Notas breves de Teoria de Erros

Filomena Teodoro e João Soares

1/3/2004

1 Introdução

1.1 Sistemas de representação numérica

Em muitas áreas tradicionais da Matemática como, por exemplo, a Análise, não é absolutamente necessário recorrer a uma descrição numérica das entidades que são manipuladas. Por exemplo, é perfeitamente legítimo escrever $x = \sqrt{2}$ ou $y = 2\pi\sqrt{5}$ e aceitar esta descrição destes números. Mas, quando pretendemos concretizar cálculos, de modo a obtermos respostas a um problema concreto, estas representações não são adequadas, pois o cálculo em questão é efectuado em máquinas que podem apenas manipular um número finito de símbolos e realizar um número finito de operações.

Podemos então definir formalmente a base numérica b, onde $b \in \mathbb{N}_{\geq 2}$, (porque é que o caso $b = 1$ é excluído?) como sendo um conjunto de símbolos $D_b = \{d_0, ..., d_{b-1}\}$, tais que, dado $x \in \mathbb{R}$, temos a representação:

$$x = \pm \left(\sum_{n=-\infty}^{N} d_{k_n} b^n \right), \; d_{k_n} \in D_b$$

onde N é o menor inteiro tal que $10^{N+1} \geq x$. Os elementos do conjunto D_b dizem-se os *dígitos* da base b. Se escrevermos a expressão acima na forma

$$x = \pm \left(\sum_{n=0}^{N} d_{k_n} b^n + \sum_{n=1}^{\infty} d_{k_n} b^{-n} \right) \tag{1}$$

o primeiro e segundo termos são denominados, respectivamente, a *parte inteira* e a *parte fraccionária* de x.

Exemplos:

(i) $b = 10$ (base decimal). Temos $D_b = \{0, 1, ..., 9\}$ e, por exemplo, as raízes de $x^2 - 100$, são representadas nesta base por $x = 10$ e $x = -10$.

(ii) $b = 2$ (base binária). Temos $D_b = \{0, 1\}$ e, neste caso $x = 8$ (na base 10) é representado por $x = 1000$.

Estes exemplos mostram que é necessário ter cuidado quando são utilizadas simultaneamente bases distintas, por isso, é necessário adoptar uma

notação que torne explícita a base na qual um determinado número está a ser representado. Assim, no exemplo (2) acima escreve-se $x = (8)_{10}$ e $x = (1000)_2$.

É necessário também definir algoritmos que permitam converter a representação de x na base b_1, $(x)_{b_1}$, calcular a representação correspondente noutra base b_2.

1.2 Conversão entre bases

Dado então $x \in \mathbb{R}$ e uma base b_1, separa-se x nas partes inteira e fraccionária, tal como foram definidas acima, em (1):

$$x = (d_N ... d_0)_{b_1} + (d_{-1} ... d_{-n} ...)_{b_1}$$

Note-se que a parte inteira de x tem, necessariamente, um número finito de dígitos, mas isto não tem que ocorrer com a parte fraccionária (porquê?). O método de conversão desta representação de x para outra base b_2 é genérico mas, de momento, vamos restringir-nos ao caso particular $b_1 = 10$ e $b_2 = 2$.

i) Parte inteira. Seja $x = 1026.4356$, então $(d_N ... d_0)_{10} = (1026)_{10}$, e dividindo sucessivamente por 2, obtemos:

$$\begin{aligned}
1026 &= 513 \times 2 + \mathbf{0} \\
513 &= 256 \times 2 + \mathbf{1} \\
256 &= 128 \times 2 + \mathbf{0} \\
128 &= 64 \times 2 + \mathbf{0} \\
64 &= 32 \times 2 + \mathbf{0} \\
32 &= 16 \times 2 + \mathbf{0} \\
16 &= 8 \times 2 + \mathbf{0} \\
8 &= 4 \times 2 + \mathbf{0} \\
4 &= 2 \times 2 + \mathbf{0} \\
2 &= \mathbf{1} \times 2 + \mathbf{0}
\end{aligned}$$

Note-se que, pelas propriedades da divisão inteira, os restos só podem ser 0 ou 1. Mais, afirmamos que estes restos são exactamente os dígitos da representação da parte inteira de x na base 2:

$$(1026)_{10} = (10000000010)_2$$

Para verificar esta afirmação, basta escrever:

$$\begin{aligned}(10000000010)_2 &= \sum_{n=0}^{9} b_n 2^n = \\ &= \mathbf{1} \times 2^{10} + \mathbf{0} \times 2^8 + \cdots + \mathbf{1} \times 2^1 + \mathbf{0} = \\ &= (1026)_{10}\end{aligned}$$

e notar que o resto da primeira divisão é simplesmente o último 0 da representação binária. O resto da segunda divisão é o 1 que vem a seguir, pois a primeira divisão anulou o factor multiplicativo 2^1 e por aí adiante.

(ii) Relativamente à parte fraccionária, o processo é diferente. Primeiro, notemos que esta é sempre menor que 1, logo, multiplicando sucessivamente por 2 e conservando apenas a parte fraccionária do resultado, obtemos um número cuja parte inteira é 0 ou 1. Continuando com o exemplo acima:

$$\begin{aligned}&0.4356 \times 2 = \mathbf{0}.8712 \\ &0.8712 \times 2 = \mathbf{1}.7424 \\ &0.7424 \times 2 = \mathbf{1}.4848 \\ &0.4848 \times 2 = \mathbf{0}.9696 \\ &0.9696 \times 2 = \mathbf{1}.9392 \\ &...\end{aligned}$$

A sequência de $\{0, 1\}$ gerada pela parte inteira dos resultados destas multiplicações sucessivas é exactamente a representação, em base 2, da parte fraccionária do nosso número original.

O resultado final é então:

$$(1026.4356)_{10} = (10000000010.01101...)_2$$

Note-se que, na base 2, a parte fraccionária tem um número infinito de dígitos não-nulos, o que não acontece com a representação em base 10.

