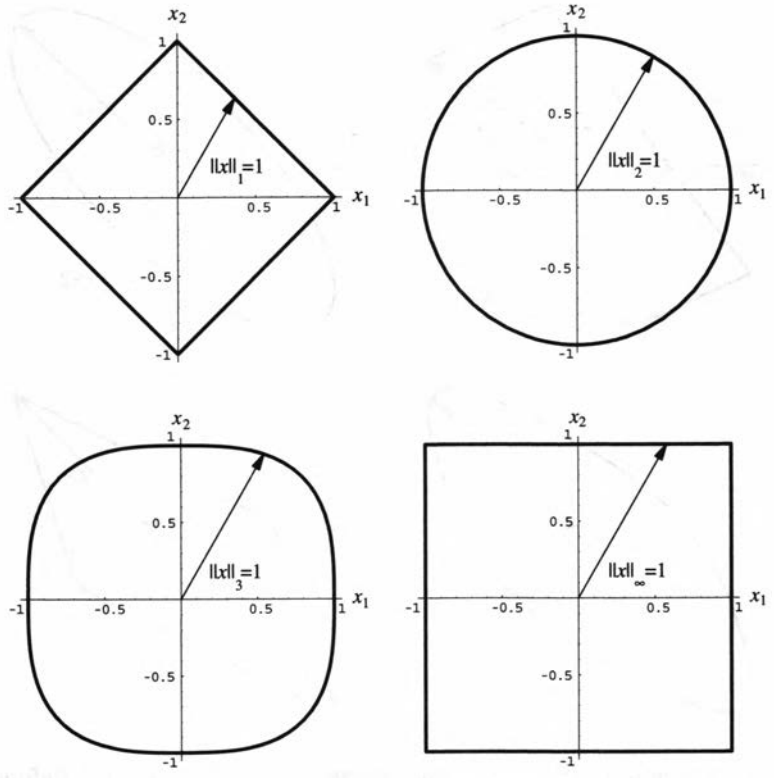


Figura 1.18 Contornos delimitados por vectores com normas $\|x\|_p$ unitárias em $\mathbb{R}^2$.

Este conceito está ilustrado na Figura 1.19, para a matriz

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \tag{1.27}$$

através da delimitação dos contornos das extremidades dos vectores $y = Ax$ obtidos usando vectores x unitários, para várias casos da família de *normas-p*. Os vectores originais x que permitem determinar $\|A\|_p$ estão também individualizados em cada caso. Neste exemplo, $\|A\|_1 = 6$, $\|A\|_2 \simeq 5{,}12$, $\|A\|_3 \simeq 5{,}06$ e $\|A\|_\infty = 5$.

No caso da família de *normas-p* temos

$$\|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p$$

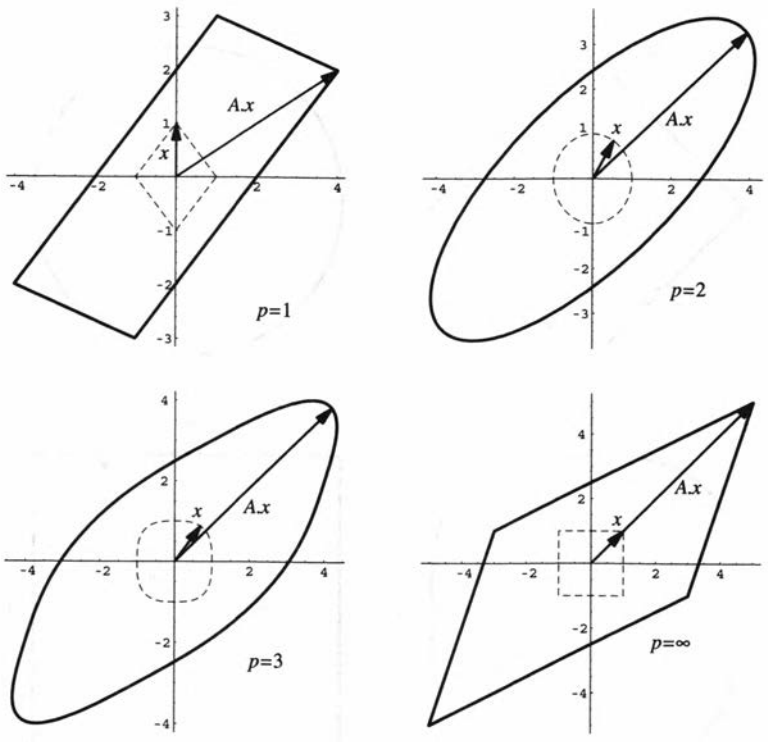


Figura 1.19 Determinação de algumas *normas-p* matriciais para A definida por (1.27).

sendo possível demonstrar que, para $A \in \mathbb{R}^{m \times n}$ (Golub e Van Loan, 1989):

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| \qquad \text{(máxima soma por colunas)}$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}| \qquad \text{(máxima soma por linhas)}$$

A simetria das expressões anteriores ilustra bem o facto de as diferentes normas traduzirem, de modo semelhante, a noção de comprimento dos vectores aos quais são aplicadas (ou de comprimento dos vectores de saída correspondentes, no caso de matrizes). É possível por exemplo mostrar as seguintes propriedades adicionais (Golub e Van Loan, 1989):

$$\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty \tag{1.28}$$

$$\frac{1}{\sqrt{m}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1 \tag{1.29}$$

Para além das propriedades anteriores, qualquer norma matricial *induzida* satisfaz ainda as desigualdades

$$\|Ax\| \leq \|A\|\|x\| \tag{1.30}$$

$$\|AB\| \leq \|A\|\,\|B\| \tag{1.31}$$

para $A \in \mathbb{R}^{m\times n}$ e $B \in \mathbb{R}^{n\times q}$, uma vez que a partir da definição, no 1º caso:

$$\|A\| = \sup_{\|x\|\neq 0} \frac{\|Ax\|}{\|x\|} \geq \frac{\|Ax\|}{\|x\|} \quad \Rightarrow \quad \|Ax\| \leq \|A\|\|x\|$$

Assim, no segundo caso pode facilmente escrever-se:

$$\|AB\| = \max_{\|x\|=1} \|ABx\| \leq \max_{\|x\|=1} \|A\|\|Bx\| = \|A\| \max_{\|x\|=1} \|Bx\| = \|A\|\|B\| \quad \Rightarrow$$

$$\Rightarrow \quad \|AB\| \leq \|A\|\,\|B\|$$

Os resultados (1.30) e (1.31) devem ser comparados com a propriedade (1.11), notando a necessidade da introdução da desigualdade no caso vectorial. Isso pode ser interpretado como o facto de o resultado depender não apenas do "tamanho" de x (i.e., da sua norma), mas também da direcção que representa, e das *direcções características* das matrizes A e B.

Raio espectral e valores singulares de A

Para uma matriz quadrada genérica $A \in \mathbb{R}^{n\times n}$ o seu *raio espectral* $\rho(A)$ é definido através de

$$\rho(A) = \max_{1\leq i\leq n} |\lambda_i(A)|$$

onde λ_i $(i = 1, 2, \ldots, n)$ são os valores próprios de A. É possível demonstrar que

$$\rho(A) = \inf_{\|\cdot\|} \|A\|$$

ou seja, o valor do raio espectral de qualquer matriz coincide com a *menor norma subordinada* que é possível definir para essa matriz.

Para matrizes simétricas, $\|A\|_2 = \rho(A)$. No caso geral tem-se:

$$\|A\|_2 = \sqrt{\rho(A^{\mathrm{T}}A)}$$

Os valores próprios de $A^{\mathrm{T}}A$ são todos não-negativos. As suas raízes quadradas são denominadas *valores singulares* de A, $\sigma_i(A)$, $i = 1, 2, \ldots, n$. Deste modo, a equação anterior indica que $\|A\|_2$ é igual a σ_1, o *maior* valor singular de A.

Exemplo 1.6 (Raio espectral de A)

Os valores próprios de A definida por (1.27), obteníveis a partir da resolução da sua equação característica $|\lambda I - A| = 0$, são $\lambda_1 = 5$, $\lambda_2 = -2$, e portanto $\rho(A) = 5$, que coincide com $\|A\|_\infty$. Neste caso também

$$A^{\mathrm{T}}A = \begin{bmatrix} 10 & 10 \\ 10 & 20 \end{bmatrix}$$

e portanto:

$$\|A\|_2 = \sigma_1(A) = \sqrt{5(3+\sqrt{5})} \simeq 5{,}12$$

Aplicações na análise de erros

Como referido atrás, a noção de norma é extremamente útil para quantificar o tamanho de entidades vectoriais. Este conceito será usado nos Capítulos seguintes para medir (ou comparar) o tamanho dos erros cometidos na aproximação de quantidades vectoriais.

Por exemplo, se $f(x) \in \mathbb{R}^m$ for uma função vectorial com argumento $x \in \mathbb{R}^n$ também vectorial, as normas atrás definidas permitem generalizar facilmente a este caso as definições de erros introduzidas na Secção 1.2.1. Aqui, dado um valor $x \in \mathbb{R}^n$ e uma aproximação deste $\bar{x} \in \mathbb{R}^n$, o erro absoluto da aproximação pode neste caso ser definido como

$$e(\bar{x}) = \|x - \bar{x}\|$$

para uma norma específica, de forma semelhante a (1.3). Para uma aproximação $\bar{f}(x)$ do valor exacto $f(x)$, o erro relativo correspondente é

$$r(\bar{f}) = \frac{e(\bar{f})}{\|f\|} = \frac{\|f - \bar{f}\|}{\|f\|}$$

também por generalização de (1.4). Se f coincidir com um operador linear $f(x) = Ax$, e $\bar{f} = A\bar{x}$, então o erro *máximo* induzido ao utilizarmos uma quantidade $\bar{x}$ próxima de x pode ser majorado (i.e., estimado por excesso) fazendo

$$e(\bar{f}) = \|Ax - A\bar{x}\| = \|A(x - \bar{x})\| \leq \|A\| e(\bar{x})$$

atendendo a (1.30). A norma de A actua neste caso como factor máximo de amplificação no erro original presente em x.

A escolha de uma norma específica nos resultados anteriores não é, em geral, particularmente importante, dadas as propriedades (1.28) e (1.29). Habitual-

mente são escolhidas as normas que mais simplificam o cálculo dos resultados pretendidos.

CAPÍTULO 2

Equações não-lineares

DADA uma equação não-linear $f(x) = 0$, com f e x escalares, um problema bastante comum consiste na determinação das *raízes* desta equação. As raízes são valores x^* que satisfazem exactamente a equação anterior:

$$f(x^*) = 0$$

Os valores x^* são também conhecidos como *zeros* da função $f(x)$. Alguns exemplos de equações deste tipo são:

$$x - \sin(x) = 0 \qquad \text{e} \qquad x - e^{-x} = 0$$

Para estes exemplos, assim como para a maioria das equações não-lineares, não existe um método fechado (envolvendo apenas um número finito de operações básicas) que permita a obtenção exacta das suas raízes. Este é portanto um problema numérico que é resolvido habitualmente através de métodos aproximados.

Por exemplo, com funções $f(x)$ polinomiais, a *fórmula resolvente* para polinómios de 2º grau (considerada no Exemplo 1.5) é conhecida desde há mais de 4000 anos. Fórmulas de natureza idêntica (embora mais complicadas) para polinómios de 3º e 4º graus, estão também disponíveis desde a época do Renascimento. No entanto, não são conhecidas fórmulas resolventes semelhantes para equações polinomiais de ordem superior[1].

[1]No princípio do século passado foi demonstrado que para a determinação dos zeros de polinómios de ordem igual ou superior a 5, não pode ser escrita qualquer fórmula finita envolvendo apenas as operações aritméticas básicas, conjuntamente com a extracção de raízes. Ainda no século passado, a teoria de grupos desenvolvida por Galois, veio permitir determinar rigorosamente quando é que uma equação polinomial pode (ou não) ser resolvida de forma exacta. Uma ideia do esforço envolvido na determinação exacta de raízes de equações de ordem superior pode ser encontrada em Wolfram Research (1996).

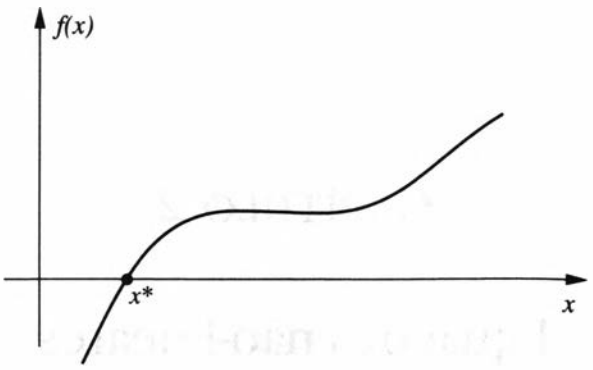


Figura 2.1 Localização aproximada dos zeros de $f(x)$.

2.1 Métodos gráficos, rearranjos e aproximações da solução

Os métodos conceptualmente mais simples para a localização de raízes de equações recorrem à representação gráfica de $f(x)$ (Figura 2.1). Estes métodos permitem, em muitos casos, determinar rapidamente o número e o valor aproximando das raízes x^*. Possuem ainda a vantagem de fornecerem indicações adicionais sobre a velocidade de variação de $f(x)$ no intervalo considerado, que pode ser correlacionada com a magnitude das derivadas da função.

Com a profusão de calculadoras com capacidades gráficas, e programas que permitem fazer a representação gráfica de funções, estes métodos tornaram-se a forma mais expedita de localizar as raízes pretendidas. No entanto, estes métodos requerem algum esforço e cuidado no traçado da curva, evitando possíveis perdas de informação sobre o comportamento da função, sobretudo em zonas de variação mais rápida.

Este procedimento pode também ser aplicado a *sistemas de equações*, envolvendo mais do que uma equação e uma incógnita. No entanto, o número adicional de dimensões complica consideravelmente a representação, limitando a utilidade destes métodos. Muitas vezes esta técnica é combinada com o uso de rearranjos algébricos das equações iniciais, conforme ilustrado no exemplo seguinte.

Exemplo 2.1 (Sistema de 2 equações não-lineares)

Considere-se o sistema de 2 equações não-lineares a 2 incógnitas:

$$\begin{cases} f_1(x,y) = 0 \\ f_2(x,y) = 0 \end{cases} \tag{2.1}$$

A aplicação directa do método gráfico à resolução deste sistema de equações requer a realização de 2 gráficos tridimensionais de $f_1(x,y)$ e $f_2(x,y)$, sendo adicionalmente necessário localizar os pontos onde as 2 funções se anulam simultaneamente. Esta é uma tarefa morosa, mesmo para funções com comportamentos simples, como ilustrado na Figura 2.2.

Se em vez disto for possível, por rearranjo algébrico (ou manipulação simbólica), expressar a primeira equação de (2.1) na forma

$$f_1(x,y) = 0 \quad \Rightarrow \quad x = g(y) \quad (\text{ou } y = h(x))$$

então, substituindo a variável explicitada na segunda equação de (2.1) fica

$$f_2(g(y),y) = 0 \quad \text{ou seja} \quad \boxed{\phi(y) = 0}$$

tendo sido obtida uma única equação a uma incógnita. A raiz desta equação y^* pode agora ser localizada mais facilmente usando o método gráfico unidimensional anterior. Após isto é possível calcular $x^* = g(y^*)$, constituindo (x^*, y^*) a solução do sistema de equações original (2.1). Este procedimento pode ser generalizado a casos de maior dimensão, usando eliminações sucessivas. No entanto, o seu uso requer algum cuidado, relativamente à análise dos pontos de singularidade das diferentes funções consideradas. A aplicação deste método ao exemplo considerado na Figura 2.2, substituindo $y = \exp(-x)$ na 2ª equação, encontra-se ilustrada na Figura 2.3.

Outras vezes é possível obter uma estimativa rápida da solução, por aproximação da função $f(x)$. A técnica de aplicação deste procedimento varia no entanto de caso para caso, conforme ilustrado no seguinte exemplo.

Exemplo 2.2 (Raízes de um polinómio (Rosa, 1988))

Considere-se o problema de determinação de uma raiz positiva da equação polinomial:

$$f(x) = x^5 - x - 500 = 0$$

Neste caso, o conhecimento antecipado da existência de uma raiz no intervalo $x \in [0, 10]$ pode ser usado para simplificar a determinação da solução. Com

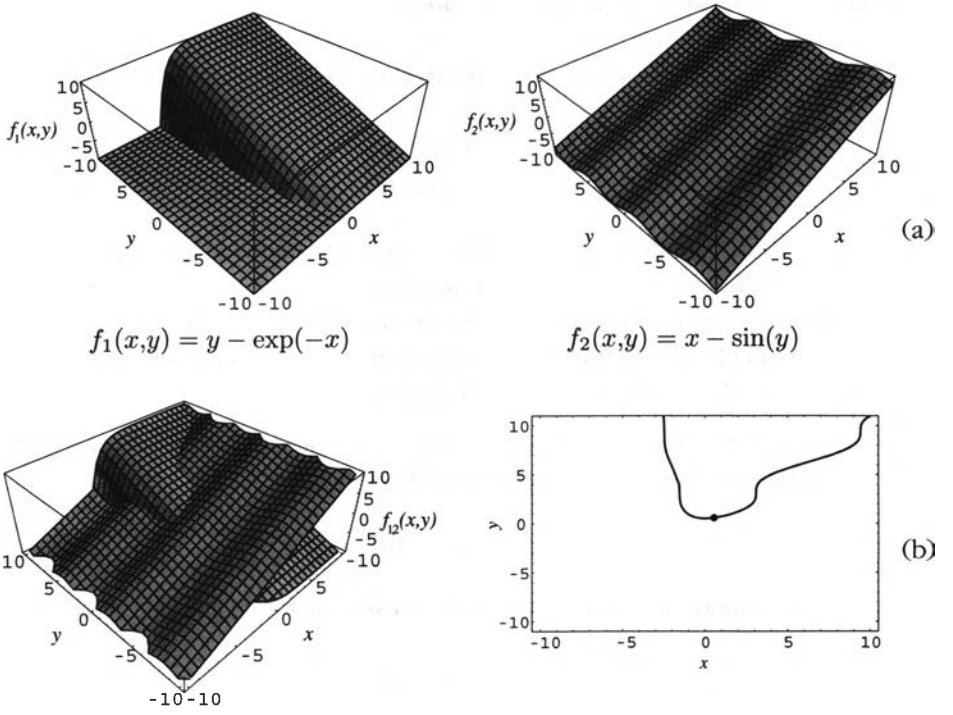


Figura 2.2 Localização das raízes de um sistema de duas equações: (a) Gráficos tridimensionais. (b) Intersecção das duas superfícies e solução do sistema de equações (ponto •).

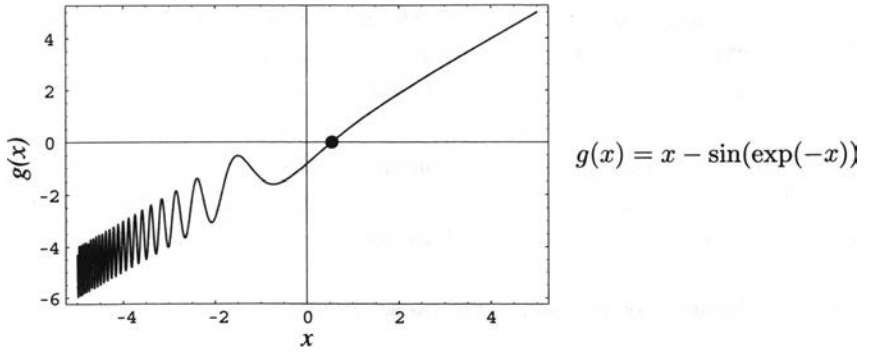


Figura 2.3 Localização de uma das raízes do exemplo considerado na Figura 2.2.

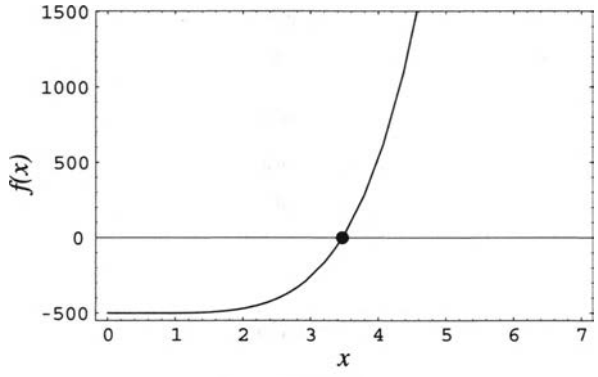


Figura 2.4 Localização aproximada dos zeros de $f(x)$ no Exemplo 2.2.

efeito:

$$\text{Para } x = 5, \quad f(5) = \underbrace{5^5}_{3125} - 5 - 500 > 0;$$
$$\text{Para } x = 3, \quad f(3) = \underbrace{3^5}_{243} - 3 - 500 < 0.$$

Em qualquer dos casos, o segundo termo do polinómio possui uma magnitude bastante inferior ao valor dos outros dois termos, podendo ser desprezado, sem erro significativo. Assim:

$$x^5 - x - 500 = 0 \quad \rightarrow \quad x^5 - 500 \simeq 0 \quad \Rightarrow \quad x^* \simeq \sqrt[5]{500} = 3{,}466$$

Para comparação, a solução exacta é $x^* = 3{,}471$, tendo sido obtida uma aproximação com 3 algarismos significativos correctos. A localização aproximada dos zeros de $f(x)$, usando o método gráfico anterior, encontra-se ilustrada na Figura 2.4.

2.2 Métodos iterativos

Sensivelmente todos os métodos estudados para a resolução numérica de equações não-lineares (bem como para outras classes de problemas) são métodos aproximados, de *natureza iterativa*. Estes métodos possuem uma forma geral do tipo:

$$x_{k+1} = \phi(x_k), \qquad k = 0, 1, 2, 3, \ldots \tag{2.2}$$

Todos eles utilizam uma estimativa inicial x_0 da solução, geralmente fornecida pelo utilizador, e iteram-na, gerando uma sequência infinita de valores:

$$\begin{aligned} x_1 &= \phi(x_0) \\ x_2 &= \phi(x_1) \\ x_3 &= \phi(x_2) \\ &\vdots \end{aligned}$$

Geralmente o valor pretendido x^* é obtido como limite da sequência anterior:

$$x^* = \lim_{k \to \infty} x_k$$

A aplicação prática destes métodos requer a interrupção da sequência anterior, após um número máximo (finito) de iterações. O erro da aproximação correspondente, após k iterações, pode ser definido como $e_k = |x^* - x_k|$, assumindo a natureza de um *erro de truncatura*.

2.2.1 Razão e ordem de convergência

A rapidez de convergência de um método iterativo genérico da forma (2.2) pode ser caracterizada do seguinte modo:

Se existirem constantes $0 \leq c < 1$, $p \geq 1$ e $k_0 > 0$ tais que, para todas as iterações $k \geq k_0$, se tem

$$\frac{|x^* - x_{k+1}|}{|x^* - x_k|^p} \leq c \qquad (2.3)$$

então o método tem *ordem de convergência* p. A constante c designa-se neste caso por *razão de convergência*.

Quanto maior for o índice p e menor a constante c, maior é a rapidez de convergência do método. Um método iterativo com $p = 1$ é descrito como possuindo ordem de convergência *linear*; se $p = 2$ o método têm ordem de convergência *quadrática*. Para $1 < p < 2$, o método tem ordem de convergência *superlinear*, verificando-se também neste caso

$$\frac{|x^* - x_{k+1}|}{|x^* - x_k|} \leq c_k \qquad \text{com} \qquad \lim_{k \to \infty} c_k = 0.$$

A equação anterior é por isso usada, por vezes, como definição alternativa da propriedade de convergência superlinear.

Para métodos com ordem de convergência baixa, é sobretudo a constante c que determina a sua velocidade de convergência. No entanto, o efeito de p é geralmente o mais importante, sobretudo quando ele é superior à unidade. Métodos com p elevado podem requerer contudo a continuidade (e o conhecimento) das derivadas de ordem superior de $f(x)$, não sendo sempre prática a sua aplicação. Como se pode adivinhar, existe em geral um compromisso ideal entre o número de iterações necessárias (minimizado para métodos de ordem elevada) e o esforço requerido por iteração (em geral maior nos métodos com ordem de convergência elevada), considerando a rapidez de obtenção da solução de uma equação não-linear. Este assunto é analisado na Secção 2.10.

2.3 Método das substituições sucessivas

Este é talvez o método iterativo mais simples para resolução de equações não-lineares, sendo também designado como *método do ponto fixo*. Se for possível re-escrever a equação

$$f(x) = 0 \tag{2.4}$$

na forma equivalente

$$x - g(x) = 0 \tag{2.5}$$

então é possível construir o método iterativo

$$x_{k+1} = g(x_k) \tag{2.6}$$

dado que os pontos x^* que verificam (2.6) são também soluções de (2.4) e (2.5). Nem sempre $g(x)$ tem de ser definida de forma única, como ilustrado no seguinte exemplo.

Exemplo 2.3 (Polinómio de 5º grau)

Considerando novamente a equação polinomial de 5ª ordem $x^5 - x - 500 = 0$, é possível rearranjar esta equação de forma a obter imediatamente pelo menos 2 fórmulas recursivas:

$$x_{k+1} = \sqrt[5]{x_k + 500} \tag{2.7}$$

$$x_{k+1} = x_k^5 - 500 \tag{2.8}$$

Na Tabela 2.1 representam-se os valores sucessivos obtidos por aplicação de (2.7) e (2.8), usando a estimativa inicial $x_0 = 3$. Como se pode observar, a 1ª expressão converge para uma raiz finita do polinómio, ao passo que a 2ª parece divergir para $-\infty$.

Tabela 2.1 Aplicação do método das substituições sucessivas com base em (2.7) e (2.8).

k	Equação (2.7)	Equação (2.8)
0	3	3
1	3,46987	$-257{,}000$
2	3,47052	$-1{,}12115 \times 10^{12}$
3	3,47052	$-1{,}77145 \times 10^{60}$
4	3,47052	$-1{,}74438 \times 10^{301}$
5	...	...

Problema 2.1 *Utilize a expressão* (2.3) *para caracterizar a rapidez de convergência do método iterativo* (2.7)*, determinando as constantes* c *e* p *aplicáveis a este caso.*

O exemplo anterior ilustra o facto de a convergência (ou não) do método das substituições sucessivas depender mais da forma particular de $g(x)$ escolhida, do que da equação inicial. Em qualquer dos casos a implementação prática de métodos iterativos requer a definição de *critérios de paragem*, estabelecendo limites no prosseguimento dos cálculos. Os critérios mais comuns são:

Limite máximo no número de iterações: Este critério especifica o número máximo de aplicações sucessivas da expressão iterativa (2.2), sendo sempre usado conjuntamente com um dos critérios de erro seguintes.

Erro da aproximação: Geralmente, a quantidade $e_k = |x^* - x_k|$ apenas pode ser
58 calculada após ser conhecido x^*. Como tal, é frequentemente substituída pelo critério de erro

$$|x_k - x_{k-1}| \leq \epsilon_1 \tag{2.9}$$

que já pode ser facilmente calculado. No entanto deve notar-se que

$$|x_k - x_{k-1}| \leq \epsilon_1 \not\Rightarrow |x^* - x_k| \leq \epsilon_1$$

i.e., o progresso para a solução pode tornar-se diminuto, sem que a solução tenha sido atingida. Deste modo, o critério (2.9) apenas é indicativo do progresso para a solução em métodos que ofereçam garantias quanto à sua rapidez de convergência, ou quando é usado conjuntamente com critérios de erro no valor da função, do tipo (2.12).

Este critério possui ainda a desvantagem de ser dependente da escala do

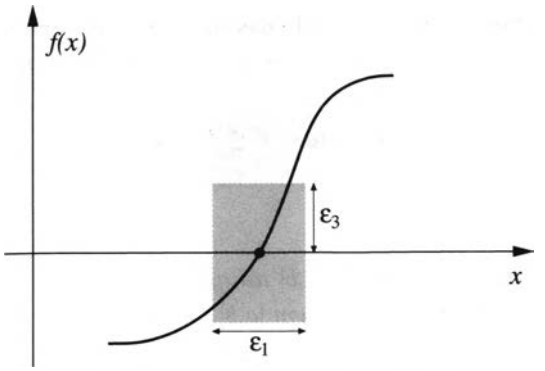


Figura 2.5 Critérios de paragem relativos ao erro da solução e ao valor da função.

problema (i.e., da magnitude de x^*), sendo por vezes substituído por

$$\frac{|x_k - x_{k-1}|}{|x_k|} \leq \epsilon_2 \tag{2.10}$$

O critério de erro relativo revela contudo dificuldades na sua aplicação quando $x^* \simeq 0$, sendo frequentemente substituído pela quantidade

$$\frac{|x_k - x_{k-1}|}{1 + |x_k|} \leq \epsilon_2 \tag{2.11}$$

que tem como limites as equações anteriores (2.9) e (2.10), quando a magnitude de x_k tende para zero ou para infinito, respectivamente.

Valor da função: Neste caso, o critério de paragem tem a forma

$$|f(x_k)| \leq \epsilon_3 \tag{2.12}$$

avaliando o resíduo de uma quantidade que deveria ser exactamente zero na solução.

Muito frequentemente é usado um critério de cada uma das classes anteriores, sendo os cálculos interrompidos apenas quando todas as condições consideradas forem simultaneamente satisfeitas (Figura 2.5).

É também necessário conhecer melhor quais as *condições de convergência* do método das substituições sucessivas. Condições suficientes são fornecidas pelo seguinte teorema:

Teorema 2.1 (Convergência do método das substituições sucessivas) *Considerando a equação $x = g(x)$, com $g : [a,b] \to [a,b]$ diferenciável no intervalo usado, se*

$$\max_{x\in[a,b]} |g'(x)| \leq L < 1 \tag{2.13}$$

então:

1. *Existe um único ponto $x^* \in [a,b]$ tal que $x^* = g(x^*)$. Este ponto é designado como* ponto fixo *de $g(x)$ em $[a,b]$.*

2. *A partir de qualquer $x_0 \in [a,b]$, x^* pode ser calculado iterando*

$$x_{k+1} = g(x_k), \qquad k = 0, 1, 2, \ldots$$

   uma vez que $\lim_{k\to\infty} x_k = x^*$.

3. *Tem-se ainda:*

$$|x^* - x_{k+1}| \leq L|x^* - x_k| \tag{2.14}$$

A equação (2.14) pode também ser escrita na forma:

$$|g(x^*) - g(x_k)| \leq L|x^* - x_k| \tag{2.15}$$

Uma equação deste tipo é designada habitualmente por *condição de Lipschitz.*

Demonstração Da fórmula de Taylor com resto apropriado, vem

$$g(x_k) = g(x_{k-1}) + g'(\eta)(x_k - x_{k-1})$$

com $\eta \in [\min\{x_{k-1}, x_k\}, \max\{x_{k-1}, x_k\}]$. Rearranjando a equação anterior fica:

$$|x_{k+1} - x_k| = |g(x_k) - g(x_{k-1})| = |g'(\eta)||x_k - x_{k-1}| \quad \Rightarrow$$
$$\Rightarrow \ |x_{k+1} - x_k| \leq L|x_k - x_{k-1}| \tag{2.16}$$

Aplicando esta expressão repetidamente tem-se

$$|x_{k+1} - x_k| \leq L|x_k - x_{k-1}| \leq L^2|x_{k-1} - x_{k-2}| \leq \cdots \leq L^k|x_1 - x_0|$$

o que significa que a sequência converge (uma vez que $L < 1$). Neste caso, quanto menor for L, maior será a rapidez de convergência da sequência.

Outra aplicação da fórmula de Taylor em torno de x_k dá-nos

$$g(x^*) = g(x_k) + g'(\xi)(x^* - x_k)$$

com $\xi \in [\min\{x_k, x^*\}, \max\{x_k, x^*\}]$, de onde resulta:

$$|x^* - x_{k+1}| = |g(x^*) - g(x_k)| = |g'(\xi)||x^* - x_k| \quad \Rightarrow$$
$$\Rightarrow \quad |x^* - x_{k+1}| \leq L|x^* - x_k| \quad (2.17)$$

A aplicação repetida desta equação garante portanto a convergência para a solução:

$$|x^* - x_{k+1}| \leq L|x^* - x_k| \leq L^2|x^* - x_{k-1}| \leq \cdots \leq L^{k+1}|x^* - x_0|$$

Exemplo 2.4 (Substituições sucessivas com equações polinomiais)

É instrutivo verificar se a condição de convergência (2.13) se aplica a ambas as fórmulas iterativas do Exemplo 2.3. Para a fórmula $x_{k+1} = g_1(x) = x_k^5 - 500$, temos:

$$g_1'(x) = 5x^4 > 1, \quad \forall x \in [1{,}10]$$

Para $x_{k+1} = g_2(x_k) = \sqrt[5]{x_k + 500}$ fica

$$g_2'(x) = \frac{1}{5(x + 500)^{4/5}} < 1, \quad \forall x \in [0{,}10]$$

e portanto a condição suficiente só é verificada no segundo caso.

A situação verificada neste exemplo é bastante comum com equações polinomiais. Daí que a fórmula iterativa preferida para a determinação de raízes de equações polinomiais genéricas

$$a_x x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0 \quad (2.18)$$

seja geralmente

$$x_{k+1} = \pm \sqrt[n]{-a_{n-1}x^{n-1} - \cdots - a_1 x - a_0}$$

obtida por rearranjo de (2.18), a partir da potência de ordem mais elevada.

Se n for par, torna-se necessário optar por um dos sinais na equação anterior, sendo para o efeito necessário conhecer antecipadamente o sinal da raiz procurada.

A aplicação do método das substituições sucessivas possui uma interpretação gráfica simples. A construção efectuada possui geralmente um aspecto típico em degraus ou em espiral (Figura 2.6).

