Matemática Computacional
Cap.1 - Conceitos básicos do Cálculo Científico

Sumário

1 Erros em cálculo numérico

Ao longo de um processo de cálculo para resolver um problema, surgem erros de diversos tipos, que podem ser agrupados em quatro categorias:

1. Erros do modelo matemático
2. Erros dos dados
3. Erros computacionais
4. Erros do método numérico

Não iremos abordar os erros do modelo matemático, que podem surgir quando o modelo não traduz fielmente a realidade, e os *erros dos dados* serão tidos em conta apenas na propagação de erros em funções e algoritmos.

Os *erros computacionais* surgem porque os computadores têm memória limitada, o que não permite representar números nem efetuar cálculos com precisão infinita. Os *erros do método numérico* devem-se ao uso de fórmulas que apenas dão valores aproximados para a solução do problema matemático em causa, por exemplo, truncatura de séries e utilização de polinómios de Taylor para aproximar funções. Em Matemática Computacional, o nosso objectivo principal é o estudo dos *métodos numéricos* básicos do cálculo científico, incluindo a análise teórica das suas propriedades de convergência, a implementação computacional dos algoritmos correspondentes e a sua aplicação a exemplos da engenharia, da física, da economia, etc. A análise dos métodos numéricos centra-se no estudo de condições que garantem a sua convergência e na dedução de fórmulas de *majoração dos erros*.

Relativamente à notação que será usada, no caso de aproximação de números reais, sendo $\tilde{x}$ um valor aproximado de x, o que se representa por $x \approx \tilde{x}$, define-se *erro de* $\tilde{x}$ por $e_{\tilde{x}} = x - \tilde{x}$. Para $x \neq 0$, define-se ainda $\delta_{\tilde{x}} = \dfrac{e_{\tilde{x}}}{x}$. O *erro absoluto de* $\tilde{x}$ é dado por $|e_{\tilde{x}}| = |x - \tilde{x}|$ e o *erro relativo de* $\tilde{x}$ é dado por $|\delta_{\tilde{x}}| = \dfrac{|e_{\tilde{x}}|}{|x|}$. O erro relativo percentual de $\tilde{x}$ é $100\%|\delta_{\tilde{x}}|$. Estas noções podem ser generalizadas a vetores e matrizes. Isto será feito no capítulo dedicado à resolução de sistemas de equações.

2 Representação de números no computador

2.1 Sistemas de ponto flutuante

Vamos introduzir os sistemas de ponto flutuante e estudar as suas propriedades e limitações. Uma dificuldade que podemos imediatamente identificar ao passar da matemática para os

computadores é a impossibilidade inerente de representar um número real em memória finita. Isso implica incorrer em algum erro de aproximação ao realizar um cálculo computacional.

Definição 2.1. *Sejam $\beta \in \mathbb{N} \setminus \{1\}$, $n \in \mathbb{N}$, $t_{min}, t_{max} \in \mathbb{Z}$. Designa-se por sistema de ponto flutuante na base β, com n dígitos na mantissa $(0.a_1...a_n)_\beta$ e expoente t variando entre t_{min} e t_{max} o conjunto*

$$\begin{aligned}\mathbb{F} &= \mathbb{F}(\beta, n, t_{min}, t_{tmax}) \\ &= \{0\} \cup \{x \in \mathbb{Q} : x = (-1)^s(0.a_1a_2...a_n)_\beta \times \beta^t,\ a_1 \neq 0,\ s \in \{0,1\},\ t \in \mathbb{Z} \cap [t_{min}, t_{max}]\}.\end{aligned}$$

A mantissa pode ser escrita como

$$\begin{aligned}(0.a_1a_2...a_n)_\beta &= a_1(0.10...0)_\beta + a_2(0.010...0)_\beta + ... + a_n(0.0...01)_\beta \\ &= a_1\beta^{-1} + a_2\beta^{-2} + ... + a_n\beta^{-n}\end{aligned}$$

onde $a_i \in \{0, ..., \beta - 1\}$, para $i = 1, ..., n$, sendo $a_1 \neq 0$. A condição $a_1 \neq 0$ corresponde a uma normalização e garante a unicidade da representação no sistema de ponto flutuante $\mathbb{F}$. Além disso, impede que se utilize memória desnecessariamente para guardar zeros que surgiriam a seguir ao ponto caso não fosse $a_1 \neq 0$. É claro que, em geral, $t_{min} < 0$ e $t_{max} > 0$.

Exemplo 2.1. *Qual é o menor número positivo pertencente a $\mathbb{F} := \mathbb{F}(\beta, n, t_{min}, t_{max})$? E o maior número positivo? Quanto é $\#\mathbb{F}$?*

O menor número positivo é

$$\begin{aligned}x_{min} &= (0.10...0)_\beta \times \beta^{t_{min}} \\ &= \beta^{-1} \times \beta^{t_{min}} = \beta^{t_{min}-1}\end{aligned} \tag{1}$$

e o maior número positivo é

$$\begin{aligned}x_{max} &= (0.(\beta-1)(\beta-1)...(\beta-1))_\beta \times \beta^{t_{max}} \\ &= (1 - (0.0...01)_\beta) \times \beta^{t_{max}} = (1 - \beta^{-n}) \times \beta^{t_{max}} = \beta^{t_{max}} - \beta^{t_{max}-n}.\end{aligned} \tag{2}$$

Vejamos agora quantos números positivos pertencem a $\mathbb{F}$. Em

$$x = (-1)^s(0.a_1a_2...a_n)_\beta \times \beta^t$$

o dígito a_1 pode tomar $\beta - 1$ valores e os restantes dígitos $a_2, ..., a_n$ podem tomar β valores cada um. O expoente t pode tomar $t_{max} - t_{min} + 1$ valores. Assim, existem $(\beta - 1) \times \beta^{n-1} \times (t_{max} - t_{min} + 1)$ números positivos em $\mathbb{F}$ e

$$\#\mathbb{F} = 1 + 2 \times (\beta - 1) \times \beta^{n-1} \times (t_{max} - t_{min} + 1).$$

2.2 Arredondamentos, *underflow* e *overflow*. Unidade de arredondamento do sistema (*machine epsilon*).

Um sistema de ponto flutuante é um conjunto finito, onde apenas é possível obter resultados com precisão finita. Isto leva-nos à definição e estudo de arredondamentos, *underflow* e *overflow*.

Pretendemos responder à seguinte questão: Dado um número $x \in \mathbb{R}$, se $x \notin \mathbb{F}$, como é feita a sua representação, $\text{fl}(x)$, no sistema $\mathbb{F}$?