Para verificarmos que este algoritmo funciona, basta ver que a representação da parte fraccionária em base 2 é:

$$\sum_{n=1}^{\infty} \frac{d_n}{2^n} = \frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + ...$$

e que a primeira multiplicação por 2 isola o dígito d_1, a segunda o dígito d_2, etc.

Quanto à conversão da base 2 para a base 10, podemos utilizar directamente a definição, pois a representação em qualquer base é escrita na forma de um somatório ou série e podemos calcular (pelo menos nalguns casos) a respectiva soma. Seja $x = (1001.1111)_2$, então temos:

$$x = 2^3 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} = (9.9375)_{10}$$

Finalmente, notemos que estes algoritmos são válidos para a conversão entre quaisquer bases, pois dependem apenas de propriedades elementares da multiplicação e divisão inteira. Por isto, não vamos fazer aqui a demonstração no caso geral, mas apenas dar alguns exemplos:

Exemplos:
Conversão para a base decimal.
(a) Base 2 → Base 10:

$$\begin{aligned}(101101)_2 &= \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = \\ &= (45)_{10}\end{aligned}$$

(b) Base 8 → Base 10:

$$\begin{aligned}(427)_8 &= \mathbf{4} \times 8^2 + \mathbf{2} \times 8^1 + \mathbf{7} \times 8^0 = \\ &= (279)_{10}\end{aligned}$$

(c) Base 16 → Base 10[1]:

$$\begin{aligned}(1A0F)_{16} &= \mathbf{1} \times 16^3 + \mathbf{10} \times 16^2 + \mathbf{0} \times 16^1 + \mathbf{15} \times 16^0 = \\ &= (6671)_{10}\end{aligned}$$

Neste exemplo, vemos a razão de ser da notação $(...)_b$, que temos vindo a utilizar. Por exemplo, o conjunto de símbolos 427, sem mais nenhuma indicação, pode ser, em função da base, a representação de vários números distintos.

[1] A base 16, dita hexadecimal, tem 15 dígitos. Os dez primeiros são representados pelos dígitos decimais $\{0, ..., 9\}$ e os restantes seis, pelas letras maiúsculas $\{A, B, C, D, E, F\}$.

Note-se também que, neste exemplo, apenas estão conversões da parte inteira. Se os números envolvidos tivessem parte fraccionária diferente de zero, esta teria que ser tratada separadamente, mas o processo é idêntico:

Exemplo:
Converter $(0.467)_8$ para a base decimal.
Temos:

$$\begin{aligned}(0.467)_8 &= \mathbf{4}\times 8^{-1}+\mathbf{6}\times 8^{-2}+\mathbf{7}\times 8^{-3}= \\ &= (0.607421875)_{10}\end{aligned}$$

O problema inverso, ou seja, a conversão de um número em base 10 para outra base b, é resolvido, no caso da parte inteira, por divisões sucessivas por b e, para a parte fraccionária, por multiplicações sucessivas por b. Isto é uma consequência das expressões de representação gerais, que foram enunciadas acima, e da unicidade dos restos na divisão inteira. Novamente, ilustramos o processo através de exemplos.

Exemplo:
Converter $(176)_{10}$ para as bases 2, 8 e 16.
(a):

$$\begin{aligned}176 &= 88\times 2+\mathbf{0}\\ 88 &= 44\times 2+\mathbf{0}\\ 44 &= 22\times 2+\mathbf{0}\\ 22 &= 11\times 2+\mathbf{0}\\ 11 &= 5\times 2+\mathbf{1}\\ 5 &= 2\times 2+\mathbf{1}\\ 2 &= 2\times \mathbf{1}+\mathbf{0}\end{aligned}$$

Logo $(176)_{10}=(10110000)_2$.
(b):

$$\begin{aligned}176 &= 22\times 8+\mathbf{0}\\ 22 &= \mathbf{2}\times 8+\mathbf{6}\end{aligned}$$

Logo $(176)_{10}=(260)_8$.
(c):

$$176 = \mathbf{11}\times 16+\mathbf{0}$$

Logo $(176)_{10}=(B0)_{16}$.

Para números com parte fraccionária não nula, temos:

Exemplo:
Converter $(0.625)_{10}$ para a base 8. Temos:

$$0.625 \times 8 = \mathbf{5}.0$$

Logo $(0.625)_{10} = (0.5)_8$.
Converter $(0.1)_{10}$ para a base 2:

$$\begin{array}{l} 0.1 \times 2 = \mathbf{0}.2 \\ 0.2 \times 2 = \mathbf{0}.4 \\ 0.4 \times 2 = \mathbf{0}.8 \\ 0.8 \times 2 = \mathbf{1}.6 \\ 0.6 \times 2 = \mathbf{1}.2 \\ 0.2 \times 2 = \mathbf{0}.4 \\ ... \end{array}$$

Logo $(0.1)_{10} = (0.00011\,(0011))_2$.

Dada uma base b, será a representação de um número x relativamente a esta base, $(x)_b$, única? Em geral, não. Mas os casos em que a unicidade falha podem ser caracterizados muito precisamente.

Dada então uma base b, seja $d_b = b-1$ o maior dígito dessa base e consideremos os números que, relativamente a b, têm parte fraccionária $(0.d_b d_b d_b ...)_b$. Então, temos:

$$\begin{aligned} (0.d_b d_b d_b ...)_b &= \sum_{n=1}^{\infty} d_b b^{-n} = \\ &= d_b \left(\sum_{n=1}^{\infty} b^{-n} \right) = d_b \left(\frac{1}{b-1} \right) = \\ &= 1 = (1)_b \end{aligned}$$

Ou seja, $(0.d_b d_b d_b ...)_b = (1)_b$. Então, em vez de escrevermos, por exemplo, $(3.99...)_{10}$, optamos pela representação finita $(4)_{10}$. (Porque é que a representação de 1 é a mesma em qualquer base?)