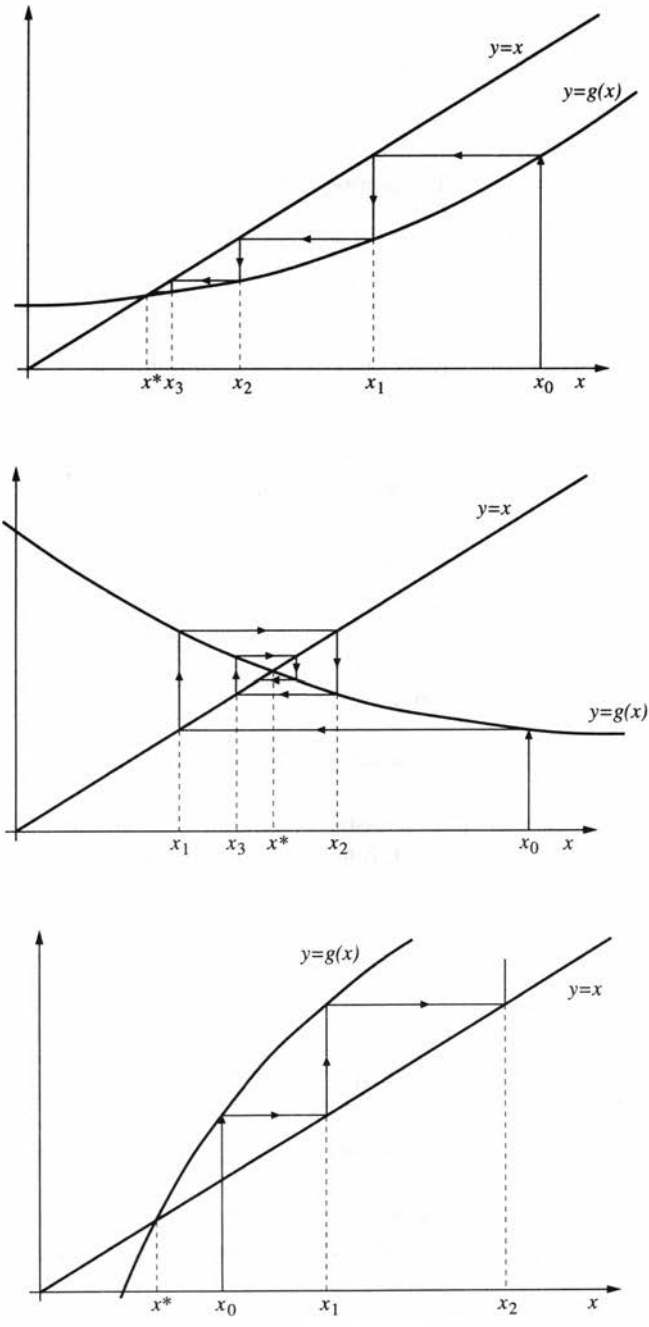


Figura 2.6 Ilustração gráfica do método das substituições sucessivas. Cada situação corresponde a um valor de $g'(x)$ numa gama distinta: a) $0 \leq g'(x) < 1$; b) $-1 < g'(x) \leq 0$; c) $g'(x) > 1$.

Estes gráficos são obtidos começando pelo desenho da curva $y = g(x)$ e da recta $y = x$. A iteração é depois iniciada, obtendo-se o ponto $y_1 = g(x_0)$, dado pela intersecção de uma recta vertical $x = x_0$ com a curva $y = g(x)$; o valor correspondente pode ser lido directamente no eixo das ordenadas. Este valor pode ser copiado para o eixo das abcissas (onde é necessário para a segunda iteração), através da sua reflexão na recta $y = x$, obtendo-se x_1. Este valor é usado para a segunda iteração, obtendo-se $y_2 = g(x_1)$, e assim sucessivamente.

Problema 2.2 *Obtenha a representação gráfica correspondente à Figura 2.6, para o caso em que* $g'(x) < -1$.

Problema 2.3 *Construa a representação gráfica correspondente à aplicação das fórmulas iterativas* (2.7) *e* (2.8)*, consideradas no Exemplo 2.3.*

De modo semelhante à demonstração do Teorema 2.1, a constante L da condição de Lipschitz (2.15) pode ser usada para demonstrar várias propriedades desta classe genérica de métodos iterativos:

Erro na iteração k

Os critérios de paragem (2.9) a (2.11), embora de aplicação simples, não permitem quantificar directamente o verdadeiro erro cometido na localização da raiz x^* no final da iteração k, dado por $|x^* - x_k|$. Como tal, corre-se o risco de interromper o processo iterativo devido à aplicação de um dos critérios anteriores, sem localizar a raiz com a exactidão desejada.

No entanto, é possível escrever

$$x^* - x_k = (x^* - x_{k+1}) + (x_{k+1} - x_k)$$

e portanto:

$$|x^* - x_k| \leq |x^* - x_{k+1}| + |x_{k+1} - x_k|$$

No caso do método das substituições sucessivas, usando (2.17) a equação anterior fica:

$$|x^* - x_k| \leq L|x^* - x_k| + |x_{k+1} - x_k|$$

Agrupando o termo $|x^* - x_k|$ no lado esquerdo, tem-se:

$$|x^* - x_k| \leq \frac{1}{(1-L)}|x_{k+1} - x_k|$$

De igual modo, usando (2.16) vem:

$$|x^* - x_k| \leq \frac{1}{(1-L)}|x_{k+1} - x_k| \leq$$
$$\leq \frac{L}{(1-L)}|x_k - x_{k-1}| \leq \cdots \leq \frac{L^k}{(1-L)}|x_1 - x_0| \quad (2.19)$$

Esta expressão permite portanto delimitar o erro na localização da solução x^* no final da iteração k, com base na quantidade $|x_k - x_{k-1}|$, ou outro valor anterior da mesma natureza. Pode observar-se que o factor de amplificação do erro $|x_k - x_{k-1}|$ depende apenas da constante L; valores desta constante próximos da unidade conduzem a intervalos de maior amplitude na localização de x^*.

Problema 2.4 *Usando os valores da Tabela 2.1 e com base em* (2.19) *estime o erro na localização da raiz de* (2.7) *após 2 iterações.*

Exactidão final atingível

Dahlquist e Björck (1974) desenvolveram uma análise semelhante à anterior para quantificar a *incerteza final* na localização de raízes de equações não-lineares através de métodos desta categoria, quando a iteração é prolongada indefinidamente. Esta análise é efectuada com base na sequência de *valores calculados* $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$, que podem diferir dos seus congéneres exactos $x_1, x_2, \ldots, x_k$ uma vez que se admite que em cada iteração o erro no cálculo de $g(x_k)$ seja de δ_k:

$$\bar{x}_{k+1} = g(\bar{x}_k) + \delta_k, \qquad k = 0, 1, 2, \ldots$$

Subtraindo a equação anterior de $x^* = g(x^*)$ vem

$$(x^* - \bar{x}_{k+1}) = g(x^*) - g(\bar{x}_k) - \delta_k$$
$$= g(x^*) - [g(x^*) + g'(\xi_k)(\bar{x}_k - x^*)] - \delta_k$$

usando o teorema do valor médio, com $\xi_k \in [\min\{\bar{x}_k, x^*\}, \max\{\bar{x}_k, x^*\}]$. Tem-se portanto

$$(x^* - \bar{x}_{k+1}) = g'(\xi_k)(x^* - \bar{x}_k) - \delta_k$$

de onde resulta, somando e subtraindo nesta equação a quantidade $g'(\xi_k)\bar{x}_{x+1}$:

$$(x^* - \bar{x}_{k+1}) - g'(\xi_k)(x^* - \bar{x}_{k+1}) = g'(\xi_k)(\bar{x}_{k+1} - \bar{x}_k) - \delta_k \Rightarrow$$
$$[1 - g'(\xi_k)]\,(x^* - \bar{x}_{k+1}) = g'(\xi_k)(\bar{x}_{k+1} - \bar{x}_k) - \delta_k$$

Assumindo um limite superior para os erros

$$\delta_k \leq \Delta g, \quad \forall k = 0,1,2,\ldots$$

e uma condição de Lipschitz (2.13) para $g(x)$

$$|g'(\xi_k)| \leq L < 1, \quad \forall k = 0,1,2,\ldots$$

resulta da equação anterior:

$$|x^* - \bar{x}_{k+1}| \leq \frac{L}{(1-L)}|\bar{x}_{k+1} - \bar{x}_k| + \frac{\Delta g}{(1-L)}$$

Quando $k \rightarrow \infty$, se $|\bar{x}_{k+1} - \bar{x}_k| \rightarrow 0$, o último termo desta equação torna-se dominante, podendo ser considerado uma limitação *final* na incerteza de delimitação da raiz x^*. No limite, este termo apenas depende do erro de aproximação típico *por cada iteração* (Δg), e da constante L; nestas condições, o uso posterior da expressão $x_{k+1} = g(x_k)$ não produzirá (estatisticamente) nenhuma melhoria na estimativa da solução já disponível. Tal como observado na Secção anterior, este comportamento é particularmente pertinente quando a constante L da condição de Lipschitz se aproxima da unidade.

2.4 Delimitação prévia da solução e métodos abertos

Os métodos numéricos estudados para resolução de equações não-lineares podem ser agrupados em duas classes genéricas, consoante envolvam ou não uma fase de pré-processamento destinada à *delimitação prévia* da solução. Os métodos que não necessitam desta etapa são por isso frequentemente designados como *métodos abertos*. O método das substituições sucessivas descrito anteriormente pertence à classe de métodos abertos.

Nos métodos que delimitam previamente a solução, o objectivo da etapa de delimitação consiste em encontrar um intervalo $[a,b]$, tal que $f(a) \cdot f(b) < 0$. Se $f(x)$ for uma função contínua, é então possível garantir que existe sempre um número ímpar (1, 3, 5, etc.) de zeros de $f(x)$ em $[a,b]$ (Figura 2.7). Estes métodos possuem portanto como principal vantagem a facilidade com que se pode demonstrar a sua convergência, após a delimitação prévia da solução[2]. No entanto, os métodos que requerem uma pré-delimitação da solução possuem alguns inconvenientes:

- Pode ser necessário, durante a fase de pré-processamento, avaliar $f(x)$

[2]Para garantir a convergência de um método deste tipo basta apenas mostrar que ele permite gerar intervalos $[a_k,b_k]$ de amplitude sucessivamente mais pequena, com $\lim_{k\rightarrow\infty}(b_k - a_k) = 0$, sempre com $f(a_k) \cdot f(b_k) < 0$.

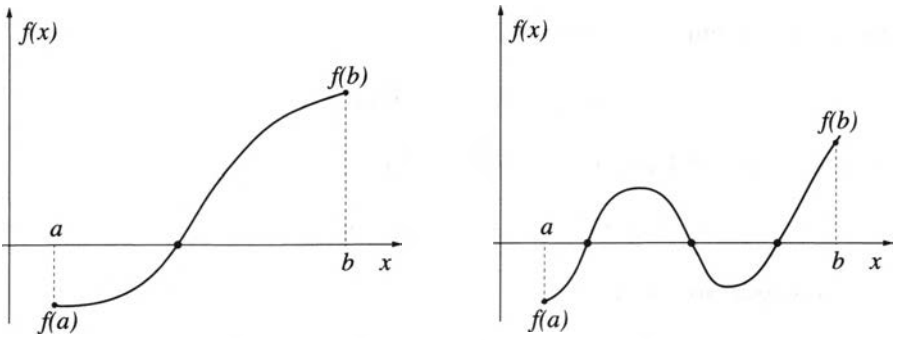


Figura 2.7 Delimitação prévia da solução.

bastantes vezes, o que pode ser desvantajoso, sobretudo para problemas de grande dimensão.

- Podem ser ignoradas regiões contendo um número par de zeros da função.

- Caso sejam identificados intervalos com um número ímpar de raízes, como é possível garantir a convergência para uma destas raízes específicas?

Uma comparação mais extensiva entre estas classes de métodos deverá ter como base as vantagens e desvantagens de métodos representativos das duas classes. Alguns métodos mais comuns são apresentados seguidamente.

2.5 Método da bissecção

Este método corresponde talvez à implementação mais simples das técnicas que recorrem à delimitação prévia da solução. Admitindo que, na iteração k, se tem um intervalo $[a_k, b_k]$, com $f(a_k) \cdot f(b_k) < 0$, este método sugere que seja calculado o valor da função no ponto intermédio do intervalo:

$$x_{k+1} = \frac{a_k + b_k}{2}$$

Consoante o sinal de $f(x_{k+1})$, o novo intervalo escolhido poderá ser $[a_k, x_{k+1}]$ ou $[x_{k+1}, b_k]$. Em qualquer dos casos, o novo intervalo terá uma amplitude igual a metade do intervalo anterior. O intervalo escolhido deverá ser tal que

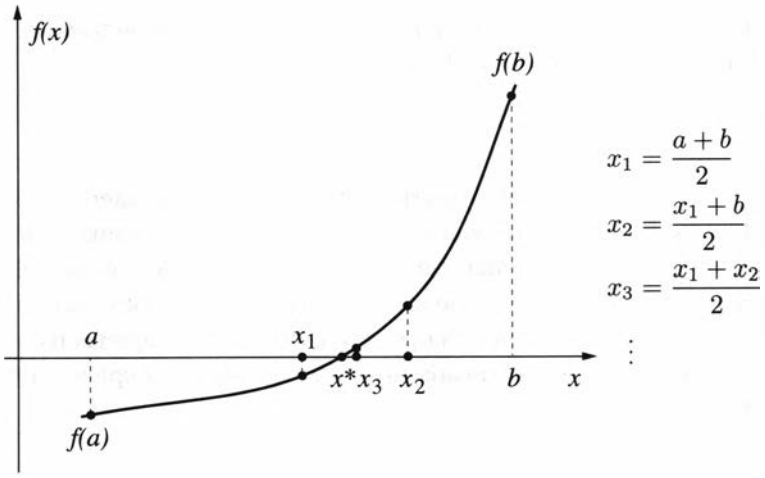


Figura 2.8 Aplicação do método da bissecção.

se tenha sempre $f(a_{k+1}) \cdot f(b_{k+1}) < 0$ (Figura 2.8). Deste modo:

$$\text{se} \quad f(x_{k+1}) \cdot f(a_k) > 0, \quad \text{então} \quad \begin{cases} a_{k+1} = x_{k+1} \\ b_{k+1} = b_k \end{cases}$$

$$\text{caso contrário, faz-se} \quad \begin{cases} a_{k+1} = a_k \\ b_{k+1} = x_{k+1} \end{cases}$$

Como se pode verificar, se $f(a_0) \cdot f(b_0) < 0$, e $f(x)$ for contínua, o método converge sempre para uma das soluções de $f(x) = 0$. O progresso efectuado ao fim de k iterações pode ser antecipado:

$$\begin{aligned} (b_1 - a_1) &= \frac{1}{2}(b_0 - a_0) \\ (b_2 - a_2) &= \frac{1}{2}(b_1 - a_1) = \frac{1}{2^2}(b_0 - a_0) \\ &\vdots \\ (b_k - a_k) &= \frac{1}{2^k}(b_0 - a_0) \end{aligned} \tag{2.20}$$

Deste modo, se $|b_k - a_k|$ for usado como uma estimativa superior da incerteza na aproximação de x^* (ao fim de k iterações), a expressão anterior permite prever quantas iterações são necessárias para reduzir a amplitude de um intervalo inicial arbitrário $|b_0 - a_0|$ a um valor pré-determinado.

Problema 2.5 *Utilizando as equações* (2.3) *e* (2.20), *mostre que este método tem* convergência linear *($p = 1$), com* razão de convergência $c = 1/2$.

Deve notar-se que os valores de c e p determinados no problema anterior não dependem da função $f(x)$ a avaliar, uma vez que não é usada nenhuma informação específica sobre o seu comportamento no intervalo em estudo. Como tal, não é de estranhar que este método requeira em geral muitas iterações, quando comparado com os métodos seguintes. Na prática, este método é apenas recomendado para a determinação aproximada de zeros de funções simples, e de avaliação rápida.

2.6 Método da regula falsi

Este método recorre a uma estratégia semelhante ao método da bissecção, partindo também de um intervalo inicial $[a_0, b_0]$ que deve delimitar a solução. No entanto, em vez de em cada iteração se calcular o valor de $f(x)$ no ponto médio do intervalo, a função é aproximada neste intervalo por uma recta, que passa pelos pontos $\big(a_k, f(a_k)\big)$ e $\big(b_k, f(b_k)\big)$. À medida que a amplitude do intervalo $[a_k, b_k]$ diminui, esta aproximação torna-se cada vez melhor, gerando uma sucessão de aproximações convergente para a solução.

Na iteração k, a recta que passa pelos extremos do intervalo tem como equação:

$$y(x) = f(a_k) + \underbrace{\frac{f(b_k) - f(a_k)}{(b_k - a_k)}}_{\text{declive } d_k} (x - a_k) \tag{2.21}$$

Nesta equação, para que $y(x_{k+1}) = 0$, fica

$$\boxed{x_{k+1} = a_k - \frac{f(a_k)}{d_k}} \tag{2.22}$$

com:

$$d_k = \frac{f(b_k) - f(a_k)}{(b_k - a_k)}$$

A aplicação do método da regula falsi encontra-se ilustrada na Figura 2.9. Tal

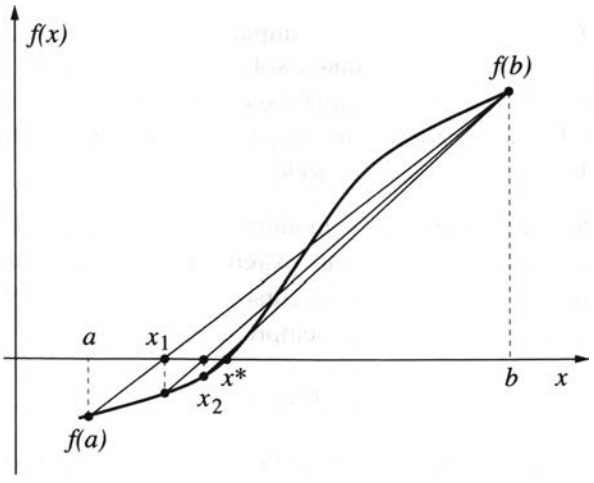


Figura 2.9 Aplicação do método da regula falsi.

como no método anterior:

$$\text{se} \quad f(x_{k+1}) \cdot f(a_k) > 0, \quad \text{então} \quad \begin{cases} a_{k+1} = x_{k+1} \\ b_{k+1} = b_k \end{cases}$$

$$\text{caso contrário, faz-se} \quad \begin{cases} a_{k+1} = a_k \\ b_{k+1} = x_{k+1} \end{cases}$$

Em algumas situações (por exemplo, se $f(x)$ for uma função convexa), é possível demonstrar que este método possui igualmente *convergência linear* (Rosa, 1988). Neste caso, a rapidez de convergência poderá ser inferior à do método da bissecção, se a constante c para o método da regula falsi em (2.3) for superior a $1/2$. Este comportamento depende contudo da natureza específica da função. Como o método requer a mesma informação por iteração (uma avaliação da função $f(x)$), este pode ser preferido ao método da bissecção, depois de implementadas algumas modificações no seu modo de aplicação.

2.6.1 Método da regula falsi modificado

Observando a Figura 2.9 pode constatar-se que neste método, por vezes, um dos extremos do intervalo se mantém fixo até ao final das iterações. Este fenómeno ocorre habitualmente quando $f(x)$ mantém a sua curvatura num intervalo $[a_k, b_k]$ da sequência de iterações, embora também possa ocorrer noutros casos,

como na Figura 2.9. Nesta situação a amplitude do intervalo $[a_k, b_k]$ não converge para zero, coincidindo no limite a solução x^* com um dos seus extremos. Este facto impossibilita o uso da amplitude do intervalo final como uma estimativa do erro na determinação da solução, prejudicando também frequentemente a velocidade de convergência do método.

Para evitar este comportamento, a modificação mais comum consiste em diminuir artificialmente a ordenada do extremo que se mantém fixo, até este ponto ser finalmente substituído no decurso do processo iterativo. Uma forma empírica de forçar isto é a seguinte: sempre que

$$f(x_k) \cdot f(x_{k-1}) > 0$$

ou seja, sempre que sejam geradas duas iterações consecutivas do mesmo lado da raiz, é usada na iteração seguinte *apenas metade do valor da função no extremo oposto do intervalo.*

Por exemplo na Figura 2.9, tanto $f(x_1) < 0$ como $f(x_2) < 0$, e logo neste caso a condição $f(x_1) \cdot f(x_2) > 0$ é válida. Deste modo, o método modificado irá utilizar na terceira iteração os pontos de coordenadas $a_3 = x_2, f(a_3) = f(x_2)$, e $b_3 = b, f(b_3) = f(b)/2$, para traçar a recta correspondente, usando novamente a equação (2.22). A aplicação deste método modificado é ilustrada na Figura 2.10. Como se pode observar, esta modificação permite uma redução significativa da amplitude do intervalo na terceira iteração. Após esta iteração, o intervalo que delimita a solução é bastante menor que o intervalo que se obteria por aplicação do método da bissecção com a função ilustrada nas Figuras 2.9 e 2.10. Por esta razão, a aplicação do método de regula falsi modificado é em geral preferida ao método da bissecção.

A aplicação deste método pode ser sumarizada através do Algoritmo 2.1.

2.7 Método das secantes

Contrariamente aos métodos anteriores, este método (e os seguintes) já não depende de uma delimitação prévia da solução, antes do início da aplicação do método iterativo. O método das secantes utiliza apenas duas estimativas x_k e x_{k-1} da solução, juntamente com os respectivos valores de $f(x)$ nestes pontos, para estimar a localização do zero de $f(x)$. Usando novamente a equação (2.21), tem-se neste caso

$$x_{k+1} = a_k - \frac{a_k}{d_k} \tag{2.23}$$

com:

$$d_k = \frac{f(x_k) - f(x_{k-1})}{(x_k - x_{k-1})} \tag{2.24}$$

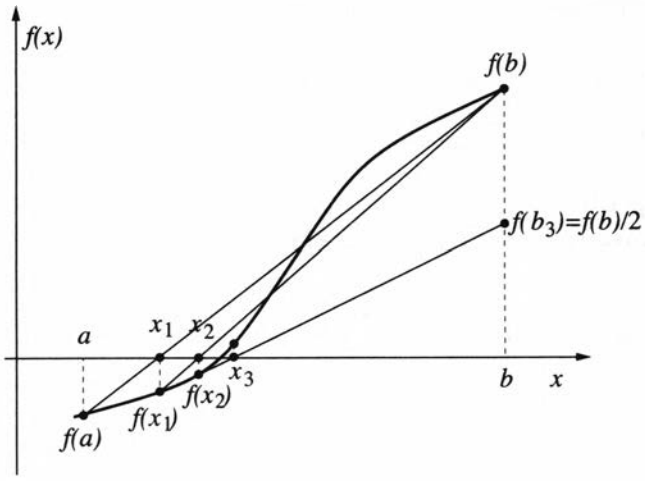


Figura 2.10 Aplicação do método da regula falsi modificado.

A principal diferença destas equações relativamente a (2.22) é que no caso presente é possível que os dois pontos base x_k e x_{k-1} estejam situados do mesmo lado da raiz, (i.e., com $f(x_k) \cdot f(x_{k-1}) > 0$), o que não era permitido no método anterior.

Este facto pode originar a divergência do método, com algumas funções. Por exemplo, se os valores de $f(x_k)$ e $f(x_{k-1})$ forem aproximadamente iguais, a recta traçada é quase horizontal, estando o seu ponto de intersecção com o eixo das abcissas bastante longe da zona onde se encontram x_k e x_{k-1}. O método pode igualmente divergir se os pontos iniciais estiverem "bastante" afastados da solução. Consequentemente, os resultados disponíveis para a análise da convergência deste método recorrem à definição de uma vizinhança "suficientemente próxima" da solução, onde devem estar contidas as estimativas iniciais x_0 e x_1, para assegurar a convergência do método. A dimensão dessa vizinhança depende de $f(x)$, sendo dificilmente quantificável na prática (Rosa, 1988).

Pode demonstrar-se que quando o método das secantes converge, a sua *ordem de convergência* é $p = (1 + \sqrt{5})/2 \simeq 1{,}618$ (Kincaid e Cheney, 1991). Esta convergência será portanto mais rápida do que a de qualquer um dos métodos estudados na classe dos métodos fechados. Tendo presente as semelhanças entre (2.22) e (2.23) constata-se pois que *manter a solução delimitada, nos métodos considerados anteriormente, teve como principal consequência uma diminuição da velocidade de convergência do método iterativo*, embora ofereça a segurança adicional da convergência assegurada.

O método das secantes é bastante utilizado, sobretudo com funções cujo desco-

Algoritmo 2.1 (*Regula falsi* modificado)

```
Dados a, b com f_a = f(a), f_b = f(b) e f_a f_b < 0
Para k = 1, 2, ..., N_iter.max. fazer
    d_k = (f_b − f_a)/(b − a)
    x_k = a − f_a/d_k
    f_k = f(x_k)
    Se convergência não verificada
        Se f_a f_k > 0 fazer
            a = x_k ; f_a = f_k
        Caso contrário fazer
            b = x_k ; f_b = f_k
        Fim_Se
        Se k ≥ 2 fazer
            Se f_k f_{k−1} > 0 fazer
                Se f_a f_k > 0 fazer
                    f_b = f_b/2
                Caso contrário fazer
                    f_a = f_a/2
                Fim_Se
            Fim_Se
        Fim_Se
        f_{k−1} = f_k
    Fim_Se
Fim_Para
Assinalar a não-convergência do algoritmo
```

nhecimento (ou dificuldade no cálculo) das derivadas impeça o uso do método de Newton, descrito na Secção seguinte.

2.8 Método de Newton

Este método é também conhecido por *método de Newton-Raphson*, ou das *tangentes*, distinguido-se dos métodos anteriores pelo uso de uma fórmula recursiva de um só ponto, do tipo $x_{k+1} = \phi(x_k)$. A ideia base do método consiste em desenvolver uma aproximação local da função em causa, usando esta aproximação para obter o zero da função.

Dado um ponto x_k tal que $f(x_k) \neq 0$ (senão x_k seria já uma solução), vamos escolher um novo ponto x_{k+1} de modo a que $f(x_{k+1}) = 0$. Escrevendo a

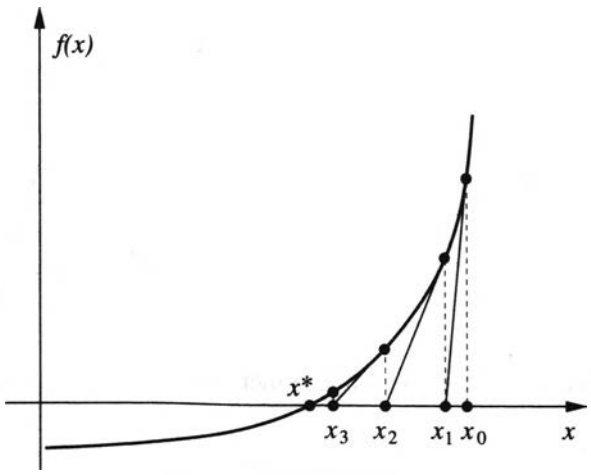


Figura 2.11 Interpretação geométrica da aplicação do método de Newton.

fórmula de Taylor desta expressão, tomando x_k como ponto base, temos

$$0 = f(x_{k+1}) = f(x_k) + f'(x_k)(x_{k+1} - x_k) + \frac{f''(\eta)}{2!}(x_{k+1} - x_k)^2$$

com $\eta \in [\min\{x_k, x_{k+1}\}, \max\{x_k, x_{k+1}\}]$. Geralmente o ponto η não é conhecido, sendo possível escrever, de forma aproximada

$$0 \simeq f(x_k) + f'(x_k)(x_{k+1} - x_k) \tag{2.25}$$

por truncatura da série correspondente a seguir ao termo de primeira ordem. Resolvendo a equação anterior em ordem a x_{k+1}, resulta a expressão iterativa do método de Newton:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{2.26}$$

Deve ser notado que a equação (2.25) constitui uma *aproximação linear* da função $f(x)$. De facto, a recta $y = f(x_k) + f'(x_k)(x - x_k)$ é tangente à curva $y = f(x)$ no ponto x_k. Daí resulta a interpretação geométrica do método ilustrada na Figura 2.11, sendo esta a razão pela qual o método também é conhecido como *método das tangentes*.

Tal como o método das secantes, o método de Newton pode divergir com algumas estimativas iniciais. Este facto é ilustrado na Figura 2.12. No entanto, quando converge, este método é ainda mais rápido que o método das secantes, possuindo *convergência quadrática* ($p = 2$, na equação 2.3). Este resultado é estabelecido de forma rigorosa através do seguinte teorema:

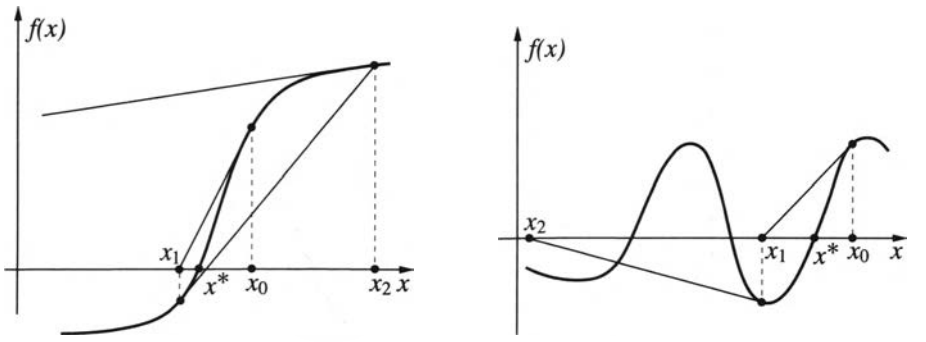


Figura 2.12 Situações onde o método de Newton diverge.

Teorema 2.2 (Convergência do método de Newton) *Se $f(x)$ for uma função contínua, com primeira e segunda derivadas contínuas em $[a,b]$, verificando-se $f(a) \cdot f(b) < 0$ e $f'(x) \neq 0$, $\forall x \in [a,b]$, então o método iterativo de Newton converge* quadraticamente *para $x^* \in [a,b]$ (com $f(x^*) = 0$), desde que x_0 esteja* suficientemente próximo *de x^*.*

Uma perspectiva histórica do desenvolvimento deste método pode ser encontrada em Ypma (1995); Sebah e Gourdon (2001).

A principal desvantagem do método de Newton face ao método das secantes reside na necessidade do cálculo das derivadas. Esta desvantagem é mais severa para funções complexas, ou quando é necessária a resolução de sistemas de equações não-lineares de grandes dimensões, envolvendo muitas variáveis. Frequentemente, estas derivadas são aproximáveis por diferenças finitas (usando, por exemplo, os métodos estudados na Secção 1.2.7), embora essas aproximações possam diminuir ligeiramente a ordem de convergência do método.

Tal como no caso das secantes, a convergência deste método pode depender da proximidade entre x_0 e x^*. Uma vantagem importante do método de Newton é que a sua convergência pode ser *globalizada*, i.e., assegurada a partir de qualquer estimativa inicial da solução, por transformação de (2.26) num método de *Optimização*. A descrição das modificações necessárias está fora do âmbito destas notas, podendo ser consultada por exemplo em Dennis e Schnabel (1983). O uso desta técnica não se estende contudo ao caso do método das secantes, sendo esta uma importante vantagem do método de Newton face ao primeiro.

Apesar do método de Newton ser em geral eficiente, existem algumas situações onde a sua convergência pode ser bastante lenta, por exemplo quando existe "ruído" ou variações rápidas nos valores de $f(x)$, como ilustrado na Figura 2.13.

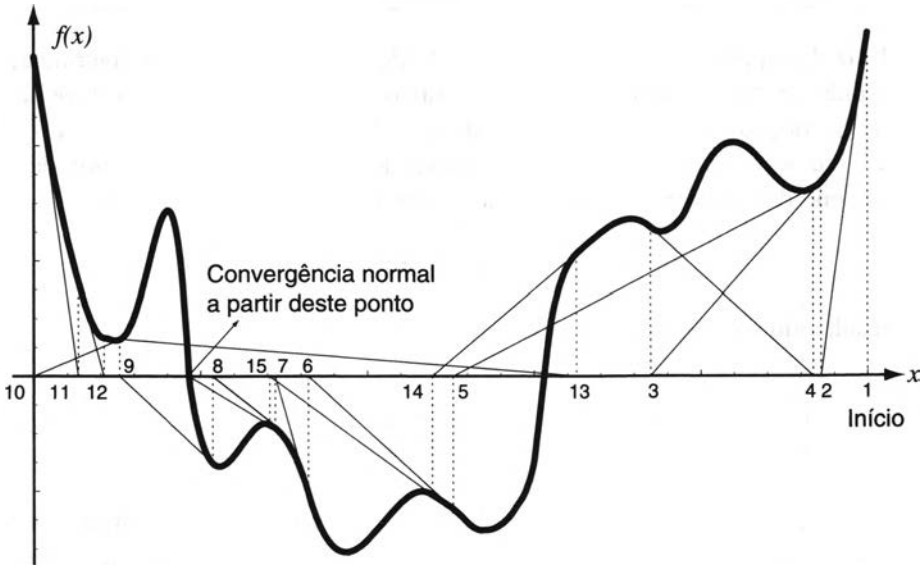


Figura 2.13 Dificuldades de convergência no método de Newton (Rice, 1993). Partindo do ponto 1, o método de Newton comporta-se quase como um método de pesquisa aleatória, até a estimativa chegar a uma vizinhança próxima de uma das soluções. Isto acontece na iteração número 16. A partir daqui, ocorre a convergência muito rápida do método.

2.8.1 Condições suficientes de convergência

A expressão (2.26) tem uma forma genérica $x_{k+1} = \phi(x_k)$, sendo as condições suficientes de convergência do Teorema 2.1 aqui aplicáveis. Neste caso, tem-se:

$$\phi(x) = x - f(x)/f'(x), \qquad \text{e logo} \qquad \phi'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}$$

Consequentemente o método de Newton é sempre convergente quando, nas condições de aplicação do Teorema 2.1, se tem:

$$|\phi'(x)| = \frac{|f(x)f''(x)|}{[f'(x)]^2} \leq L < 1, \qquad \forall x \in [a,b] \tag{2.27}$$

Na prática esta condição raramente é aplicada, uma vez que requer o conhecimento da segunda derivada de $f(x)$ num intervalo englobando a solução.

Exemplo 2.5 (Determinação de raízes quadradas)

Uma das aplicações possíveis do método de Newton consiste na determinação rápida de raízes quadradas, em dispositivos electrónicos que não disponham desta função. Com efeito, para calcular $\sqrt{a}$, pode escrever-se $x = \sqrt{a} \Rightarrow x^2 - a = 0$, coincidindo a raiz positiva desta equação com o valor de $\sqrt{a}$ pretendido. Aplicando o método de Newton à equação anterior fica

$$f(x) = x^2 - a, \quad f'(x) = 2x \quad \text{e} \quad f''(x) = 2$$

resultando a expressão recursiva:

$$\boxed{x_{k+1}} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2 - a}{2x_k} = \boxed{\frac{1}{2}\left(x_k + \frac{a}{x_k}\right)} \tag{2.28}$$

A aplicação recursiva desta equação é bastante simples[3]. No entanto, a sua aplicabilidade depende muito das suas condições de convergência. Usando (2.27) tem-se como condição suficiente

$$|\phi'(x)| = \left|\frac{(x^2 - a)2}{4x^2}\right| \leq L < 1$$

ou seja:

$$\frac{(x^2 - a)}{2x^2} < 1 \quad \wedge \quad \frac{(x^2 - a)}{2x^2} > -1 \quad \Leftrightarrow$$
$$\Leftrightarrow \quad x^2 - a < 2x^2 \quad \wedge \quad x^2 - a > -2x^2 \quad \Leftrightarrow$$
$$\Leftrightarrow \quad x^2 > -a \quad \wedge \quad 3x^2 > a$$

A 1ª condição é sempre satisfeita para $x \in \mathbb{R}$, resultando apenas como condição de convergência (para raízes positivas):

$$\boxed{x > \sqrt{\frac{a}{3}}}$$

Consequentemente, desde que os valores produzidos por (2.28) satisfaçam a relação anterior, o método iterativo tem convergência garantida. O conhecimento deste facto simplifica bastante a tarefa de especificação de estimativas iniciais para este processo iterativo.