Se $0 < |x| < x_{min}$, onde x_{min} é dado por (1), não há precisão suficiente para representar x em $\mathbb{F}$. Diz-se que x está no nível de *Underflow* e será $\text{fl}(x) = 0$, ou seja, ocorre *underflow* quando números próximos de 0 são arredondados para 0.

Se $|x| > x_{max}$, sendo x_{max} dado em (2), então x está na zona de *Overflow* e eventualmente é apresentado INF como output.

Para $x \in [-x_{max}, -x_{min}] \cup [x_{min}, x_{max}]$, partindo da representação

$$x = (-1)^s (0.a_1 a_2 ... a_n a_{n+1} ...)_\beta \times \beta^t,\ a_1 \neq 0,$$

define-se

- Arredondamento por corte ou truncatura

$$\text{fl}(x) = \text{fl}_c(x) = (-1)^s (0.a_1 a_2 ... a_n)_\beta \times \beta^t$$

- Arredondamento simétrico (quando β é par)

$$\text{fl}(x) = \text{fl}_s(x) = \begin{cases} (-1)^s (0.a_1 a_2 ... a_n)_\beta \times \beta^t,\ 0 \leq a_{n+1} < \frac{\beta}{2}, \\ \text{fl}\left((-1)^s \left((0.a_1 a_2 ... a_n)_\beta + \beta^{-n}\right) \times \beta^t\right),\ \frac{\beta}{2} \leq a_{n+1} \leq \beta - 1, \end{cases}$$

  onde o último fl significa apenas uma eventual normalização.

Assim, $\text{fl}_s(x)$ é o número de $\mathbb{F}$ mais perto de x, enquanto que $\text{fl}_c(x)$ é o número de $\mathbb{F}$ mais perto de x que está entre 0 e x.

Exemplo 2.2. *1. A representação de* $\pi = 3.1415926535... = 0.31415926535... \times 10^1$ *num sistema de ponto flutuante* $\mathbb{F}(10, 7)$ *é*

$$\text{fl}_c(\pi) = 0.3141592 \times 10^1,$$

$$\text{fl}_s(\pi) = \text{fl}((0.3141592 + 0.0000001) \times 10^1) = 0.3141593 \times 10^1$$

consoante seja feito arredondamento por corte ou simétrico.

Num sistema $\mathbb{F}(10, 4)$*, as representações de* $x = -0.0013295$ *e* $y = 0.9999801$ *são*

$$\text{fl}_c(x) = \text{fl}_c(-0.13295 \times 10^{-2}) = -0.1329 \times 10^{-2}$$

$$\text{fl}_s(x) = \text{fl}(-(0.1329 + 0.0001) \times 10^{-2}) = -0.1330 \times 10^{-2}$$

e

$$\text{fl}_c(y) = \text{fl}_c(0.9999801 \times 10^0) = 0.9999 \times 10^0$$

$$\text{fl}_s(x) = \text{fl}((0.9999 + 0.0001) \times 10^0) = \text{fl}(1.000 \times 10^0) = 0.1000 \times 10^1.$$

Teorema 2.1. *Para qualquer* $x \in \mathbb{R}$ *com representação em* $\mathbb{F}$*, tem-se*

$$|e_{\text{fl}(x)}| \leq \beta^{t-n}, \quad |\delta_{\text{fl}(x)}| \leq \beta^{1-n}, \text{ em arredondamento por corte,}$$

$$|e_{\text{fl}(x)}| \leq \frac{1}{2}\beta^{t-n}, \quad |\delta_{\text{fl}(x)}| \leq \frac{1}{2}\beta^{1-n}, \text{ em arredondamento simétrico.}$$

Demonstração. Para o erro absoluto de arredondamento de um número real

$$x = (-1)^s(0.a_1a_2...a_na_{n+1}...)_\beta \times \beta^t$$

tem-se

$$|x - \text{fl}_c(x)| = (0.0...0a_{n+1}...)_\beta \times \beta^t = (0.a_{n+1}...)_\beta \times \beta^{t-n} \leq \beta^{t-n}$$

e

$$|x - \text{fl}_s(x)| \leq \frac{1}{2}\beta^{t-n}.$$

As desigualdades para os erros relativos decorrem de

$$|x| \geq (0.10...0...)_\beta \times \beta^t = \beta^{t-1} \Rightarrow \frac{1}{|x|} \leq \beta^{1-t}.$$

□

A quantidade β^{1-n}, que permite controlar os erros relativos de arredondamento, depende apenas do sistema de ponto flutuante.

Definição 2.2. *A* unidade de arredondamento do sistema de ponto flutuante $\mathbb{F}(\beta, n, t_{min}, t_{max})$ *é*

$$\epsilon_M = \beta^{1-n}, \text{ quando é utilizado arredondamento por corte,}$$

ou

$$\epsilon_M = \frac{1}{2}\beta^{1-n}, \text{ quando é utilizado arredondamento simétrico.}$$

A ϵ_M também se chama precisão do sistema de ponto flutuante, precisão da máquina, machine epsilon *ou epsilon da máquina.*

Em geral tem-se

$$0 < x_{min} \ll \epsilon_M \ll x_{max}.$$

2.3 Os padrões IEEE (Institute of Electrical and Electronics Engineers)

Há décadas atrás, cada computador tinha seu próprio sistema numérico de ponto flutuante (representação e aritmética de números reais no hardware). Alguns eram decimais, alguns usavam 2 como base, outros usavam base 8 ou 16. E todos tinham uma precisão diferente. Em 1985, o *IEEE Standards Board* e o *American National Standards Institute* adotaram o padrão ANSI/IEEE 754-1985 para a aritmética binária de ponto flutuante. Foi o culminar de quase uma década de trabalho realizado por um grupo interdisciplinar de 92 pessoas, incluindo engenheiros, matemáticos, cientistas da computação, académicos, fabricantes de computadores e empresas de microprocessadores. Todos os computadores projetados desde 1985 utilizam formatos internos e aritmética de ponto flutuante IEEE. Isto não significa que todos obtenham exatamente os mesmos resultados, porque há alguma flexibilidade dentro do padrão, mas significa que existe um modelo independente da máquina para a realização da aritmética de ponto flutuante. Em precisão simples são usados 32 bits (4 bytes) para a representação numérica de números reais: 1 bit para o sinal, 8 bits para o expoente e 23 bits para a mantissa. A precisão dupla utiliza 64 bits (8 bytes) para a representação numérica: 1 bit para o sinal, 11 bits para o expoente e 52 bits para a mantissa.