1.3 Representação em vírgula flutuante

Na secção anterior, foi demonstrado que qualquer número $x \in \mathbb{R}$ admite uma representação em termos dos dígitos de uma base b. Mas, para a implementação de métodos computacionais, é necessário ir mais longe, dado que a representação de cada número tem que ser finita. Isto significa que existirão números que não podem ser representados exactamente no sistema adoptado. Por exemplo:

$$\sqrt{2} = 1.41421356237...$$

é um número irracional, o que significa que a sua representação exacta em qualquer base inteira exige um número infinito de dígitos. Dado que, na prática, só podemos trabalhar com um número finito de dígitos, é necessário recorrer a uma aproximação de $\sqrt{2}$.

Os sistemas de representação numérica computacional mais utilizados no presente, dizem-se de *vírgula flutuante* e são definidos, para $b \geq 2$ e $|e| \leq 10^m - 1$:

$$FP(b, n, m, *) = \{x \in \mathbb{R} : x = \pm (0.d_1...d_n)_b \times b^e,\ d_i \in D_b,\ d_1 \geq 1\} \quad (2)$$

isto é, o sistema $FP(b, n, m, *)$ é o conjunto de todos os números da forma:

$$x = \pm (0.d_1...d_n)_b \times b^e \quad (3)$$

onde o parâmetro n indica o número de dígitos da parte fraccionária, ou *mantissa*, de (3), m é o número máximo de dígitos do expoente e e $*$ indica o tipo de arredondamento utilizado, que será definido mais adiante. A restrição $d_1 \geq 1$ implica que o primeiro dígito imediatamente à direita do ponto decimal não pode ser zero. Os sistemas numéricos que obedecem a esta restrição, e que serão os considerados nestas notas, dizem-se *normalizados*.

Um ponto importante relativamente a (2) é que este conjunto é *finito*. Além disso, se $x \in FP(b, n, m, *)$, $x \neq 0$, então

$$b^{-1-e_m} \leq |x| \leq \left(1 - b^{-n}\right) b^{e_m} \quad (4)$$

onde $e_m = 10^m - 1$ é o maior expoente atingível com m dígitos[2].

Fixado um sistema $FP(b, n, m, *)$, e dado $x \in \mathbb{R}$, $x \neq 0$, podem ocorrer três situações distintas:

[2]Por convenção, o expoente é sempre representado em base 10.

i) $x \in FP(b, n, m, *)$.

ii) $|x| > (1 - b^{-n})\, b^{e_m}$ (*overflow*) ou $|x| > b^{-1-e_m}$ (*underflow*). Neste caso, x está fora dos limites atingíveis do sistema e é necessário implementar procedimentos correctivos para estas situações.

iii) $|x|$ satisfaz (4), mas $x \not\in FP(b, n, m, *)$. Neste caso, é necessário escolher $\hat{x} \in FP(b, n, m, *)$ de modo a que $\hat{x}$ seja uma boa aproximação de x. Esta escolha é feita através da função

$$\mathrm{fl} : \mathbb{R} \to FP(b, n, m, *)$$

cuja definição precisa depende do processo de aproximação escolhido.

Dado que a escolha de uma função fl implica a existência de um erro sempre que $x \not\in FP(b, n, m, *)$, é necessário definir mais precisamente as medidas de erro que vão ser utilizadas.

1.4 Erros absolutos e relativos

Dado x, os erros absoluto e relativo da aproximação $\hat{x}$ de x, são definidos por:

(i) $\epsilon_x = \hat{x} - x$ é o erro absoluto de $\hat{x}$.

(ii) $\delta_x = \dfrac{\epsilon_x}{x} = \dfrac{\hat{x} - x}{x}$, $x \neq 0$. é o erro relativo de $\hat{x}$.

Uma definição alternativa do erro relativo é

$$\hat{x} = x\,(1 + \delta_x) \tag{5}$$

As diferenças entre estas duas medidas de erro são ilustradas no exemplo seguinte:

Sejam $x = 0.6\,(6)$, $y = 0.006(6)$ e $\hat{x} = 0.666$, $\hat{y} = 0.006$ aproximações por truncatura a três casas decimais. Os erros absolutos são:

$$\epsilon_x = \epsilon_y = -0.0006\,(6)$$

enquanto que os erros relativos são:

$$\begin{aligned} \delta_x &= -0.001 \\ \delta_y &= -0.1 \end{aligned}$$

e este resultado mostra que, enquanto o erro absoluto é insensível à magnitude da quantidade, o mesmo não se passa com o erro relativo.

Uma medida alternativa da qualidade de uma aproximação $\hat{x}$, definida em através destas medidas de erro, é o número de *dígitos significativos*, que são definidos da seguinte forma: se $|\delta_x|$ satisfaz

$$|\delta_x| \leq \frac{1}{2} b^{-n+1} \tag{6}$$

para algum $n \geq 0$, então $\hat{x}$ é uma aproximação de x com n algarismos significativos.

É conveniente escrever esta definição em termos do erro absoluto ϵ_x. Para isso, note-se que existe $m \geq 0$ tal que $b^m \leq x < b^{m+1}$e, sendo $|\epsilon_x| = |x|\,|\delta_x|$, obtem-se:

$$|\epsilon_x| = |x|\,|\delta_x| \leq \frac{1}{2} b^{m+1-n} \tag{7}$$

Uma justificação para as expressões (6), (7) acima é dada nos exemplos seguintes:

Exemplos:

i) Seja $\hat{x} = 0.25$, uma aproximação de x. Sabendo que $|\epsilon_x| \leq 0.5 \times 10^{-2} = 0.005$, será esta aproximação de x "fiável"? Se os dois dígitos de $\hat{x}$ à direita do ponto decimal coincidirem com os dígitos correspondentes de x, então $|\epsilon_x|$ terá forçosamente que verificar:

$$|\epsilon_x| < 0.009(9) = 0.01$$

então $|\epsilon_x| \leq 0.005 < 0.01$ e, por (7) $\hat{x}$ é uma representação de x com 2 os dois dígitos significativos.

ii) Seja $\hat{x} = 29.2$ e $|\epsilon_x| \leq 0.5$. Temos $t = 2$ e $n = 2$, logo $\hat{x}$ tem dois algarismos significativos, que coincidem com os dois dígitos à esquerda do ponto decimal.

iii) Seja $\hat{x} = 0.005894$ e $|\epsilon_x| \leq 0.5 \times 10^{-6}$. Temos $t = -2$ e $n = 4$, logo $\hat{x}$ tem 4 algarismos significativos que, neste caso, são 5894. Note que, por (7), os zeros imediatamente à direita do ponto decimal não são algarismos significativos.