[3]Usando, por exemplo, aritmética binária, torna-se necessário efectuar apenas uma operação de adição e de divisão por iteração, uma vez que a multiplicação pelo coeficiente 1/2 corresponde simplesmente ao deslocamento da vírgula fraccional de uma unidade para a esquerda.

Problema 2.6 *Os microprocessadores com menos recursos, usados por exemplo em controladores autónomos, implementam por vezes em* hardware *apenas as operações de adição (e subtracção), e multiplicação. Neste caso, as operações de divisão necessitam ser implementadas em* software. *Devido à sua rapidez de convergência, o método de Newton pode ser útil neste caso, reformulando os termos b/a através do produto $b \times (1/a)$. A quantidade $(1/a)$ pode ser calculada através da solução da equação não-linear:*

$$f(x) = a - 1/x = 0$$

Usando esta informação, e considerando $a > 0$, efectue uma análise semelhante à do exemplo anterior, determinando como o método de Newton poderia ser usado para avaliar os termos $(1/a)$, e as respectivas condições de convergência do procedimento.

2.8.2 Zeros múltiplos

A presença de soluções múltiplas *simultâneas* de $f(x) = 0$ (Figura 2.14) pode também introduzir dificuldades na aplicação do método de Newton. O conceito de zeros múltiplos é facilmente apreensível no caso de funções polinomiais, podendo ser entendido a outras funções através da seguinte definição (Pina, 1995):

Definição 2.1 *A* multiplicidade *m de uma raiz x^* da equação $f(x) = 0$ corresponde ao* maior (supremo) *dos valores de k onde:*

$$\lim_{x \to x^*} \frac{|f(x)|}{|x - x^*|^k} = c \quad \text{(finito)}$$

As raízes com multiplicidade $m = 1$ são ditas simples, *com multiplicidade $m = 2$ são* duplas, *e assim sucessivamente.*

Caso $f(x)$ seja suficientemente diferenciável na vizinhança de um zero x^* de multiplicidade m tem-se também

$$f(x^*) = f'(x^*) = \cdots = f^{(m-1)}(x^*) = 0$$

com $f^{(m)}(x^*) \neq 0$. Neste caso, na vizinhança de x^* é possível aproximar

$$f(x) \simeq (x - x^*)^m g(x)$$

onde $g(x)$ é uma função contínua com $g(x^*) \neq 0$, em $x = x^*$.

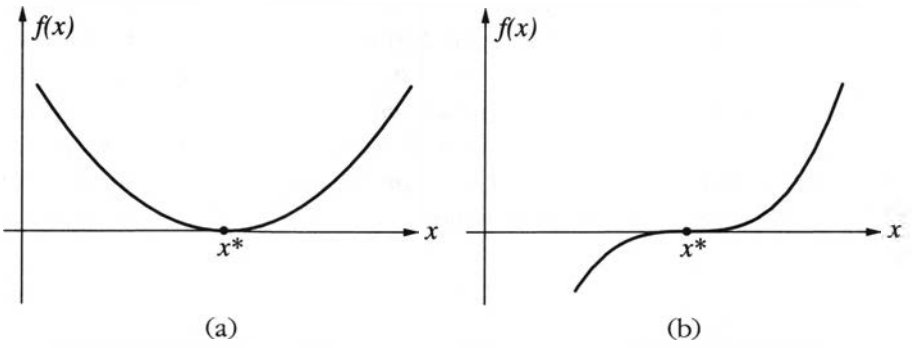


Figura 2.14 Funções com múltiplos zeros simultâneos: (a) Multiplicidade dupla. (b) Multiplicidade tripla.

Para $m \geq 2$ torna-se portanto necessário alterar a equação (2.26), para calcular o limite de $f(x_k)/f'(x_k)$ à medida que nos aproximamos da solução. Contudo, a definição deste limite pode requerer a avaliação de derivadas de ordem superior de $f(x)$, tornando por vezes complexa a aplicação do método de Newton nestes casos. Estas dificuldades são igualmente partilhadas pelos outros métodos descritos anteriormente. Por exemplo no caso do método das secantes, como se viu, é também utilizada uma "estimativa" da derivada da função na vizinhança de x^*, podendo ocorrer uma divisão por zero caso não sejam tomadas medidas especiais na implementação deste método. Também a aplicabilidade dos métodos anteriores que recorrem à delimitação prévia das raízes se encontra bastante restringida nestes casos, e em especial quando não existe uma variação de sinal de $f(x)$ de cada um dos lados de x^* (por exemplo na Figura 2.14 (a)).

Mesmo quando as dificuldades numéricas são evitadas, o método de Newton na forma (2.26) pode apenas apresentar convergência linear para raízes de multiplicidade $m \geq 2$, sendo conhecidas algumas modificações para restaurar a sua ordem de convergência quadrática (Chapra e Canale, 1989).

Problema 2.7 *Uma alternativa sugerida para ultrapassar as dificuldades de aplicação do método de Newton a um problema $f(x) = 0$ com raízes múltiplas simultâneas consiste em definir uma nova função*

$$\phi(x) = \frac{f(x)}{f'(x)}$$

e aplicar antes o método de Newton à equação $\phi(x) = 0$.

1. *Em que condições terá $\phi(x)$ zeros nas mesmas localizações que $f(x)$?*

2. *Mostre que a aplicação do método de Newton a* $\phi(x) = 0$ *conduz à expressão iterativa:*

$$x_{k+1} = x_k - \frac{f(x_k)f'(x_k)}{[f'(x_k)]^2 - f(x_k)f''(x_k)} \tag{2.29}$$

3. *Compare o desempenho numérico de* (2.26) *e de* (2.29) *no problema:*

$$f(x) = x^3 - 3x^2 + 3x - 1 = 0$$

Utilize a estimativa inicial $x_0 = 0{,}5$ *em ambos os casos.*

4. *Tendo presente o conceito de* índice de condição *descrito na Secção 1.2.6, e o Exemplo 1.5, comente a robustez numérica da aplicação da expressão* (2.29) *nos pontos de estacionaridade de* $f(x)$, *i.e., onde* $f'(x) = 0$. *Sugestão: Utilize a equação* $f(x) = (x-1)^2 \sin(x) = 0$.

2.9 Comparação das velocidades de convergência

Tal como foi mencionado, é de esperar que os métodos iterativos com maior ordem de convergência se aproximem mais rapidamente da solução, por cada iteração efectuada. Este facto é ilustrado no exemplo seguinte. No entanto, e para além deste critério, a escolha do algoritmo mais apropriado para a resolução de um dado problema deve ainda ter em consideração outros critérios, tais como a natureza da função, a disponibilidade de derivadas da mesma, ou a exactidão pretendida para a solução.

Exemplo 2.6 (Convergência de diferentes métodos iterativos)

Para comparar a eficiência dos métodos iterativos descritos, considere-se a resolução numérica da equação

$$f(x) = 10e^{-10x} - \frac{1}{10} = 0 \tag{2.30}$$

com solução analítica:

$$x^* = \frac{\ln 100}{10} \simeq 0{,}4605$$

O comportamento desta função no intervalo $x^* \in [0,2]$ encontra-se representado na Figura 2.15. Este é típico de numerosas funções encontradas na prática: possui um andamento regular, com fortes variações de curvatura em determinadas regiões do seu domínio. Para melhor ilustração, são representadas na Figura 2.16 as 1ª e 2ª derivadas desta função.

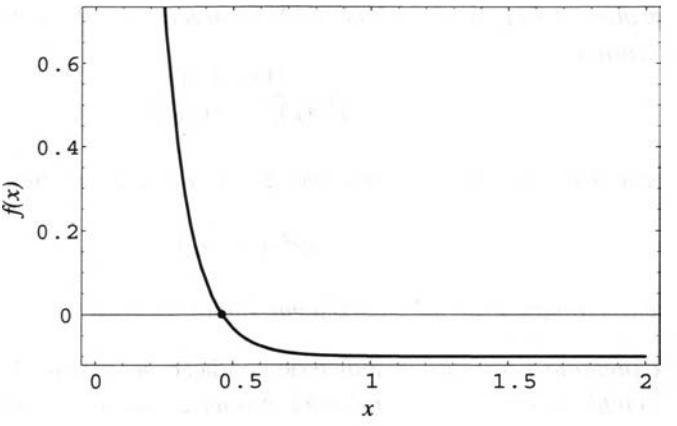


Figura 2.15 Localização das raízes da equação (2.30).

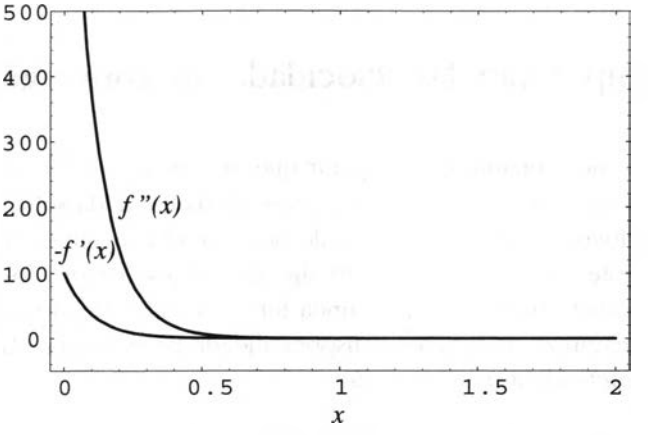


Figura 2.16 Primeiras derivadas de $f(x)$ na equação (2.30).

A aplicação do método de Newton ou de outros que aproximem o comportamento por rectas para $x > 1$ aparenta ser problemática neste problema, sendo fácil antecipar neste caso a divergência dos métodos anteriores. O comportamento mais seguro corresponde neste caso ao dos métodos que efectuam uma delimitação prévia das soluções do problema.

Partindo das estimativas iniciais $x_0 = 0{,}35$ e $x_{-1} = 1$ (quando necessário para iniciar o procedimento iterativo ou para delimitar previamente a solução), a Figura 2.17 ilustra a variação do erro absoluto na localização desta raiz, para cada um dos métodos atrás descritos[4]. A informação contida na Figura 2.17 pode

[4]Nesta ilustração, os cálculos numéricos correspondentes foram efectuados com uma precisão superior à habitual, por forma a permitir uma visualização correcta do desempenho relativo dos vários métodos numa gama *alargada* de precisões.

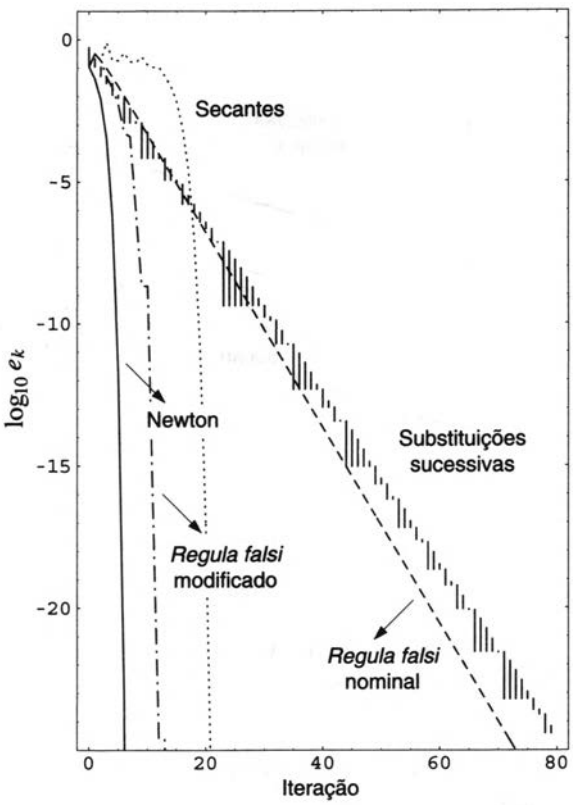


Figura 2.17 Evolução dos erros absolutos $e_k = |x_k - x^*|$ durante a aplicação dos diferentes métodos iterativos a (2.30).

também ser usada para confirmar numericamente a ordem de convergência de cada um dos métodos usados. Definindo o erro absoluto $e_k = |x^* - x_k|$, para um método iterativo com ordem de convergência p, tem-se da equação (2.3)

$$e_{k+1} \simeq c e_k^p$$

ou seja:

$$\log e_{k+1} \simeq \log c + p \log e_k$$

Deste modo, um gráfico de $\log e_{k+1}$ em função de $\log e_k$ permitirá confirmar a ordem de convergência do respectivo método iterativo através do *declive* da recta obtida. A Figura 2.18 apresenta esta informação correspondente à aplicação dos diferentes métodos testados com a equação (2.30).

Como se pode observar, dentro dos algoritmos estudados, o método de Newton é aquele que apresenta a convergência mais rápida, permitindo localizar a raiz pretendida com qualquer exactidão (na gama estudada) em menos de

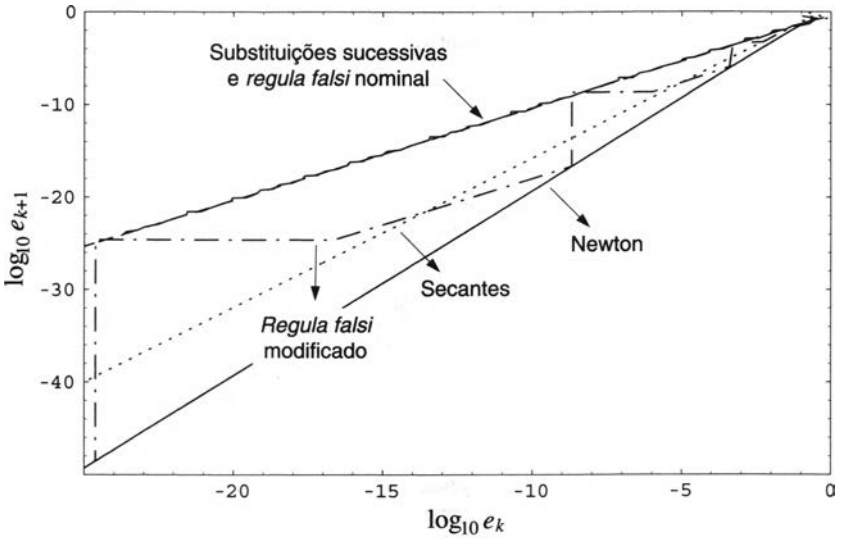


Figura 2.18 Confirmação numérica das ordens de convergência dos diferentes métodos iterativos do Exemplo 2.6.

7 iterações. O método da *regula falsi* modificado tem também neste caso um desempenho bastante eficiente, embora requeira cerca do dobro das iterações do método de Newton para localizar a raiz com a mesma incerteza. Como o método da *regula falsi* modificado não requer a avaliação das derivadas de $f(x)$, o desempenho prático destes dois métodos pode ser considerado como comparável, *nesta aplicação*.

A Figura 2.18 revela também que, para esta função, a convergência do método da *regula falsi* oscila entre a 1ª e a 2ª ordens, consoante o método utiliza o algoritmo nominal ou as modificações nele introduzidas para acelerar a sua convergência. Por si só, o método da *regula falsi* nominal apenas consegue atingir uma convergência linear neste problema, devido à convexidade de $f(x)$. A este comportamento está também associado o facto de as estimativas produzidas pelo método da *regula falsi* nominal estarem localizadas *sempre* à direita de x^*, coincidindo o extremo esquerdo do intervalo final de delimitação da solução com o valor homólogo do intervalo inicial! Devido a isto, o seu desempenho neste problema torna-se comparável ao do método das substituições sucessivas.

Na Figura 2.18 é também possível constatar a ordem de convergência *superlinear*[5] do método das secantes. Este método revela-se contudo mais sensível às estimativas iniciais que os métodos anteriores; por exemplo, a sua aplica-

[5]Com um coeficiente $p > 1$, na equação (2.3).

ção a partir de $x_0 = 0$ diverge, enquanto que o método de Newton demora apenas 2 ou 3 iterações adicionais para convergir a partir deste ponto. Neste exemplo, as secantes produzidas durante as primeiras iterações do método são bastante afastadas das respectivas tangentes, atrasando um pouco o progresso deste método durante a sua fase inicial.

2.10 Outros métodos

Usando as metodologias empregues na elaboração dos métodos das secantes e de Newton, é possível construir outros algoritmos que aproximem a função em causa por polinómios de ordem superior. Por exemplo, generalizando a formulação do método das secantes por forma a aproximar $f(x)$ por um polinómio de segunda ordem, obtém-se o *método de Muller* (Conte e de Boor, 1981). Este método tem convergência quadrática, na proximidade da solução, podendo também ser utilizado para a determinação de pares de raízes complexas de equações. Para equações polinomiais, analisadas em maior detalhe na Secção seguinte, é também preferível a utilização de métodos específicos.

Uma outra alternativa para a determinação de raízes de equações não-lineares gerais é o método de Brent. Este método combina a delimitação prévia das raízes, o método da bissecção e uma aproximação quadrática da função em causa, proporcionando um método robusto e rápido de resolução de equações não-lineares, permitindo uma convergência *superlinear*, usando apenas valores da função $f(x)$ (Brent, 1973). Devido à sua complexidade, a generalização deste algoritmo para sistemas de equações não-lineares é no entanto complexa. Uma implementação computacional para o problema unidimensional é apresentada em Press et al. (1992).

Na escolha de métodos iterativos deve ter-se em consideração o número de iterações necessárias (relacionado com a ordem de convergência do método), e o esforço ou volume de cálculos necessário *por iteração*. Por sua vez, estas características estão também relacionadas com o custo relativo da avaliação da função. Com funções cuja avaliação seja demorada (requerendo, por exemplo, a implementação de outros métodos numéricos), são preferíveis métodos que requeiram relativamente menos avaliações de $f(x)$ (o que geralmente coincide com métodos de ordem mais elevada). Para funções menos complexas, pode ser preferível a utilização de métodos iterativos também mais simples, uma vez que o tempo gasto no cálculo de $f(x)$ não vai ser tão desfavorável. Finalmente, deve também ser ponderada a complexidade da tarefa relativa ao cálculo (ou estimativa) das derivadas necessárias.

2.11 Equações polinomiais

Os métodos gerais descritos anteriormente podem ser utilizados para resolver equações polinomiais

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0 \tag{2.31}$$

com $a_n \neq 0$. No entanto, a sua forma específica, associada à sua ubiquidade, tornam vantajoso o estudo de alguns aspectos particulares destas equações. As funções polinomiais são bastante flexíveis, adaptando-se bem à descrição aproximada do comportamento da maioria das funções "regulares", sendo ainda bastante fáceis de manipular.

Em alguns casos é necessário conhecer uma solução da equação anterior com significado especial. Por exemplo, um problema de pH pode requerer a determinação de uma raiz positiva de uma equação deste tipo. Um problema de estabilidade de sistemas dinâmicos pode requerer encontrar (ou delimitar) a raiz de maior magnitude, ou com maior parte real, de um polinómio. Noutros casos pode ser necessário encontrar *todas* as raízes de uma equação polinomial. Algumas destas podem ser complexas, sendo necessárias algumas modificações nos métodos anteriores para lidar com este facto[6].

Um resultado importante é o Teorema Fundamental da Álgebra, que garante que *todos os polinómios de grau n têm exactamente n raízes (ou zeros), reais ou complexas.* É também conhecido o facto de que se os coeficientes $\{a_0, a_1, \ldots, a_n\}$ forem reais, as raízes complexas do polinómio (2.31) aparecem em pares conjugados $x_{1,2} = a \pm jb$. Como consequência, qualquer polinómio de grau ímpar (i.e., com n ímpar), com coeficientes reais, tem pelo menos uma raiz real.

Existem outros resultados que permitem estabelecer limites para a localização de raízes de equações polinomiais, descritos seguidamente.

2.11.1 Regra dos sinais de Descartes

Esta regra especifica que o número de raízes *reais positivas* de $p(x) = 0$, com $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, é igual ao número de *variações de sinal* da sucessão de coeficientes $a_0, a_1, \ldots, a_n$, ou um número inferior, da mesma paridade.

[6]Por exemplo, o método de Newton, implementado usando aritmética complexa, pode ser usado para determinar raízes complexas de equações; existem no entanto outros métodos mais eficientes para a resolução de equações polinomiais.

Por exemplo, se

$$p(x) = x^4 \underbrace{-3}_{1+} x^3 \underbrace{+4}_{1+} x^2 + 5x \underbrace{-3}_{1} = 0 \tag{2.32}$$

este polinómio terá 3 ou 1 raízes positivas. Se o número de variações de sinal for par, é possível que um dado polinómio tenha 0 raízes reais positivas. Os termos com coeficientes nulos podem ser ignorados. Por exemplo

$$p(x) = x^4 + 4x^2 + 5x \underbrace{-3}_{1} = 0$$

terá apenas uma raiz real positiva.

O número de raízes negativas pode ser contado do mesmo modo, usando a equação $p(-x) = 0$. No primeiro dos exemplos anteriores, temos

$$\begin{aligned} p(-x) = (-x)^4 - 3(-x)^3 + 4(-x)^2 + 5(-x) - 3 = 0 \quad &\Leftrightarrow \\ x^4 + 3x^3 + 4x^2 \underbrace{-5}_{1} x - 3 = 0 \end{aligned}$$

ou seja apenas uma raiz real negativa. Resumindo toda a informação relativa a este polinómio, temos dois cenários possíveis:

1 raiz real negativa		1 raiz real negativa
3 raízes reais positivas	ou	1 raiz real positiva
		2 raízes complexas conjugadas
4 raízes reais		2 raízes reais + 2 raízes complexas

Esta regra pode também ser aplicada por forma a delimitar o número de raízes *reais* de um polinómio à esquerda e à direita de um ponto arbitrário x_0 no eixo real[7]. Para o efeito, basta aplicar a regra de Descartes ao polinómio $p(x + x_0)$ e contar o número de variações de sinais dos seus coeficientes (ou do seu simétrico). Por exemplo, no polinómio (2.32) o número de raízes à direita de $x_0 = 1$ pode ser estimado a partir do número de variações de sinal dos coeficientes de:

$$\begin{aligned} g(x) = p(x+1) &= (x+1)^4 - 3(x+1)^3 + 4(x+1)^2 + 5(x+1) - 3 \\ &= x^4 + x^3 + x^2 + 8x + 4 \end{aligned}$$

Este polinómio não apresenta variações de sinal nos seus coeficientes, e logo

[7]Desde que x_0 não coincida com uma raiz do próprio polinómio $p(x)$.

$p(x)$ não tem raízes reais à direita de $x_0 = 1$. O seu simétrico

$$g(-x) = x^4 - x^3 + x^2 - 8x + 4$$

apresenta 4 variações de sinal nos seus coeficientes. Conjugando esta informação com a informação obtida anteriormente são possíveis pois os seguintes cenários:

1 raiz real negativa		1 raiz real negativa
3 raízes reais positivas (≤ 1)	ou	1 raiz real positiva (≤ 1)
4 raízes reais		2 raízes complexas conjugadas
		2 raízes reais + 2 raízes complexas

2.11.2 Delimitação de zeros de polinómios

Se

$$\rho = 1 + \max_{0 \leq i \leq n-1} \frac{|a_i|}{|a_n|} \tag{2.33}$$

então todos os zeros de $p(x)$ em (2.31) estão localizados no interior de um círculo no plano complexo, com centro na origem e raio ρ (Conte e de Boor, 1981).

Para o polinómio anterior temos:

$$\rho = 1 + \max\left\{\frac{3}{1}, \frac{4}{1}, \frac{5}{1}, \frac{3}{1}\right\} = 6$$

Os zeros do polinómio, juntamente com o círculo resultante da aplicação deste critério estão ilustrados na Figura 2.19. Como se pode observar, este critério produz frequentemente uma estimativa conservadora (i.e., bastante alargada) da localização das raízes do polinómio. A transformação de variáveis usada com a regra de Descartes permite também, de forma semelhante, desenhar um círculo centrado em qualquer ponto do plano complexo que englobe todos os zeros de um polinómio. A aplicação repetida deste critério produz frequentemente regiões bastante alargadas, não sendo particularmente eficaz a sua intersecção com vista à obtenção de uma região mais reduzida demarcando, de forma eficaz, a localização das raízes de equações polinomiais.

Uma outra possibilidade de delimitação da região que contém as raízes de um polinómio $p(x)$ dado por (2.31) consiste na construção do polinómio (Kincaid e Cheney, 1991):

$$q(x) = x^n p(1/x) \tag{2.34}$$

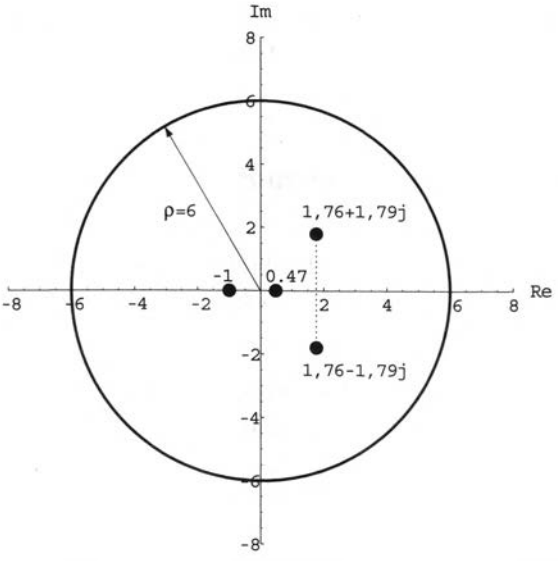


Figura 2.19 Localização das raízes do polinómio, e a região resultante da aplicação de (2.33).

Neste caso:

$$\begin{aligned} q(x) &= x^n \left(a_n \left(\frac{1}{x}\right)^n + a_{n-1} \left(\frac{1}{x}\right)^{n-1} + \cdots + a_0 \right) \\ &= a_n + a_{n-1} z + a_{n-2} z^2 + \cdots + a_0 z^n \end{aligned}$$

Facilmente se verifica que $q(x)$ possui os mesmos coeficientes que $p(x)$, embora utilizados pela ordem inversa. Assim, $q(x)$ tem também ordem máxima n. A partir de (2.34) facilmente se verifica que, para qualquer valor $x_0 \neq 0$, a equação $p(x_0) = 0$ é equivalente a $q(1/x_0) = 0$ (i.e., as raízes destes dois polinómios são inversas). Deste modo, a aplicação do critério (2.33) a $q(x)$ permite concluir que se todos os zeros de $q(x)$ estão localizados no interior de um círculo de raio ρ centrado na origem, então todas as raízes de $p(x) = 0$ diferentes de 0 estão fora de um círculo correspondente, de raio $1/\rho$, centrado também na origem.

Para o polinómio anterior temos

$$q(x) = -3x^4 + 5x^3 + 4x^2 - 3x + 1$$

e:

$$\rho = 1 + \max\left\{\frac{5}{3}, \frac{4}{3}, \frac{3}{3}, \frac{1}{3}\right\} = \frac{8}{3}$$

Consequentemente, todas as raízes de $p(x)$ estão localizadas no exterior de um círculo centrado na origem e com raio $1/\rho = 3/8 = 0{,}375$.

2.11.3 Manipulação de polinómios

A forma expandida (2.31) não é normalmente empregue no cálculo do valor de funções polinomiais. Existem duas razões fundamentais que desfavorecem a sua aplicação:

- Esta forma requer demasiadas operações, quando comparada com outras alternativas (são necessárias no mínimo n adições e $2n-1$ multiplicações).
- Para polinómios de ordem moderadamente elevada, as potências x^n tornam-se quantidades grandes. A sua adição algébrica pode provocar importantes problemas de perda de precisão no cálculo do valor da função.

Na avaliação do valor de funções polinomiais é habitualmente usada a forma factorizada:

$$p(x) = ((\cdots(a_n x + a_{n-1})x + \cdots a_2)x + a_1)x + a_0$$

Esta expressão é conhecida como a *forma de Horner*, requerendo n adições e n multiplicações, e não envolvendo o cálculo de potências. Usando esta forma, o polinómio (2.32) pode ser escrito como

$$p(x) = (((x-3)x+4)x+5)x-3$$

88 em vez de:

$$p(x) = x^4 - 3x^3 + 4x^2 + 5x - 3$$

Esta forma é frequentemente codificada de modo recursivo através do seguinte algoritmo:

Algoritmo 2.2 (Método de Horner para avaliar $p_0 = p(x_0)$)

$p_0 = a_n$
Para $i = n-1, n-2, \ldots, 0$ **fazer**
    $p_0 = p_0 x_0 + a_i$
Fim_Para
Devolver p_0.

A aplicação deste algoritmo para calcular $p(x_0)$ é também frequentemente ilustrada pelo seguinte quadro (para quantidades reais):

	a_n	a_{n-1}	a_{n-2}	a_{n-3}	$\cdots$	a_0	
$+\ x_0$		$x_0 a_n$	$x_0 b_{n-1}$	$\cdots$	$\cdots$		
	$\underbrace{a_n}_{b_n}$	$\underbrace{(x_0 a_n + a_{n-1})}_{b_{n-1}}$	$\underbrace{(x_0 b_{n-1} + a_{n-2})}_{b_{n-2}}$	$\cdots$	$\cdots$	$\boxed{b_0} \equiv p(x_0)$	

Por exemplo, para calcular $p(3)$ em (2.32), o quadro correspondente será

	1	-3	4	5	-3
$+\ 3$		3	0	12	51
	1	0	4	17	$\boxed{48}$

sendo $p(3) = 48$. Existem modificações possíveis neste quadro para facilitar o cálculo de valores de funções com argumentos x_0 complexos, ou para polinómios com coeficientes complexos. Uma outra característica importante da forma de Horner reside na facilidade com que é paralelizado, para execução eficiente em sistemas multi-processador (Dowd e Severance, 1998).

O desempenho das formas polinomiais anteriores é comparado na Figura 2.20, usando neste caso um exemplo bastante conhecido, onde $p(x) = (x-1)^6$ (Conte e de Boor, 1981). Neste caso compara-se o comportamento numérico desta função, avaliada usando as formas *expandida*

$$p(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$$

e *factorizada*

$$p(x) = (((((x-6)x+15)x-20)x+15)x-6)+1$$

na vizinhança da sua raiz múltipla. Como se pode observar, as dificuldades na localização exacta são consideravelmente superiores no primeiro caso. As vantagens da utilização da forma factorizada, tanto em termos de precisão da resposta como em termos do número total de operações necessárias, são óbvias neste exemplo.

Problema 2.8 *Considerando a Figura 2.20, investigue as principais razões que justificam que a amplitude do "ruído" numérico seja diferentes nos dois lados do eixo das ordenadas, em cada um dos casos. (Sugestão: Considere o efeito do cancelamento de termos de magnitudes semelhantes, analisado no Exemplo 1.5.)*

O exemplo anterior mostra que na avaliação de funções polinomiais, o uso da forma de Horner pode, em geral, ser vantajoso relativamente à sua forma

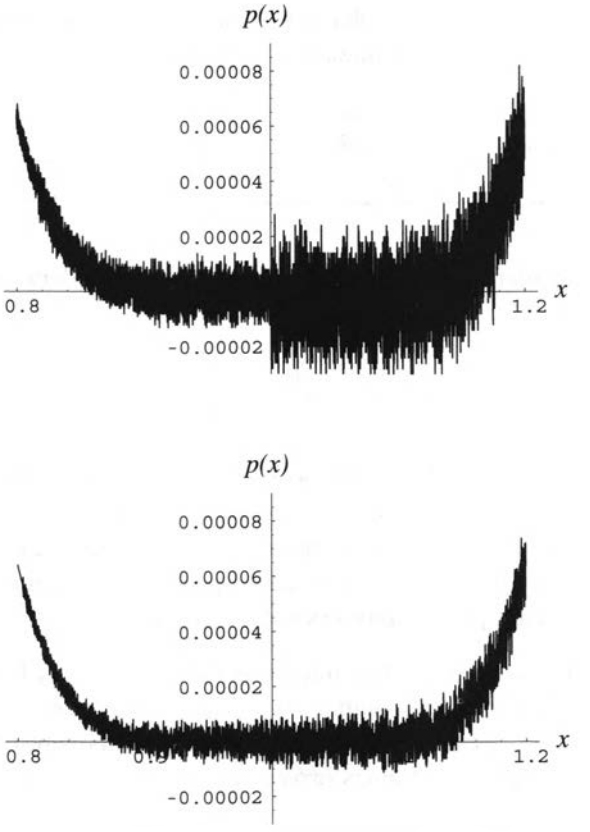


Figura 2.20 Gráfico dos valores do polinómio $p(x) = (x-1)^6$, efectuando os cálculos com uma precisão de 7 algarismos. Em (a) é usada a *forma expandida* do polinómio, e em (b) a *forma de Horner* correspondente. As curvas representadas foram traçadas por amostragem da função num conjunto discreto de pontos.

expandida. Contudo, este 1º método não é por si só totalmente imune à produção de erros numéricos diversos (como se pode observar no exemplo anterior), especialmente em redor das raízes do polinómio em causa. Um volume substancial de literatura mostra que existem diversas alternativas, envolvendo nalguns casos menos operações algébricas, noutros casos permitindo aproximações mais precisas do que a forma de Horner correspondente (Press et al., 1992; Higham, 1996; Knuth, 1997; McNamee, 2002). Apesar disto, este método pode ser considerado como um bom ponto de partida, correspondente a um algoritmo simples e eficiente para o cálculo de funções polinomiais.

O método anterior pode também ser usado para obter as derivadas de funções polinomiais que, sendo também polinómios, deverão ser avaliadas do mesmo

modo. Isto é ilustrado no algoritmo seguinte, avaliando simultaneamente a função e a sua derivada (Press et al., 1992).

Algoritmo 2.3 (Método de Horner para avaliar $p_0 = p(x_0)$ e $d_0 = p'(x_0)$)

$p_0 = a_n$
$d_0 = 0$
`Para` $i = n-1, n-2, \ldots, 0$ `fazer`
    $d_0 = d_0 x_0 + p_0$
    $p_0 = p_0 x_0 + a_i$
`Fim_Para`
`Devolver` p_0, d_0.

Deflação de polinómios

O método de Horner pode também ser usado para efectuar a *deflação de polinómios*, quando se pretendem encontrar todas as raízes de $p(x)$. Com efeito, depois de conhecida uma primeira raiz x_1, é preferível factorizar

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = (x - x_1) q(x)$$

sendo $q(x)$ um polinómio de grau $n-1$, e prosseguir encontrando as raízes de $q(x)$. Isto evita que o método iterativo volte a convergir para as raízes já encontradas (a menos que estas sejam múltiplas). Nesta expressão, os coeficientes de $q(x)$ correspondem aos termos b_i obtidos na construção do quadro anterior.