Concretamente, no padrão IEEE 754, define-se a representação dos números de ponto flutuante diferentes de zero por

$$x = (-1)^s(d_1.d_2...d_n)_2 \times 2^t,\ d_1 \neq 0,\ s \in \{0,1\}, t \in \mathbb{Z} \cap [t_{max}, t_{min}].$$

A condição de normalização $d_1 \neq 0$ no caso da base $\beta = 2$ significa simplesmente $d_1 = 1$, pelo que podemos considerar a representação

$$x = (-1)^s(1 + 0.a_1...a_N)_2 \times 2^t,\ s \in \{0,1\}, t \in \mathbb{Z} \cap [t_{max}, t_{min}],$$

onde se devem tomar os seguintes valores para $N = n - 1$, t_{max} e t_{min}

Sistema	N	t_{min}	t_{max}
IEEE precisão simples	23	-126	127
IEEE precisão dupla	52	-1022	1023

Exemplo 2.3. *Algumas correspondências com o sistema decimal usual:*

$$(0.10...0)_2 = 2^{-1} = (0.500)_{10}$$

$$(0.010...0)_2 = 2^{-2} = (0.250)_{10}$$

$$(0.0010...0)_2 = 2^{-3} = (0.125)_{10}$$

$$(1.0010...0)_2 = (1 + 2^{-3})_2 = (1 + 0.125)_{10} = (1.125)_{10}.$$

Nestes sistemas tem-se

Sistema	N	x_{min}	x_{max}	ϵ_M
IEEE precisão simples	23	2^{-126}	$(1.1...1)_2 \times 2^{127}$	$2^{-23} \approx 0.2 \times 10^{-6}$
IEEE precisão dupla	52	2^{-1022}	$(1.1... ...1)_2 \times 2^{1023}$	$2^{-52} \approx 0.2 \times 10^{-15}$

o que está de acordo com o que nos é fornecido pelo MATLAB (padrão IEEE com precisão dupla):

```
>>  realmin
ans = 2.2251x 10^(-308)

>> realmax
ans = 1.7977 x 10^308

>> eps
ans = 2.2204x10^(-16)
```

Há a registar várias evoluções do padrão ANSI/IEEE 754-1985: em 1989, recebeu reconhecimento internacional como IEC 559; em 2008, surgiu uma versão atualizada, denominada IEEE 754-2008, a qual é uma revisão significativa e substitui o padrão de ponto flutuante IEEE 754-1985; em 2011, foi aprovado o padrão internacional ISO/IEC/IEEE 60559:2011; em 2019 deu-se uma revisão menor IEEE 754-2019.

Atualmente o padrão IEEE também especifica formatos de precisão quádrupla (16 bytes) que alguns compiladores C e C++ implementam como dados de tipo `long double`.

2.4 Cálculo em sistemas de ponto flutuante

Vamos considerar a implementação das *operações aritméticas elementares* em sistemas de ponto flutuante de acordo com a seguinte condição : se $x, y \in \mathbb{F}$ e $\mathtt{op} \in \{+, -, \times, /\}$ então

$$\mathrm{fl}(x \,\mathtt{op}\, y) = (x \,\mathtt{op}\, y)(1+\delta), \quad |\delta| \leq \epsilon_M.$$

Define-se $\mathtt{op}_{\mathbb{F}} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ por

$$x \,\mathtt{op}_{\mathbb{F}}\, y = \mathrm{fl}(x \,\mathtt{op}\, y) \tag{3}$$

assumindo que continuamos a ter $\mathrm{fl}(x \,\mathtt{op}\, y) \in \mathbb{F}$ e que são usados algarismos de guarda para efetuar as operações $x \,\mathtt{op}\, y$. Deste modo o erro de arredondamento que surge em $x \,\mathtt{op}_{\mathbb{F}}\, y$ satisfaz

$$|\delta_{x \,\mathtt{op}_{\mathbb{F}}\, y}| = \frac{|x \,\mathtt{op}\, y - x \,\mathtt{op}_{\mathbb{F}}\, y|}{|x \,\mathtt{op}\, y|} \leq \epsilon_M.$$

As *funções elementares* cos, sin, exp, etc, são implementadas de forma análoga. Se $x \in \mathbb{F} \cap \mathrm{Dom}(g)$ então

$$\mathrm{fl}(g(x)) = g(x)(1+\delta), \quad |\delta| \leq \epsilon_M,$$

o que permite definir $g_{\mathbb{F}} : \mathbb{F} \cap \mathrm{Dom}(g) \to \mathbb{F}$ por

$$g_{\mathbb{F}}(x) = \mathrm{fl}(g(x)). \tag{4}$$

Tem-se ainda

$$|\delta_{g_{\mathbb{F}}(x)}| = \frac{|g(x) - g_{\mathbb{F}}(x)|}{|g(x)|} \leq \epsilon_M.$$

De uma forma geral, define-se *algoritmo* como sendo uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito com uma quantidade de esforço finita. No contexto do Cálculo Científico, um algoritmo é uma sequência finita de cálculos elementares, tal como definidos em (3) e (4).

Um *programa* corresponde a um algoritmo escrito numa linguagem de programação (linguagem que é entendida pelo computador).

Exemplo 2.4. *De acordo com a associatividade da soma em* $\mathbb{R}$*, a soma de três números reais positivos* $S = a + b + c$ *pode ser feita usando os algoritmos*

$$(1)\, S_1 = (a+b)+c; \qquad\qquad (2)\, S_2 = a+(b+c).$$

Para $a = 2745.568$*,* $b = 34.68734$*,* $c = 0.0003$*, vamos efetuar o cálculo de* S *num sistema* $\mathbb{F}(10,7)$ *com arredondamento simétrico usando os dois algoritmos. Estes números têm representação exata em* $\mathbb{F}$*, ou seja, não é necessário efetuar arredondamento (apenas normalização) para os representar:*

$$\mathrm{fl}(a) = 0.2745568 \times 10^4, \quad \mathrm{fl}(b) = 0.3468734 \times 10^2, \quad \mathrm{fl}(c) = 0.3000000 \times 10^{-3}.$$