1.5 Estimativas de erro em sistemas de vírgula flutuante

A função de aproximação fl mais simples é denominada *truncatura* ou *corte* a n casas decimais: sempre que um número não possa ser representado exactamente no sistema, desprezam-se os dígitos à direita da posição n. Por exemplo, seja $x = 10.256$ e[3] $FP(10, 4, 2, T)$. Para representar exactamente este número, a mantissa seria 0.10256, o que é impossivel, pois $n = 4$. A truncatura de 010256 é 0.1025, logo a representação de 10.256 em $FP(10, 4, 2, T)$ é:

$$\hat{x} = \text{fl}(x) = 0.1025 \times 10^2$$

Apesar da simplicidade da truncatura, a aproximação mais utilizada é o chamado *arredondamento simétrico*, normalmente denotado por $FP(b, n, m, A)$. Aqui, determina-se $\hat{x} \in FP(b, n, m, A)$ que minimiza $|\epsilon_x|$. Considerando novamente o exemplo $x = 0.10256 \times 10^2$, existem dois candidatos possíveis $\hat{x}_1, \hat{x}_2 \in FP(10, 4, 2, T)$:

$$\begin{aligned} \hat{x}_1 &= 0.1025 \times 10^2 \\ \hat{x}_2 &= 0.1026 \times 10^2 \end{aligned}$$

com os erros $|\epsilon_{x_1}| = 0.006$ e $|\epsilon_{x_2}| = 0.004$, logo $\hat{x} = \text{fl}(x) = 0.1026 \times 10^2$. Note-se que $\hat{x}_1$ coincide com a aproximação obtida por truncatura, e que o erro absoluto associado é mais elevado. Em geral, o erro introduzido pela truncatura é dado, no pior caso, por:

$$|\epsilon_x| = 0.\underbrace{0...0}_{n}9\,(9) \times 10^e = 0.\underbrace{0...0}_{n-1}1 \times 10^e = 10^{e-n}$$

Note-se também que, em alguns casos, o arredondamento simétrico não determina univocamente a aproximação $\text{fl}(x)$, pois é possível que existam dois elementos de $FP(b, n, m, A)$ equidistantes de x. Um caso típico é o seguinte: determinar a representação de 0.75 em $FP(10, 1, 1, A)$.

Neste caso, os números $0.7, 0.8 \in FP(10, 1, 1, A)$ têm ambos o erro de aproximação de 0.05, logo, qualquer um deles seria uma representação aceitável de 0.75 e, para decidir a questão, temos de introduzir regras adicionais. Uma das mais utilizadas é a regra *Round-to-even*, que opta pela

[3] O T na definição do sistema é uma abreviatura de truncatura.

aproximação que deixa o último dígito da mantissa par[4]. Neste caso, teríamos $\mathrm{fl}(0.75) = 0.8$.

No caso geral, os erros absolutos introduzidos pela truncatura e arredondamento são majorados por:

$$\begin{aligned} \left|\epsilon_x^T\right| &\leq b^{e-n} \\ \left|\epsilon_x^A\right| &\leq \frac{1}{2} b^{e-n} \end{aligned}$$

E, para o erro relativo:

$$\begin{aligned} \left|\delta_x^T\right| &\leq b^{-n+1} \\ \left|\delta_x^A\right| &\leq \frac{1}{2} b^{-n+1} \end{aligned}$$

O lado direito destas duas últimas desigualdades é uma característica que mede a qualidade do sistema de vírgula flutuante adoptado, é denominada *unidade de arredondamento* e designada por u. Da definição (5) de δ_x, obtemos um modelo para fl:

$$\mathrm{fl}(x) = (1 + \delta_x)\, x, \quad |\delta_x| \leq u$$

1.6 Operações aritméticas em vírgula flutuante

Para um dado sistema de representação numérica, temos agora que definir as operações aritméticas elementares neste sistema. Qualquer que seja a definição final, estas operações têm que ser compatíveis com as operações aritméticas conhecidas, em particular, para números que são exactamente representáveis no nosso sistema, estas têm que coincidir com as operações aritméticas clássicas.

O problema é que, por exemplo, dados dois números $x, y \in FP(b, n, m, *)$, pode acontecer que $x + y \notin FP(b, n, m, *)$. E esta afirmação é válida para as outras operações aritmeticas.

A solução deste problema envolve uma aproximação (normalmente, a mesma aproximação que é feita para a representação), logo, introduz mais erros, que têm de ser contabilizados para termos uma ideia de quão fiáveis são estas operações.

[4]Esta regra fundamenta-se na distribuição estatística dos erros de arredondamento, e tende a evitar a acumulação de desvios sistemáticos sempre que um cálculo é repetido muitas vezes.

A definição geral é a seguinte: dada uma operação aritmética $\circledast \in \{+, -, \times, /\}$ e $x, y \in FP(b, n, m, *)$, definimos:

$$\text{fl}(x) \circledast \text{fl}(y) = \text{fl}(x \circledast y)$$

Ou seja, dados dois números pertencentes ao sistema, primeiro calculamos o resultado da operação aritmética com os algoritmos habituais. Em geral, o resultado não está normalizado, nem vai pertencer ao sistema, por isso, normalizamos e voltamos a aplicar o processo de aproximação.

Exemplo:
Calcular, em $F(10, 4, 2, T)$, $x + y$, para $x = 1.256879$ e $y = 0.985441$.
Temos $\hat{x} = 1.256$ e $\hat{y} = 0.9854$, logo $z = \hat{x} + \hat{y} = 0.1256 + 0.9854 = 1.111$. Então $\hat{z} = 0.1111$.

2 Propagação de erros. Fórmula fundamental do cálculo de erros.

2.1 Fórmula fundamental do cálculo de erros:

Na secção anterior vimos a necessidade de, nos processos de cálculo, controlar os efeitos dos erros existentes nos valores numéricos (aproximados) que são utilizados na prática.