Exemplificando com o polinómio anterior de quarta ordem, e admitindo que já é conhecida a raiz $x^* = -1$, temos o quadro seguinte

	1	−3	4	5	−3
+ (−1)		−1	4	−8	3
	1	−4	8	−3	0

e logo:

$$x^4 - 3x^3 + 4x^2 + 5x - 3 = (x+1) \underbrace{(x^3 - 4x^2 + 8x - 3)}_{q(x)}$$

2.11.4 Outros métodos

Os algoritmos descritos anteriormente apenas permitem a determinação de raízes reais, quando implementados em aritmética real, e com estimativas iniciais

também reais. Contudo, se os polinómios não possuírem raízes reais, os métodos podem divergir. A determinação de raízes complexas pode ser efectuada usando uma implementação dos métodos numéricos anteriores em aritmética complexa (por exemplo com o método de Newton). Se os polinómios tiverem coeficientes reais, as suas raízes complexas aparecem em pares conjugados, sendo mais eficiente a sua determinação também aos pares. Para além do método de Muller, já mencionado, os algoritmos mais indicados para o efeito são os métodos de Bairstow ou de Laguerre (Kincaid e Cheney, 1991), ou ainda o método de Jenkins-Traub (Jenkins e Traub, 1970; Pan, 1997).

Problema 2.9 *Usando o método de Horner, construa o algoritmo correspondente ao método de Newton para a determinação de zeros de funções polinomiais:*

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

2.12 Sistemas de equações não-lineares

Os métodos anteriores podem ser adaptados para a resolução de *sistemas* de equações não-lineares

$$f(x) = 0$$

onde $f(\cdot)$ e $x \in \mathbb{R}^n$. A aplicação das transformações sugeridas na Secção 2.1 nem sempre é viável, sendo portanto necessária, em alguns casos, a sua solução conjunta. Um estudo aprofundado da robustez e convergência destes métodos requer o uso de conceitos de Optimização, estando por isso fora do âmbito destas notas. Aqui, procurar-se-á sobretudo fornecer uma ideia geral do funcionamento destas extensões.

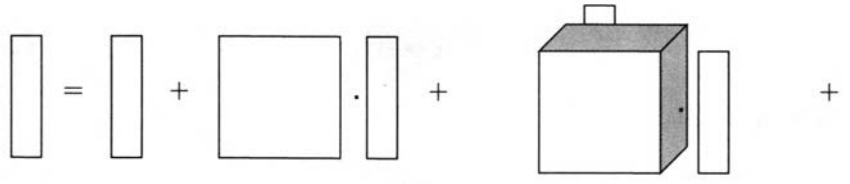
2.12.1 Método de Newton

Tal como no caso escalar, o método de Newton tem por base uma expansão semelhante à série de Taylor de uma função vectorial $f(x) \in \mathbb{R}^n$, com $x \in \mathbb{R}^n$ também. Tomando como ponto base $x_0 \in \mathbb{R}^n$, esta expansão pode ser escrita numa forma simplificada usando uma notação vectorial:

$$f(x) = f(x_0) + \left(\frac{\partial f^{\mathrm{T}}}{\partial x}\right)^{\mathrm{T}}_{x=x_0} (x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}} \left(\frac{\partial^2 f}{\partial x^2}\right)_{x=x_0} (x - x_0) + \\ + O(\|x - x_0\|^3)$$

Nesta expressão, a quantidade $O(\|x - x_0\|^3)$ representa os termos restantes da expansão[8], de ordem igual ou superior a 3. É no entanto importante realçar que não é possível expressar aqui o resto de ordem n de forma semelhante a (1.8), uma vez que o teorema do valor médio não é aplicável nesta situação[9]. Para uma melhor elucidação, a estrutura vectorial desta equação é a seguinte:

$$f(x) = f(x_0) + \left(\frac{\partial f^{\mathrm{T}}}{\partial x}\right)^{\mathrm{T}}_{x=x_0}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}}\left(\frac{\partial^2 f}{\partial x^2}\right)_{x=x_0}(x - x_0) +$$



$$+ O(\|x - x_0\|^3)$$

A matriz de primeiras derivadas tem o nome de *Jacobiano*, possuindo a forma:

$$\frac{\partial f^{\mathrm{T}}}{\partial x} \equiv \nabla f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{2.35}$$

Esta matriz não é simétrica, requerendo algum cuidado na sua utilização[10]. Como se pode observar, o conjunto das segundas derivadas forma um tensor (uma entidade com 3 dimensões), não sendo contudo necessária a sua utilização no método de Newton.

Tal como no caso escalar, o método de Newton pode ser deduzido partindo de um ponto x_k onde $f(x_k) \neq 0$. Pretende-se escolher um incremento $h \in \mathbb{R}^n$, de forma a que $f(x_k + h) = 0$. Expandindo esta equação, fica:

$$f(x_k + h) = f(x_k) + \nabla f(x_k)^{\mathrm{T}} h + \cdots = 0$$

[8] A notação $O(\cdot)$ ("ordem de") é usada por vezes para identificar os termos assimptoticamente dominantes.

[9] Por exemplo, cada componente f_i poderá satisfazer individualmente este teorema num ponto intermédio η diferente. Esta situação é portanto distinta da expansão de uma função escalar $f(x)$ com $x \in \mathbb{R}^n$, considerada na Secção 1.2.5.

[10] Alguns autores preferem definir a matriz Jacobiana na forma transposta da equação (2.35). Isto permite poupar o sinal de transposto em muitas das equações seguintes. Na prática, qualquer destas definições é equivalente, desde que usada de forma coerente.

Truncando esta equação depois dos termos de primeira ordem vem:

$$f(x_k) + \nabla f(x_k)^{\mathrm{T}} h \simeq 0$$

A notação desta última equação pode ser simplificada, denotando a matriz Jacobiana $\nabla f(x_k)$ apenas por J_k, e $f(x_k)$ por f_k. Se J_k for uma matriz *não-singular* a equação anterior permite obter, de forma única, uma correcção h_k que nos aproxima da solução. Este valor pode ser obtido a partir da solução do sistema de equações lineares de ordem n

$$J_k^{\mathrm{T}} h_k = -f_k \tag{2.36}$$

ou de forma explícita através de:

$$h_k = x_{k+1} - x_k = -(J_k^{\mathrm{T}})^{-1} f_k = -J_k^{-\mathrm{T}} f_k$$

Embora estas duas formas sejam teoricamente equivalentes quando J_k é uma matriz não-singular, na prática prefere-se o uso da equação (2.36), uma vez que a resolução de um sistema de equações lineares é geralmente um processo mais rápido e mais robusto, do ponto de vista numérico, do que o cálculo da inversa da matriz correspondente, conforme analisado no Capítulo seguinte.

O método de Newton aplicado a um sistema de equações não-lineares requer portanto a resolução de um sistema de equações lineares, de igual dimensão, *em cada iteração*. Alguns métodos eficientes para a resolução de sistemas lineares são descritos no Capítulo seguinte.

Os requisitos necessários para a convergência do método de Newton baseado em (2.36), onde $\|x^* - x_k\| \to 0$, são conhecidos desde o trabalho de Kantarovich (1948):

94

1. $f(x)$ deve ser diferenciável *duas vezes*, devendo existir x^* tal que $f(x^*) = 0$.
2. $|J_k| \neq 0$, para $k = 1, 2, \ldots$, por forma a garantir a unicidade de h_k.
3. o ponto inicial x_0 deve estar localizado "próximo" da solução.

Tal como no caso escalar, quando o método de Newton (2.36) converge, possui convergência *quadrática* na vizinhança da solução, verificando-se neste caso:

$$\frac{\|x^* - x_{k+1}\|}{\|x^* - x_k\|^2} \leq c$$

Esta equação é semelhante a (2.3), devendo notar-se neste caso o uso da *norma* em vez do módulo do erro de aproximação (no caso escalar), como medida desta distância. Os restantes critérios de paragem, definidos na Secção 2.2.1

para o caso escalar, podem ser adaptados de modo semelhante a esta situação, através do uso das normas vectoriais definidas na Secção 1.3.

Exemplo 2.7 (Método de Newton com 2 equações não-lineares)

A Tabela 2.2 ilustra a aplicação do método de Newton ao sistema não-linear:

$$\begin{cases} y - \exp(-x) = 0 \\ x - \sin(y) = 0 \end{cases} \tag{2.37}$$

Como se pode verificar, 5 iterações são suficientes para produzir uma estimativa da solução com cerca de 5 algarismos significativos correctos.

Apesar da sua convergência rápida, uma das principais desvantagens do método de Newton é o facto de este requerer derivadas das funções correspondentes. No caso de sistemas de equações não-lineares, este requisito facilmente se torna limitante, mesmo em problemas de média dimensão. Por exemplo, um sistema de dimensão $n = 10$ requer o cálculo de $10 \times 10 = 100$ derivadas individuais. Para $n = 100$ já são necessárias 10^4 derivadas. As principais dificuldades são o volume de cálculos envolvidos, e a possibilidade de ocorrência de erros na codificação das derivadas, quando estas são fornecidas pelo utilizador.

Uma possibilidade de obviar esta segunda limitação consiste na utilização de métodos de *diferenciação automática* (Bischof et al., 1992; Bischof, 1999; Nocedal e Wright, 1999), ou de linguagens de manipulação simbólica de expressões, como o *Mathematica* ou o *Maple* (Apêndice A), para gerar derivadas analíticas de $f(x)$. Outras alternativas são analisadas nas duas Secções seguintes.

2.12.2 Aproximação das derivadas por diferenças finitas

A necessidade do cálculo de derivadas no método de Newton pode ser contornada, aproximando a matriz Jacobiana por diferenças finitas, tal como descrito na Secção 1.2.7. Usando, por exemplo, diferenças progressivas de 1ª ordem, e designando $f(x) \equiv f([x_1, x_2, \ldots, x_n]^{\mathrm{T}})$, por forma a explicitar os argumentos escalares individuais de f, a generalização da equação (1.19) ao caso vectorial pode ser escrita como

$$\frac{\partial f}{\partial x_i} \simeq \frac{f([x_1, \ldots, x_i + h_i, \ldots, x_n]^{\mathrm{T}}) - f([x_1, \ldots, x_i, \ldots, x_n]^{\mathrm{T}})}{h_i} \in \mathbb{R}^n$$
$$i = 1, 2, \ldots, n \tag{2.38}$$

Tabela 2.2 Resultados de aplicação do método de Newton ao sistema (2.37).

k	x_k	f_k	J_k	h_k	$\|x^* - x_k\|_2$
0	$\begin{bmatrix} 2 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1{,}86466 \\ 1{,}0907 \end{bmatrix}$	$\begin{bmatrix} 0{,}135335 & 1 \\ 1 & 0{,}416147 \end{bmatrix}$	$\begin{bmatrix} -0{,}333511 \\ -1{,}81953 \end{bmatrix}$	2,03259
1	$\begin{bmatrix} 1{,}66649 \\ 0{,}180471 \end{bmatrix}$	$\begin{bmatrix} -8{,}43809 \times 10^{-3} \\ 1{,}487 \end{bmatrix}$	$\begin{bmatrix} 0{,}188909 & 1 \\ 1 & -0{,}983759 \end{bmatrix}$	$\begin{bmatrix} -1{,}24696 \\ 0{,}244 \end{bmatrix}$	1,18826
2	$\begin{bmatrix} 0{,}41953 \\ 0{,}424471 \end{bmatrix}$	$\begin{bmatrix} -0{,}232884 \\ 7{,}69142 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 0{,}657355 & 1 \\ 1 & -0{,}911257 \end{bmatrix}$	$\begin{bmatrix} 0{,}127907 \\ 0{,}148804 \end{bmatrix}$	0,200065
3	$\begin{bmatrix} 0{,}547437 \\ 0{,}573275 \end{bmatrix}$	$\begin{bmatrix} -5{,}15513 \times 10^{-3} \\ 5{,}05105 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 0{,}57843 & 1 \\ 1 & -0{,}840129 \end{bmatrix}$	$\begin{bmatrix} -4{,}84587 \times 10^{-4} \\ 5{,}43543 \times 10^{-3} \end{bmatrix}$	$5{,}46057 \times 10^{-3}$
4	$\begin{bmatrix} 0{,}546953 \\ 0{,}578710 \end{bmatrix}$	$\begin{bmatrix} -6{,}79258 \times 10^{-8} \\ 8{,}03456 \times 10^{-6} \end{bmatrix}$	$\begin{bmatrix} 0{,}578711 & 1 \\ 1 & -0{,}837169 \end{bmatrix}$	$\begin{bmatrix} -5{,}37407 \times 10^{-6} \\ 3{,}17796 \times 10^{-6} \end{bmatrix}$	$6{,}18113 \times 10^{-6}$
5	$\begin{bmatrix} 0{,}546947 \\ 0{,}578714 \end{bmatrix}$	$\begin{bmatrix} -8{,}35665 \times 10^{-12} \\ 2{,}76190 \times 10^{-12} \end{bmatrix}$	–	–	$6{,}98337 \times 10^{-8}$

atendendo à necessidade de avaliar individualmente o efeito de cada componente escalar x_i. Uma forma equivalente, mais compacta, desta equação é

$$\frac{\partial f}{\partial x_i} \simeq \frac{f(x + h_i e_i) - f(x)}{h_i}, \qquad i = 1, 2, \ldots, n \tag{2.39}$$

onde e_i representa a coluna i da matriz identidade de dimensão n (ou seja, o vector unitário na direcção da coordenada i)[11]. A equação (2.39) permite portanto obter o vector correspondente à linha i da matriz Jacobiana (2.35), definida anteriormente. Deste modo, a estimativa completa da matriz J_k requer $n + 1$ avaliações de $f(x)$, por iteração do método de Newton[12].

Para que a estimativa produzida seja precisa, é relevante a análise efectuada na Secção 1.2.7, relativamente aos cuidados a tomar na especificação de cada h_i. No entanto, a escolha óptima destes parâmetros pode tornar-se agora uma tarefa mais difícil, atendendo à natureza vectorial de $f(\cdot)$, neste caso.

Os parâmetros h_i utilizados influenciam também a rapidez de convergência deste método de Newton modificado. Consoante os valores utilizados (tendo em atenção os limites impostos pela precisão dos valores de $f(x)$ e pela natureza da aproximação usada), a ordem de convergência deste método pode ser *superlinear* ou *quadrática* (Dennis e Schnabel, 1983).

Exemplo 2.8 (Aproximação por diferenças finitas)

A Tabela 2.3 ilustra os valores das estimativas da solução do problema anterior, produzidas usando o método de Newton modificado, onde as derivadas foram aproximadas por (2.39), com $h_i = 0{,}01$. Como se pode verificar, os valores produzidos são bastante próximos dos representados na Tabela 2.2. Para valores menores de h_i, os resultados produzidos pelos dois métodos são praticamente indistinguíveis neste problema.

Em exemplos como o anterior, onde não existem dificuldades em aproximar adequadamente todas as derivadas, este método parece ser a variante do método de Newton mais fácil de implementar. Caso esta abordagem não permita obter estimativas suficientemente precisas das derivadas para ser usado, é recomendável a modificação considerada na próxima Secção. Uma vez que esta aproximação é relativamente fácil de programar, é geralmente sempre recomendado o seu uso numa fase inicial da solução de sistemas de equações, para validar as derivadas analíticas quando estas são fornecidas pelo utilizador.

[11] Na prática, a operação $h_i e_i$ não é efectuada, sendo antes usada a forma (2.38). A notação usada em (2.39) apenas é preferida por vezes para indicar a operação correspondente numa forma mais compacta.

[12] Portanto com um volume de cálculos comparável, em geral, ao requerido na avaliação das derivadas analíticas de $f(\cdot)$, mas eventualmente uma tarefa bastante mais simples de programar.

Tabela 2.3 Resultados da aplicação do método de Newton modificado, com $h_i = 0{,}01$.

k	x_k^T	k	x_k^T
0	[2 2]	3	[0,547432 0,573225]
1	[1,693 0,176267]	4	[0,546936 0,578718]
2	[0,411658 0,416546]	5	[0,546948 0,578714]

2.12.3 Método das secantes

Outra possibilidade, que também evita a necessidade do cálculo de derivadas analíticas de $f(\cdot)$, consiste na generalização do método das secantes (Secção 2.7) a sistemas de equações não-lineares.

Conforme descrito na Secção 2.7, qualquer generalização da equação (2.24) terá de satisfazer no caso vectorial

$$B_k \underbrace{(x_k - x_{k-1})}_{s_k} = \underbrace{(f(x_k) - f(x_{k-1}))}_{y_k}$$

onde $B_k \in \mathbb{R}^{n \times n}$ é a *matriz das secantes*. Como esta equação é utilizada com muita frequência, é comum definir as quantidades $s_k = x_k - x_{k-1}$ e $y_k = f(x_k) - f(x_{k-1})$, o que permite a sua escrita numa forma mais compacta:

$$B_k s_k = y_k \tag{2.40}$$

Tal como no caso escalar, esta equação é utilizada no método das secantes
98 para obter a matriz B_k, que desempenha neste caso um papel semelhante à matriz Jacobiana J_k do método de Newton. Para $n > 1$, surge contudo uma dificuldade na sua aplicação: o sistema de equações (2.40) comporta n equações individuais, mas envolve n^2 incógnitas (os coeficientes de B_k), sendo insuficiente para a especificação única da matriz B. Este problema pode ser ultrapassado de dois modos:

- Utilizando tantos pontos anteriores x_k, x_{k-1}, x_{k-2}, ... quantos os necessários para definir os declives das secantes que passam por todos eles. Por exemplo, para $n = 2$ seriam necessários 3 pontos para definir o plano secante que passa por x_k, x_{k-1} e x_{k-2}.
- Usando condições adicionais para além de (2.40), para definir unicamente B_k a partir dos pontos x_k e x_{k-1}. Neste caso a equação (2.40) servirá como base para a definição de uma *família* de métodos secantes.

A segunda estratégia é a mais utilizada na prática. De um conjunto quase infinito de possibilidades, qualquer escolha razoável para B_k, para além de satisfazer (2.40), deve ainda obedecer aos seguintes requisitos:

- Constituir uma "boa" aproximação de J_k, tentando preservar a rapidez de convergência observada no método de Newton.
- Facilitar o seu uso num método iterativo, nomeadamente requerendo apenas um volume de cálculo moderado.

Um dos algoritmos desta família com mais sucesso é o *método de Broyden (1965)*. Neste método, B_k é escolhido através da resolução do seguinte problema de optimização

$$\boxed{\begin{array}{ll} \min\limits_{B_k} & \|B_k - B_{k-1}\|_F \\ \text{s.a} & B_k s_k = y_k \end{array}}$$

onde $\|\cdot\|_F$ representa a *norma de Frobenius*:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$

Esta norma especial foi escolhida porque permite a resolução analítica do problema anterior, obtendo-se (Dennis e Schnabel, 1983):

$$B_k = B_{k-1} + \frac{(y_k - B_{k-1}s_k)s_k^{\mathrm{T}}}{s_k^{\mathrm{T}}s_k} \tag{2.41}$$

Como estratégia, o método efectua portanto apenas as alterações "*mínimas*" necessárias na matriz B, entre iterações sucessivas, por forma a respeitar sempre a equação das secantes (2.40). Deste modo, informação anterior eventualmente presente na matriz B numa dada iteração é preservada, tanto quanto possível, desde que esta seja compatível com a equação das secantes, aplicada na iteração presente.

As correcções efectuadas na matriz B, em cada iteração, envolvem um esforço da ordem das n^2 operações. A expressão (2.41) é também conhecida como uma *actualização de característica unitária* ("rank-1 update"), uma vez que a matriz $(y_k - B_{k-1}s_k)s_k^{\mathrm{T}}/(s_k^{\mathrm{T}}s_k)$ tem apenas característica 1.

Depois de conhecido B_k, o algoritmo prossegue de forma semelhante ao método de Newton, obtendo-se em cada iteração uma correcção $h_k = x_{k+1} - x_k$ através de:

$$B_k h_k = -f_k \qquad \text{ou} \qquad h_k = -B_k^{-1} f_k \tag{2.42}$$

As diferentes fases do método iterativo podem ser descritas através do seguinte algoritmo:

Algoritmo 2.4 (Método de Broyden)

Dados x_0 **e** B_0
Enquanto convergência ou limite de iterações não verificados
  Resolver $B_k s_{k+1} = -f_k$ **em ordem a** s_{k+1}
  $x_{k+1} = x_k + s_{k+1}$
  $y_{k+1} = f(x_{k+1}) - f(x_k)$
  $B_{k+1} = B_k + (y_{k+1} - B_k s_{k+1}) s_{k+1}^{\mathrm{T}} / (s_{k+1}^{\mathrm{T}} s_{k+1})$
Fim_Enquanto

Como se pode observar, este algoritmo requer, para além de um ponto base x_0, uma matriz inicial de secantes B_0. Uma vez que o método é algo conservador quanto às alterações efectuadas nesta matriz, convém que ela seja "próxima" da matriz Jacobiana equivalente, tendo este facto geralmente um impacto positivo na rapidez de convergência do método. Por essa razão, B_0 é frequentemente construída através de uma aproximação de J_0 por diferenças finitas, embora sem necessariamente muito rigor.

Exemplo 2.9 (Método de Broyden com 2 equações não-lineares)

A aplicação do método de Broyden ao sistema não-linear (2.37) encontra-se ilustrada na Tabela 2.4. Como se pode observar, a convergência é agora algo mais lenta que no caso do método de Newton, embora ainda possa ser considerada rápida. A matriz inicial das secantes B_0 fornecida era próxima do Jacobiano correspondente; como tal, as actualizações posteriores de B_k permi-
100 tiram mantê-la sempre próxima do Jacobiano do problema, apesar das variações observadas nos seus coeficientes.

Problema 2.10 *Estime a ordem de convergência do método das secantes, usando como base os dados numéricos do exemplo anterior.*

Para além de (2.41), são também usadas na prática outras expressões semelhantes. Por exemplo, em vez de actualizar directamente B_k é possível trabalhar com a sua inversa $H_k = B_k^{-1}$. Isto permite a obtenção de h_k directamente através de

$$h_k = -H_k f_k \tag{2.43}$$

em vez de usar (2.42), e portanto sem envolver a resolução de um sistema de equações lineares, ou a inversão de uma matriz. Isto é possível, por exemplo, através do uso do seguinte lema:

Tabela 2.4 Resultados de aplicação do método de Broyden ao sistema (2.37).

k	x_k	f_k	B_k	h_k	$\|x^* - x_k\|_2$
0	$\begin{bmatrix}2\\2\end{bmatrix}$	$\begin{bmatrix}1{,}86466\\1{,}0907\end{bmatrix}$	$\begin{bmatrix}0{,}1 & 1\\1 & 0{,}5\end{bmatrix}$	$\begin{bmatrix}-0{,}166705\\-1{,}84799\end{bmatrix}$	2,03259
1	$\begin{bmatrix}1{,}83329\\0{,}152006\end{bmatrix}$	$\begin{bmatrix}-7{,}88012\times10^{-3}\\1{,}68187\end{bmatrix}$	$\begin{bmatrix}0{,}100382 & 1{,}00423\\0{,}918563 & -0{,}402761\end{bmatrix}$	$\begin{bmatrix}-1{,}75081\\0{,}182855\end{bmatrix}$	1,35527
2	$\begin{bmatrix}8{,}24876\times10^{-2}\\0{,}334861\end{bmatrix}$	$\begin{bmatrix}-0{,}585962\\-0{,}246151\end{bmatrix}$	$\begin{bmatrix}0{,}431451 & 0{,}969653\\1{,}05764 & -0{,}417286\end{bmatrix}$	$\begin{bmatrix}0{,}400798\\0{,}425964\end{bmatrix}$	0,524583
3	$\begin{bmatrix}0{,}483286\\0{,}760825\end{bmatrix}$	$\begin{bmatrix}0{,}144071\\-0{,}206234\end{bmatrix}$	$\begin{bmatrix}0{,}60025 & 1{,}14905\\0{,}816008 & -0{,}674089\end{bmatrix}$	$\begin{bmatrix}0{,}104195\\-0{,}179813\end{bmatrix}$	0,192918
4	$\begin{bmatrix}0{,}58748\\0{,}581012\end{bmatrix}$	$\begin{bmatrix}2{,}52862\times10^{-2}\\3{,}86101\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}0{,}661254 & 1{,}04377\\0{,}909155 & -0{,}834837\end{bmatrix}$	$\begin{bmatrix}-4{,}0913\times10^{-2}\\1{,}69356\times10^{-3}\end{bmatrix}$	$4{,}05979\times10^{-2}$
5	$\begin{bmatrix}0{,}546567\\0{,}582706\end{bmatrix}$	$\begin{bmatrix}3{,}77181\times10^{-3}\\-3{,}71775\times10^{-3}\end{bmatrix}$	$\begin{bmatrix}0{,}56922 & 1{,}04758\\0{,}99987 & -0{,}838592\end{bmatrix}$	$\begin{bmatrix}4{,}79837\times10^{-4}\\-3{,}86121\times10^{-3}\end{bmatrix}$	$4{,}01003\times10^{-3}$
6	$\begin{bmatrix}0{,}547047\\0{,}578844\end{bmatrix}$	$\begin{bmatrix}1{,}88327\times10^{-4}\\-9{,}80290\times10^{-6}\end{bmatrix}$	$\begin{bmatrix}0{,}575189 & 0{,}999552\\0{,}999559 & -0{,}836092\end{bmatrix}$	$\begin{bmatrix}-9{,}97688\times10^{-5}\\-1{,}31000\times10^{-4}\end{bmatrix}$	$1{,}64341\times10^{-4}$
7	$\begin{bmatrix}0{,}546947\\0{,}578713\end{bmatrix}$	$\begin{bmatrix}-4{,}07436\times10^{-7}\\9{,}21341\times10^{-8}\end{bmatrix}$	–	–	$3{,}24748\times10^{-7}$

Lema 2.1 (Sherman-Morrison-Woodbury) *Dados $u, v \in \mathbb{R}^n$, e uma matriz $A \in \mathbb{R}^{n \times n}$ não-singular, então:*

$$\left(A + uv^{\mathrm{T}}\right)^{-1} = A^{-1} - \frac{A^{-1}uv^{\mathrm{T}}A^{-1}}{(1 + v^{\mathrm{T}}A^{-1}u)} \tag{2.44}$$

Deve notar-se que a matriz uv^{T} possui sempre característica unitária, e que se $(A+uv^{\mathrm{T}})$ for não-singular então $(1+v^{\mathrm{T}}A^{-1}u)$ é sempre uma quantidade não-nula.

Aplicando este lema à expressão (2.41) obtém-se, após algumas manipulações algébricas (Dennis e Schnabel, 1983):

$$H_k = H_{k-1} + \frac{(s_k - H_{k-1}y_k)s_k^{\mathrm{T}}H_{k-1}}{s_k^{\mathrm{T}}H_{k-1}y_k}$$

Tal como na actualização de B_k, os produtos matriciais nesta expressão e em (2.43) envolvem um esforço global da ordem de n^2 operações, sendo a sua aplicação consequentemente mais económica do que (2.41), de acordo com a análise efectuada no Capítulo seguinte. As implementações práticas destes métodos actualizam por vezes directamente uma *factorização triangular* de B_k, o que permite utilizar qualquer um dos métodos anteriores com um volume de cálculos semelhante.

2.12.4 Outros métodos

Como mencionado no início desta Secção, foram aqui abordados apenas métodos que correspondem a generalizações directas das técnicas univariável descritas no início deste Capítulo. Um estudo aprofundado da robustez e convergência destes métodos requer o uso de conceitos de *optimização*, estando por isso fora do âmbito destas notas.

Alguns problemas relativos à solução de sistemas de equações não-lineares beneficiam também da sua reformulação como problemas de optimização, restringindo por exemplo o domínio das suas variáveis independentes (e estimativas). Outros problemas beneficiam da sua reestruturação usando métodos iterativos para a solução aproximada dos problemas lineares resultantes em cada iteração, semelhantes aos analisados no Capítulo seguinte. Neste caso, são usados procedimentos iterativos em diversos níveis do seu processo de solução, tornando-se conveniente observar algumas das recomendações do Capítulo 1 relativas à manipulação das diferentes incertezas de aproximação.

Alguns desses aspectos, conjuntamente com diversas técnicas adicionais de solução de sistemas não-lineares podem ser encontrados por exemplo em Dennis e Schnabel (1983); Nocedal e Wright (1999); Kelley (2003).

CAPÍTULO 3

Sistemas de equações lineares

A determinação da solução de sistemas de equações lineares constitui uma das tarefas numéricas mais comuns, aparecendo muito frequentemente como um subproblema resultante da aplicação de outros métodos numéricos. Por exemplo, a extensão dos algoritmos descritos no Capítulo anterior à resolução de sistemas de equações *não-lineares* requer, na maior parte dos casos (nomeadamente no método de Newton), a resolução de um sistema linear *em cada iteração*. Noutras situações, os problemas físicos podem ser formulados directamente nesta forma, sendo por vezes necessária a resolução de problemas de grandes dimensão (envolvendo, por exemplo, 10^5 a 10^6 variáveis). Torna-se assim importante o desenvolvimento de métodos eficientes para a resolução de sistemas lineares.

Um sistema de n equações lineares a n incógnitas

$$\begin{cases} a_{11}x_{11} + a_{12}x_{12} + \cdots + a_{1n}x_{1n} = b_1 \\ a_{21}x_{21} + a_{22}x_{22} + \cdots + a_{2n}x_{2n} = b_2 \\ \vdots \\ a_{n1}x_{n1} + a_{n2}x_{n2} + \cdots + a_{nn}x_{nn} = b_n \end{cases} \tag{3.1}$$

é habitualmente escrito numa forma compacta $A \cdot x = b$, usando uma notação matricial, onde $A \in \mathbb{R}^{n \times n}$ é a *matriz de coeficientes* correspondente e $b, x \in \mathbb{R}^n$ são o *termo independente* e o *vector solução*, respectivamente. Como é conhecido, para que o sistema (3.1) tenha uma solução única, é necessário que A seja *não-singular* ou, de forma equivalente, que tenha determinante não-nulo ($|A| \neq 0$).

A interpretação geométrica mais comum para a solução de um sistema de equações lineares corresponde às coordenadas de intersecção de um conjunto de rectas (para $n = 2$, Figura 3.1), planos ou hiperplanos (de dimensão $n - 1$) para dimensões superiores. Se $|A| \simeq 0$, o problema resultante pode ser numericamente *mal-condicionado*. As rectas ou planos correspondentes podem

ser neste caso quase paralelos, e qualquer pequeno desvio na especificação dos seus coeficientes pode introduzir grandes alterações na localização do seu ponto de intersecção, e portanto na resolução do problema. Este comportamento foi observado no Exemplo 1.4, e será analisado em maior detalhe na Secção 3.2.3.

3.1 Casos especiais

O esforço necessário para a determinação da solução de $A \cdot x = b$ depende fortemente da estrutura da matriz A. Por exemplo, se $A = I$, onde I é a matriz identidade de ordem n, a resolução do sistema é $x = b$ (trivial). Se A for uma matriz diagonal, o sistema

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

terá a solução (também simples)

$$\begin{cases} x_1 = b_1/a_{11} \\ x_2 = b_2/a_{22} \\ \vdots \\ x_n = b_n/a_{nn} \end{cases} \tag{3.2}$$

para $a_{ii} \neq 0, \forall i = 1, \ldots, n$. A determinação da solução para este caso envolve portanto apenas n divisões (em geral).

3.1.1 Resolução de sistemas triangulares

Se A for triangular superior ou inferior, a resolução do sistema correspondente pode ser encontrada usando o método de *substituição regressiva* ou *progressiva*, respectivamente. Exemplificando com um sistema 3×3 triangular superior

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

temos inicialmente, a partir da terceira equação $x_3 = b_3/a_{33}$. Substituindo este resultado na segunda equação, e resolvendo-a em ordem a x_2 fica $x_2 = (b_2 - a_{23}x_3)/a_{22}$. Finalmente x_1 pode ser obtido da primeira equação, substituindo

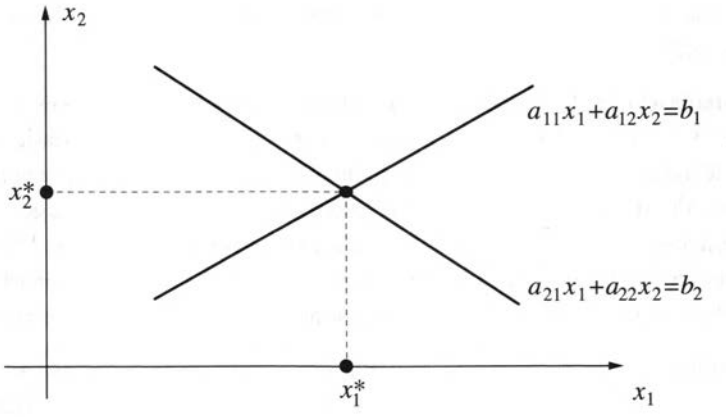


Figura 3.1 Determinação da intersecção de 2 rectas.

nesta os valores das variáveis x_2 e x_3 calculados anteriormente, e resolvendo-a em ordem a x_1.

O método da substituição regressiva para um sistema geral de dimensão n pode portanto ser resumido através do seguinte algoritmo:

Algoritmo 3.1 (Substituição regressiva)

$x_n = b_n/a_{nn}$
`Para` $i = n-1, n-2, \ldots, 1$ `fazer`

$$x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right)/a_{ii}$$

`Fim_Para`

Problema 3.1 *Usando como base o algoritmo anterior, escreva o algoritmo correspondente para o método de* substituição progressiva, *aplicado em sistemas triangulares inferiores.*

3.1.2 Complexidade computacional

Conforme já mencionado, uma característica fundamental dos algoritmos para a resolução de sistemas lineares é a sua *eficiência*, tendo em vista a sua aplicação em problemas de dimensão elevada, ou bastante frequentes[1]. Esta eficiência está relacionada com a *complexidade computacional* de cada algoritmo, ex-

[1]Por exemplo quando esta tarefa surge como subproblema resultante da aplicação de outros métodos numéricos.

pressa através do volume de operações envolvido na resolução de um sistema geral de ordem n.

Como analisado, o algoritmo para a resolução de sistemas diagonais correspondente a (3.2) envolve apenas n divisões, sendo a sua complexidade habitualmente designada por $O(n)$ — ordem de n. Noutros casos (por exemplo, na resolução de sistemas triangulares) é necessário proceder à contagem manual das operações elementares efectuadas para determinar a complexidade correspondente do método. Este valor pode posteriormente ser relacionado com o tempo de execução da tarefa, para vários tamanhos do problema a resolver[2].

Na generalidade dos processadores actuais, o tempo requerido para a execução de cada operação elementar entre duas quantidades reais (soma, subtracção, multiplicação ou divisão) não é idêntico, podendo variar significativamente de acordo com a natureza da operação efectuada. Torna-se por isso conveniente agrupar as operações realizadas por classes, de acordo com a sua complexidade relativa (por ordem crescente):

- adições e subtracções.
- multiplicações e divisões.
- operações transcendentes, tais como o cálculo de raízes quadradas e de funções trigonométricas.

É comum considerar nesta análise apenas as operações relativas a quantidades de vírgula flutuante, supondo-se que as operações com inteiros são significativamente mais rápidas[3]. A análise correspondente do algoritmo de substituição regressiva anterior é efectuada no exemplo seguinte.