Utilizando o algoritmo (1), ou seja, $z_1 = a + b$*,* $z_2 = z_1 + c$*, para calcular* S_1 *em* $\mathbb{F}$*, obtém-se*

$$\begin{aligned}
\widetilde{z_1} &= \mathrm{fl}(0.2745568 \times 10^4 + 0.3468734 \times 10^2) = \mathrm{fl}(0.2745568 \times 10^4 + 0.003468734 \times 10^4) \\
&= \mathrm{fl}(0.278025534 \times 10^4) = 0.2780255 \times 10^4 \\
\widetilde{z_2} &= \mathrm{fl}(0.2780255 \times 10^4 + 0.3000000 \times 10^{-3}) \\
&= \mathrm{fl}(0.2780255 \times 10^4 + 0.00000003 \times 10^4) \\
&= \mathrm{fl}(0.27802553 \times 10^4) = 0.2780255 \times 10^4 = (S_1)_{\mathbb{F}}.
\end{aligned}$$

A execução em $\mathbb{F}$ *do algoritmo (2),* i.e*,* $w_1 = b + c$*,* $w_2 = a + w_1$*, produz o seguinte valor para* S_2*:*

$$\begin{aligned}
\widetilde{w_1} &= \mathrm{fl}(0.3468734 \times 10^2 + 0.3000000 \times 10^{-3}) \\
&= \mathrm{fl}(0.3468734 \times 10^2 + 0.000003 \times 10^2) = \mathrm{fl}(0.3468764 \times 10^2) \\
&= 0.3468764 \times 10^2 \\
\widetilde{w_2} &= \mathrm{fl}(0.2745568 \times 10^4 + 0.3468764 \times 10^2) = \mathrm{fl}(0.278025564 \times 10^4) \\
&= 0.2780256 \times 10^4 = (S_2)_{\mathbb{F}}.
\end{aligned}$$

Tem-se $S = 2780.25564$ *(cálculo efetuado com mais precisão) e*

$$\left|\delta_{(S_1)_{\mathbb{F}}}\right| = \frac{|S - (S_1)_{\mathbb{F}})|}{S} \approx 2.30195 \times 10^{-7},$$

$$\left|\delta_{(S_2)_{\mathbb{F}}}\right| = \frac{|S - (S_2)_{\mathbb{F}})|}{S} \approx 1.29484 \times 10^{-7}.$$

Ambos os cálculos apresentam erros relativos muito pequenos, mas o erro correspondente ao segundo algoritmo é menor. Como $(a+b)+c \neq a+(b+c)$ *em* $\mathbb{F}$*, conclui-se que* a adição não é associativa em sistemas de ponto flutuante.

3 Propagação de erros no cálculo de funções e na execução de algoritmos

O erro de arredondamento, por si só potencialmente preocupante, quando transportado várias vezes ao longo de muitos cálculos, pode ter um *efeito de bola de neve*. Vamos tentar perceber como isto acontece.

3.1 Propagação de erros nas operações aritméticas elementares

Começamos por entender a sensibilidade das operações aritméticas elementares a eventuais variações nos seus argumentos. Para tal, sejam $x, y \in \mathbb{R}\setminus\{0\}$ e $\tilde{x}, \tilde{y} \in \mathbb{R}$ valores aproximados. Sendo $e_{\tilde{x}} = x - \tilde{x}$, $e_{\tilde{y}} = y - \tilde{y}$, $\delta_{\tilde{x}} = \frac{e_{\tilde{x}}}{x}$ e $\delta_{\tilde{y}} = \frac{e_{\tilde{y}}}{y}$, tem-se a seguinte propagação de erros nas operações aritméticas elementares (supondo que são efetuadas em $\mathbb{R}$):

- Ao aproximar $x + y \approx \tilde{x} + \tilde{y}$ em $\mathbb{R}$, os erros satisfazem

  $$e_{\tilde{x}+\tilde{y}} = x + y - (\tilde{x} + \tilde{y}) = e_{\tilde{x}} + e_{\tilde{y}}$$

  e

  $$\delta_{\tilde{x}+\tilde{y}} = \frac{e_{\tilde{x}+\tilde{y}}}{x+y} = \frac{1}{x+y}e_{\tilde{x}} + \frac{1}{x+y}e_{\tilde{y}} = \frac{x}{x+y}\frac{e_{\tilde{x}}}{x} + \frac{y}{x+y}\frac{e_{\tilde{y}}}{y} = \frac{x}{x+y}\delta_{\tilde{x}} + \frac{y}{x+y}\delta_{\tilde{y}}.$$

  Podemos supor que $x, y > 0$. Se os erros relativos de $\tilde{x}$ e $\tilde{y}$ são pequenos, ou seja, se $|\delta_{\tilde{x}}|, |\delta_{\tilde{y}}| \ll 1$ e $x, y > 0$ então o erro relativo da soma também é pequeno pois

  $$|\delta_{\tilde{x}+\tilde{y}}| \leq \frac{x}{x+y}|\delta_{\tilde{x}}| + \frac{y}{x+y}|\delta_{\tilde{y}}|$$

  e $\frac{x}{x+y}, \frac{y}{x+y} < 1$.

- De forma análoga, os erros inerentes à aproximação $x - y \approx \tilde{x} - \tilde{y}$ são dados por

  $$e_{\tilde{x}-\tilde{y}} = x - y - (\tilde{x} - \tilde{y}) = e_{\tilde{x}} - e_{\tilde{y}}$$

  e

  $$\delta_{\tilde{x}-\tilde{y}} = \frac{e_{\tilde{x}-\tilde{y}}}{x-y} = \frac{1}{x-y}e_{\tilde{x}} - \frac{1}{x-y}e_{\tilde{y}} = \frac{x}{x-y}\delta_{\tilde{x}} - \frac{y}{x-y}\delta_{\tilde{y}}.$$

  Ao contrário do que acontece com a soma, $\delta_{\tilde{x}}$ e $\delta_{\tilde{y}}$ podem ser pequenos e $\delta_{\tilde{x}-\tilde{y}}$ ser muito grande, concretamente quando x e y são números positivos muito próximos. À perda de precisão daí resultante dá-se o nome de *Cancelamento subtrativo*.