Por exemplo, na subtracção de dois números aproximadamente iguais, erros pequenos podem ser amplificados no resultado final, levando à perda de dígitos significativos.

Exemplo: sejam $\hat{x} = 67.5796$ e $\hat{y} = 67.5722$, ambos com seis dígitos significativos. Então $\hat{x} - \hat{y} = 0.0074$, que tem apenas *dois* dígitos significativos. Este é um exemplo típico do fenómeno de cancelamento subtractivo, que ocorre na subtracção de dois números com valores muito próximos.

Um problema importante no Cálculo Numérico é a quantificação de fenómenos deste tipo. Qual é o factor de amplificação de erros de aproximação, para uma dada operação? Será que depende da forma como a operação é implementada, ou seja, do algoritmo escolhido?. Qual é a contribuição para o erro no resultado que provém da própria operação? Neste capítulo, vai ser

dada uma introdução às técnicas e resultados que nos permitem responder a este tipo de questões.

Até que seja dito o contrário, todas as funções são contínuas, com derivadas contínuas e definidas num intervalo $I \subset \mathbb{R}^n$ limitado e fechado.

Seja então $f(x)$, definida em $[a,b]$. Pretende-se calcular $y = f(x)$, conhecendo apenas um valor aproximado, $\tilde{x}$, de x. O teorema de Lagrange[5] garante a existência de um ponto $c \in]a,b[$ tal que:

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

Escrevendo $a = x$ e $b = x + h$, temos

$$f(x+h) = f(x) + f'(c)\, h$$

Se x fôr o valor exacto da quantidade e $\tilde{x} = x + \epsilon_x$ o seu valor aproximado, a expressão acima pode escrever-se na forma

$$\epsilon_{f(x)} = f'(c)\, \epsilon_x$$

Na maior parte dos casos práticos, não conhecemos o ponto c, por isso calculamos apenas um limite superior para o valor absoluto do erro:

$$\left|\epsilon_{f(x)}\right| \leq |f'|_M \, |\epsilon_x|$$

onde $|f'|_M = \sup_{y \in [x, x+h]} |f'(y)|$.

Usualmente, f depende de várias variáveis, $x_1, ..., x_n$. A generalização da expressão para n variáveis é

$$|\epsilon_f| \leq \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}\right|_M |\epsilon_{x_i}|$$

onde os números $\left|\frac{\partial f}{\partial x_i}\right|_M$ são majorações do módulo das derivadas parciais de f numa vizinhança de $(x_1, ..., x_n)$. Antes de passarmos a um exemplo concreto, notemos que esta expressão fornece um limite superior para o erro absoluto de y, na forma de uma soma ponderada dos erros cometidos em cada uma das variáveis x_i. Esta expressão vai ser utilizada daqui para a frente,

[5] É assumido que f satisfaz todas as condições do teorema.

sempre que queiramos estimar o erro (absoluto) cometido no cálculo de uma função. Pela sua importância, é normalmente chamada *fórmula fundamental do cálculo de erros (absolutos).*

Exemplo: calcular a precisão com que é medida a área de um círculo de raio $\tilde{r} = 1.25\text{m}$, sendo π aproximado por $\tilde{\pi} = 3.14$. Todos os dígitos são significativos.

Temos: $|\epsilon_r| \leq 0.005\text{m}$, $|\epsilon_\pi| \leq 0.002$ (porquê?) e $\tilde{A} = \tilde{\pi}\tilde{r}^2 = 4.90625\text{m}^2$.

Sendo $\dfrac{\partial A}{\partial r} = 2\pi r$ e $\dfrac{\partial A}{\partial \pi} = r^2$, vem:

$$\begin{aligned} |\epsilon_A| &\leq \left|\frac{\partial A}{\partial r}\right| |\epsilon_r| + \left|\frac{\partial A}{\partial \pi}\right| |\epsilon_\pi| = \\ &= |2\pi r|_M |\epsilon_r| + \left|r^2\right|_M |\epsilon_\pi| \end{aligned}$$

Substituíndo valores (atenção aos valores utilizados nos factores de peso), temos:

$$|\epsilon_A| \leq |2 \times 3.142 \times 1.255|\, 0.005 + \left|1.255^2\right| 0.002 = 0.04258215\text{m}^2$$

De forma mais compacta, $|\epsilon_A| \leq 0.05\text{m}^2$, ou seja, podemos garantir dois dígitos significativos neste cálculo da área.

Exemplo: calcular a área de uma coroa circular, definida por duas circunferências concêntricas de raios a e b, respectivamente. Sabemos que $\tilde{a} = 5.0\text{cm}$, com todos os dígitos significativos, $b \in [2.0, 2.1]\text{cm}$ e o valor aproximado de π é o mesmo do exemplo anterior. Temos:

$$\begin{aligned} \tilde{A} &= \tilde{\pi}\left(\tilde{a}^2 - \tilde{b}^2\right) = \\ &= 3.14 \times \left(5.0^2 - 2.05^2\right) = \\ &= 65.30915\text{cm}^2 \end{aligned}$$

A estimativa do erro e_A é dada por:

$$\begin{aligned} |\epsilon_A| &\leq \left|\frac{\partial A}{\partial a}\right| |\epsilon_a| + \left|\frac{\partial A}{\partial b}\right| |\epsilon_b| + \left|\frac{\partial A}{\partial \pi}\right| |\epsilon_\pi| = \\ &= |-2\pi b|_M |\epsilon_a| + |2\pi a|_M |\epsilon_b| + \left|a^2 - b^2\right|_M |\epsilon_\pi| \end{aligned}$$

Temos $|\epsilon_a| \leq 0.05\text{cm}$ e $|\epsilon_\pi| \leq 0.0016$. Quanto à estimativa para $|\epsilon_b|$, note que $b \in [2.0, 2.1]$, logo os piores casos são $b + |\epsilon_b| = 2.1$ e $b - |\epsilon_b| = 2.0$, donde tiramos $\tilde{b} = 2.05$ e $|\epsilon_b| = 0.05$. Introduzindo estes valores na expressão acima, obtemos $|\epsilon_b| \leq 2.3\text{cm}^2$.