Exemplo 3.1 (Análise do método de substituição regressiva)

No Algoritmo 3.1 a instrução $x_n = n_n/a_{nn}$ requer 1 divisão. O ciclo seguinte é repetido $n-1$ vezes, devendo este factor ser usado para multiplicar o número de operações correspondentes à linha compreendida no interior deste ciclo.

Em termos de adições e subtracções, a linha $x_i = (b_i - \sum_{j=i+1}^{n} a_{ij}x_j)/a_{ii}$

[2]Várias características dos micro-processadores actuais dificultam contudo esta correlação. Por exemplo, a maioria possui uma estrutura *super-escalar*, que permite executar mais do que uma instrução por ciclo (i.e., em simultâneo), possivelmente fora da ordem em que estas são especificadas no programa. É também frequente a existência de instruções combinadas que permitem realizar uma operação de adição e outra de multiplicação em simultâneo na mesma unidade de vírgula flutuante. Nestas condições, o tempo de execução de um dado programa irá depender não só do número de instruções que este contempla, como do seu tipo, e do modo como estas podem ser ordenadas para tirar partido do paralelismo existente. Dowd e Severance (1998) consideram em detalhe estes aspectos relativos à optimização de programas de cálculo numérico.

[3]Isto facilita os aspectos relativos à contabilização do uso de variáveis inteiras em ciclos iterativos e como índices de variáveis não-escalares.

corresponde a

$$1 + [n - (j+1)] = n - i \quad \text{adições.}$$

O total de operações deste tipo será portanto:[4]

$$\sum_{i=1}^{n-1}(n-i) = (n-1)n - \sum_{i=1}^{n-1} i = (n-1)n - \frac{n(n-1)}{2} = \boxed{\frac{n^2}{2} - \frac{n}{2}}$$

No interior do ciclo temos 1 divisão e

$$n - (i+1) + 1 = n - i \quad \text{multiplicações.}$$

O número total de operações deste tipo será portanto:

$$1 + \sum_{i=1}^{n-1}(n-i+1) = 1 + (n-1)(n+1) - \sum_{i=1}^{n-1} i = \boxed{\frac{n^2}{2} + \frac{n}{2}}$$

Conclui-se portanto que a resolução de sistemas triangulares envolve aproximadamente tanto $n^2/2$ adições / subtracções como multiplicações / divisões de vírgula flutuante. Trata-se portanto de um algoritmo com complexidade de $O(n^2)$, uma vez que para n elevado, este é o termo dominante em cada um dos resultados de contagem anteriores.

3.1.3 Matrizes com estrutura especial

Deve notar-se que as indicações anteriores relativas à complexidade dos métodos de resolução apenas são válidas para matrizes A gerais, com relativamente poucos elementos nulos na sua parte triangular principal (ou diagonal), e sem outra estrutura especial; se isto não se verificar, o número de operações necessárias pode ser consideravelmente inferior a este valor, através da omissão de um grupo significativo de passos que produziriam valores nulos, ou outras quantidades facilmente antecipáveis. Por esta razão, as bibliotecas numéricas existentes para a determinação da solução de sistemas lineares possuem habitualmente subrotinas ou funções específicas para vários tipos de matrizes. Alguns dos tipos de matrizes tratados habitualmente em separado são:

- matrizes *triangulares*, sendo os sistemas resolvidos pelos métodos anteriores.
- matrizes *por bandas*, onde todos os elementos não-nulos estão situados na sua diagonal principal, e nas diagonais imediatamente acima e

[4] Usando o facto de que $\sum_{i=1}^{n} i = n(n+1)/2$.

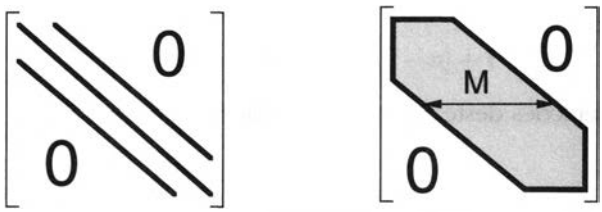


Figura 3.2 Estrutura de matrizes tridiagonais e por bandas (com largura M).

abaixo desta (Figura 3.2). Estas matrizes são frequentemente originadas em problemas descritos por equações que descrevem a interacção entre elementos próximos (ou *vizinhos*), como por exemplo problemas de redes, de diagramas de fabrico (tal como o Exemplo 3.6, considerado na Secção 3.3), ou problemas resultantes da discretização de equações diferenciais.

Uma matriz A com apenas M diagonais não-nulas é designada como uma matriz por bandas com *largura* M. Como caso especial, as matrizes com $M = 3$ (tendo portanto apenas uma diagonal acima e outra imediatamente abaixo da diagonal principal) são designadas como *tridiagonais*. Estes sistemas são tratados muitas vezes por redução à forma triangular (superior ou inferior), seguida da aplicação de uma versão especializada do método de substituição regressiva ou progressiva, que tenha em consideração os elementos nulos da matriz original.

- matrizes *simétricas*, onde $A = A^{\mathrm{T}}$. Neste caso, a simetria da matriz pode ser usada para prever o resultado de algumas operações (dado que alguns coeficientes da matriz são idênticos entre si), não sendo sequer necessário o armazenamento da matriz inteira.

Os métodos existentes para a resolução de equações lineares podem habitualmente ser enquadrados numa das duas classes seguintes:

- métodos *directos*, onde o problema original é convertido, através da aplicação de uma sequência de operações elementares, num problema suficientemente simples para permitir a sua resolução imediata (por exemplo, um sistema triangular, ou diagonal).

- métodos *iterativos*, onde se parte de uma solução aproximada para o problema, e se refina sucessivamente esta estimativa (por iteração), até determinados critérios de paragem serem obedecidos, de forma semelhante aos métodos estudados no Capítulo anterior.

A parte restante deste Capítulo é dedicada à análise de cada uma destas classes de métodos.

3.2 Métodos directos

Salvo indicação em contrário, os métodos considerados nesta Secção dizem respeito a uma matriz genérica $A \in \mathbb{R}^{n\times n}$, *densa* (i.e., com poucos elementos nulos), e sem nenhuma estrutura especial.

3.2.1 Eliminação Gaussiana

O método de *eliminação Gaussiana* efectua a conversão do sistema $Ax = b$ num sistema triangular equivalente, por aplicação de operações elementares, sendo depois usada a substituição regressiva (ou progressiva) com o sistema triangular resultante, para a obtenção da solução completa.

Operações e matrizes elementares

As operações elementares possíveis no processo de eliminação Gaussiana são as seguintes:

- Multiplicar uma equação por uma constante não-nula.
- Permutar duas equações.
- Adicionar um múltiplo de uma equação a outra.

Cada uma destas operações pode ser representada através de uma *matriz elementar* M equivalente, da forma

$$M = I - uv^{\mathrm{T}} \tag{3.3}$$

onde $I \in \mathbb{R}^{n\times n}$ corresponde à matriz identidade de ordem n, e $u, v \in \mathbb{R}^n$ são vectores arbitrários.

A aplicação de uma operação elementar à matriz A equivale assim à sua multiplicação pela matriz M correspondente. Para formar a matriz M relativa a uma dada operação elementar basta realizar a operação pretendida sobre a matriz identidade correspondente.

Designando por e_i a coluna i da matriz identidade de dimensão compatível temos, como exemplos de cada uma das categorias anteriores, para $n = 3$:

- $M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} = I - (1-\alpha)e_2 e_2^{\mathrm{T}} \quad \Rightarrow$

$$\Rightarrow \quad M_1 A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ \alpha a_{21} & \alpha a_{22} & \alpha a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Esta operação corresponde à multiplicação da 2ª linha de A pelo escalar α.

- $M_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I - uu^{\mathrm{T}} \quad \Rightarrow \quad M_2 A = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

  com $u = e_1 - e_2$. Aqui a 1ª linha de A foi permutada com a sua 2ª linha.

- $M_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \alpha & 0 & 1 \end{bmatrix} = I + \alpha e_3 e_1^{\mathrm{T}} \quad \Rightarrow$

$$\Rightarrow \quad M_3 A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \alpha a_{11} + a_{31} & \alpha a_{12} + a_{32} & \alpha a_{13} + a_{33} \end{bmatrix}$$

Agora a 1ª linha foi multiplicada por α e adicionada à 3ª linha de A.

Em vez da multiplicação à esquerda, se A for multiplicada por M à direita obtém-se um resultado equivalente, mas desta vez sobre as *colunas* de A (por exemplo, colunas permutadas em vez de linhas permutadas, colunas multiplicadas por α, etc.).

As matrizes do tipo de M_2 anterior são também conhecidas como *matrizes de permutação*. Como se pode observar, no caso geral, estas matrizes possuem apenas um coeficiente unitário por linha (e coluna), sendo os restantes coeficientes todos nulos. Se $P \in \mathbb{R}^n$ for uma matriz de permutação é possível demonstrar que $P^{\mathrm{T}} = P^{-1}$, i.e., as matrizes de permutação são *ortogonais*. O produto de matrizes de permutação origina também sempre uma matriz de permutação.

Todas as matrizes elementares dos vários tipos anteriores são não-singulares, podendo a sua inversa ser obtida através de uma forma particular do lema de Sherman-Morrison-Woodbury (equação 2.44):

$$(I - uv^{\mathrm{T}})^{-1} = I - \frac{uv^{\mathrm{T}}}{v^{\mathrm{T}}u - 1} \tag{3.4}$$

A partir desta expressão facilmente se constata que as inversas de matrizes elementares são também elas próprias matrizes elementares, da forma (3.3).

Deve notar-se que, embora o uso de matrizes elementares seja uma forma conveniente para expressar as operações efectuadas, estas matrizes não são necessariamente criadas explicitamente durante a sua utilização. Por exemplo,

as matrizes permutação anteriores são frequentemente representadas usando apenas um vector de inteiros, contendo em cada posição o índice do elemento unitário em cada coluna. Deste modo, em vez de criar a matriz

$$P_4 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

a informação equivalente pode ser codificada através de um vector

$$p_4 = \begin{bmatrix} 3 & 2 & 1 & 4 \end{bmatrix}$$

sendo o resultado da sua aplicação facilmente obtido a partir desta forma condensada. Para cada uma das categorias anteriores, o resultado das operações correspondentes sobre matrizes desse tipo pode ser igualmente antecipado em vez de ser obtido através do cálculo do produto matricial explícito, tendo em conta o elevado número de coeficientes nulos presente nestas matrizes elementares.

Aplicação do processo de eliminação

O resultado da aplicação das operações elementares durante o processo de eliminação Gaussiana é frequentemente descrito usando a matriz aumentada $[A|b]$. Pretende-se portanto efectuar a transição

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \rightarrow \cdots \rightarrow \left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} & b'_n \end{array}\right]$$

usando as operações elementares anteriores.

A eliminação Gaussiana é habitualmente aplicada por colunas. Assumindo que $a_{11} \neq 0$, procede-se numa primeira fase ao anulamento dos coeficientes da primeira coluna abaixo da diagonal principal. Para isso adiciona-se a cada linha cujo coeficiente se quer anular um múltiplo apropriado da primeira linha. Por exemplo, para anular a_{21}, adiciona-se à segunda linha a primeira linha multiplicada por $-a_{21}/a_{11}$. A segunda linha da matriz fica então

$$\left[\begin{array}{ccccc|c} 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \end{array}\right]$$

designando o expoente (2) a segunda versão dos coeficiente a_{ij} respectivos.

Como se poderá confirmar com o decorrer da aplicação do método, os coeficientes $a_{22}^{(2)}, a_{23}^{(2)}, \ldots, a_{2n}^{(2)}$ irão ser mantidos até ao final, a menos que exista uma permuta de variáveis. Para anular a_{31}, o procedimento é semelhante, havendo neste caso necessidade de multiplicar a primeira linha por $-a_{31}/a_{11}$ antes de a somar à terceira linha. A terceira linha da matriz fica portanto:

$$\left[\begin{array}{cccc|c} 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} & b_3^{(2)} \end{array}\right]$$

Esta fase termina com a adição da 1ª linha multiplicada por $(-a_{n1}/a_{11})$ à n-ésima linha da matriz aumentada. As quantidades

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \qquad (i = 2, 3, \ldots, n) \tag{3.5}$$

utilizadas durante esta fase são denominadas de *multiplicadores*, aparecendo também noutros métodos de resolução. O coeficiente a_{11} está sempre em denominador dos multiplicadores criados nesta fase, sendo por isso designado de *pivot*. No final, a matriz aumentada terá portanto o aspecto

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{array}\right]$$

com

$$\begin{cases} a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1} a_{ij}^{(1)} \\ b_i^{(2)} = b_i^{(1)} - m_{i1} b_1^{(1)} \end{cases} \qquad (i,j = 2, 3, \ldots, n)$$

sendo os multiplicadores m_{i1} definidos por (3.5).

É fácil confirmar que o resultado desta operação pode também ser obtido multiplicando A à esquerda por uma matriz elementar apropriada que, para a condensação da 1ª coluna, possuirá a forma

$$M_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -m_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & 0 & \cdots & 1 \end{bmatrix}$$

onde os coeficientes m_{i1} correspondem aos multiplicadores definidos por (3.5).

Em termos da definição (3.3), esta matriz corresponde a $M_1 = I - v_1 e_1^{\mathrm{T}}$, onde

$$v_1 = \begin{bmatrix} 0 \\ m_{21} \\ \vdots \\ m_{n1} \end{bmatrix}$$

e e_1 corresponde à 1ª coluna da matriz identidade.

Na fase seguinte, para anular os termos da segunda coluna abaixo da diagonal principal, definem-se de forma semelhante os multiplicadores

$$m_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} \qquad (i = 3, \ldots, n) \tag{3.6}$$

desde que $a_{22}^{(2)} \neq 0$. Para anular $a_{32}^{(2)}$ será necessário multiplicar a segunda linha por $-a_{32}^{(2)}/a_{22}^{(2)}$ e adicioná-la à terceira linha, que fica então:

$$\left[\begin{array}{cccccc|c} 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} & \cdots & a_{3n}^{(3)} & b_3^{(3)} \end{array}\right]$$

O aspecto da matriz aumentada no final desta segunda fase será portanto:

$$\left[\begin{array}{ccccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} & b_3^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} & b_n^{(3)} \end{array}\right] \tag{3.7}$$

Este resultado pode ser igualmente obtido formando uma matriz elementar M_2 correspondente à eliminação da 2ª coluna. O produto $M_2 \cdot M_1 \cdot [A|b]$ produzirá assim uma matriz com a estrutura de (3.7). Neste caso M_2 terá a forma

$$M_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & -m_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -m_{n2} & \cdots & 0 \end{bmatrix}$$

com os multiplicadores m_{i2} definidos por (3.6). Esta matriz corresponde a $M_2 = I - v_2 e_2^{\mathrm{T}}$, onde

$$v_2^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & m_{32} & \cdots & m_{n2} \end{bmatrix}$$

e e_2 corresponde à 2ª coluna da matriz identidade.

Prosseguindo de forma idêntica para as restantes colunas ter-se-á, no final, a matriz triangular superior:

$$\left[\begin{array}{ccccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} & b_3^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} & b_n^{(n)} \end{array}\right] \tag{3.8}$$

De uma forma genérica, para a coluna $k < n$, se $a_{kk}^{(k)} \neq 0$, temos portanto

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad (i = k+1, \ldots, n) \tag{3.9}$$

e:

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{ij}^{(k)} \\ b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_i^{(k)} \end{cases} \qquad (i,j = k+1, \ldots, n)$$

Não é necessário guardar todas as versões sucessivas dos coeficientes $a_{ij}^{(k)}$, uma vez que a aplicação do algoritmo apenas requer a última versão de cada coeficiente. Isto permite uma redução significativa do espaço de memória necessário.

A estrutura final do processo de eliminação (3.8) pode ser obtida, de forma equivalente, através da sequência de multiplicações

116

$$M_{n-1} \cdots M_2 M_1 [A|b] \tag{3.10}$$

onde

$$M_k = I - v_k e_k^{\mathrm{T}} \tag{3.11}$$

tem a estrutura:

$$M_k = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & -m_{k+1,k} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -m_{n,k} & 0 & \cdots & 1 \end{bmatrix} \quad \text{e} \quad v_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ m_{k+1,k} \\ \vdots \\ m_{nk} \end{bmatrix} \tag{3.12}$$

Em termos práticos, o produto das matrizes M_i em (3.10) não é efectuado explicitamente, uma vez que estas possuem demasiados elementos nulos. É possível verificar que o produto $M_{n-1} \cdots M_2 M_1$ vai corresponder a uma matriz *triangular inferior*, uma vez todas as matrizes elementares usadas em cada um dos passos da eliminação Gaussiana é também triangular inferior.

Uma implementação deste método é considerada no seguinte algoritmo:

Algoritmo 3.2 (Eliminação Gaussiana)

Para $k = 1, 2, \ldots, n-1$ **fazer** (varrer colunas)
  Para $i = k+1, k+2, \ldots, n$ **fazer** (criar multiplicadores)
    $m = a_{ik}/a_{kk}$
    Para $j = k+1, k+2, \ldots, n$ **fazer** (à direita da diagonal)
      $a_{ij} = a_{ij} - m\, a_{kj}$
    Fim_Para
    $b_i = b_i - m\, b_k$
  Fim_Para
Fim_Para

Complexidade do método de eliminação Gaussiana

Usando uma metodologia semelhante à do Exemplo 3.1, é possível demonstrar que o processo de eliminação Gaussiana envolve

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} \quad \text{adições / subtracções}$$
$$\frac{n^3}{3} + n^2 - \frac{n}{3} \quad \text{multiplicações / divisões}$$

incluindo a fase de substituição regressiva. Este processo pode portanto ser considerado como tendo uma complexidade de $O(2n^3/3)$.

O processo de eliminação Gaussiana pode também ser repetido com a matriz triangular superior obtida, eliminando desta vez os elementos *acima da diagonal*. Daí resulta uma matriz diagonal, permitindo obter a solução apenas com n divisões. Este método é conhecido como o método de *Gauss-Jordan*. O número de operações necessárias é neste caso

$$\frac{n^3}{2} - \frac{n}{2} \quad \text{adições / subtracções}$$
$$\frac{n^3}{2} + \frac{n^2}{2} \quad \text{multiplicações / divisões}$$

portanto superior ao método de eliminação de Gauss[5], sendo por isso menos usado.

Qualquer destes dois algoritmos requer um esforço consideravelmente superior aos dos métodos de substituição progressiva e regressiva descritos anteriormente, com complexidades de $O(n^2)$. É útil estabelecer uma comparação entre estas grandezas e as correspondentes a outras tarefas comuns em Álgebra Linear:

$$\begin{array}{lll} y = Ax & (A \in \mathbb{R}^{n\times n},\, x \in \mathbb{R}^n) & \rightarrow \; O(2n^2) \\ C = A + B & (A, B \in \mathbb{R}^{n\times n}) & \rightarrow \; O(n^2) \\ C = A + xy^{\mathrm{T}} & (A \in \mathbb{R}^{n\times n},\, x, y \in \mathbb{R}^n) & \rightarrow \; O(2n^2) \\ C = A \cdot B & (A, B \in \mathbb{R}^{n\times n}) & \rightarrow \; \boldsymbol{O(2n^3)} \end{array}$$

Pode verificar-se que o processo de multiplicação de matrizes densas é a tarefa de maior complexidade entre as consideradas, apesar da resolução de sistemas lineares poder parecer aparentemente "mais difícil". Mesmo o cálculo explícito da matriz inversa A^{-1} através da redução de $[A|I]$ pelo método de Gauss-Jordan requer apenas n^3 multiplicações e divisões, e um número semelhante de operações de adição / subtracção (Meyer, 2000), sendo portanto um processo de complexidade comparável à multiplicação de matrizes densas[6].

Necessidade do processo de permuta

O método de eliminação correspondente ao Algoritmo 3.2 apenas pode ser usado quando os coeficientes $a_{kk}^{(k)}$ são diferentes de zero. Quando um destes coeficientes se anula, o método deixa de poder ser aplicado, devido à divisão
118 por zero que ocorre no cálculo dos multiplicadores. Isto não significa necessariamente que o sistema não tenha solução. Por exemplo o sistema

$$\begin{cases} x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{3.13}$$

não pode ser resolvido pelo método anterior. Ele é, no entanto, equivalente ao sistema

$$\begin{cases} x_1 + x_2 = 2 \\ x_2 = 1 \end{cases}$$

[5]Para n elevado, a diferença entre $O(n^3/2)$ e $O(n^3/3)$ pode corresponder a 50% do volume total de cálculos!

[6]De igual modo, pode concluir-se que a resolução de sistemas lineares formando explicitamente A^{-1} requer cerca de 3 vezes o esforço da solução do mesmo sistema através da eliminação Gaussiana.

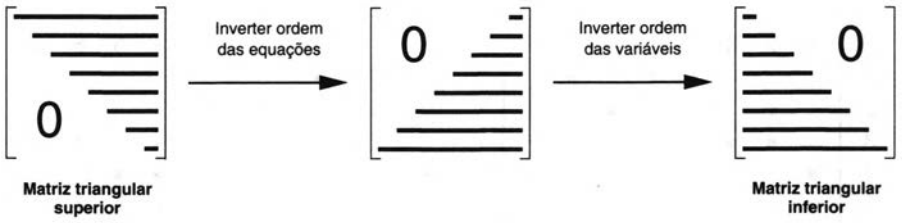


Figura 3.3 Conversão de uma matriz triangular superior em inferior, por permuta de equações e variáveis.

onde o método anterior já é aplicável. Uma estratégia simples para minimizar este problema consiste portanto na permuta de equações para evitar o aparecimento de *pivots* nulos durante a eliminação Gaussiana.

O aparecimento de pivots bastante pequenos, com $|a_{kk}^{(k)}| \simeq 0$, pode também colocar alguns problemas. Como estas quantidades aparecem como divisores em (3.9), existe a possibilidade de serem criados multiplicadores com magnitude elevada. A adição algébrica das linhas correspondentes pode causar posteriormente problemas de perda de precisão da solução, ao serem subtraídas quantidades elevadas, com sinais contrários, tal como analisado no Exemplo 1.5. Por esta razão, *deve ser evitado o uso de pivots de magnitude reduzida*. Para isto, é conveniente modificar o método anterior, introduzindo a possibilidade de reordenação sistemática das equações. Surgem assim os métodos de *eliminação Gaussiana com pivotagem*. Estes métodos permitem minimizar os erros inerentes à resolução de sistemas lineares usando aritmética com precisão finita.

Problema 3.2 *Aplique o processo de eliminação Gaussiana apresentado anteriormente ao sistema linear*

$$\begin{cases} \epsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases}$$

semelhante a (3.13). *Em que medida o cálculo numérico da solução poderá neste caso ser afectado por problemas de perda de precisão, quando* $|\epsilon| \to 0$*?*

Pivotagem em sistemas lineares

Num sistema linear é possível permutar a *ordem pela qual as equações são escritas*; isto corresponde à troca de *linhas* da matriz aumentada $[A|b]$. A *permuta da ordem das variáveis* é também possível, correspondendo neste caso a uma troca das colunas de A. Na Figura 3.3 é ilustrada a aplicação destes mecanismos na conversão de uma matriz triangular superior em inferior.

Problema 3.3 *Reescreva as operações realizadas na Figura 3.3 em termos do produto de A por matrizes elementares correspondentes.*

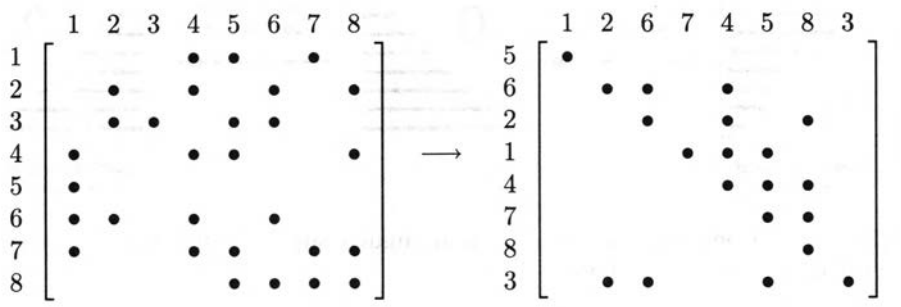


Figura 3.4 Exemplo de rearranjo preliminar da matriz A de um sistema de equações, usando o método de Markovitz (Mah, 1990). Apenas são representados os elementos não-nulos da matriz. Os números no topo e no lado esquerdo da matriz indicam a numeração das equações e variáveis antes e depois de realizado o reordenamento.

Existem essencialmente duas formas de implementar o rearranjo sistemático de sistemas lineares. As operações de pivotagem podem ser implementadas em 1º lugar através de um *rearranjo preliminar* do sistema de equações, antes de este ser resolvido. Em alguns casos, esta abordagem permite converter um sistema "desordenado" num outro sistema equivalente, com uma estrutura próxima de um caso especial (por exemplo quase diagonal, ou quase triangular). Isto acontece frequentemente com sistemas *esparsos* (i.e., quando muitos coeficientes da matriz A são nulos), tais como os resultantes da resolução de diagramas de fabrico de processos químicos ("flowsheeting"). Neste tipo de sistemas, geralmente de grandes dimensões, pode nem ser necessário (ou conveniente) armazenar a matriz A explicitamente. Alguns métodos para efectuar o rearranjo preliminar deste tipo de matrizes são descritos por Mah (1990). Depois de rearranjado, o sistema é convertido num sistema triangular, com um esforço tipicamente bastante inferior ao necessário para processar uma matriz A genérica equivalente (Figura 3.4[7]).

Este tipo de análise, numa fase prévia à resolução, é particularmente vantajoso em diversas situações:

- Quando permite reduções significativas no esforço necessário para a resolução do sistema de equações resultante.

[7]Para garantir a sua aplicabilidade, o método usado nesta ilustração elimina também alguns coeficientes não-nulos, em posições chave da matriz; isto pode ser constatado atendendo a que a matriz inicial apresenta 29 coeficientes não-nulos, enquanto que a matriz final apenas possui 20 elementos não-nulos. No entanto, o método procura sobretudo tirar partido efeito do reordenamento das equações e variáveis no esforço necessário para a resolução do sistema. Uma implementação computacional deste algoritmo pode ser encontrada nas rotinas MA28 e MA48 da biblioteca HSL (AEA Technology, 1978).

- Quando é necessária a resolução consecutiva de vários sistemas de equações, com a mesma estrutura de elementos não-nulos. Isto pode acontecer, por exemplo, na aplicação de um método iterativo como o método de Newton, considerado anteriormente, a um sistema de equações não-lineares. Neste caso, a informação obtida na fase de pré-processamento é gerada antes da resolução do 1º sistema linear, sendo depois usada para acelerar a resolução dos vários problemas lineares considerados.

Uma outra estratégia consiste em proceder ao rearranjo das equações *à medida que a determinação da solução prossegue.* Isto pode ser acomodado no método de eliminação Gaussiana de várias formas distintas:

1. No modo de *pivotagem parcial*, no início da eliminação relativa à coluna k, em vez de a_{kk} é seleccionado o pivot a_{pk}, onde:

$$a_{pk} = \max_{k \leq i \leq n} |a_{ik}|$$

Isto significa que, se $p \neq q$, as linhas (equações) k e p são permutadas. Por exemplo, com a matriz

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix} \tag{3.14}$$

o pivot é escolhido como o elemento com máxima magnitude entre os coeficientes $a_{22}^{(2)}$, $a_{32}^{(2)}$, e $a_{42}^{(2)}$. A linha onde este ocorre é permutada com a segunda linha, se necessário. Evita-se assim, tanto quanto possível, a divisão por quantidades pequenas. É fácil verificar que neste caso todos os multiplicadores passam a satisfazer:

$$|m_{ik}| = \frac{|a_{ik}^{(k)}|}{|a_{kk}^{(k)}|} \leq 1 \qquad (i = k+1, \ldots, n)$$

Em termos da utilização de matrizes elementares, para descrever o processo de eliminação com permuta de linhas é necessário alterar (3.10) para a forma

$$(M_{n-1}P_{n-1} \cdots M_2P_2M_1P_1)\ [A|b] \tag{3.15}$$

prevendo a necessidade de aplicar uma permuta de linhas P_i antes de eliminar cada coluna de A.

2. No modo de *pivotagem completa* é escolhido para pivot o elemento com

maior valor absoluto na submatriz a_{kj}, com $k,j \geq i$. Se necessário, procede-se à permuta *de linhas e colunas* em A para colocar esse elemento na posição pivotal. No caso da matriz (3.14), o pivot será portanto escolhido como o elemento de maior magnitude da submatriz:

$$\begin{bmatrix} a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix}$$

Este elemento é depois trazido para a posição do coeficiente $a_{22}^{(2)}$, por permuta de linhas e colunas de A, se necessário. Esta estratégia proporciona uma maior segurança quanto à escolha dos pivots, à custa de uma implementação mais complexa (é necessário registar as alterações das variáveis originais do sistema). Usando matrizes elementares, a sequência de operações necessária pode ser descrita através do produto de matrizes

$$(M_{n-1}P_{n-1}\cdots M_2P_2M_1P_1)\ [A|b]\ (Q_1Q_2\cdots Q_{n-1}), \tag{3.16}$$

onde P_i e Q_i correspondem a matrizes permutação, usadas para colocar as linhas e colunas da matriz A nas posições adequadas.

3. Na resolução de sistemas esparsos, a aplicação de qualquer um dos métodos anteriores de pivotagem pode introduzir o aparecimento de um número significativo de coeficientes não-nulos na parte da matriz ainda não eliminada, devido ao rearranjo de equações e variáveis efectuado, e às restantes operações elementares que se tornam necessárias em seguida. Este fenómeno, designado por *enchimento* ou *fill-in*, é geralmente indesejado, uma vez que pode aumentar consideravelmente o esforço computacional necessário na resolução do sistema (induzindo a necessidade da realização de mais operações elementares), e o espaço de memória necessário para guardar a matriz A. Nestes casos, pode ser importante atingir um compromisso equilibrado entre a estabilidade numérica do algoritmo de resolução e a sua rapidez de aplicação. Para o efeito é frequentemente usada uma regra de *pivotagem com limiar* (ou "threshold pivoting") onde os pivots escolhidos satisfazem o critério

$$|a_{kk}^{(k)}| \geq u|a_{ik}^{(k)}|, \quad \text{para } i > k$$

sendo $0 < u \leq 1$ um parâmetro ajustável, escolhido de forma apropriada. Este modo de pivotagem é particularmente útil para a resolução de sistemas esparsos, sendo o seu desempenho com sistemas densos bastante semelhante ao modo de pivotagem parcial, descrito anteriormente.

A complexidade computacional dos algoritmos resultantes é idêntica à do método original de eliminação Gaussiana, uma vez que as operações de permuta necessárias nos elementos de A não são contabilizadas na medida de complexidade usada. É importante realçar que, com alguns tipos de matrizes, a pivotagem pode ser uma operação supérflua, nomeadamente em situações onde A é uma matriz *diagonal dominante*. Diz-se que uma matriz $A \in \mathbb{R}^{n\times n}$ é diagonal dominante (por linhas) se

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \qquad (i = 1,2,\ldots,n)$$

e *estritamente* diagonal dominante caso:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \qquad (i = 1,2,\ldots,n)$$

É possível demonstrar que o método de eliminação Gaussiana com escolha parcial de pivots não efectua qualquer permuta de linhas quando aplicado a uma matriz *A estritamente diagonal dominante* (Kincaid e Cheney, 1991).

Problema 3.4 *Efectue as modificações necessárias no Algoritmo 3.2 para implementar uma estratégia de pivotagem parcial.*

Resolução de sistemas tridiagonais

Como mencionado, são frequentes as aplicações que envolvem a resolução de sistemas com matrizes por bandas, e em especial matrizes *tridiagonais*, onde todos os elementos não nulos estão localizados na diagonal principal, e nas diagonais imediatamente acima e abaixo da diagonal principal. Devido ao número considerável de coeficientes nulos, é conveniente armazenar neste caso os elementos da matriz em 3 vectores a, d e c (de tamanho n ou $n-1$), de acordo com a forma:

$$\begin{bmatrix} d_1 & c_1 & & & & \\ a_1 & d_2 & c_2 & & & \\ & a_2 & d_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & d_{n-1} & c_{n-1} \\ & & & & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

Estes sistemas podem ser resolvidos com versões especializadas dos algoritmos anteriores, atendendo à localização dos coeficientes não-nulos do problema. Em particular, se não for necessária a utilização de pivotagem[8], é apenas necessário eliminar a diagonal abaixo da diagonal principal para se obter uma matriz triangular superior. Também durante a fase de substituição regressiva cada equação envolve apenas duas variáveis, tornando este processo bastante mais económico que o caso geral. O *algoritmo de Thomas* permite resolver estes sistemas com um esforço de cerca de $3n$ adições e subtracções, e $5n$ multiplicações.

Algoritmo 3.3 (Algoritmo de Thomas)

Para $i = 2, 3, \ldots, n$ **fazer** (eliminar subdiagonal)

$m = a_{i-1}/d_{i-1}$

$d_i = d_i - m\, c_{i-1}$

$b_i = b_i - m\, b_{i-1}$

Fim_Para

$x_n = b_n/d_n$

Para $i = n-1, n-2, \ldots, 1$ **fazer** (substituição regressiva)

$x_i = (b_i - c_i x_{i+1})/d_i$

Fim_Para

Implementações eficientes para a resolução geral de sistemas com matrizes por bandas podem ser encontrados na biblioteca LAPACK[9] (Anderson et al., 1992).

Escala do problema e escolha dos pivots

Em qualquer dos métodos anteriores a escolha do pivot pode depender fortemente da escala do problema, nomeadamente das unidades usadas para representar as diferentes variáveis físicas intervenientes. O uso de unidades do Sistema Internacional (por exemplo, com concentrações expressas em mol/m^3, temperaturas em K, e energia em J) implica muito frequentemente que existam no mesmo problema grandezas com várias ordens de magnitude de diferença entre si. Esta diferença pode condicionar o desempenho das estratégias anteri-

[8] Por exemplo com matrizes simétricas *positivas definidas* (Kincaid e Cheney, 1991). Uma matriz real simétrica A é positiva definida quando $\forall x \in R^n$, $x \neq 0 \Rightarrow x^T A x > 0$. Neste caso todos os valores próprios de A são também positivos. Se uma matriz simétrica tiver valores próprios nulos, esta será singular.

[9] Descrita no Apêndice A.

ores, introduzindo erros desnecessários. Por exemplo, o sistema

$$\begin{bmatrix} 10 & 1000 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1000 \\ 2 \end{bmatrix}$$

é equivalente ao sistema:

$$\begin{bmatrix} 0{,}01 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

No entanto, no primeiro caso, a existência de coeficientes de maior magnitude na primeira equação pode influenciar artificialmente a escolha dos multiplicadores. Torna-se portanto útil "condicionar" os sistemas lineares antes da sua resolução.