- Para $x \times y \approx \tilde{x} \times \tilde{y}$, tem-se

$$\begin{aligned} e_{\tilde{x}\times\tilde{y}} &= x \times y - \tilde{x} \times \tilde{y} = x \times y - (\tilde{x} - x + x) \times (\tilde{y} - y + y) \\ &= x \times y - (x - e_{\tilde{x}}) \times (y - e_{\tilde{y}}) = y e_{\tilde{x}} + x e_{\tilde{y}} - e_{\tilde{x}} e_{\tilde{y}} \end{aligned}$$

  e

$$\delta_{\tilde{x}\times\tilde{y}} = \frac{e_{\tilde{x}\times\tilde{y}}}{x \times y} = \frac{y}{x \times y} e_{\tilde{x}} + \frac{x}{x \times y} e_{\tilde{y}} - \frac{e_{\tilde{x}} e_{\tilde{y}}}{x \times y} = \delta_{\tilde{x}} + \delta_{\tilde{y}} - \delta_{\tilde{x}} \delta_{\tilde{y}}.$$

  Supondo que $|\delta_{\tilde{x}}|, |\delta_{\tilde{y}}| \ll 1$, podemos tomar a aproximação (linearização dos erros)

$$\delta_{\tilde{x}\times\tilde{y}} \approx \delta_{\tilde{x}} + \delta_{\tilde{y}}.$$

- Na aproximação $\frac{x}{y} \approx \frac{\tilde{x}}{\tilde{y}}$ ocorrem os seguintes erros

$$e_{\frac{\tilde{x}}{\tilde{y}}} = \frac{x}{y} - \frac{\tilde{x}}{\tilde{y}} = \frac{x}{y} - \frac{x - e_{\tilde{x}}}{y - e_{\tilde{y}}} = \frac{y e_{\tilde{x}} - x e_{\tilde{y}}}{y(y - e_{\tilde{y}})}$$

  e

$$\delta_{\frac{\tilde{x}}{\tilde{y}}} = \frac{e_{\frac{\tilde{x}}{\tilde{y}}}}{\frac{x}{y}} = \frac{y e_{\tilde{x}} - x e_{\tilde{y}}}{x(y - e_{\tilde{y}})} = \frac{y}{y - e_{\tilde{y}}} \left(\frac{1}{x} e_{\tilde{x}} - \frac{1}{y} e_{\tilde{y}} \right) = \frac{1}{1 - \delta_{\tilde{y}}} \left(\delta_{\tilde{x}} - \delta_{\tilde{y}} \right).$$

  Supondo que $|\delta_{\tilde{y}}| \ll 1$, é válida a aproximação (linearização dos erros)

$$\delta_{\frac{\tilde{x}}{\tilde{y}}} \approx \delta_{\tilde{x}} - \delta_{\tilde{y}}.$$

3.2 Propagação de erros no cálculo de funções

Começamos pelo caso de uma função real de variável real. Seja $f \in C^2(I)$ onde I é um intervalo (compacto, conexo) que contém x e $\tilde{x}$. Consideremos as aproximações $x \approx \tilde{x}$ e $f(x) \approx f(\tilde{x})$ em $\mathbb{R}$. Da expansão de Taylor

$$f(\tilde{x}) = f(x) + (\tilde{x} - x) f'(x) + (\tilde{x} - x)^2 \frac{f''(x + \theta(\tilde{x} - x))}{2}, \quad 0 < \theta < 1,$$

onde θ depende de x e $\tilde{x}$, obtém-se a seguinte relação para os erros de $\tilde{x}$ e $f(\tilde{x})$

$$e_{f(\tilde{x})} = f'(x) e_{\tilde{x}} - \frac{f''(x + \theta(\tilde{x} - x))}{2} e_{\tilde{x}}^2.$$

Para os erros relativos tem-se

$$\delta_{f(\tilde{x})} = \frac{x f'(x)}{f(x)} \delta_{\tilde{x}} - \frac{x^2 f''(x + \theta(\tilde{x} - x))}{2 f(x)} \delta_{\tilde{x}}^2$$

e supondo que $|\delta_{\tilde{x}}| \ll 1$, podemos tomar a aproximação

$$\delta_{f(\tilde{x})} \approx \frac{xf'(x)}{f(x)}\delta_{\tilde{x}}.$$

Seja agora $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, onde D é convexo. Sejam $x, \tilde{x} \in D$, tais que $x = (x_1, ..., x_n) \approx (\tilde{x}_1, ..., \tilde{x}_n)$, o que leva a tomar a aproximação $f(x) \approx f(\tilde{x})$. Supondo $f \in C^2(D)$, partimos da expansão de Taylor

$$f(\tilde{x}) = f(x) + (\tilde{x} - x) \cdot \nabla f(x) + \frac{1}{2}(\tilde{x} - x)^\top Hf(x + \theta(\tilde{x} - x))(\tilde{x} - x), \quad 0 < \theta < 1,$$

onde Hf é a matriz Hesseana de f, para obter a seguinte relação entre $e_{f(\tilde{x})}$ e $e_{\tilde{x}}$

$$e_{f(\tilde{x})} = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k}(x) e_{\tilde{x}_k} - \frac{1}{2} \sum_{k,l=1}^{n} e_{\tilde{x}_k} e_{\tilde{x}_l} \frac{\partial f^2}{\partial x_k \partial x_l}(x + \theta(\tilde{x} - x)).$$

Introduzindo os erros relativos $\delta_{\tilde{x}_i}$, $i = 1, ..., n$, e desprezando o termo

$$\frac{1}{2} \sum_{k,l=1}^{n} \delta_{\tilde{x}_k} \delta_{\tilde{x}_l} \frac{x_k x_l \frac{\partial^2 f}{\partial x_k \partial x_l}(x + \theta(\tilde{x} - x))}{f(x)},$$

na hipótese de erros relativos pequenos, obtém-se

$$\delta_{f(\tilde{x})} \approx \sum_{k=1}^{n} p_{f,k}(x) \delta_{\tilde{x}_k}, \quad p_{f,k}(x) := \frac{x_k \frac{\partial f}{\partial x_k}(x)}{f(x)}.$$

Definição 3.1. *Os coeficientes $p_{f,1}(x)$,..., $p_{f,n}(x)$ de ponderação dos erros relativos de $\tilde{x}$ chamam-se* números de condição *de f em x.*

3.3 Propagação de erros em algoritmos (executados em sistemas de ponto flutuante)

Ao calcular o valor de uma função num sistema de ponto flutuante, esta é decomposta em funções elementares, de acordo com (3) e (4). Em geral, poderá haver vários algoritmos possíveis para efetuar o cálculo de uma função, correspondentes a várias expressões equivalentes para a definir. Assim, a propagação dos erros dos dados e dos sucessivos erros de arredondamento irá depender do algoritmo implementado.

Vamos estudar um exemplo concreto.