2.2 Algumas aplicações

Nesta secção, vamos calcular as estimativas do erro para as operações aritméticas elementares.

(1) Soma: $y = x_1 + x_2$. Temos:

$$|\epsilon_{x_1+x_2}| \leq |\epsilon_{x_1}| + |\epsilon_{x_2}|$$

pois[6] $\dfrac{\partial y}{\partial x_i} = 1$, $i = 1, 2$

(2) Subtracção: $y = x_1 - x_2$. Temos:

$$|\epsilon_{x_1-x_2}| \leq |\epsilon_{x_1}| + |\epsilon_{x_2}|$$

(Porquê? Compare com a adição.)

(3) Produto: $y = x_1 \times x_2$. Temos:

$$|\epsilon_{x_1\times x_2}| \leq \max\left\{|x_1|, |x_2|\right\}\left(|\epsilon_{x_1}| + |\epsilon_{x_2}|\right)$$

(4) Divisão: $y = \dfrac{x_1}{x_2}$. Temos:

$$\left|\epsilon_{\frac{x_1}{x_2}}\right| \leq \max\left\{\left|\frac{1}{x_2}\right|, \left|\frac{x_1}{x_2^2}\right|\right\}\left(|\epsilon_{x_1}| + |\epsilon_{x_2}|\right)$$

Note que a propagação dos erros para a soma e a subtracção são dadas pelas mesmas expressões. Isto parece contradizer o exemplo dado no início deste capítulo, onde falámos do fenómeno de cancelamento subtractivo. Todavia, é preciso não esqueçer que, por enquanto, estamos apenas a falar de erros *absolutos*. Mais adiante, veremos que estes erros transportam apenas parte da informação necessária. Logo, não existe aqui nenhuma contradição. Como motivação para o uso desta medida de erro, considere o exemplo seguinte:

Exemplo:
$\tilde{x} = 2112.9$, com $|\epsilon_x| \leq 0.1$. Podemos garantir que $x \in [2112.8, 2113]$.
Da mesma forma, dado $\tilde{y} = 5.3$ com $|\epsilon_y| \leq 0.1$, temos $y \in [5.2, 5.4]$.

Os majorantes do erro (absoluto) de x e y são iguais. Nesta situação, o que é que podemos dizer acerca da precisão com que estes números estão representados?

[6] Se $y = \sum_{i=1}^{n} x_i$ temos, por indução, $|e_y| \leq \sum_{i=1}^{n} |e_{x_i}|$.

Sem entrar em conta com a ordem de grandeza de cada um dos números, não podemos responder a esta questão. É neste sentido que dizemos que o erro absoluto não transporta toda a informação.

Neste exemplo concreto, é intuitivo dizer que a precisão de x é maior que a de y, mas como é que podemos precisar esta intuição? Para isso, é necessário recorrer aos erros relativos.

2.3 Propagação de erros relativos:

O erro relativo é definido por[7] $\delta_x = \frac{\epsilon_x}{x} \simeq \frac{\epsilon_x}{\tilde{x}}$. Voltando ao exemplo da secção anterior, temos:

$$|\delta_x| \leq \frac{|\epsilon_x|}{|\tilde{x}|} = \frac{0.1}{2112.9} = 4.7 \times 10^{-5}$$
$$|\delta_y| \leq \frac{|\epsilon_y|}{|\tilde{y}|} = \frac{0.1}{5.3} = 0.02$$

Estes resultados tornam clara (e precisa) a diferença de precisão nos dois casos.

Considere agora uma função $y = f(x_1, ..., x_n)$, a fórmula fundamental dos erros, deduzida atrás, é:

$$|\epsilon_y| \leq \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right|_M |\epsilon_{x_i}|$$

Temos $\delta_y = \frac{\epsilon_y}{y} = \frac{\epsilon_y}{f(x_1, ..., x_n)}$ e $\delta_{x_i} = \frac{\epsilon_{x_i}}{x_i}$. Substituíndo, obtemos a expressão:

$$|\delta_y| \leq \sum_{i=1}^{n} \left| \frac{x_i \frac{\partial f}{\partial x_i}}{f(x_1, ..., x_n)} \right|_M |\delta_{x_i}| = \sum_{i=1}^{n} |p_i| \, |\delta_{x_i}|$$

Esta expressão é análoga à fórmula fundamental para erros absolutos e podemos justificadamente chamá-la de fórmula fundamental para erros *rela-*

[7] Esta aproximação é normalmente utilizada para estimar o erro relativo, pois não temos conhecimento do valor exacto de x.

tivos. Os factores de peso p_i

$$p_i = \frac{x_i \dfrac{\partial f}{\partial x_i}}{f(x_1, ..., x_n)}$$

são denominados *números de condição* (relativamente ás variáveis x_i) e medem as contribuições dos erros relativos de cada uma destas variáveis no resultado final.

Voltemos ás operações aritméticas:

(i) Multiplicação: $f(x, y) = xy$. Os números de condição são $p_x = \dfrac{x\frac{\partial f}{\partial x}}{xy} = \dfrac{xy}{xy} = 1$ e $p_y = 1$. Então:

$$|\delta_{xy}| \leq |\delta_x| + |\delta_y|$$

(ii) Divisão: $f(x, y) = \dfrac{x}{y}$. Temos $p_x = \dfrac{x\frac{\partial f}{\partial x}}{x/y} = 1$ e $p_y = -1$. Então:

$$\left|\delta_{x|y}\right| \leq |\delta_x| + |\delta_y|$$

(iii) Soma e subtracção: $f(x, y) = x \pm y$. Temos $p_x = \dfrac{x}{x \pm y}$ e $p_y = -\dfrac{y}{x \pm y}$. Então:

$$|\delta_{x\pm y}| \leq \max\left\{\left|\frac{x}{x \pm y}\right|, \left|\frac{y}{x \pm y}\right|\right\}(|\delta_x| + |\delta_y|)$$

Note-se que, apesar da semelhança destas expressões, o comportamento em termos de erros relativos da subtracção é distinto do da soma. Se x e y forem aproximadamente iguais, então os números de condicionamento da subtracção podem ser muito elevados. Esta é uma das manifestações do fenómeno de cancelamento subtractivo, de que já falámos antes. Como ilustração, temos o seguinte exemplo:

Exemplo:

Determinar o número de algarismos significativos do resultado de cada uma das operações xy, x/y, $x + y$ e $x - y$, sabendo que $\tilde{x} = 1010$ e $\tilde{y} = 1000$ (todos os algarismos são significativos).