Condicionamento de sistemas lineares

O objectivo desta tarefa consiste em obter um sistema equivalente, com uma matriz que contenha apenas coeficientes de magnitudes reduzidas (e semelhantes entre eles), uma vez que a sensibilidade a problemas de escala será em geral menor neste caso. O condicionamento de sistemas lineares pode ser efectuado através da introdução de *factores de escala* para as diferentes equações, e ainda através da *normalização de variáveis*:

1. Quando as componentes x_i são calculadas com precisão semelhante (i.e., quando possuem aproximadamente o mesmo número de algarismos significativos), os resíduos $|A_i x - b_i|$ (onde A_i representa a linha i da matriz A) podem assumir magnitudes bastante diferentes. Por exemplo, é expectável que um balanço energético $\sum_i H_i x_i$ = constante, onde cada termo H_i é da ordem de 10^6 ou 10^9J produza um resíduo maior que um balanço mássico $\sum_i x_i = 1$, onde as fracções x_i são inferiores à unidade. Neste caso é aconselhável introduzir *factores de escala* para cada equação, no sentido de tornar os resíduos comparáveis. Isto pode ser feito através da introdução de uma matriz elementar diagonal

$$D_e = \begin{bmatrix} d_1 & & & 0 \\ & d_2 & & \\ & & \ddots & \\ 0 & & & d_n \end{bmatrix}$$

usada para pré-multiplicar o sistema inicial:

$$Ax = b \quad \Leftrightarrow \quad (D_e A)x = (D_e b) \tag{3.17}$$

Para minimizar os erros resultantes deste processamento, os factores de escala são muitas vezes escolhidos na forma $d_i = \beta^{e_i}$, correspondendo β à base de numeração utilizada (por exemplo decimal, binária, etc.), e sendo e_i um inteiro que permite obter d_i com uma magnitude próxima do valor pretendido[10].

2. Quando se prevê que as componentes da solução tenham magnitudes bastante diferentes é possível proceder à sua *normalização*, definindo novas quantidades $x_{i,N} = x_i / x_{i,\text{ref}}$, onde $x_{i,\text{ref}}$ representa um valor de referência (típico) para a componente i da solução. Estas componentes podem também ser combinadas numa matriz elementar diagonal:

$$D_v = \begin{bmatrix} x_{1,\text{ref}} & & & 0 \\ & x_{2,\text{ref}} & & \\ & & \ddots & \\ 0 & & & x_{n,\text{ref}} \end{bmatrix}$$

Efectuando esta substituição em (3.17), resulta o sistema equivalente

$$(D_e A D_v) x_N = (D_e b) \tag{3.18}$$

com $x = D_v x_N$. Para matrizes D_v e D_e escolhidas de forma apropriada, é preferível resolver (3.18) em vez do sistema original $Ax = b$, uma vez que a resolução do primeiro será mais imune a erros numéricos durante o processo de eliminação Gaussiana.

Em algumas situações, o desconhecimento do significado físico do problema não permite efectuar estas transformações (por exemplo, em bibliotecas matemáticas de uso geral). Nestes casos é contudo possível aplicar métodos genéricos para *equilibrar* ou *balancear* a matriz A, antes de resolver o sistema
126 $Ax = b$ (Golub e Van Loan, 1989). Tal como no caso anterior, o sistema original é transformado num equivalente

$$(D_e A D_v) x^* = (D_e b) \quad \Leftrightarrow \quad \boxed{A^* x^* = b^*}$$

onde os elementos da matriz A^* têm magnitudes semelhantes.

Existem vários modos de atingir este objectivo. Uma abordagem procura obter

$$\forall i = 1, 2, \ldots, n \quad \max_j |a^*_{ij}| \simeq \max_i |a^*_{ij}|, \quad \forall j = 1, 2, \ldots, n$$

ou seja, aproximar as magnitudes máximas em cada coluna e linha correspondentes. A biblioteca LAPACK (Anderson et al., 1992) inclui este pré-processamento

[10] Isto permite manter intacta a mantissa dos coeficientes originais, ajustando apenas o seu expoente (representado como uma quantidade inteira, como descrito na Secção 1.1.2).

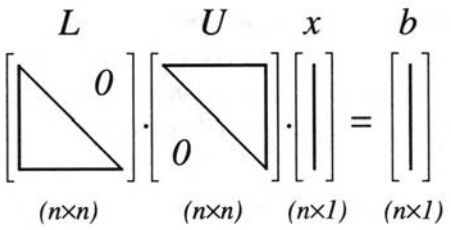


Figura 3.5 Estrutura do sistema de equações $Ax = b$ com a decomposição LU.

nas rotinas para resolução de sistemas lineares gerais. É também possível efectuar a minimização directa da soma dos quadrados dos valores absolutos dos logaritmos dos elementos não-nulos de A (Curtis e Reid, 1972); este método é usado na biblioteca HSL (AEA Technology, 1978). Um método alternativo é apresentado em Gill et al. (1981, Secção 8.7.3.2).

3.2.2 Decomposição LU

A decomposição LU de uma matriz $A \in \mathbb{R}^{n \times n}$

$$A = L \cdot U$$

produz duas matrizes também de dimensões $n \times n$, sendo L triangular inferior e U triangular superior. A principal vantagem desta decomposição reside na facilidade de resolução de um sistema de equações $Ax = b$, depois de conhecidos os seus factores L e U (Figura 3.5). Com efeito, partindo da forma

$$LU\,x = b \tag{3.19}$$

e usando a propriedade associativa do produto matricial é possível escrever $L(Ux) = b$, definindo:

$$Ux = y \tag{3.20}$$

Substituindo esta expressão em (3.19) resulta:

$$Ly = b$$

Este sistema é triangular inferior, podendo ser resolvido facilmente por substituição progressiva, com um esforço de $O(n^2)$ operações. Conhecido y, basta resolver o sistema (3.20) (também triangular) para obter a solução x do sistema de equações original. A decomposição LU de uma matriz $A \in \mathbb{R}^{n \times n}$ permite pois resolver um sistema $Ax = b$ genérico com um esforço de apenas cerca de n^2 adições / subtracções e um número idêntico de multiplicações / divisões.

Existem muitas semelhanças entre a decomposição LU e o método de eliminação Gaussiana descrito anteriormente. Por exemplo, tendo presente a equação (3.10), e uma vez que o resultado final do processo de eliminação é uma matriz triangular superior (aqui designada por U') pode escrever-se

$$(M_{n-1}\cdots M_2M_1)A = U' \quad \Rightarrow \quad A = (M_{n-1}\cdots M_2M_1)^{-1}U' \quad \Rightarrow$$
$$\Rightarrow \quad A = (M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1})\,U'$$

sabendo-se que a inversa de cada uma das matrizes elementares M_k existe, e pode ser calculada através de (3.4). Usando (3.12), obtém-se

$$M_k^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & m_{k+1,k} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & m_{n,k} & 0 & \cdots & 1 \end{bmatrix}$$

(portanto apenas com a troca de sinal dos elementos m_{ik} na coluna k), e de (3.11) resulta

$$M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1} = (I + v_1e_1^{\mathrm{T}})(I + v_2e_2^{\mathrm{T}})\cdots(I + v_{n-1}e_{n-1}^{\mathrm{T}}) =$$
$$= I + v_1e_1^{\mathrm{T}} + v_2e_2^{\mathrm{T}} + \cdots + v_{n-1}e_{n-1}^{\mathrm{T}} =$$
$$= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{bmatrix}$$

atendendo à ortogonalidade entre a maioria dos vectores e_i e v_i. Como o produto das matrizes elementares é uma matriz triangular inferior, é possível pois estabelecer as relações

$$\boxed{L = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}} \quad \text{e} \quad \boxed{U = U'} \tag{3.21}$$

entre (pelo menos) uma possibilidade de efectuar a decomposição LU e o processo de eliminação Gaussiana.

Cálculo da decomposição LU

O modo mais directo de obter as matrizes L e U parte da definição:

$$\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & nn \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{3.22}$$

Facilmente se verifica que esta igualdade corresponde a n^2 equações não-lineares (uma para cada coeficiente de A), envolvendo $n(n+1)$ incógnitas[11]. O sistema não terá portanto solução única (em geral), sendo possível fixar condições adicionais de modo a simplificar a sua resolução.

Uma escolha frequente consiste em *igualar à unidade* os elementos da diagonal de uma das matrizes triangulares. Consoante a matriz escolhida surgem duas variantes na decomposição LU:

- na *variante de Doolittle* (a mais usada) faz-se $l_{ii} = 1$.
- na *variante de Crout* escolhe-se $u_{ii} = 1$.

No final de cada uma destas especificações o sistema contém exactamente n^2 incógnitas, podendo ser resolvido unicamente. Estabelecida a unicidade da decomposição, facilmente se conclui que as equações (3.21) identificam as matrizes L e U da variante de Doolittle.

A decomposição pode ser obtida directamente a partir de (3.22), varrendo alternadamente as linhas e colunas de A a partir do canto superior esquerdo, e expressando cada nova variável encontrada em função das já conhecidas. Obtém-se assim um algoritmo com três ciclos iterativos:

[11] Cada matriz triangular tem $n(n+1)/2$ coeficientes não nulos.

Algoritmo 3.4 (Factorização LU de Doolittle)

Para $k = 1, 2, \ldots, n$ **fazer** (índice de varrimento)
$l_{kk} = 1$
Para $j = k, k+1, \ldots, n$ **fazer** (linhas)
$$u_{kj} = a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj}$$
Fim_Para
Para $i = k+1, k+2, \ldots, n$ **fazer** (colunas)
$$l_{ik} = \left(a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk}\right) / u_{kk}$$
Fim_Para
Fim_Para

As semelhanças entre este algoritmo e o correspondente à eliminação Gaussiana são bastante óbvias. Em particular, as condições para a existência da factorização (pivots u_{kk} não nulos) são partilhadas pelo método de eliminação Gaussiana sem pivotagem. Este algoritmo requer também $O(n^3/3)$ operações[12], possuindo uma complexidade idêntica à do método de eliminação. Deste modo, resolver um único sistema linear através de qualquer destes dois métodos tem um custo semelhante; no entanto, quando é necessário obter a solução de mais do que um sistema linear com a mesma matriz A torna-se preferível o uso da decomposição LU. Embora o 1º passo em cada uma destas alternativas tenha um custo de $O(2n^3/3)$ — tanto para resolver o 1º sistema linear por eliminação, como para calcular os factores L e U e a solução a partir destes — estando disponíveis os factores L e U torna-se possível resolver os sistemas seguintes com um custo de apenas $O(n^2)$ operações cada, o que é bastante vantajoso.

Exemplo 3.2 (Cálculo da matriz inversa)

Como mencionado, a inversão de matrizes não corresponde ao método mais conveniente para a resolução de sistemas de equações lineares; no entanto, é necessário, em algumas situações, formar explicitamente a inversa de uma matriz quadrada.

Um método para obter A^{-1} consiste na resolução do conjunto de n sistemas lineares $AX = I$, onde I representa a matriz identidade de ordem correspondente, e $X \in \mathbb{R}^{n \times n}$ a solução pretendida. Neste caso, a utilização da factorização LU é preferível ao uso da eliminação Gaussiana, uma vez que permite

[12]Tanto em termos de multiplicações / divisões como de adições / subtracções.

resolver os n sistemas lineares equivalentes $Ax_k = e_k$, $k = 1, 2, \ldots, n$, onde e_k corresponde à coluna k da matriz identidade, com um esforço individual de $O(n^2)$ operações, depois de obtida a factorização LU de A. Os passos envolvidos neste processo são:

- Definir $y_j = Ux_j$ e resolver $Ly_j = e_j$, por substituição progressiva.
- Depois de conhecido y_j, resolver $Ux_j = y_j$ por substituição regressiva, para obter o vector coluna x_j de X.

Este método possui também a vantagem de permitir a utilização de código desenvolvido para a resolução de sistemas lineares na inversão de matrizes.

Uma outra possibilidade de obtenção da inversa seria utilizar $A^{-1} = (LU)^{-1} = U^{-1}L^{-1}$, e calcular as inversas de cada um destes factores em separado (que correspondem também a matrizes triangulares). No entanto, este método revela-se mais trabalhoso, devido à necessidade de inversão separada, seguida da multiplicação dos factores obtidos que, como se viu, é sempre uma operação de complexidade considerável.

Problema 3.5 *Identifique a complexidade do cálculo de A^{-1} através do método descrito no exemplo anterior. Compare este valor com o avançado para o outro método descrito no final da Secção relativa à complexidade do processo de eliminação Gaussiana.*

A forma do Algoritmo 3.4 permite que a decomposição se processe armazenando os elementos l_{ij} e u_{ij} no espaço de memória correspondente à própria matriz A, uma vez que os elementos a_{ij} deixam de ser necessários depois de usados pela primeira vez. Neste caso, usando por exemplo a forma de Doolittle, obtém-se no final:

$$[A] \rightarrow \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & u_{nn} \end{bmatrix}$$

Decomposição LU com pivotagem

Se as quantidades u_{kk} no Algoritmo 3.4 se tornarem nulas (ou, de forma equivalente, os pivots m_{ij} em (3.9) se anularem), a factorização LU não se encontra definida. As considerações desenvolvidas no método de eliminação Gaussiana relativamente à necessidade de evitar pivots com magnitude reduzida são também aplicáveis a este caso. A pivotagem é uma operação que melhora a

estabilidade numérica da decomposição LU, podendo também ser usada nas formas de pivotagem de *linhas* (parcial), ou de *linhas e colunas* (completa).

Quando é utilizada a permuta de linhas, tem-se a partir de (3.15)

$$(M_{n-1}P_{n-1}\cdots M_2P_2M_1P_1)\ A = U$$

onde cada P_i representa uma matriz permutação correspondente à troca de 2 linhas[13]. É possível demonstrar que este reordenamento de A pode ser conseguido, de forma equivalente, *antes do processo de factorização ser iniciado*, tal como aconteceria com a aplicação do método de eliminação Gaussiana (Meyer, 2000). Deste modo, a equação anterior pode ser escrita na forma

$$(\bar{M}_{n-1}\cdots\bar{M}_2\bar{M}_1)\underbrace{(P_{n-1}\cdots P_2P_1)}_{P}A = U$$

onde as matrizes $\bar{M}_i$ representam as operações elementares originais, tendo agora em atenção o ordenamento diferente das linhas de A. Isto permite expressar a decomposição LU com pivotagem de linhas na forma

$$P\cdot A = L\cdot U \tag{3.23}$$

com $L = \bar{M}_1^{-1}\bar{M}_2^{-1}\cdots\bar{M}_{n-1}^{-1}$. À semelhança do método de eliminação Gaussiana, a decomposição com pivotagem de linhas (3.23) encontra-se sempre definida, desde que A seja não-singular. Este facto pode ser usado para diagnosticar a singularidade de uma matriz.

No caso de utilização do processo de *pivotagem completa* (de linhas e colunas) na decomposição LU, a equação (3.16) permite obter, de modo semelhante

132

$$P\cdot A\cdot Q = L\cdot U$$

onde P representa a matriz de permutações relativa à permuta de linhas e Q a matriz referente à troca de colunas. Este método é mais robusto que a decomposição com pivotagem parcial, uma vez que possibilita uma maior liberdade na escolha dos pivots em cada passo da factorização.

Exemplo 3.3 (Cálculo de determinantes)

Para além da sua utilidade no cálculo da matriz inversa, o conhecimento dos factores L e U permite também obter rapidamente o determinante de uma

[13]E, deste modo, cada matriz P_i será simétrica, diferindo da matriz identidade correspondente em apenas 2 linhas e 2 colunas.

matriz quadrada. Com efeito, a partir de $A = LU$ tem-se:

$$|A| = |LU| = |L||U|$$

Uma vez que o determinante de matrizes triangulares é dado pelo produto dos seus elementos diagonais fica apenas

$$|A| = |U| = \prod_{k=1}^{n} u_{kk}$$

na variante de Doolittle, confirmando-se uma vez mais que os elementos diagonais de U necessitam de ser todos não-nulos para que a matriz A correspondente seja invertível.

Quando é usada uma decomposição com pivotagem parcial tem-se:

$$|P||A| = |U|$$

Uma vez que $P^{-1} = P^{\mathrm{T}}$ (devido à sua ortogonalidade) e atendendo a $|P| = |P^{\mathrm{T}}|$, resulta da equação anterior:

$$|A| = |P| \prod_{k=1}^{n} u_{kk} \tag{3.24}$$

É conhecido o facto de que $|P| = 1$ se o número de permutações de linhas for par, e $|P| = -1$ em caso ímpar, o que permite avaliar facilmente $|A|$ a partir da equação anterior. A utilização da decomposição com pivotagem completa pode ser analisada de forma semelhante.

É importante notar que a implementação directa de (3.24) pode produzir facilmente situações de "overflow" ou "underflow", mesmo que a matriz A seja não singular, ou que $|A|$ tenha uma magnitude razoável. Para evitar isto é conveniente utilizar neste cálculo *factores de escala* do resultado, evitando ultrapassar os limites da representação numérica utilizada.

Decomposição de Cholesky

Quando a matriz A é *simétrica* ($A = A^{\mathrm{T}}$) é conveniente definir $L = U^{\mathrm{T}}$, preservando nestes factores a simetria de A. Este método é conhecido como a *decomposição de Cholesky*[14]

$$A = LL^{\mathrm{T}} \tag{3.25}$$

[14]Tarefa que em princípio deverá envolver metade do esforço anterior, uma vez que neste caso apenas se torna necessário calcular um dos factores.

onde L é uma matriz triangular inferior (desempenhando o papel da matriz L anterior). No entanto, esta decomposição apenas se encontra definida para matrizes simétricas e *positivas definidas*.

O cálculo da decomposição de Cholesky pode ser efectuado a partir da definição, usando um varrimento das equações (3.22) por colunas (a partir da esquerda), de modo semelhante ao usado na decomposição LU. Por exemplo, é possível mostrar que as equações relativas aos elementos diagonais de L são neste caso

$$\sum_{j=1}^{k-1} l_{kj}^2 + l_{kk}^2 = a_{kk} \qquad (k = 1, 2, \ldots, n) \tag{3.26}$$

de onde se retira

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2} \qquad (k = 1, 2, \ldots, n)$$

desde que a quantidade no interior da raiz quadrada seja positiva[15]. O algoritmo completo da decomposição pode ser escrito na forma:

Algoritmo 3.5 (Decomposição de Cholesky)

`Para` $k = 1, 2, \ldots, n$ `fazer` (índice de varrimento)

$$l_{kk} = \left(a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2\right)^{1/2}$$

`Para` $i = k+1, k+2, \ldots, n$ `fazer` (abaixo da diagonal)

$$l_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj}\right) / l_{kk}$$

`Fim_Para`

`Fim_Para`

Tal como as factorizações anteriores, este algoritmo possui uma complexidade de $O(n^3/3)$[16]. É importante notar que, a partir de (3.26) se tem

$$a_{kk} = \sum_{j=1}^{k} l_{kj}^2$$

e portanto $|l_{kj}| \leq \sqrt{a_{kk}}$, para $j = 1, 2, \ldots, k$. Daqui é possível concluir que os elementos de L possuem sempre magnitudes limitadas pela raiz quadrada

[15] O que apenas é possível garantir para matrizes A positivas definidas, e daí a necessidade de restrição da decomposição a matrizes deste tipo.

[16] Cerca de metade do número de operações correspondentes à decomposição LU equivalente.

do elemento diagonal de A correspondente, nunca assumindo portanto valores consideráveis relativamente a A, *mesmo sem o uso de pivotagem.*

Decomposições LDU e $\mathbf{LDL^T}$

De uma forma mais genérica, é ainda possível definir a decomposição

$$A = LDU \tag{3.27}$$

onde L e U são matrizes triangulares inferior e superior (respectivamente), com elementos diagonais *unitários*, e D é uma matriz diagonal. Nesta decomposição, é fácil associar a variante LU de Doolittle ao produto $L(DU)$, e a variante LU de Crout ao produto $(LD)U$. Com matrizes simétricas esta decomposição pode ser escrita na forma

$$A = LDL^T$$

pressupondo que não é necessário efectuar permutações para obter esta decomposição, especialmente no caso de matrizes simétricas *indefinidas*[17]. Para matrizes positivas definidas esta forma é equivalente à decomposição de Cholesky, tendo como principal vantagem a possibilidade de evitar o cálculo de raízes quadradas, uma vez que os elementos de D correspondem aos quadrados dos elementos diagonais respectivos na decomposição de Cholesky.

No caso de matrizes simétricas, e ao contrário da factorização de Cholesky, a decomposição $\mathrm{LDL^T}$ pode ser possível com algumas matrizes simétricas não-definidas. A sua existência não pode ser contudo assegurada no caso geral, mesmo considerando a possibilidade de serem efectuadas permutações simétricas de linhas e colunas de A (na forma $PAP^T = LDL^T$), como se ilustra no exemplo seguinte (Gill et al., 1991):

Exemplo 3.4 (Decomposição $\mathrm{LDL^T}$ inexistente)

Para

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(que constitui já por si uma matriz permutação), qualquer permutação simétrica produz de novo A, uma vez que os elementos da diagonal se mantêm sempre na diagonal após este processo. A decomposição $\mathrm{LDL^T}$ poderia ser escrita

[17] Para preservar a simetria da decomposição seria necessário efectuar simultaneamente a mesma permuta numa dada linha e coluna.

neste caso através de:

$$\begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \begin{bmatrix} 1 & l_{21} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

O conjunto de equações anteriores originaria no entanto as contradições $d_1 = 0$ e $d_1 l_{21} = 1$, que inviabilizam a existência desta factorização.

Mesmo quando existente, a decomposição $\mathrm{LDL}^{\mathrm{T}}$ de matrizes indefinidas pode apresentar propriedades numéricas pouco satisfatórias, sendo esta factorização habitualmente usada apenas (tal como a decomposição de Cholesky) com matrizes simétricas positivas definidas.

3.2.3 Condição numérica de uma matriz e análise de erros

No final da aplicação de um método directo para a resolução de um sistema linear, se os cálculos forem efectuados com precisão finita (por exemplo, num computador), é natural que a solução obtida $\bar{x}$ difira da solução exacta do sistema, x^*. O erro $\Delta x = e(\bar{x}) \equiv \|x^* - \bar{x}\|$ dependerá em geral da natureza do método usado, bem como do próprio sistema linear. Usando o conceito de *normas* de vectores e matrizes, definido na Secção 1.3, torna-se possível estimar o efeito de erros ou incertezas na especificação dos coeficientes da matriz A ou do vector b na solução encontrada:

1. Para perturbações ΔA na matriz dos coeficientes é possível escrever o sistema equivalente

$$(A + \Delta A)(x + \Delta x) = b$$

sendo $x + \Delta x$ a solução exacta deste sistema modificado. Desenvolvendo a equação anterior vem

$$Ax + A\Delta x + \Delta A x + \Delta A \Delta x = b \quad \Rightarrow \quad A\Delta x + \Delta A(x + \Delta x) = 0$$

de onde se obtém, supondo A invertível:

$$\Delta x = -A^{-1}\Delta A(x + \Delta x)$$

Tomando normas nesta equação fica

$$\|\Delta x\| \leq \|A^{-1}\| \, \|\Delta A\| \, \|x + \Delta x\|$$

e portanto:

$$\frac{\|\Delta x\|}{\|x+\Delta x\|} \leq \|A^{-1}\|\,\|A\|\,\frac{\|\Delta A\|}{\|A\|} \tag{3.28}$$

2. Com perturbações Δb no lado direito é possível escrever

$$A(x+\Delta x) = b + \Delta b$$

de onde resulta

$$A\Delta x = \Delta b \quad \Rightarrow \quad \Delta x = A^{-1}\Delta b$$

assumindo que a matriz A é invertível. Aplicando normas nesta expressão resulta

$$\|\Delta x\| \leq \|A^{-1}\|\,\|\Delta b\| = \|A^{-1}\|\,\|Ax\|\,\frac{\|\Delta b\|}{\|b\|} \leq \|A^{-1}\|\,\|A\|\,\|x\|\,\frac{\|\Delta b\|}{\|b\|}$$

e logo:

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A^{-1}\|\,\|A\|\,\frac{\|\Delta b\|}{\|b\|} \tag{3.29}$$

A quantidade

$$\kappa(A) = \|A^{-1}\|\,\|A\| \geq 0$$

que desempenha em (3.28) e (3.29) o papel de "factor de amplificação" do erro original na solução é designada por *número de condição* ou *índice de condição* da matriz A. O seu valor exacto depende da norma escolhida, embora a sua magnitude seja em geral bastante insensível à definição usada[18].

É possível mostrar também que $\kappa(A) \geq 1$, para qualquer matriz A não-singular, e qualquer norma induzida. Tendo presente (1.30) e as propriedades da matriz identidade I de ordem correspondente, tem-se

$$\|I\| = \|I^{-1}\| = 1 \quad \Rightarrow \quad \kappa(I) = 1$$
$$\|I\| = 1 \quad \Rightarrow \quad 1 = \|A^{-1}A\| \leq \|A^{-1}\|\|A\| = \kappa(A)$$

e consequentemente $\kappa(A) \geq 1$. Matrizes com índices de condição baixos (i.e., próximos da unidade) originam sistemas lineares ditos *bem-condicionados*. De acordo com (3.28) e (3.29), se este índice for elevado, qualquer pequena incerteza na especificação dos dados do problema tem frequentemente como consequência um erro elevado na solução obtida. Os sistemas lineares com $\kappa(A)$ elevados correspondem pois a problemas *mal-condicionados* (Secção 1.2.6). As

[18]Tendo por exemplo presentes os limites (1.28) e (1.29).

matrizes A correspondentes são designadas por vezes como *quase-singulares*, uma vez que no caso limite, com $\kappa(A) = \infty$, se tem A singular. Estes conceitos estão ilustrados no exemplo seguinte.

Exemplo 3.5 (Exemplo 1.4 revisitado)

De forma equivalente à proposta no Problema 1.10 é possível calcular $\kappa(A)$ para avaliar o condicionamento do sistema descrito no Exemplo 1.4. A partir de (1.15) tem-se:

$$A = \begin{bmatrix} 1 & 2 \\ 1/2 & 1+\epsilon \end{bmatrix}, \quad A^{-1} = \frac{1}{\epsilon}\begin{bmatrix} 1+\epsilon & -2 \\ -1/2 & 1 \end{bmatrix}$$

Neste caso

$$\|A\|_1 = \max\left\{\frac{3}{2}, 2+|1+\epsilon|\right\}, \quad \|A\|_\infty = \max\left\{3, \frac{1}{2}+|1+\epsilon|\right\}$$

e:

$$\|A^{-1}\|_1 = \max\left\{\frac{3}{|\epsilon|}, \frac{1}{2|\epsilon|}+\left|\frac{1+\epsilon}{\epsilon}\right|\right\},$$

$$\|A^{-1}\|_\infty = \max\left\{\frac{3}{2|\epsilon|}, \frac{2}{|\epsilon|}+\left|\frac{1+\epsilon}{\epsilon}\right|\right\}$$

No caso limite

$$\lim_{\epsilon\to 0}\|A\|_1 = \lim_{\epsilon\to 0}\|A\|_\infty = 3$$

mas:

$$\lim_{\epsilon\to 0}\|A^{-1}\|_1 = \lim_{\epsilon\to 0}\|A^{-1}\|_\infty = \infty$$

Portanto $\lim_{\epsilon\to 0}\kappa(A) = \infty$, devido ao termo $\|A^{-1}\|$.

Em termos numéricos, é preferível avaliar a proximidade de uma matriz A da singularidade usando $\kappa(A)$ em vez do seu determinante, como se pode constatar a partir dos exemplos

$$A_1 = \begin{bmatrix} 0{,}1 & & & \\ & 0{,}1 & & \\ & & \ddots & \\ & & & 0{,}1 \end{bmatrix} = 0{,}1\ I_{20} \quad \Rightarrow \quad \kappa(A_1) = 1,\ |A_1| = 10^{-20}$$

$$A_2 = \begin{bmatrix} 10 & & & \\ & \ddots & & \\ & & 10 & \\ & & & 10^{-6} \end{bmatrix} \quad (n = 20) \quad \Rightarrow \quad \kappa(A_2) = 10^7, \ |A_2| = 10^{13}$$

dentro da família de normas-p definida na Secção 1.3. Se o último coeficiente de A_2 não for exacto, a matriz A_2 pode ser considerada próxima da singularidade, ao contrário de A_1, apesar de $|A_2| \gg |A_1|$. Para matrizes diagonais do tipo $A = \alpha I$, o determinante destas matrizes corresponde ao produto dos seus valores próprios, bastante dependente da escala do problema que originou A, enquanto que o índice de condição definido anteriormente não depende desta escala.

Dada a importância deste parâmetro, surge frequentemente a necessidade de calcular $\kappa(A)$. O seu valor pode ser determinado exactamente quando A possui uma estrutura simples, facilmente invertível. Na maioria dos casos, torna-se mais económica a determinação apenas de uma *estimativa* da sua magnitude, evitando o cálculo de A^{-1} apenas para este fim. A maioria dos algoritmos disponíveis para este fim utiliza os factores L e U de A (ou uma factorização equivalente) para estimar este índice com um esforço de $O(n^2)$ operações (Golub e Van Loan, 1989; Demmel, 1997). Este processo torna expedito o cálculo deste índice simultaneamente com (ou após) a resolução de um sistema de equações lineares.

3.2.4 Melhoramento iterativo da solução

Para além do efeito de propagação de eventuais incertezas nos coeficientes do sistema linear, devido ao índice de condição da matriz A, a solução obtida $\bar{x}$ é também sensível a erros de arredondamento e truncatura produzidos durante a aplicação de um método directo, com efeitos potencialmente cumulativos. No final da aplicação é sempre possível calcular o resíduo correspondente:

$$r = A\bar{x} - b$$

Se esta quantidade for pequena a solução calculada pode ser tomada como uma boa aproximação da solução exacta[19]. Quando isto não acontece é possível refinar $\bar{x}$ *iterativamente*, produzindo uma sequência de vectores

$$\bar{x}^{(0)}, \bar{x}^{(1)}, \bar{x}^{(2)}, \ldots$$

[19]Pelo menos no sentido de que $A\bar{x}$ é uma boa aproximação de b.

eventualmente convergente para a solução exacta x^*, onde $\|\bar{x}^{(k)} - x^*\| \to 0$. Definindo por exemplo

$$x = \bar{x}^{(0)} + \Delta x^{(0)} \tag{3.30}$$

vem de $Ax = b$

$$A(\bar{x}^{(0)} + \Delta x^{(0)}) = b$$

ou seja:

$$A\Delta x^{(0)} = b - A\bar{x}^{(0)} = r^{(0)} \tag{3.31}$$

Este sistema de equações pode ser resolvido em ordem a $\Delta x^{(0)}$, de forma a corrigir a solução inicial $\bar{x}^{(0)}$ em (3.30)[20]. Depois de obtido $\Delta x^{(0)}$, a solução inicial pode ser melhorada através de:

$$\bar{x}^{(1)} = \bar{x}^{(0)} + \Delta x^{(0)}$$

Como a resolução do sistema (3.31) está também sujeita a erros, a estimativa $\bar{x}^{(1)}$ poderá não corresponder ainda à solução exacta. O procedimento anterior poderá então ser aplicado novamente para melhorar esta estimativa. Definindo novamente

$$x = \bar{x}^{(1)} + \Delta x^{(1)}$$

e resolvendo, de modo semelhante a (3.31), o sistema linear

$$A\Delta x^{(1)} = b - A\bar{x}^{(1)} = r^{(1)}$$

obtém-se a segunda correcção da solução. Este processo iterativo pode ser repetido de forma idêntica, até o valor das correcções sucessivas ser desprezável.

Deve notar-se que apesar do processo anterior permitir melhorar com frequência a estimativa da solução, este nem sempre é convergente, no sentido em que $\|\Delta x^{(k)}\| \to 0$. Se os erros cometidos na resolução dos sistemas (3.31) de-
140 penderem da condição da matriz A, a solução $\Delta x^{(k)}$ pode não ter a precisão necessária para melhorar a estimativa da solução, a partir de determinada fase. Para minimizar este comportamento, sempre que possível, é vantajoso efectuar o cálculo dos resíduos $r^{(k)} = b - A\bar{x}^{(k)}$ em precisão superior (por exemplo, dupla) à utilizada nas restantes operações (Golub e Van Loan, 1989).

3.3 Métodos indirectos (iterativos)

A resolução de sistemas lineares $Ax = b$ por métodos *indirectos* ou *iterativos* envolve o refinamento de uma estimativa inicial da solução $x^{(0)}$ através de um

[20]No caso de ter sido usada a decomposição LU para resolver o sistema inicial, este passo adicional representará apenas um acréscimo de $O(n^2)$ operações.

processo iterativo, gerando uma sequência de valores:

$$x^{(0)}, x^{(1)}, x^{(2)}, \ldots$$

Colocam-se neste caso as possibilidades de:

- o processo iterativo divergir;
- quando o processo iterativo converge, o volume de cálculos necessários para atingir uma exactidão pré-definida ser desconhecido;

sendo oportuno indagar a aplicabilidade destes métodos face aos seus congéneres directos (de natureza exacta), estudados anteriormente.

Os métodos iterativos são competitivos quando o número de operações necessárias para a resolução de um sistema linear é bastante inferior às $O(2n^3/3)$ operações necessárias para a sua resolução através de um método directo. Deve também ter-se em consideração que a resolução de sistemas lineares é frequentemente um passo intermédio na aplicação de um algoritmo mais complexo, não sendo necessário por isso obter sempre a sua solução com toda a precisão disponível (pelo menos antes das etapas finais). Assim, os métodos iterativos são geralmente aplicados na resolução de sistemas de *grandes dimensões*, geralmente *esparsos* (com muitos coeficientes nulos), tais como os resultantes da resolução de diagramas de fabrico ("flowsheeting").

Exemplo 3.6 (Resolução de diagramas de fabrico)

Considere-se por exemplo o diagrama de fabrico simplificado da Figura 3.6. Nesta Figura, o bloco correspondente a cada unidade principal é descrito pelo conjunto de variáveis u_i, sendo as correntes que as unem representadas pelos vectores x_i. Neste caso é natural escrever as equações correspondentes ao bloco U_1 na forma:

$$f_1(x_1, x_2, x_5, u_1) = 0$$

De igual modo, o segundo bloco pode ser representado através de

$$f_2(x_2, x_3, u_2) = 0$$

e o último bloco por:

$$f_3(x_3, x_4, x_5, u_3) = 0$$

Como se pode verificar, qualquer destes conjuntos de equações envolve apenas um subconjunto das variáveis totais $\{x_1, x_2, x_3, x_4, x_5, u_1, u_2, u_3\}$. Assim, o sistema de equações que descreve todo o processo $f(x_1, x_2, x_3, x_4, x_5, u_1, u_2,$

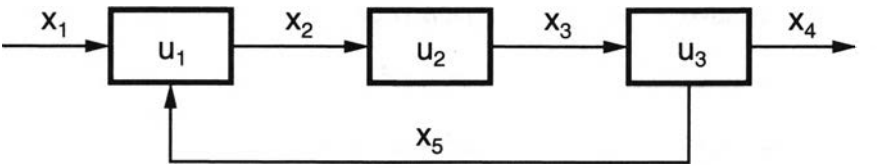


Figura 3.6 Exemplo de um diagrama de fabrico simplificado.