Exemplo 3.1. *Pretende-se calcular* $f(x) = f(x_1, x_2) := \exp(\sin(x_1) + x_2)$. *Um algoritmo possível é*

$$z_1 = \sin(x_1)$$
$$z_2 = z_1 + x_2$$
$$z_3 = \exp(z_2)$$

Considerando a sua execução num sistema de ponto flutuante, devemos ter em atenção que surge um erro de arredondamento em cada passo do algoritmo, de modo que

$$\begin{aligned}
\delta_{\tilde{z}_1} &\approx \frac{x_1 \cos(x_1)}{\sin(x_1)}\delta_{\tilde{x_1}} + \delta_{arr_1} = x_1 \cot(x_1)\delta_{\tilde{x_1}} + \delta_{arr_1},\\
\delta_{\tilde{z}_2} &\approx \frac{z_1}{z_1 + x_2}\delta_{\tilde{z_1}} + \frac{x_2}{z_1 + x_2}\delta_{\tilde{x_2}} + \delta_{arr_2},\\
\delta_{\tilde{z}_3} &\approx z_2\delta_{\tilde{z}_2} + \delta_{arr_3}.
\end{aligned}$$

Com o objetivo de entender a influência dos erros dos dados e dos arredondamentos parciais no resultado final $f_{\mathbb{F}}(\tilde{x})$, *vamos exprimir* $\delta_{f_{\mathbb{F}}(\tilde{x})}$ *em termos de* $\delta_{\tilde{x}_1}$, $\delta_{\tilde{x}_2}$ *e dos erros de arredondamento* δ_{arr_k}, $k = 1, 2, 3$. *Por substituições ascendentes sucessivas, obtemos*

$$\begin{aligned}
& \delta_{f_{\mathbb{F}}(\tilde{x})} = \delta_{\tilde{z}_3}\\
\approx\ & (z_1 + x_2)\left(\frac{z_1}{z_1 + x_2}\delta_{\tilde{z_1}} + \frac{x_2}{z_1 + x_2}\delta_{\tilde{x_2}} + \delta_{arr_2}\right) + \delta_{arr_3}\\
\approx\ & (\sin(x_1) + x_2)\left[\frac{\sin(x_1)}{\sin(x_1) + x_2}\left(\frac{x_1\cos(x_1)}{\sin(x_1)}\delta_{\tilde{x_1}} + \delta_{arr_1}\right) + \frac{x_2}{\sin(x_1) + x_2}\delta_{\tilde{x_2}} + \delta_{arr_2}\right] + \delta_{arr_3}\\
=\ & \sin(x_1)\left(\frac{x_1\cos(x_1)}{\sin(x_1)}\delta_{\tilde{x_1}} + \delta_{arr_1}\right) + x_2\delta_{\tilde{x_2}} + (\sin(x_1) + x_2)\delta_{arr_2} + \delta_{arr_3}\\
=\ & x_1\cos(x_1)\delta_{\tilde{x_1}} + x_2\delta_{\tilde{x_2}} + \sin(x_1)\delta_{arr_1} + (\sin(x_1) + x_2)\delta_{arr_2} + \delta_{arr_3}
\end{aligned}$$

Se notarmos que os números de condição de f *são dados por*

$$p_{f,1}(x) = \frac{x_1\cos(x_1)\exp(\sin(x_1) + x_2)}{\exp(\sin(x_1) + x_2)} = x_1\cos(x_1), \quad p_{f,2}(x) = \frac{x_2\exp(\sin(x_1) + x_2)}{\exp(\sin(x_1) + x_2)} = x_2$$

podemos escrever

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx p_{f,1}(x)\delta_{\tilde{x_1}} + p_{f,2}(x)\delta_{\tilde{x_2}} + \sin(x_1)\delta_{arr_1} + (\sin(x_1) + x_2)\delta_{arr_2} + \delta_{arr_3}.$$

Vamos ainda escrever um algoritmo alternativo para calcular $f(x_1, x_2)$. *Basta notar que* $f(x_1, x_2) = \exp(\sin(x_1))\exp(x_2)$ *e que esta expressão conduz à sequência de cálculos elementares*

$$w_1 = \sin(x_1)$$
$$w_2 = \exp(w_1)$$
$$w_3 = \exp(x_2)$$
$$w_4 = w_2 \times w_3$$

Para este algoritmo tem-se

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx p_{f,1}(x)\delta_{\tilde{x_1}} + p_{f,2}(x)\delta_{\tilde{x_2}} + \sin(x_1)\delta_{arr_1} + \delta_{arr_2} + \delta_{arr_3} + \delta_{arr_4}.$$

Em geral, é válida a fórmula

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx \sum_{k=1}^{n} p_{f,k}(x)\delta_{\tilde{x}_k} + \sum_{k=1}^{m} q_{a,k}(x)\delta_{\mathrm{arr}_k}$$

onde os coeficientes $q_{a,k}(x)$ dependem do algoritmo utilizado para calcular $f_{\mathbb{F}}(\tilde{x})$. Os números de condição $p_{f,k}(x)$ dependem apenas da função f e do input x, sendo independentes de qualquer algoritmo que seja utilizado para calcular $f(x)$.

4 Condicionamento de funções. Estabilidade algorítmica

4.1 Problema bem condicionado

Condicionamento é uma noção que se refere à sensibilidade de um problema matemático a pequenas variações nos seus dados. Aqui, consideramos essencialmente o cálculo de uma função cujos argumentos podem estar afetados de erros ou sofrer perturbações.

Definição 4.1. *Um problema diz-se* bem condicionado *se a pequenos erros relativos nos dados corresponde um pequeno erro relativo no resultado. Caso contrário, o problema diz-se* mal condicionado.

No caso das funções, a análise dos seus números de condição

$$p_{f,k}(x) := \frac{x_k \frac{\partial f}{\partial x_k}(x)}{f(x)}, \quad k = 1, ..., n,$$

permite decidir se o problema é bem ou mal condicionado.

Exemplo 4.1. *1. A soma, a multiplicação e a divisão de números reais positivos x, y são problemas bem condicionados, pois*

$$|p_{f,1}(x)|, |p_{f,2}(x)| \leq 1$$

para $f(x) := x_1 \texttt{op}\ x_2$, se $\texttt{op} \in \{+, \times, /\}$.

2. A subtração $f(x) := x_1 - x_2$ de dois números muito próximos é um problema mal condicionado. Tem-se

$$p_{f,1}(x) = \frac{x_1}{x_1 - x_2}, \quad p_{f,2}(x) = -\frac{x_2}{x_1 - x_2}$$

e

$$\lim_{x_1 - x_2 \to 0} |p_{f,i}(x)| = +\infty, \qquad i = 1, 2.$$

Neste caso, pode ocorrer cancelamento subtrativo.