Temos $|\epsilon_x| < 0.5$ e $|\epsilon_y| < 0.5$, logo $|\delta_x| = \dfrac{|\epsilon_x|}{x} \simeq \dfrac{|\epsilon_x|}{\tilde{x}} \leq 0.495 \times 10^{-3}$ e $|\delta_y| \leq 0.5 \times 10^{-3}$. Utilizando as expressões deduzidas acima:

$$|\delta_{xy}| \leq |\delta_x| + |\delta_y| = 0.995 \times 10^{-3}$$
$$\left|\delta_{x/y}\right| \leq |\delta_x| + |\delta_y| = 0.995 \times 10^{-3}$$

Dado que $\tilde{x}\tilde{y} = 1.010 \times 10^6$ e $\tilde{x}/\tilde{y} = 1.010$, temos $|\epsilon_{xy}| = |\delta_{xy}|\,|xy| \simeq |\delta_{xy}|\,|\tilde{x}\tilde{y}| \leq 1.005 \times 10^3$. Da mesma maneira, obtemos $\left|\epsilon_{x/y}\right| = \left|\delta_{x/y}\right|\,|x/y| \simeq \left|\delta_{x/y}\right|\,|\tilde{x}/\tilde{y}| \leq 1.005 \times 10^{-3}$. Então a multiplicação e divisão têm, pelo menos, 3 dígitos significativos.

Para a soma e subtracção, temos:

$$|\epsilon_{x\pm y}| \leq |\epsilon_x| + |\epsilon_y| \leq 0.5 + 0.5 = 1$$

Sendo os resultados aproximados $\tilde{x}+\tilde{y} = 2010$ (3 algarismos significativos) e $\tilde{x} - \tilde{y} = 10$ (1 algarismo significativo).

Vamos agora determinar os números de condição de algumas funções comuns, de modo desenvolver uma ideia melhor da informação dada por este parâmetro.

(1) $f(x) = x^n$, $n \in [0, +\infty[$:

$$p_x = \frac{nx\,(x)^{n-1}}{x^n} = n$$

Assim, se $f(x) = \sqrt{x}$ $(n = 1/2)$, temos $p_x = 1/2$, enquanto que se $f(x) = x^2$, temos $p_x = 2$. Podemos dizer, informalmente, que, quanto maior fôr o expoente n, maior será a amplificação dos erros relativos.

(2) $f(x) = \cos x$. Temos:

$$p_x = -\frac{x \sin x}{\cos x} = x \operatorname{tg} x$$

(3) $f(x) = \sin x$. Temos:

$$p_x = \frac{x \cos x}{\sin x} = x \operatorname{cotg} x$$

No caso (2), podemos dizer que o número de condicionamento é baixo nos extremos e elevado nos zeros da função $\cos x$. Para o caso (3), podemos fazer uma interpretação semelhante.

Antes de calcularmos o número de condicionamento para outras funç ões de interesse prático, vamos introduzir alguns termos utilizados para qualificar o comportamento numérico de uma função relativamente ao número de condicionamento.

Se $|p_x(x_0)| \gg 1$, dizemos que a função é mal condicionada em x_0. Na vizinhança de x_0, podemos esperar que pequenos erros em x sejam bastante amplificados.

Se $|p_x(x_0)| \ll 1$, dizemos que a função é bem condicionada em x_0. Na vizinhança de x_0, podemos esperar que erros em x não se propaguem.

Assim, a funções do tipo $f(x) = x^n$ são bem condicionadas para $n's$ pequenos (tipicamente 1,2), mas o seu condicionamento piora para valores de n elevados. Notemos também que o número de condicionamento é independente do ponto x. Isto não é verdade para a generalidade das funções, como podemos ver nos casos seguintes:

(4) $f(x) = \exp x$. Temos:

$$p_x = \frac{x \exp x}{\exp x} = x$$

(5) $f(x) = \log x$. Temos:

$$p_x = \frac{x\frac{1}{x}}{\log x} = \frac{1}{\log x}$$

Nestes dois casos, vemos que o condicionamento de $\exp x$ piora à medida que $x \to +\infty$ e que, apesar de ser bem condicionada para valores grandes de x, a função $\log x$ é muito mal condicionada para $x = 1$.

Intuitivamente, podemos pensar que o condicionamento está relacionado com o comportamento local da função, quanto mais elevada (em valor absoluto) fôr a sua derivada, maior será o efeito dos erros nas variáveis (este é o significado da fórmula fundamental para erros absolutos). Mas esta interpretação não pode ser levada à letra, pois pense-se no caso da função $\log x$ acima. Numa vizinhança suficientemente pequena de $x = 1$, a função não parece mal comportada, mas o seu condicionamento é muito elevado. A razão disto é que $\log 1 = 0$ e o cálculo deste valor particular leva à perda de um grande número de dígitos significativos.

2.4 Efeito dos erros de arredondamento:

Na análise de erros feita na secção anterior, não entrámos em conta com os erros de arredondamento. Estes ocorrem a cada passo do algoritmo uti-

lizado para o cálculo do problema, pois estamos a trabalhar dentro de um sistema $FP(b, n, m, *)$ fixo e, como já vimos anteriormente, a maior parte dos números não pertencem a estes sistemas e temos que recorrer a aproximações, que têm de ser feitas a cada passo do algoritmo.

No caso de operações elementares, temos simplesmente:

$$|\delta f_{total}| \leq |p_x|\,|\delta_x| + |\delta_{arr}|$$

O primeiro termo já foi descrito na secção anterior, enquanto que o segundo depende do sistema particular de representação adoptado.