$u_3) = 0$, quando linearizado, produzirá uma matriz A com estrutura por blocos:

$$A = \begin{bmatrix} \square\square & \square\square & \\ & \square\square & \square \\ & \square\square\square & \square \end{bmatrix}$$

Se os vectores x_1, x_2, x_3, x_4, x_5 tiverem dimensões apreciáveis (por exemplo, quando são usados modelos detalhados ou rigorosos), a matriz anterior terá provavelmente uma estrutura esparsa, com muitos coeficientes nulos.

3.3.1 Matrizes esparsas

Designa-se por *densidade* de uma matriz a fracção de coeficientes dessa matriz que são não-nulos. A Figura 3.7 ilustra a estrutura de matrizes de teste obtidas na simulação de diagramas de fabrico, sendo a origem de cada modelo descrita na Tabela 3.1. Como se pode constatar, a maioria destes exemplos, com dimensão da ordem das centenas de variáveis, possui uma densidade próxima de 1%, sendo este valor ainda inferior nos exemplos de maior dimensão.

Para matrizes com baixa densidade, a representação habitual em memória através de um *array* bidimensional não é económica, uma vez que esta forma
142 inclui muitos elementos nulos. Nesta situação prefere-se o uso de representações especiais guardando apenas os elementos diferentes de zero. Existem diversas formas que podem ser utilizadas para o efeito, variando consoante a estrutura das matrizes e a aplicação em causa. Uma das representações mais simples utiliza dois vectores inteiros para as posições de linha e coluna, e um vector real para os coeficientes *não-nulos*, na forma:

i	j	a_{ij}
1	1	a_{11}
	5	a_{15}
2	2	a_{22}
⋮	⋮	⋮

Esta representação é conhecida como armazenagem comprimida por linhas

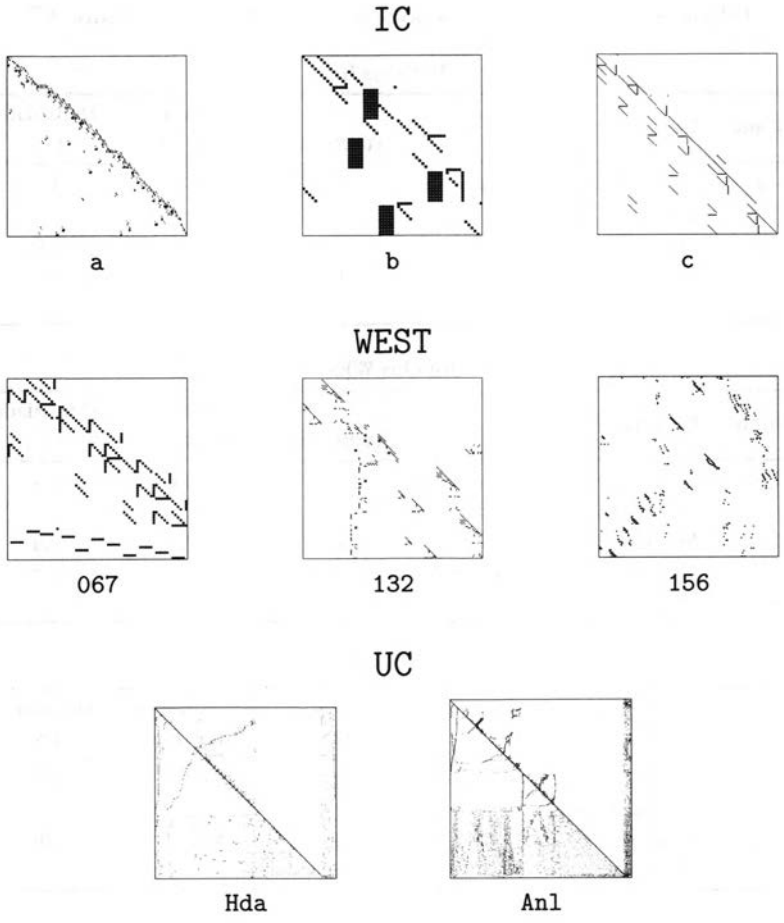


Figura 3.7 Estruturas de matrizes correspondentes a modelos de processos químicos (Duff et al., 1992). A descrição dos modelos correspondentes encontra-se na Tabela 3.1.

(*compressed row storage* — CSR). Embora esta seja uma das representações menos complexas, ela não é necessariamente a mais simples de manipular, nem a mais eficiente. Outros formatos comuns de armazenamento de matrizes esparsas podem ser encontrados em Barrett et al. (1994); Saad (2003). A representação escolhida depende por vezes da *estrutura* existente na matriz. Para além das formas especiais descritas no início deste Capítulo (triangulares, por bandas, tridiagonais, simétricas) é útil também verificar se a matriz em causa possui uma estrutura *por blocos* (Figura 3.8).

Em todos estes casos é geralmente vantajoso adoptar uma versão especializada dos algoritmos descritos anteriormente, tomando em consideração a estrutura especial da matriz.

Tabela 3.1 Descrição dos modelos representados na Figura 3.7.

Modelos IC:

Nome	Descrição	n (tamanho)	Elementos não-nulos	Densidade (%)
a	Rede de permutadores de calor	207	572	1,3
b	Processo de Cavett	59	312	9,0
c	Modelo de uma fábrica de etileno	137	411	2,2

Modelos WEST:

Nome	Descrição	n (tamanho)	Elementos não-nulos	Densidade (%)
067	Problema de Cavett com 5 componentes	67	294	6,5
132	Modelo flash rigoroso	132	414	2,4
156	Modelo de uma instalação simples	156	371	1,5

Modelos UC:

Nome	Descrição	n (tamanho)	Elementos não-nulos	Densidade (%)
Hda	Modelo de hidro-dealquilação do tolueno	2024	6505	0,16
Anl	Modelo de uma fábrica de anilina	10053	47014	0,047

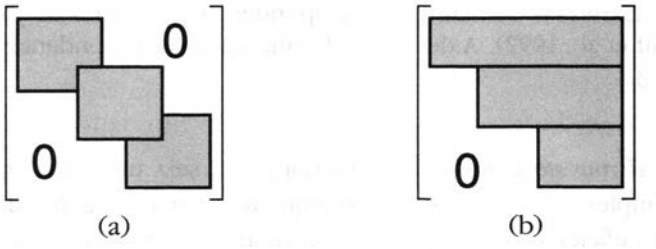


Figura 3.8 Estruturas de matrizes por blocos: (a) Diagonal por blocos. (b) Triangular superior por blocos.

Como foi referido, uma preocupação comum na resolução de sistemas lineares esparsos é a de evitar o *enchimento* da matriz, à medida que o método de resolução é aplicado. Isto corresponde ao aparecimento de mais elementos não-nulos (e portanto a um aumento da sua densidade), tornando mais morosas

(e menos atractivas) as fases posteriores do algoritmo. Por exemplo, se os factores L e U da decomposição de uma matriz tiverem bastante mais elementos diferentes de zero do que a matriz original A, a aplicação deste método poderá acarretar "bastante" trabalho nas fases de substituição regressiva e progressiva; será também necessário um espaço de memória consideravelmente superior para armazenar cada um dos factores, relativamente ao espaço necessário para guardar A.

A experiência mostra que com matrizes esparsas o grau de enchimento obtido é fortemente dependente da forma como as equações são escritas e resolvidas, sendo também influenciado pela *estratégia de pivotagem* usada. Embora as estratégias anteriores possam ser adaptadas por forma a minimizar o crescimento de termos não-nulos, este objectivo revela-se muitas vezes conflituoso com outros objectivos, tais como a obtenção da *máxima estabilidade numérica*, ou a *minimização do número total de operações* necessárias. Como referido na Secção 3.2, o rearranjo preliminar destes sistemas é geralmente vantajoso, dado permitir reduzir o número de operações elementares necessárias para a resolução do sistema, diminuindo simultaneamente a probabilidade de ocorrência de enchimento significativo, conforme observado no exemplo seguinte.

Exemplo 3.7 (Enchimento em matrizes com estrutura especial)

A factorização LU pode ser aplicada à matriz (Demmel, 1997)

$$A_1 = \begin{bmatrix} 1 & & & & & 0{,}1 \\ & 1 & & & & 0{,}1 \\ & & 1 & & & 0{,}1 \\ & & & & 1 & 0{,}1 \\ 0{,}1 & 0{,}1 & 0{,}1 & 0{,}1 & & 1 \end{bmatrix}$$

sem necessidade de pivotagem de linhas ou colunas, originando os factores:

$$A_1 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ 0{,}1 & 0{,}1 & 0{,}1 & 0{,}1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & 0{,}1 \\ & 1 & & & 0{,}1 \\ & & 1 & & 0{,}1 \\ & & & 1 & 0{,}1 \\ & & & & 0{,}96 \end{bmatrix}$$

Matrizes com a estrutura de A_1 são conhecidas como *matrizes seta*, dada a sua forma característica. O número de operações elementares necessárias para realizar esta factorização cresce neste caso apenas *linearmente* com a dimensão da matriz, em vez de depender de $O(2n^3/3)$, como no caso geral, uma vez que é apenas necessário eliminar os coeficientes da última linha. De forma semelhante, o armazenamento dos factores L e U pode ser feito num espaço

idêntico ao mínimo necessário para armazenar A_1.

Por contraste, a matriz

$$A_2 = \begin{bmatrix} 1 & 0{,}1 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0{,}1 & 1 & & & \\ 0{,}1 & & 1 & & \\ 0{,}1 & & & 1 & \\ 0{,}1 & & & & 1 \end{bmatrix}$$

obtida a partir de A_1 por inversão da ordem das linhas e colunas, origina uma decomposição LU com estrutura bastante diferente. Usando 2 casas decimais, tem-se:

$$A_2 = \begin{bmatrix} 1 & & & & \\ 0{,}1 & 1 & & & \\ 0{,}1 & -0{,}01 & 1 & & \\ 0.1 & -0{,}01 & -0{,}01 & 1 & \\ 0{,}1 & -0{,}01 & -0{,}01 & -0{,}01 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0{,}1 & 0{,}1 & 0{,}1 & 0{,}1 \\ & 0{,}99 & -0{,}01 & -0{,}01 & -0{,}01 \\ & & 0{,}99 & -0{,}01 & -0{,}01 \\ & & & 0{,}99 & -0{,}01 \\ & & & & 0{,}99 \end{bmatrix}$$

Também neste caso não foram necessárias permutações de linhas. No entanto, os factores L e U obtidos são neste caso completamente densos.

O rearranjo exacto (no sentido "óptimo") de sistemas lineares, tendo em vista a minimização do espaço de armazenagem ou do esforço de resolução é no entanto conhecido como um problema de complexidade *exponencial*, constituindo frequentemente uma etapa mais demorada que a própria resolução do sistema linear (Demmel, 1997). Por este motivo são habitualmente empregues estratégias aproximadas, com algum carácter heurístico, para melhorar a estrutura de sistemas esparsos, antes da sua resolução (Saad, 2003). Após este pré-processamento, os sistemas esparsos podem ser resolvidos através de *métodos directos* adaptados à sua estrutura especial, ou de métodos iterativos como os descritos em seguida.

3.3.2 Métodos iterativos básicos

O desenvolvimento de alguns dos principais métodos iterativos usados na resolução de sistemas lineares envolve a utilização de conceitos de Optimização, estando por isso fora do âmbito destas notas. Nesta Secção são apenas apresentados alguns métodos iterativos básicos. Um tratamento mais aprofundado deste assunto pode ser encontrado em Barrett et al. (1994); Greenbaum (1997); Saad (2003).

Existem bastantes famílias de métodos iterativos para resolução de sistemas de equações lineares. Ao contrário dos métodos directos, cuja resolução é garantida após um grau pré-determinado de esforço, não existe neste caso, em geral, uma garantia de convergência para a solução. Aqui, a escolha dos métodos mais apropriados é bastante dependente das características do problema em causa, uma vez que alguns métodos apenas funcionam quando aplicados a certas classes de problemas. Deste modo, torna-se necessário algum conhecimento prévio sobre as características do sistema, para seleccionar o método iterativo mais apropriado.

Método de Jacobi

A ideia fundamental deste método consiste em escolher, em cada equação, uma variável com coeficiente não-nulo, rearranjando o sistema de forma a expressar a variável individualizada em função das restantes. Por exemplo, o sistema de 3ª ordem

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

pode ser re-escrito na forma

$$\begin{cases} x_1 = (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \\ x_2 = (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22} \\ x_3 = (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \end{cases} \tag{3.32}$$

desde que $a_{11}, a_{22}, a_{33} \neq 0$. Esta equação poderá ser utilizada como base para a definição de um procedimento iterativo de resolução: partindo de uma estimativa inicial $x^{(0)}$, se ela for introduzida no lado direito de (3.32) será obtida uma nova estimativa $x^{(1)}$ da solução, que pode ser melhorada posteriormente por uso repetido de (3.32). É possível portanto resumir este método através do seguinte algoritmo (assumindo $a_{ii} \neq 0$, $i = 1, 2, \ldots, n$):

Algoritmo 3.6 (Método de Jacobi)

Dado $x^{(0)}$
Enquanto convergência ou limite de iterações não verificados
    Para $i = 1, 2, \ldots, n$ fazer

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)/a_{ii}$$

    Fim_Para
Fim_Enquanto

Como se pode adivinhar este é um dos métodos iterativos mais básicos. A sua principal vantagem reside na facilidade de implementação, embora a sua convergência seja em geral lenta.

Método de Gauss-Seidel

No algoritmo anterior não é usada a estimativa mais recente para cada variável. Quando este aspecto é considerado obtém-se o denominado método de *Gauss-Seidel*.

Algoritmo 3.7 (Método de Gauss-Seidel)

Dado $x^{(0)}$
Enquanto convergência ou limite de iterações não verificados
  Para $i = 1, 2, \ldots, n$ **fazer**

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

  Fim_Para
Fim_Enquanto

Tal como no caso anterior, este método nem sempre converge para a solução. Quando isso acontece, a sua convergência é, em geral, mais rápida do que o método de Jacobi.

Os dois métodos anteriores podem ser considerados membros de uma família

genérica de métodos definida por

$$Mx^{(k+1)} = Nx^{(k)} + b \tag{3.33}$$

onde $A = M - N$ corresponde a uma *partição* da matriz A. A matriz M é também conhecida como factor *pré-condicionador* do sistema de equações lineares. Estes métodos apenas possuem interesse prático quando é relativamente simples resolver um sistema com matriz de coeficientes M. Definindo os factores

$$L = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{21} & 0 & 0 & \cdots & 0 \\ a_{31} & a_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & 0 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

ou seja, representando por L, D e U a parte triangular inferior, diagonal e triangular superior de A, respectivamente, tem-se, nos casos anteriores:

Método de Jacobi: $M_J = D$ (diagonal), e $N_J = -(L + U)$.

Método de Gauss-Seidel: $M_{GS} = D + L$ (triang. inferior), e $N_{GS} = -U$.

De forma semelhante à análise de convergência do método das substituições sucessivas efectuado na Secção 2.3, a convergência da família de métodos (3.33) depende dos valores próprios da matriz $M^{-1}N$, sendo portanto influenciada tanto pela matriz A do problema, como pela forma como é definida a sua partição. Usando o conceito de *raio espectral* definido na Secção 1.3, é possível mostrar que a família de métodos (3.33) converge para a solução do problema $x^* = A^{-1}b$, a partir de qualquer estimativa inicial $x^{(0)}$, quando:

$$\rho(M^{-1}N) < 1 \tag{3.34}$$

Com efeito, dada uma partição $A = M - N$, definindo o erro na iteração k

$$e^{(k)} = x^{(k)} - x^*$$

como $Mx^* = Nx^* + b$, isto significa que

$$M\left(x^{(k+1)} - x^*\right) = N\left(x^{(k)} - x^*\right)$$

e portanto:

$$e^{(k+1)} = M^{-1}Ne^{(k)} = \left(M^{-1}N\right)^k e^{(0)}$$

Tomando normas, considerando (1.30) tem-se:

$$\|e^{(k+1)}\| \leq \| \left(M^{-1}N\right)^k \| \|e^{(0)}\| \leq \| \left(M^{-1}N\right) \|^k \|e^{(0)}\|$$

Como consequência da definição do raio espectral (Secção1.3.2)

$$\lim_{k\to\infty} \left(M^{-1}N\right)^k = 0 \in \mathbb{R}^{n\times n}$$

quando $\rho\left(M^{-1}N\right) < 1$, o que permite estabelecer esta última desigualdade como condição suficiente para a convergência de (3.33). É possível também mostrar que a condição (3.34) é ainda *necessária* para a convergência da família

de métodos iterativos definidos por (3.33) (Kincaid e Cheney, 1991). Este resultado, mais completo do que as condições de convergência do Teorema 2.1 referente ao método das substituições sucessivas (aplicável ao caso *escalar*), pode ser interpretado como uma consequência da linearidade da equação (3.33).

Tanto o método de Jacobi como o método de Gauss-Seidel são convergentes com matrizes *A diagonal dominantes* (Secção 3.2). É também conhecido o facto de que o método de Gauss-Seidel é globalmente convergente quando a matriz A é *simétrica e positiva definida* (Golub e Van Loan, 1989). Deve ser notado que a convergência (e a sua rapidez) destes métodos iterativos é bastante sensível à forma como as equações e variáveis se encontram ordenadas num problema particular. De forma semelhante, a aplicação prévia dos processos de equilibragem da matriz A, descritos na Secção 3.2.1, antes da resolução do sistema linear, assume particular importância no contexto da aplicação destes métodos iterativos.

Método SOR

Nos métodos anteriores, quando o raio espectral de $M^{-1}N$ é próximo da unidade, a convergência pode ser muito lenta, uma vez que a quantidade $\rho(M^{-1}N)^k$ tende lentamente para zero. Nestes casos é possível definir a seguinte variante, conhecida como o método SOR ("Successive Over-Relaxation"):

Algoritmo 3.8 (Método SOR)

Dado $x^{(0)}$

Enquanto convergência ou limite de iterações não verificados

  Para $i = 1, 2, \ldots, n$ **fazer**

$$x_i^{(k+1)} = \omega\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\Big)/a_{ii} + (1-\omega)x_i^{(k)}$$

  Fim_Para

Fim_Enquanto

A equação (3.33) é ainda válida para este caso, com $M_{\text{SOR}} = D + \omega L$ e $N_{\text{SOR}} = (1-\omega)D - \omega U$, onde $\omega \in \mathbb{R}$ é o parâmetro de relaxação. Quando $\omega = 1$ este algoritmo é portanto equivalente ao método de Gauss-Seidel. O algoritmo SOR é por vezes designado de *sobre-relaxação* quando são usados valores $\omega > 1$, e *sub-relaxação* se $\omega < 1$. Uma escolha criteriosa deste parâmetro permite por vezes uma redução significativa de $\rho(M^{-1}N)$, resultando numa convergência bastante mais rápida para a solução, por vezes superior a uma

ordem de magnitude. Em alguns problemas é possível determinar teoricamente o valor deste parâmetro que minimiza $\rho(M_{\text{SOR}}^{-1}N_{\text{SOR}})$ (Greenbaum, 1997).

Todos os métodos considerados anteriormente utilizam matrizes M e N com coeficientes constantes entre as várias iterações. Outros métodos onde estas matrizes vão sendo adaptadas ao progresso conseguido em cada iteração[21], podem ser encontrados em Barrett et al. (1994); Greenbaum (1997); Saad (2003).

Uma característica bastante importante dos métodos anteriores é o facto dos coeficientes da matriz A apenas serem utilizados em operações de *produto matriz-vector* (do tipo $y = Ax$). Em sistemas esparsos, esta operação pode ser executada através de um procedimento especial, construído pelo utilizador para o efeito, permitindo tirar partido da esparsidade (ou estrutura especial) de A. Deste modo, esta classe de métodos torna-se particularmente adequada para a resolução de sistemas esparsos. Esta estrutura especial pode também ser aproveitada em computadores com capacidades de processamento paralelo, permitindo uma resolução bastante mais rápida destes problemas.

[21] Denominados por isso como métodos *não-estacionários*.

Bibliografia

AEA Technology, *Harwell Subroutine Library*, Harwell Laboratory, Oxfordshire, England, (1978).

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorenson, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, (1992).

R. Barrett, M. Berry, T.F. Chan, J. Demmel, J.M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, (1994).

C. Bischof, A. Carle, G. Corliss, A. Griewank, P. Hovland, "ADIFOR — Generating Derivative Codes from Fortran Programs", *Scientific Programming*, 1, 1–29, (1992).

C. Bischof, *A Collection of Automatic Differentiation Tools*, <`http://www-unix.mcs.anl.gov/autodiff/AD_Tools/`>, (1999).

R.P. Brent, *Algorithms for Minimization Without Derivatives*, Prentice-Hall Inc., Englewood Cliffs, NJ, (1973).

C.G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations", *Math. Comp.*, 19, 577–593, (1965).

S.C. Chapra, R.P. Canale, *Numerical Methods for Scientists*, McGraw-Hill Book Company, New York, NY, (1989).

W.J. Cody, "Algorithm 666. MACHAR: A Subroutine to Dynamically Determine Machine Parameters", *ACM Trans. Math. Software*, 14(4), 303–311, (1988).

S.D. Conte, C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill Book Company, New York, NY, (1981).

A.R. Curtis, J. Reid, "On the Automatic Scaling of Matrices for Gaussian Elimination", *J. Inst. Maths. Applics.*, 10, 118–124, (1972).

G. Dahlquist, Å. Björck, *Numerical Methods*, Prentice-Hall Inc., Englewood Cliffs, NJ, (1974).

J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, (1997).

J.E. Dennis, R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall Inc., Englewood Cliffs, NJ, (1983).

N. Dershowitz, *Software Horror Stories*, <http://www-courses.cs.uiuc.edu/~cs376/horror.html>, (1998).

K. Dowd, C.R. Severance, *High Performance Computing*, O'Reilly & Associates, Sebastopol, CA, (1998).

I.S. Duff, R.G. Grimes, J.G. Lewis, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*, Relatório Técnico RAL-92-086, Rutherford Appleton Laboratory, Oxon, UK, (1992).

J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, *Linpack Users' Guide*, SIAM, Philadelphia, PA, (1979).

ESA (Agência Espacial Europeia), *ESA/CNES Joint Press Release ARIANE 501*, <http://www.esrin.esa.it/htdocs/tidc/Press/Press96/press33.html>.

P.E. Gill, W. Murray, M.H. Wright, *Practical Optimization*, Academic Press, Londres, Reino Unido, (1981).

P.E. Gill, W. Murray, M.A. Saunders, M.H. Wright, "Computing Forward-difference Intervals for Numerical Optimization", *SIAM J. Sci. Stat. Comput.*, 4(2), 310–312, (1983).

P.E. Gill, W. Murray, M.H. Wright, *Numerical Linear Algebra and Optimization*, Volume 1, Addison-Wesley Publishing Company, Redwood City, CA, (1991).

D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", *ACM Computing Surveys*, 23(1), 5–48, (1991).

G.H. Golub, C.F. Van Loan, *Matrix Computations*, 2ª edição, John Hopkins University Press, Baltimore, MD, (1989).

A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, (1997).

A.M. Hayashi, "Rough Sailing for Smart Ships — Does commercial software such as Windows NT compromise naval ship performance?", *Scientific American*, Novembro, (1998).

M. Heldoorn, *The* SIunits *LaTeX package — support for the International System of Units*, <http://www.ctan.org/tex-archive/macros/latex/contrib/supported/SIunits/>, (2000).

M. Herstine, "Algorithms For High-Precision Finite Differences", *Dr. Dobb's Journal*, 23(5), 52-58, (1998).

N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, (1996).

T. Hucle, *Collection of Software Bugs*, <http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html>, (1999).

IEEE, *IEEE Standard 754-1985 for Binary Floating-point Arithmetic*, SIGPLAN 22(2), 9–25, (1985).

M.A. Jenkins, J.F. Traub, "A three-stage-algorithm for real polynomials using quadratic iteration", *SIAM J. Num. Anal.*, 7, 545–566, (1970).

W. Kahan, *Lecture Notes on the Status of the IEEE Standard 754 for Binary Floating-Point Arithmetic*, <http://www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps>, (1996).

L.V. Kantarovich, "Functional Analysis and Applied Mathematics", *Uspehi Mat. Nauk.*, 3, 89–185, (1948).

C.T. Kelley, *Solving Nonlinear Equations with Newton's Method*, SIAM, Philadelphia, PA, (2003).

D.E. Knuth, *The Art of Computer Programming — Volume 2: Seminumerical Algorithms*, 3ª edição, Addison-Wesley Publishing Company, Reading, MA, (1997).

J. Lemos Ferreira, *Erros e algarismos significativos*, Departamento de Engenharia Química, Universidade de Coimbra, (1999).

D. Kincaid, E.W. Cheney, *Numerical Analysis*, Brooks/Cole Publishing Company, Belmont, CA, (1991).

R.S.H. Mah, *Chemical Process Structures and Information Flows*, Butterworths Publishers, Boston, MA, (1990).

J.M. McNamee, *A bibliography on roots of polynomials*, <http://www1.elsevier.com/homepage/sac/cam/mcnamee/>, (2002).

C.D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, Philadelphia, PA, (2000).

J. Nocedal, S.J. Wright, *Numerical Optimization*, Springer-Verlag, New York, NY, (1999).

M.L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, PA, (2001).

V.Y. Pan, "Solving a Polynomial Equation: Some History and Recent Progress", *SIAM Review*, 39(2), 187–220, (1997).

H. Pina, *Métodos Numéricos*, McGraw-Hill de Portugal, Alfragide, (1995).

W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, 2ª edição, Cambridge University Press, Cambridge, Reino Unido, (1992).

J.R. Rice, *Numerical Methods, Software, and Analysis*, Academic Press, Inc.,

Boston, (1993).

M.S. Rosa, *Tópicos de Análise Numérica — Métodos Numéricos*, Departamento de Matemática, Universidade de Coimbra, (1988).

Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2ª edição, SIAM, Philadelphia, PA, (2003).

P. Sebah, X. Gourdon, *Newton's Method and High Order Iterations*, <http://numbers.computation.free.fr/Constants/Algorithms/newton.html>, (2001).

G.W. Stewart, *Introduction to Matrix Computations*, Academic Press, Inc., Orlando, FL, (1973).

Unicode Consortium, *Unicode Home Page*, <http://www.unicode.org/>, (2003).

Wolfram Research, *Solving the Quintic*, <http://library.wolfram.com/examples/quintic/>, (1996).

T.J. Ypma, "Historical development of the Newton-Raphson method", *SIAM Review*, 37(4), 531–551, (1995).

APÊNDICE A

Ferramentas e recursos adicionais

A aplicação prática dos conceitos descritos nos Capítulos anteriores requer o domínio de um conjunto básico de ferramentas computacionais. Para além das capacidades de escrita e de execução de programas, estão disponíveis actualmente várias componentes que permitem simplificar consideravelmente as tarefas envolvidas na resolução de um problema numérico (por exemplo, ambientes de programação e utilitários), contribuindo para a obtenção da solução desejada mais rapidamente e com menor esforço.

Esta Secção procura efectuar uma breve descrição das principais ferramentas disponíveis, incluindo algumas referências onde pode ser obtida informação mais detalhada sobre alguns destes aspectos.

A.1 O sistema operativo UNIX

O sistema operativo UNIX foi criado no final da década de 60, nos Laboratórios Bell da empresa americana AT&T. Concebido inicialmente como um pequeno projecto, a sua popularidade foi crescendo, tendo o seu desenvolvimento (especialmente a partir da década de 80) extravasado quase por completo o local de desenvolvimento inicial. A sua popularidade pode ser atribuída ao uso de uma arquitectura modular (promovendo a especialização e a integração dos seus componentes), conjuntamente com a implementação eficiente das características fundamentais num sistema operativo moderno (por exemplo, capacidades multitarefa e multi-utilizador, memória virtual, bibliotecas partilhadas, e protocolos de comunicação).

A crescente popularidade deste sistema operativo, conjuntamente com a problemática do seu licenciamento levaram ao aparecimento, no início da década de 90, de várias reimplementações do UNIX ditas *livres*[1]. Uma das mais bem

[1] Podendo ser utilizadas, expandidas e distribuídas gratuitamente.

sucedidas actualmente é o sistema *Linux* (<http://www.linux.com>). Esta implementação começou por suportar apenas os computadores pessoais baseados na arquitectura Intel x86, estando disponível actualmente para a maioria das arquitecturas correntes de computadores pessoais e de *workstations*. O uso deste sistema operativo permite a utilização de um leque variado de ferramentas que facilitam significativamente o desenvolvimento de programas.

Existe imensa informação disponível relativa à utilização deste sistema. Por exemplo, Abrahams e Larson (1995) fornecem uma introdução relativa à utilização básica deste sistema. Em Robbins e Gilly (1999) é possível encontrar uma descrição sucinta (de referência), embora bastante completa, sobre a utilização dos seus principais comandos. Loukides (1990) centra a sua atenção sobre as ferramentas essenciais para o desenvolvimento de programas, especialmente quando escritos na linguagem Fortran 77. Informação adicional pode ser encontrada na Web, nos sítios *Linux Documentation Project* (<http://www.linuxdoc.org>) e *Scientific Applications on Linux*, disponível em <http://sal.kachinatech.com>.

A.1.1 Editores de texto

Existem muitos editores que podem ser utilizados com o sistema UNIX. Os de uso mais comum são o *Emacs* (Stallman, 1999) e o *vi* (Bostic, 1996). Qualquer deste editores é muito mais poderoso do que pode parecer à primeira vista, embora ambos requeiram uma aprendizagem algo prolongada para tirar partido de todas as suas potencialidades.

A.1.2 Linguagens de programação

Fortran

O Fortran (uma abreviatura de "FORmula TRANslation") é a linguagem de programação mais usada em aplicações relacionadas com o cálculo científico. Esta linguagem foi concebida especificamente para tarefas numéricas, podendo ser facilmente optimizada de modo a produzir programas que executam de forma relativamente rápida (i.e., com grande eficiência numérica). Outra vantagem consiste na possibilidade do uso de um conjunto vasto de bibliotecas disponíveis, contendo subrotinas escritas nesta linguagem[2]. Os dialectos mais comuns desta linguagem são o Fortran 77 e o Fortran 90/95 (definidos por standards ANSI e ISO). As principais diferenças entre estas versões encontram-se descritas, por exemplo, num documento com respostas a *questões frequentes* (FAQs)

[2]Algumas das mais importantes são descritas na Secção A.3.1.

sobre Fortran 90, disponível em <ftp://ftp.ifremer.fr/ifremer/ditigo/fortran90/engfaq>. Este documento lista ainda muitas fontes de informação adicionais relativas à linguagem Fortran (incluindo documentos introdutórios ou *tutorials*).

Alguns livros de texto com uma descrição pormenorizada da linguagem Fortran 77 são por exemplo Meissner e Organick (1980) e Page (1988)[3]. Uma descrição detalhada da linguagem Fortran 90 é fornecida por Meissner (1995). Outros locais com informação adicional sobre a linguagem Fortran são:

- *The Fortran Library*, disponível em <http://www.fortranlib.com>.
- *The Fortran Market*, disponível em <http://www.fortran.com>. Entre outra informação, este sítio inclui o texto integral do standard da linguagem Fortran 77.

C e C++

Paralelamente ao sucesso do sistema operativo UNIX, a sua linguagem de programação base (C) é também bastante usada actualmente, na maioria das tarefas genéricas de programação. Sendo uma linguagem de uso geral, é natural que possa ser considerada em alguns casos menos apropriada (ou de utilização mais complexa) que o Fortran na resolução de problemas numéricos, especialmente em aplicações de elevado desempenho (Dowd e Severance, 1998). Mais recentemente, a linguagem C++ incorpora muitas melhorias relativamente à linguagem C, úteis por exemplo na construção de aplicações de grande porte. O C++ permite também a programação estruturada com base em *objectos*, embora esta vertente não esteja ainda tão madura como as outras componentes da linguagem[4]. O uso destas linguagens encontra-se descrito em detalhe em Harbinson e Steele Jr. (1995) e Stroustrup (1997).

Compiladores e *debuggers*

Os compiladores permitem a criação de programas executáveis a partir das instruções correspondentes, escritas numa linguagem de alto nível como Fortran ou C. Para verificar a correcção dos programas criados, são por vezes utilizados *debuggers*, que permitem examinar o estado do programa, o valor das

[3]Esta última referência encontra-se também disponível em formato electrónico em <ftp://ftp.star.le.ac.uk/pub/fortran/prof77.ps.gz>.

[4]Deve notar-se que o standard ISO da linguagem C++ apenas foi aprovado em finais de 1997. Existe actualmente uma forte preocupação na uniformização dos *objectos* e *classes* disponíveis nas implementações desta linguagem, por exemplo através do uso da *Standard Template Library*, disponível em <http://www.sgi.com/Technology/STL/>.

suas variáveis, a sequência de rotinas invocadas, etc., enquanto o programa é executado.

Muita informação, específica de cada compilador, é fornecida habitualmente com estas ferramentas. Algumas opções de compilação facilitam especialmente o desenvolvimento de programas[5], devendo por isso ser usadas sobretudo durante as fases iniciais de desenvolvimento.

No sistema operativo Linux são muito frequentemente usados os compiladores `gcc` (C), `g++` (C++), `g77` (Fortran 77) e `gfortran` (Fortran 90/95), funcionando de forma totalmente integrada nas suas últimas versões. O código fonte destes programas está disponível em <`ftp://ftp.gnu.org`>, ou no arquivo local em <`http://ftp.eq.uc.pt/software/unix/gnu/`>. Os programas executáveis correspondentes à arquitectura Win32 estão também disponíveis em <`http://www.xraylith.wisc.edu/~khan/`>, ou no arquivo local <`http://ftp.eq.uc.pt/software/pc/prog/`>. Embora inclua muitas extensões, o compilador `g77` apenas implementa de forma completa o dialecto Fortran 77. Para usar a linguagem Fortran 90 estão disponíveis no ambiente Linux vários compiladores comerciais (descritos nas referências gerais anteriores sobre esta linguagem). Uma alternativa que permite a compilação e a conversão Fortran 90 → Fortran 77 é a ferramenta `vast90`, disponível gratuitamente[6] em <`http://www.psrv.com`>.

O *debugger* mais comum no ambiente UNIX é o `dbx`. A sua utilização encontra-se descrita em Loukides (1990). No Linux, com os compiladores `gcc` e `g77` pode ser utilizado o *debugger* `gdb`, preferencialmente com o seu interface visual `DDD`, ambos disponíveis em <`ftp://ftp.gnu.org`>.

Utilitários de diagnóstico

Para além dos *debuggers*, existem algumas ferramentas particularmente úteis para verificar a correcção de programas escritos na linguagem Fortran (criados pelo utilizador, ou obtidos a partir de outras fontes). Estes utilitários permitem diagnosticar também a portabilidade do software em causa, avaliando a sua conformidade com os standards oficiais. Entre as ferramentas disponíveis, destacam-se:

- `ftnchek`: Este utilitário efectua uma análise estática de programas escritos na linguagem Fortran 77, permitindo o diagnóstico de erros *semânticos*, tais como o uso de variáveis não inicializadas ou a discrepância entre

[5] Duas opções tipicamente muito úteis nos compiladores de Fortran 77 em ambientes UNIX são: `-C` para incluir código no programa que verifica se os limites dos *arrays* estão a ser respeitados, e `-u` para evitar o uso de variáveis não explicitamente declaradas.