Exemplo 4.2. *Pretende-se saber para que valores de x_1 e x_2 a função $f(x) := \exp(\sin(x_1) + x_2)$ é bem condicionada.*

Recordamos que os números de condição de f são dados por

$$p_{f,1}(x) = x_1 \cos(x_1), \quad p_{f,2}(x) = x_2$$

e portanto

$$\delta_{f(\tilde{x})} \approx x_1 \cos(x_1)\delta_{\tilde{x_1}} + x_2\delta_{\tilde{x_2}}.$$

Se $|x_1|$ e $|x_2|$ são pequenos (da ordem da unidade) então o problema é bem condicionado. Se $|x_1|$ ou $|x_2|$ são muito grandes então o problema é mal condicionado. Note-se que

$$\lim_{|x_1| \to \infty} |p_{f,1}(x)| = \lim_{|x_2| \to \infty} |p_{f,2}(x)| = +\infty.$$

Em geral, fixando $0 < \delta \ll 1$ tal que $|\delta_{\tilde{x_1}}|, |\delta_{\tilde{x_2}}| \leq \delta$ podemos dizer que se

$$|x_1| \ll \delta^{-1} \text{ e } |x_2| \ll \delta^{-1}$$

o problema é bem condicionado. No entanto, se os números da condição forem da ordem 10^k então pode perder-se até k dígitos de precisão adicional sobre o que seria perdido devido aos erros dos dados.

4.2 Estabilidade de algoritmos

A noção de estabilidade diz respeito à sensibilidade de um algoritmo utilizado para resolver um problema no computador. Mais concretamente:

Definição 4.2. *Um algoritmo diz-se* numericamente estável *se a pequenos erros relativos nos dados (input) e a pequenos valores da unidade de arredondamento do sistema de ponto flutuante corresponde um pequeno erro relativo no resultado calculado (output). Caso contrário, o algoritmo diz-se* numericamente instável.

Recordando a fórmula

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx \sum_{k=1}^{n} p_{f,k}(x)\delta_{\tilde{x}_k} + \sum_{k=1}^{m} q_{a,k}(x)\delta_{\mathrm{arr}_k}$$

conclui-se que a estabilidade de um algoritmo pode ser decidida em termos dos números de condição de f em x e dos coeficientes $q_{a,k}(x)$.

Exemplo 4.3. *O algoritmo apresentado no Exemplo 3.1 para calcular a função* $f(x_1, x_2) := \exp(\sin(x_1) + x_2)$ *é estável quando* $|x_1|$ *e* $|x_2|$ *são pequenos. Para valores de* x_1 *e* x_2 *tais que* $|x_1|$ *ou* $|x_2|$ *sejam muito grandes, o problema torna-se mal condicionado e não é possível encontrar um algoritmo estável.*

Se um problema for bem condicionado, existe um algoritmo estável para o resolver.

4.3 Aplicação

Voltamos agora ao Exemplo 1.3 da Introdução, pois estamos em condições de explicar os resultados obtidos no MATLAB com base numa análise dos algoritmos que foram utilizados.

Exemplo 4.4. *Consideremos a função*

$$f(x) := \frac{1}{x} - \frac{1}{x+1} = \frac{1}{x(x+1)} \quad (x \in \mathbb{R} \setminus \{-1, 0\}).$$

Foram efetuados os seguintes cálculos no MATLAB *(sistema IEEE-754 com precisão dupla):*

```
>> format long

>> 1/10^20-1/(10^20+1)
ans = 0

>> 1/((10^20+1)*10^20)
ans = 9.999999999999999e-41
```

Note-se que o resultado `ans = 0` *caiu na zona de* Underflow *do sistema. É claro que* $f(10^{20}) \neq 0$*, pelo que* `ans = 0` *tem erro de 100%. Também é fácil ver que devemos esperar* $f(10^{20}) \approx 10^{-40}$*.*

O **Problema** *em causa é*

$$\textit{Calcular } f(x) = \frac{1}{x} - \frac{1}{x+1} \textit{ quando } x \textit{ é muito grande.}$$

- *Observamos desde já que o problema é bem condicionado. De facto,*

$$p_f(x) = \frac{xf'(x)}{f(x)} = \frac{x \times \left(-\frac{2x+1}{x^2(x+1)^2}\right)}{\frac{1}{x(x+1)}} = -\frac{2x+1}{x+1} \tag{5}$$

e

$$\lim_{x\to\infty} p_f(x) = -2 \Rightarrow p_f(x) \approx -2, \textit{ quando } x \textit{ é muito grande.}$$

Da relação

$$\delta_{f(\tilde{x})} \approx p_f(x)\delta_{\tilde{x}} \approx -2\delta_{\tilde{x}} \textit{ para } x \textit{ muito grande,}$$

conclui-se que o cálculo de $f(x)$ *para tais valores de* x *é* bem condicionado*. Isto significa que a* função *não é muito sensível a pequenas variações no seu argumento, quando este toma valores grandes.*

- *Quanto à estabilidade algorítmica, começamos pelo estudo do* **algoritmo 1***, associado à expressão* $f(x) = \frac{1}{x} - \frac{1}{x+1}$*, com passos*

$$\begin{aligned} z_1 &= 1/x \\ z_2 &= x+1 \\ z_3 &= 1/z_2 \\ z_4 &= z_1 - z_3. \end{aligned}$$

Considerando a execução num sistema de ponto flutuante, tem-se

$$\begin{aligned} \delta_{\tilde{z}_1} &\approx -\delta_{\tilde{x}} + \delta_{arr_1}, \\ \delta_{\tilde{z}_2} &\approx \frac{x}{x+1}\delta_{\tilde{x}} + \delta_{arr_2}, \\ \delta_{\tilde{z}_3} &\approx -\delta_{\tilde{z}_2} + \delta_{arr_3}, \\ \delta_{\tilde{z}_4} &\approx \frac{z_1}{z_1 - z_3}\delta_{\tilde{z}_1} - \frac{z_3}{z_1 - z_3}\delta_{\tilde{z}_3} + \delta_{arr_4}. \end{aligned}$$