[6] Versão *pessoal* apenas disponível para o sistema operativo Linux e a arquitectura Intel x86.

o número e o tipo dos parâmetros de uma rotina. O `ftnchek` está disponível em <`http://www.dsm.fordham.edu/~ftnchek`>, e também no arquivo NETLIB[7] (no directório `ftnchek`).

- **spag**: Este sistema permite não só o *diagnóstico* (estático e dinâmico) de programas escritos na linguagem Fortran 77, como também a sua *re-estruturação*. Uma versão gratuita[8] para o sistema operativo Linux encontra-se disponível em <`http://www.polyhedron.co.uk`>.

Estilos e práticas de programação

A experiência mostra que os procedimentos usados no desenvolvimento de software têm um efeito considerável na qualidade dos programas produzidos, expressa em termos de *bugs* (ou erros de lógica), na sua facilidade de entendimento e manutenção, e na sua portabilidade.

Algumas práticas correntes que reflectem a experiência acumulada nesta área podem ser encontradas na colectânea *User Notes on Fortran Programming* (e nas referências aí incluídas), disponível no sítio <`http://www3.huji.ac.il/~agay/unfp/unfp.html`>. Coleman e Van Loan (1988) fornecem também algumas orientações deste tipo, embora tendo sobretudo em vista a eficiência numérica da implementação produzida. Dowd e Severance (1998) analisam em detalhe a escrita de fragmentos de código correspondentes a tarefas numéricas elementares muito comuns, tendo em vista optimizar a eficiência do código produzido.

Outros utilitários

Uma tarefa bastante comum na maioria das bibliotecas científicas é a determinação dos limites da representação em vírgula flutuante, sobretudo para os computadores que ainda não implementam (pelo menos de forma completa) a norma de representação IEEE 754. Rotinas que permitem efectuar um teste fiável destes limites (evitando a geração de condições de excepção numéricas) foram desenvolvidas por Cody (1988), estando disponíveis no arquivo NETLIB (no directório `toms`[9]).

Nos casos em que é usada a norma IEEE 754, o programa da Figura A.1 permite examinar em detalhe cada uma das componentes da representação de quantidades arbitrárias neste formato.

[7] Descrito na Secção A.3.

[8] Apenas disponível para a arquitectura Intel x86, com restrições de uso educacional, académico, e de demonstração.

[9] E também na biblioteca LAPACK (descrita mais à frente), através das rotinas SLAMCH e DLAMCH.

```
      PROGRAM REPIEE

c     --- Programa para verificar a estrutura da representação de
c         vírgula flutuante em memória, assumindo a representação de
c         inteiros em complemento binário (baseado em UNFP, 1998). ---

      IMPLICIT NONE
      INTEGER ians, ires, IENDI
      REAL xsp
      DOUBLE PRECISION xdp
      CHARACTER b*64

c     --- Determinar ENDIANESS ---
      ires = IENDI()
      IF (ires.EQ.0) THEN
         WRITE (*, *) 'Máquina LITTLE ENDIAN.'
      ELSE IF (ires.EQ.1) THEN
         WRITE (*, *) 'Máquina BIG ENDIAN.'
      ELSE
         WRITE (*, *) 'MAIN: ENDIANESS não determinável!'
         STOP
      END IF

c     --- Obter representação binária ---
      PRINT *, 'Introduza 1 para precisão SIMPLES ou 2 para DUPLA:'
      READ *, ians
      IF (ians.EQ.1) THEN
         WRITE (*, *) 'Introduza 1 número REAL:'
         READ (*, *) xsp
         CALL BINREP(xsp, b)
         WRITE (*, *) 'Sinal: ', b(1:1)
         WRITE (*, *) 'Expoente: ', b(2:9), ' (8 bits)'
         WRITE (*, *) 'Mantissa: (1).', b(10:32), ' (23 bits)'
      ELSE IF (ians.EQ.2) THEN
         WRITE (*, *) 'Introduza 1 número em PRECISÃO DUPLA:'
         READ (*, *) xdp
         CALL TWOB(xdp, b, ires)
         WRITE (*, *) 'Sinal: ', b(1:1)
         WRITE (*, *) 'Expoente: ', b(2:12), ' (11 bits)'
         WRITE (*, *) 'Mantissa: (1).', b(13:64), ' (52 bits)'
      ELSE
         WRITE (*, *) 'REPIEE: Precisão desconhecida.'
         STOP
      END IF
      END


      INTEGER FUNCTION IENDI()

c     --- IEND = 0 se LITTLE ENDIAN; = 1 se BIG ENDIAN: = 2 se indeterminada ---
      IMPLICIT NONE
      INTEGER*4 I, A0, A1, A2, A3
      PARAMETER (A0=48, A1=49, A2=50, A3=51)
      INTEGER ires
      COMMON /ENDIAN/ I

      I = A0 + A1*256 + A2*(256**2) + A3*(256**3)
      CALL SUBEND(ires)
      IENDI = ires
      END


      SUBROUTINE SUBEND(Ires)

      IMPLICIT NONE
```

Figura A.1: (Continua na página seguinte)

```
      INTEGER Ires
      CHARACTER I*4
      COMMON /ENDIAN/ I

      IF (I.EQ.'0123') THEN
         Ires = 0
      ELSE IF (I.EQ.'3210') THEN
         Ires = 1
      ELSE
         Ires = 2
      END IF
      END


      SUBROUTINE TWOB(Ivec, B, Iend)

      IMPLICIT NONE
      INTEGER Iend, Ivec(2)
      CHARACTER B*64, d1*32, d2*32

      IF (Iend.EQ.0) THEN
         CALL BINREP(Ivec(2), d1)
         CALL BINREP(Ivec(1), d2)
      ELSE IF (Iend.EQ.1) THEN
         CALL BINREP(Ivec(1), d1)
         CALL BINREP(Ivec(2), d2)
      ELSE
         WRITE (*, *) 'TWOB: endianess desconhecida.'
         STOP
      END IF
      B = d1//d2
      END


      SUBROUTINE BINREP(Ival, B)

      IMPLICIT NONE
      INTEGER i, Ival
      CHARACTER B*32

      IF (Ival.GE.0) THEN
         B(1:1) = '0'
         DO 50 i = 32, 2, -1
            IF (MOD(Ival, 2).EQ.0) THEN
               B(i:i) = '0'
            ELSE
               B(i:i) = '1'
            END IF
            Ival = Ival/2
50       CONTINUE
      ELSE
         B(1:1) = '1'
         Ival = ABS(Ival+1)
         DO 100 i = 32, 2, -1
            IF (MOD(Ival, 2).EQ.0) THEN
               B(i:i) = '1'
            ELSE
               B(i:i) = '0'
            END IF
            Ival = Ival/2
100      CONTINUE
      END IF
      END
```

Figura A.1 Programa em Fortran 77 para a visualização da representação em vírgula flutuante, de acordo com a norma IEEE 754.

Quando se inicia o trabalho numa arquitectura nova ou com um compilador recente, é por vezes conveniente efectuar um breve teste de validação da aritmética de vírgula flutuante usada. Embora em geral não seja possível garantir que uma determinada implementação da aritmética em vírgula flutuante produza sempre resultados apropriados, é possível, no entanto, efectuar um conjunto de testes básicos que permitem adquirir alguma confiança na sua fiabilidade. Para este fim, encontram-se disponíveis os seguintes sistemas:

- FPTEST, desenvolvido por Schryer (1981), e disponível em <`http://www.bell-labs.com/project/fptest/`>.
- PARANOIA, desenvolvido por vários autores, e disponível no arquivo NETLIB (no directório `paranoia`).

A.2 Recursos disponíveis na Internet

Para além das referências anteriores, encontram-se ainda disponíveis na Internet muitas fontes de informação, que facilitam a obtenção de soluções de problemas numéricos. Entre estas, deve ser destacado um documento com respostas a questões frequentes (FAQs) na área de Análise Numérica, disponível em <`http://www.mathcom.com/nafaq/index.html`>.

A.3 Arquivos de software matemático

O arquivo NETLIB contém software apropriado para a resolução de um conjunto vasto de problemas de natureza matemática, normalmente associados com o cálculo científico. Todo o software deste arquivo está disponível de forma gratuita, podendo ser acedido através de <`http://www.netlib.org`>. Esta colecção de software está também replicada por vários países, de forma sincronizada. Uma cópia local deste arquivo (tal como de muitas das outras ferramentas mencionadas anteriormente) encontra-se disponível em <`http://ftp.eq.uc.pt/software/math/`>.

O index GAMS ("Guide to Available Mathematic Software"), acessível através de <`http://gams.nist.gov`>, permite localizar com facilidade software apropriado no arquivo NETLIB (bem como em outras bibliotecas), para resolução de problemas específicos de natureza numérica.

A.3.1 Algumas bibliotecas matemáticas

Álgebra linear

Algumas tarefas relacionadas com aspectos de Álgebra Linear, tais como operações com vectores e matrizes, ou a resolução de sistemas de equações lineares, constituem subproblemas bastante comuns em aplicações científicas. Como tal, torna-se importante a sua implementação de forma eficiente e estandardizada, evitando a duplicação de esforços. A biblioteca BLAS (*Basic Linear Algebra Subprograms*), disponível no arquivo NETLIB, contém uma implementação eficiente e portátil dos principais tipos de operações básicas que envolvem vectores e matrizes, estando estruturada do seguinte modo:

- Nível 1: operações *vector-vector* (Lawson et al., 1979), incluindo por exemplo, copiar e permutar vectores, multiplicar um vector por um escalar, adicionar um múltiplo de um vector a outro, calcular produtos escalares ou a norma de vectores.
- Nível 2: operações *vector-matriz* (Dongarra et al., 1988), considerando a multiplicação de matrizes por vectores (eventualmente multiplicados por um escalar ou transpostos), actualizações matriciais de característica 1 ou 2, e a resolução de sistemas triangulares.
- Nível 3: operações *matriz-matriz* (Dongarra et al., 1990). Neste nível estão incluídos vários produtos e adições de matrizes, eventualmente multiplicados por escalares ou transpostos, actualizações matriciais de característica k ou $2k$, a multiplicação de uma matriz por outra triangular, bem como a resolução de sistemas triangulares de equações com vários termos independentes.

Resolução de sistemas de equações lineares

A resolução de sistemas de equações lineares e a realização de algumas tarefas relacionadas, tais como a obtenção de decomposições matriciais, o cálculo de determinantes e de matrizes inversas são considerados na biblioteca LAPACK (Anderson et al., 1992), também disponível no arquivo NETLIB. Esta biblioteca veio substituir as anteriores LINPACK e EISPACK, também disponíveis neste arquivo, mas de concepção mais antiga. A biblioteca LAPACK faz uso extensivo das operações vectoriais e matriciais de vários níveis disponíveis na biblioteca BLAS.

O uso generalizado destas bibliotecas deu recentemente origem ao projecto ATLAS, cujo objectivo é a optimização automática das bibliotecas LAPACK e

BLAS. Este software também se encontra disponível no arquivo NETLIB (no directório `atlas`).

As bibliotecas descritas anteriormente destinam-se sobretudo à resolução de sistemas de equações *densos*. O arquivo NETLIB inclui ainda várias bibliotecas para a resolução de sistemas *esparsos*[10]. Dados para testes com este tipo de sistemas podem ainda ser encontrados no sítio denominado *Matrix Market* <`http://gams.nist.gov/MatrixMarket/`>.

Resolução de equações não-lineares

Como foi mencionado atrás, a resolução de equações ou de sistemas de equações não-lineares beneficia geralmente com a sua transformação (mesmo implícita) num problema equivalente de Optimização. Como tal, não é de estranhar que alguns dos métodos mais eficientes para a resolução destes problemas sejam descritos conjuntamente com algoritmos para a optimização de problemas não-lineares.

O directório `opt` do arquivo NETLIB contém diversas subrotinas que permitem a resolução de equações não-lineares. Entre estas, deve ser destacada a subrotina `NNES` ("Nonmonotonic Nonlinear Equation Solver") que implementa, de forma globalizante, os métodos de Newton e quasi-Newton descritos na Secção 2.12, conjuntamente com outros métodos de resolução. A biblioteca MINPACK, também no arquivo NETLIB, possui diversas rotinas para a resolução de equações não-lineares. Bibliotecas adicionais podem ser encontradas na Internet: KINSOL (Serban et al., 1998), SNES (Balay et al., 2001), e NITSOL (Walker, 2003).

Bibliotecas de índole geral

Adicionalmente às referências anteriores, é ainda possível o recurso a algumas bibliotecas genéricas, abrangentes de um leque muito vasto de problemas numéricos. Estão nesta categoria as bibliotecas HSL, IMSL, e NAG. Apesar da conveniência do seu largo espectro de aplicabilidade, a principal desvantagem é que o seu uso (tal como a incorporação de eventuais melhorias propostas pelos utilizadores) se encontra dependente de um licenciamento comercial prévio. Para além das questões de preço, as aplicações possíveis dos programas desenvolvidos com base nestas rotinas encontram-se por vezes fortemente limitadas pelas próprias condições de licenciamento. Informação adicional sobre estas bibliotecas pode ser encontrada em:

- HSL (Harwell): <`http://www.cse.clrc.ac.uk/Activity/HSL`>

[10] Consultar por exemplo os directórios `sparse`, `sparse-blas`, `sparspak` e `templates`.

- IMSL: <http://www.vni.com>
- NAG: <http://www.nag.com>

Uma alternativa livre é a biblioteca GSL ("GNU Scientific Library"), disponível em <http://www.gnu.org/software/gsl/>.

A.4 Outras linguagens de índole matemática

Para além das linguagens de programação descritas anteriormente, encontram-se também disponíveis outros sistemas que podem funcionar de forma interactiva permitindo, conjuntamente com o uso de uma linguagem própria de nível bastante elevado, o desenvolvimento rápido e a experimentação com software matemático (sobretudo ao nível de protótipos). Nesta Secção são descritos alguns dos exemplos mais comuns de sistemas deste tipo.

A.4.1 Linguagens matriciais

O sistema mais divulgado nesta categoria é o *Matlab*, descrito em <http://www.mathworks.com>. Tal como o nome sugere, este sistema é particularmente adequado para o cálculo com matrizes e vectores. A funcionalidade básica do sistema é frequentemente expandida através de *packages*, que funcionam como bibliotecas especiais. A interactividade deste sistema é conseguida à custa do uso de uma linguagem *interpretada*, o que torna a execução dos programas escritos nesta linguagem algo mais lenta do que os produzidos por linguagens *compiladas* como o Fortran ou o C.

Com base no sucesso desta linguagem, e tendo em consideração a sua natureza proprietária, foram desenvolvidos outros sistemas semelhantes, com características compatíveis. Um dos sistemas deste tipo mais utilizados é o *Octave*[11]. A linguagem base de programação destes sistemas (conjuntamente com alguns exemplos de código) encontra-se disponível em várias referências, por exemplo em Coleman e Van Loan (1988); Higham e Higham (2000). Informação adicional sobre esta linguagem pode ser encontrada no manual do sistema *Octave*, bem como em muitos outros documentos introdutórios adicionais (na forma de *tutorials*) disponíveis na Web[12]. Outra linguagem de natureza semelhante é o *Scilab* (Scilab Group, 1989).

[11] Disponível gratuitamente em <http://www.che.wisc.edu/octave/> ou no arquivo GNU em <http://www.gnu.org>.

[12] Alguns destes são Sigmon (1992) e Gockenbach (1999).

A.4.2 Linguagens de manipulação simbólica

A característica distintiva desta classe de sistemas é a possibilidade de efectuarem *manipulações simbólicas* em expressões algébricas arbitrárias. Conjugada com a capacidade de cálculo numérico normal, e com capacidades gráficas extensivas, estas potencialidades tornam os sistemas desta categoria bastante poderosos na resolução de problemas gerais de âmbito matemático. No entanto, a sua generalidade é contrabalançada pela menor rapidez de processamento (em geral são também usadas linguagens interpretadas), conjuntamente com os requisitos adicionais de memória que as representações matemáticas usadas (mais gerais) requerem. O seu uso, em tarefas de carácter predominantemente numérico, deve ser portanto considerado através da ponderação de todos estes factores.

Os sistemas mais populares nesta categoria são o *Mathematica* <`http://www.wolfram.com`>, o *Maple* <`http://www.maplesoft.com`>, e o *Maxima*, disponível em <`http://maxima.sourceforge.net/`>. Todos eles são sistemas comerciais, com linguagens de programação próprias (e bastante diferentes entre si). Por exemplo, a linguagem de sistema *Mathematica* encontra-se descrita em Wolfram (1996); uma introdução mais sucinta está disponível em Blachman (1992). Uma comparação das capacidades e dos desempenhos relativos destes sistemas pode ser encontrada em <`http://math.unm.edu/~wester/cas_review.html`>.

Referências

P.W. Abrahams, B.R. Larson, *Unix for the Impatient*, Addison-Wesley Publishing Company, Reading, MA, (1995).

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorenson, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, (1992).

S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, *PETSc, Portable Extensible Toolkit for Scientific Computation*, <`http://www-unix.mcs.anl.gov/petsc/petsc-as/`>, (2001).

N. Blachman, "Introduction to Mathematica", *The* Mathematica *Conference*, Boston, MA, (1992). (Disponível em <`http://www.mathsource.com/MathSource22/General/Tutorials/General/0203-847/IntroToM.ps`>).

K. Bostic, *Vi/Ex Reference Manual*, University of California, Berkeley, CA, (1996). (Incluído com o programa `nvi`, disponível em <`ftp://mongoose.bostic.com/pub/`>).

W.J. Cody, "Algorithm 666. MACHAR: A Subroutine to Dynamically Determine Machine Parameters", *ACM Trans. Math. Software*, 14(4), 303–311, (1988).

T.F. Coleman, C. Van Loan, *Handbook for Matrix Computations*, SIAM, Philadelphia, PA, (1988).

J.J. Dongarra, J. du Croz, S. Hammarling, R.J. Hanson, "An Extended Set of FORTRAN Basic Linear Algebra Subprograms", *ACM Trans. Math. Software*, 14(1), 1–17, (1988).

J.J. Dongarra, J. du Croz, I. Duff, S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms", *ACM Trans. Math. Software*, 16(1), 1–17, (1990).

M.S. Gockenbach, *A Practical Introduction to Matlab*, <http://www.math.mtu.edu/~msgocken/intro/intro.ps>, (1999).

S.P. Harbinson, G.L. Steel Jr., *C, A Reference Manual*, 4ª edição, Prentice Hall Inc., Englewood Cliffs, NJ, (1995).

D.J. Higham, N.J. Higham, *MATLAB Guide*, SIAM, Philadelphia, PA, (2000).

C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage", *ACM Trans. Math. Software*, 5(3), 308–323, (1979).

M. Loukides, *UNIX for FORTRAN Programmers*, O'Reilly and Associates, Inc., Sebastopol, CA, (1990).

L.P. Meissner, E.I. Organick, *Fortran 77, Featuring Structured Programming*, Addison-Wesley Publishing Company, Reading, MA, (1980).

L.P. Meissner, *Fortran 90*, PWS Publishing Company, Boston, MA, (1995).

C.G. Page, *Professional Programmer's Guide to Fortran 77*, Pitman, Londres, (1988).

A. Robbins, D. Gilly, *UNIX in a Nutshell*, O'Reilly and Associates, Inc., Sebastopol, CA, (1999).

N.L. Schryer, *A Test of a Computer's Floating-Point Arithmetic Unit*, Computer Science Technical Report no. 89, AT&T Bell Laboratories, Murray Hill, NJ, (1981).

Scilab Group, *Scilab, A Free Scientific Software Package*, <http://www.scilab.org/>, (1989).

R. Serban, C. Woodward, A. Hindmarsh, *SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers)*, <http://www.llnl.gov/casc/sundials/>, (1998).

K. Sigmon, *MATLAB Primer*, 2ª edição, <x<ftp://ftp.math.ufl.edu/pub/matlab/primer35.ps>, (1992).

R. Stallman, *GNU Emacs Manual*, 13ª edição, Free Software Foundation, Boston, MA, (1999). (Manual disponível conjuntamente com o programa, em <ftp://ftp.gnu.org>).

B. Stroustrup, *The C++ Programming Language*, Addison-Wesley Publishing Company, Reading, MA, (1997).

H. Walker, *NITSOL: A Newton Iterative Solver for Nonlinear Systems*, <http://users.wpi.edu/~walker/NITSOL/>, (2003).

S. Wolfram, *The Mathematica Book*, 3ª edição, Cambridge University Press, Cambridge, UK, (1996).

APÊNDICE B

Exercícios de aplicação

ESTA Secção contém alguns exercícios de aplicação, recolhidos de exames da disciplina de Prática de Computação da licenciatura em Engenharia Química do DEQ/FCTUC, particularmente no ano lectivo de 1998/1999.

B.1 Capítulo 1 — Introdução

1. (Fev/99)

   (a) Descreva sucintamente (estabelecendo as distinções possíveis entre) os conceitos de *condicionamento de um problema* e *estabilidade de um algoritmo*.

   (b) A representação de quantidades reais no computador HAL 9000 (de grande porte) é feita usando um formato semelhante ao standard IEEE 754. No HAL, cada número real é representado usando um conjunto de 36 bits, sendo um destes bits reservado para o sinal, 8 bits para o expoente, e os restantes para a mantissa. Tal como no standard IEEE, os números são representados em base binária, em formato normalizado. Qual a precisão esperada na representação de quantidades reais, em *precisão simples*, nesta máquina?

   (c) Qual a razão que torna particularmente perigosas as operações de subtracção (e adição aritmética) de quantidades quase iguais, na implementação de métodos numéricos?

2. (Set/99)

   (a) O formato IEEE 754 utiliza um total de 64 bits para representar quantidades em precisão dupla. Destes, 52 bits são usados para representar a mantissa, sendo os restantes atribuídos ao expoente. Com base

nesta informação determine aproximadamente o *número mais negativo* (i.e., o número negativo de maior magnitude) que é possível representar neste formato.

(b) Indique de forma clara mas sucinta as relações entre *erro relativo*, *erro absoluto*, *número de algarismos significativos* e *número de casas decimais correctas*.

B.2 Capítulo 2 — Equações não-lineares

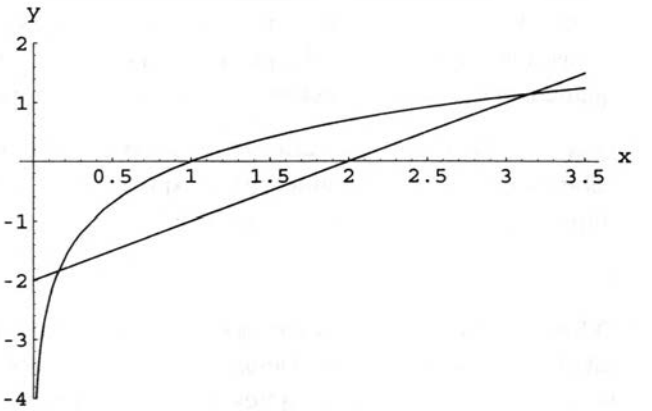
1. (Fev/99) O método de Steffensen para resolução de equações não-lineares é uma variante do método das substituições sucessivas, com a vantagem de garantir uma *convergência quadrática*. Se as equações $f(x) = 0$ e $x = g(x)$ forem equivalentes, então a fórmula de recorrência do método de Steffensen toma a forma:

$$x_{k+1} = x_k - \frac{[g(x_k) - x_k]^2}{g[g(x_k)] - 2g(x_k) + x_k}$$

(a) Qual a relação entre as estimativas produzidas por este método, perto da solução, para ser verificada uma convergência *quadrática*?

(b) Determine o zero da função $f(x) = \exp(x) - x - 2$, usando como estimativa inicial $x_0 = 1$. Utilize como critério de paragem $|f(x)| \leq 10^{-3}$.

2. (Set/99) Determine os pontos de intersecção das curvas $y(x) = \ln(x)$ e $y(x) = x - 2$ representadas na figura, utilizando o método de Newton. Escolha as estimativas iniciais e o critério de paragem que lhe pareçam adequados. Averigue ainda por que razão o método não converge para as estimativas iniciais $x = 0.4$ e $x = 1$.



3. (Set/99) A um grupo do ano anterior foi pedida a implementação computacional do método da *regula falsi* para a determinação do zero de uma função univariável. A listagem estava assim no relatório:

```
      FUNCTION RTFLSP(FUNC,X1,X2,Xacc)

      IMPLICIT NONE
      INTEGER MAXIT
      REAL RTFLSP, X1, X2, Xacc, FUNC
      EXTERNAL FUNC

c     --- No. maximo de iteracoes ---
      PARAMETER (MAXIT=30)
      REAL del, dx, f, fh, fl, sawp, xh, xl

c     --- Usando o metodo da regula falsi, encontrar a raiz da funcao
c         FUNC, situada entre X1 e X2. A raiz, devolvida em RTFLSP, e'
c         refinada ate a precisao da sua localizacao ser + ou - XACC.

      fl = FUNC(X1)
      fh = FUNC(X2)
      IF ( fl*fh.LT.0. ) THEN
         PRINT * , 'A raiz deve estar entre X1 e X2!'
         STOP
      ENDIF
      IF ( fl.LT.0. ) THEN
         xl = X1
         xh = X2
      ELSE
         xl = X2
         xh = X1
         swap = fl
         fl = fh
         fh = swap
      ENDIF
      dx = xh - xl
      DO j = 1 , MAXIT
         RTFLSP = xl + dx*fh/(fl-fh
         f = FUNC(RTFLSP)
         IF ( f.LT.0. ) THEN
            del = xh - RTFLSP
            xh = RTFLSP
            fh = f
         ELSE
            del = xl - RTFLSP
            xl = RTFLSP
            fl = f
         ENDIF
         dx = xh - xl
         IF ( ABS(del).LT.Xacc .OR. f.EQ.0. ) RETURN
      ENDDO
      PRINT * , 'No. maximo de iteracoes excedido em rtflsp."
      END
```


O assistente achou estranho e resolveu compilar este programa, usando o comando `g77 -c rtflsp.f`, encontrando o seguinte diagnóstico:

```
rtflsp.f: In function 'rtflsp':
rtflsp.f:28:
            swap = fl
            ^
Invalid declaration of or reference to symbol 'swap' at (^) [initially
 seen at (^)]
rtflsp.f:33:
         DO j = 1 , MAXIT
            ^
Invalid declaration of or reference to symbol 'j' at (^) [initially seen
at (^)]
rtflsp.f:34:
            RTFLSP = xl + dx*fh/(fl-fh
                                 1     2
Missing close parenthese at (2) needed to match open parenthese at (1)
rtflsp.f:48:
         PRINT * , 'No. maximo de iteracoes excedido em rtflsp."
                   1                                           2
Character constant at (1) has no closing apostrophe at (2)
Exit 1
```

(a) Identifique e corrija todos os erros de *sintaxe* no programa anterior.

(b) Após a eliminação dos erros de sintaxe, verificou-se que o programa ainda não produzia os resultados pretendidos. Corrija-o e transforme-o numa rotina que implemente o método da *regula falsi modificado*.

4. (Dez/99)

(a) Deduza as equações do método de Newton para um sistema de 2 equações não-lineares a 2 incógnitas (Sugestão: Tal como para o método unidimensional, use a fórmula de Taylor das funções em torno de um ponto base, e verifique quais as correcções nas variáveis independentes que permitem que estas equações se anulem).

(b) Utilize este método para determinar a solução do seguinte sistema, com um erro relativo de 0,5%:

$$\begin{cases} x_1^2 + x_2 = 3 \\ x_1 + \ln x_2 = 1 \end{cases}$$

(c) Será possível utilizar o método de Newton para determinar as raízes complexas de equações polinomiais $p(x) = 0$? Justifique.

5. (Dez/99) O programa seguinte pretende testar a rotina **newton** que implementa o método de Newton-Raphson.

```
      program teste    ! esta e' a primeira linha do programa
      external func
      double precision x1, x2, tol, newton
c
      x1  =  1.0d-2
      x2  = 10.0d0
      tol =  0.0d-4
c
      xsol = newton( func, x1, x2, tol )
c
      write(*,*) ' Resultado xsol = ' xsol
      end
c
c     ----------------------------------------------------------------
c
      double precision function newton( func, x1, x2, tol )
c
      integer itmax
      parameter( itmax = 50 )        ! No. maximo de iteracoes
c
      double precision x1, x2, tol, xnewt, df, dx, f
c
c     * Usando o metodo de Newton-Raphson, encontrar a raiz da funcao f
c       (ver rotina func) situada entre x1 e x2.  A raiz e' devolvida
c       na variavel newton e obtida com uma precisao de + ou - tol.
c
      xnewt = 0.5d0*( x1 + x2 )           ! Estimativa inicial
c
      do j=1, itmax
         call func( xnewt, df, f )
         dx = f / df
         xnewt = xnewt + dx
         if ( (x1 - xnewt)*(xnewt - x2) .gt. 0.d0 ) then
            write(*,*) 'xnewt esta fora do intervalo!; xnewt = ', xnewt
            stop
         else if ( dabs( dx ) .lt. tol ) then
```

```
            newton = xnewt
          endif
        end do
        write(*,*) 'newton atingiu o numero  max. de iteraccoes'
        end
c
        subroutine func( x, f, df )
        double precision x, f, df
        f = x**2 - x**3 + 1.d0/x
        df = 2.d0*x - 3.d0*x - 1.d0/x**2
        return
        end
```

Ao compilar o programa obtém-se a seguinte informação:

```
   MAIN teste:
Error on line 9: Declaration error for xsol: attempt to use undefined
   variable
Error on line 11: invalid hexadecimal character
Error on line 11: syntax error
   newton:
Error processing entries before line 27: Declaration error for func:
   attempt to use undefined variable
Error on line 29: Declaration error for j: attempt to use undefined
   variable
```

(a) Identifique e corrija os erros de sintaxe acima indicados.

(b) Após a eliminação dos erros de sintaxe, verificou-se que o programa ainda não produzia os resultados pretendidos. Identifique e corrija todos os erros lógicos e de programação, por forma a este programa produzir os resultados pretendidos.

B.3 Capítulo 3 — Sistemas de equações lineares

1. (Fev/99) Um dos algoritmos mais eficientes para calcular a inversa de uma matriz $A \in \mathbb{R}^{n\times n}$ consiste em: (i) efectuar a decomposição LU da matriz A; (ii) resolver o sistema de equações $AX = I$ (na realidade n sistemas de n equações), onde $I \in \mathbb{R}^{n\times n}$ é a matriz identidade. A inversa de A é precisamente a matriz solução X. Aplique este algoritmo à seguinte matriz:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 10 \\ 2 & 4 & 17 \end{bmatrix}$$

2. (Fev/99) A resolução de sistemas lineares $Ax = b$, quando A é uma matriz triangular superior, pode ser feita de forma eficiente usando o método da *substituição regressiva*. O programa seguinte (ficheiro `teste.f`) corresponde à primeira tentativa de implementação deste algoritmo, efectuada pelos vossos colegas:

```
      program vector  ! esta e' a linha 1
c
      integer i, index, j, n
      parameter( n = 4 )
      double precision a(n,n), b(n) x(n), soma
c
      do i = 1, n
```

```
         read(*,*) (a(i,j), j = 1, n) b(i)
      end do
c
      do index = 1, n
         i = n + 1 - indx
         som = 0.d0
c
         do j = i + 1, n
            soma = soma + a(i,i)*x(j)
         end do
         x(i) = (b(i) - soma/a(i,j)
c
      write(*,*)
      write(*700) (x(i), i = 1, n)
  700 format( 'vector x:', 10(1x, f8.5  )
      end
```

Para testar este programa foi criado o seguinte ficheiro de dados, com o nome `teste.dat`:

```
3. 2. 3. 1.    17.
0. 6. 0. 4.    16.
0. 0. 8. 5.    53.
0. 0. 0. 7.     7.
```

Infelizmente ao compilar o programa anterior, usando o comando
`f77 teste.f -o teste.x`
foram encontrados os seguintes erros de sintaxe:

```
teste.f:
MAIN vector:
"teste.f", line 5: Error: syntax error at or near variable name "x"
"teste.f", line 5: Warning: local variable "b" never used
"teste.f", line 8: Error: syntax error at or near variable name "b"
"teste.f", line 11: Error: do not closed
"teste.f", line 18: Error: unbalanced parentheses, statement skipped
"teste.f", line 18: Error: unclassifiable statement
"teste.f", line 21: Error: syntax error at or near integer constant "700"
"teste.f", line 22: Error: unbalanced parentheses, statement skipped
"teste.f", line 22: Error: unclassifiable statement
Compilation failed
```


(a) Identifique e corrija todos os erros de *sintaxe* no programa anterior.

(b) Após a eliminação dos erros de sintaxe, verificou-se que a solução obtida não era a esperada. Identifique os restantes erros e deficiências desta implementação, de forma a que ela permita encontrar a solução correcta de um sistema triangular superior genérico.

3. (Set/99) Antes da resolver um sistema de equações $A.x = b$, usa-se frequentemente um procedimento para *equilibrar* ou *balancear* a matriz A. Quais as vantagens deste passo adicional? Descreva uma possível implementação deste procedimento.

4. (Set/99) Considere o seguinte sistema de equações lineares:

$$\begin{cases} 3x_1 + x_2 + 2x_3 = 9 \\ x_1 + 5x_2 + 4x_3 + x_4 = 2 \\ 2x_1 + 4x_2 + 7x_3 + 5x_4 = 9 \\ x_2 + 5x_3 + 4x_4 = 2 \end{cases}$$

   (a) Qual o método numérico mais eficiente para a sua resolução? Indique uma estimativa do número de operações necessárias para a sua solução.

   (b) Resolva este sistema, usando o método da alínea anterior.

5. (Dez/99)

   (a) Em que circunstâncias devem ser preferidos os métodos iterativos para a resolução de sistemas lineares, face aos seus congéneres de natureza directa?

   (b) Quais as principais diferenças entre o método de Jacobi e o método de Gauss-Seidel para a resolução de sistemas lineares?

Nuno M.C. de Oliveira licenciou-se em Engenharia Química pela Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC) em 1987. Em 1994 obteve o Ph.D. em Engenharia Química, na Universidade de Carnegie Mellon, em Pittsburgh, PA, EUA, e a equivalência ao grau de Doutor em Engenharia Química, pela Universidade de Coimbra. Exerce, desde esse ano, as funções de Professor Auxiliar no Departamento de Engenharia Química da FCTUC.

A componente docente do seu percurso académico tem sido dedicada à leccionação de várias disciplinas da área de Processos Químicos, como Sistemas e Processos Químicos, Dinâmica de Sistemas, Controlo Automático, Sistemas Avançados de Controlo, e Projecto Industrial.

ISBN 972-8704-74-7
9 789728 704742

Série
Ensino
•
Imprensa da Universidade de Coimbra
Coimbra University Press
2006