Por substituições sucessivas, obtemos

$$\begin{aligned}
\delta_{\tilde{z}_4} &\approx \frac{1/x}{1/x-1/z_2}\delta_{\tilde{z}_1} - \frac{1/z_2}{1/x-1/z_2}\delta_{\tilde{z}_3} + \delta_{arr_4} \\
&\approx \frac{1/x}{1/x-1/z_2}\left(-\delta_{\tilde{x}} + \delta_{arr_1}\right) - \frac{1/z_2}{1/x-1/z_2}\left(-\delta_{\tilde{z}_2} + \delta_{arr_3}\right) + \delta_{arr_4} \\
&= \frac{1/x}{1/x-1/(x+1)}\left(-\delta_{\tilde{x}} + \delta_{arr_1}\right) - \frac{1/(x+1)}{1/x-1/(x+1)}\left(-\delta_{\tilde{z}_2} + \delta_{arr_3}\right) + \delta_{arr_4} \\
&\approx \frac{1/x}{1/x-1/(x+1)}\left(-\delta_{\tilde{x}} + \delta_{arr_1}\right) - \frac{1/(x+1)}{1/x-1/(x+1)}\left(-\frac{x}{x+1}\delta_{\tilde{x}} - \delta_{arr_2} + \delta_{arr_3}\right) + \delta_{arr_4},
\end{aligned}$$

onde podemos fazer as simplificações

$$\frac{1/x}{1/x-1/(x+1)} = \frac{1/x}{1/(x(x+1))} = \frac{1}{1/(x+1)} = x+1$$

e

$$\frac{1/(x+1)}{1/x-1/(x+1)} = x.$$

Assim, o **algoritmo 1**, *quando executado num sistema de ponto flutuante, produz a seguinte propagação de erros*

$$\begin{aligned}
\delta_{f_{\mathbb{F}}(\tilde{x})} &\approx (x+1)\left(-\delta_{\tilde{x}} + \delta_{arr_1}\right) - x\left(-\frac{x}{x+1}\delta_{\tilde{x}} - \delta_{arr_2} + \delta_{arr_3}\right) + \delta_{arr_4} \\
&= p_f(x)\delta_{\tilde{x}} + (x+1)\delta_{arr_1} + x\delta_{arr_2} - x\delta_{arr_3} + \delta_{arr_4},
\end{aligned}$$

onde usámos (5) *na última igualdade. No que se refere aos erros de arredondamento* δ_{arr_k}, *assumimos que* $|\delta_{arr_k}| \leq \epsilon_M$. *No* MATLAB *tem-se* $\epsilon_M \approx 10^{-15}$. *Este algoritmo é* <u>instável</u> *para* x *grande devido aos termos* $(x+1)\delta_{arr_1}$ *e* $x\delta_{arr_2}$ *que constituem grandes ampliações dos erros de arredondamento. No último passo do algoritmo ocorre cancelamento subtrativo quando* x *é muito grande.*

Para $x = 10^{20}$, *que tem representação exata no* MATLAB, *tem-se*

$$\begin{aligned}
\tilde{z}_1 &= 10^{-20} \\
\tilde{z}_2 &= \mathrm{fl}(10^{20}+1) = 10^{20} \\
\tilde{z}_3 &= 1/\tilde{z}_2 = 10^{-20} \\
\tilde{z}_4 &= \tilde{z}_1 - \tilde{z}_3 = 0
\end{aligned}$$

pelo que

$$\delta_{\tilde{x}} = 0, \quad \delta_{arr_1} = 0, \quad \delta_{arr_2} = 1/10^{20}, \quad \delta_{arr_3} = \delta_{arr_4} = 0$$

e portanto

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx x\delta_{arr_2} = 1.$$

Isto está de acordo com o erro de 100% que já tínhamos identificado no início.

• *A execução do* **algoritmo 2**, *o qual resulta da implementação da expressão* $f(x) = \frac{1}{x(x+1)}$,

$$\begin{aligned} w_1 &= x + 1 \\ w_2 &= x \times w_1 \\ w_3 &= 1/w_2 \end{aligned}$$

produz a seguinte propagação de erros quando executado num sistema de ponto flutuante $\mathbb{F}$*:*

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx p_f(x)\delta_{\tilde{x}} - \delta_{arr_1} - \delta_{arr_2} + \delta_{arr_3}, \quad |\delta_{arr_k}| \leq \epsilon_M.$$

O algoritmo 2 é <u>estável</u> *para* x *muito grande.*

• *Podemos ainda considerar o* **algoritmo 3**

$$\begin{aligned} y_1 &= x^2 \\ y_2 &= x + y_1 \\ y_3 &= 1/y_2 \end{aligned}$$

implementação da expressão $f(x) = \frac{1}{x+x^2}$, *o qual produz a seguinte propagação de erros:*

$$\delta_{f_{\mathbb{F}}(\tilde{x})} \approx p_f(x)\delta_{\tilde{x}} - \frac{x}{x+1}\delta_{arr_1} - \delta_{arr_2} + \delta_{arr_3}, \quad |\delta_{arr_k}| \leq \epsilon_M.$$

Este algoritmo também é <u>estável</u> *para* x *muito grande.*

Em resumo:

1. As noções de *bem condicionado/mal condicionado* referem-se ao problema, mais concretamente à sensibilidade do problema a pequenas variações nos seus dados. *Estável/instável* são noções referentes a um algoritmo, ou seja, ao processo e sistema de cálculo numérico.

2. A propagação de erros de arredondamento num algoritmo instável pode produzir resultados numéricos desastrosos.

3. Podem surgir maus resultados com a execução de um algoritmo no computador porque:

(a) O problema é mal condicionado para certos valores de input;

(b) O algoritmo implementado é numericamente instável. O cancelamento subtrativo num passo do algoritmo pode causar instabilidade numérica.

Note-se que muitas funções se comportam qualitativamente de maneira diferente quando o seu argumento é 0 e quando é $\varepsilon \neq 0$ muito próximo de 0. Basta pensar no que acontece ao dividir por 0 em vez de ε por muito pequeno que este seja.

Referências

[1] C. Alves, Fundamentos de Análise Numérica, 2001.

[2] D. Goldberg, What every computer scientist should know about floating-point arithmetic, *ACM Comput. Surv.*(1991), **23**, 1, 5–48.

[3] J. M. Ortega, Numerical Analysis: a second course, Classics in Applied Mathematics; Vol. 3, SIAM, 1990.

[4] C. Moler, Numerical Computing with Matlab, SIAM, 2004.

[5] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres, Handbook of Floating-Point Arithmetic, Handbook of Floating-Point Arithmetic, Birkhauser, 2010.

[6] M. L. Overton, Numerical computing with IEEE floating point arithmetic, SIAM, 2001.

[7] A. Quarteroni, R. Sacco e F. Saleri, Cálculo Científico com Matlab e Octave, Springer-Verlag, 2007 (traduzido por Adélia Sequeira).

[8] F. Romeiras, Matemática Computacional, Apontamentos das aulas, 2008.