Exemplo:

Determinar um majorante para o erro absoluto de $\tilde{y} = \cos(\tilde{x})$, quando $\tilde{x} = 1.5$, com $\delta_x = 0.001$ e o cálculo é efectuado no sistema $FP(10, 4, 2, T)$.

Temos $\cos(\tilde{x}) = \mathrm{fl}(0.07073720) = 0.7073 \times 10^{-1}$ e:

$$|\delta_{\cos}| \leq (-x \operatorname{tg} x)_M\,|\delta_x| \simeq |-\tilde{x} \operatorname{tg} \tilde{x}|\,|\delta_x|$$

Logo:

$$|\delta_{total}| \leq |\delta_{\cos}| + |\delta_{arr}|$$

Dado que $|\delta_{arr}| \leq 10^{-4}$, temos:

$$\begin{array}{l} |\delta_{total}| \leq |\delta_{\cos}| + |\delta_{arr}| \simeq \\ \simeq |-1.5 \operatorname{tg} 1.5|\,|0.001| + 10^{-4} \simeq \\ \simeq 2.1252 \times 10^{-2} \end{array}$$

e $|e_y| = |\delta_{total}|\,|y| \simeq |\delta_{total}|\,|\tilde{y}| \simeq 1.5033 \times 10^{-3}$.

A decomposição do erro total nestas duas componentes distintas, leva-nos a definir os conceitos, extremamente importantes na prática, de condicionamento e estabilidade.

O primeiro, que já foi discutido atrás, depende essencialmente da estrutura da função ou modelo matemático com que pretendemos fazer os cálculos. Nomeadamente, vimos exemplos de funções bem e mal condicionadas, e vimos também que podemos estimar a propagação de erros conhecendo apenas as derivadas parciais da função numa vizinhança dos valores das variáveis. Tudo isto é pré-algorítmico e dizemos que um problema numérico é estável, se pequenas variações (erros) nos valores das variáveis não são amplificados no resultado final. Na linguagem que temos estado a utilizar, isto significa

que nenhum dos pesos que afectam cada um dos factores δ_{x_i} (os números de condição) pode ser muito elevado.

Quanto ao segundo, quando é que dizemos que um método é numericamente estável? Para tornar precisa esta noção, temos que ter em conta as outras fontes de erro, nomeadamente os erros de arredondamento introduzidos em cada passo do algoritmo implementado. Neste caso, não podemos esperar ter à nossa disposição uma fórmula fundamental de erros *genérica*, mas ainda assim, podemos escrever:

$$|\delta_f| \leq \sum_{i=1}^{n} |p_i| \, |\delta_{x_i}| + \sum_{i=1}^{k} A_i \, |\delta_i^{arr}|$$

Onde os pesos A_i têm de ser estimados caso-a-caso, através da análise do algoritmo em questão. Então, um método numérico diz-se *estável*, se nenhum dos coeficientes $|p_i|$ ou A_i é elevado. Este é um conceito mais forte que o bom condicionamento, onde apenas exigimos que nenhum dos coeficientes $|p_i|$ seja elevado. Então, um problema pode ser bem condicionado, mas o algoritmo que o resolve instável. Por outro lado, um problema mal condicionado não é *sempre* instável.

Como introdução, vejamos o seguinte exemplo:

Exemplo:

Seja $f(x,y) = x^2 - y^2 = (x-y)(x+y)$. Usamos apenas operações elementares para calcular f, mas as duas expressões, algebricamente equivalentes, que definem esta função, sugerem duas maneiras distintas de aplicar estas operações, ou seja, dão a dois algoritmos distintos.

Algoritmo A:

$$\begin{aligned} z_1 &= x \times x \\ z_2 &= y \times y \\ z_3 &= z_1 - z_2 \end{aligned}$$

Algoritmo B:

$$\begin{aligned} z_1 &= x + y \\ z_2 &= x - y \\ z_3 &= z_1 \times z_2 \end{aligned}$$

Dados $x = 0.3237$, $y = 0.3134$ e supondo que utilizamos o sistema $FP(10,4,2,A)$, vamos comparar o desempenho numérico destes dois algoritmos. O resultado exacto é $f(x,y) = 0.656213 \times 10^{-3}$.

Temos:

$$\begin{array}{ll} \tilde{z}_1 = \mathrm{fl}\,(x \times x) = 0.1048 & \tilde{z}_1 = \mathrm{fl}\,(x + y) = 0.6371 \\ \tilde{z}_2 = \mathrm{fl}\,(y \times y) = 0.9822 \times 10^{-1} & \tilde{z}_2 = \mathrm{fl}\,(x - y) = 0.1030 \times 10^{-1} \\ \tilde{z}_3 = \mathrm{fl}\,(\tilde{z}_1 - \tilde{z}_2) = 0.6580 \times 10^{-2} & \tilde{z}_3 = \mathrm{fl}\,(\tilde{z}_1 \times \tilde{z}_2) = 0.6562 \times 10^{-2} \end{array}$$

Destes resultados, podemos então calcular $\delta f^A_{total} = -0.27232 \times 10^{-2}$ e $\delta f^B_{total} = 0.00198 \times 10^{-2}$, ou seja, o erro relativo introduzido no resultado final pelo algoritmo A é superior ao introduzido pelo algoritmo B. Para detectarmos a fonte desta diferença, temos que analisar os erros introduzidos a cada passo, em cada um dos algoritmos.

Para o algoritmo A, temos:

$$\begin{aligned} |\delta_{z_1}| &\leq |\delta_x| + |\delta_x| + |\delta_{arr1}| \\ |\delta_{z_2}| &\leq |\delta_y| + |\delta_y| + |\delta_{arr2}| \\ |\delta_{z_3}| &\leq \left|\tfrac{z_1}{z_3}\right|_M |\delta_{z_1}| + \left|\tfrac{z_2}{z_3}\right|_M |\delta_{z_2}| + |\delta_{arr3}| \end{aligned}$$

Logo, substituindo as duas primeiras expressões na terceira:

$$\begin{aligned} |\delta_{z_3}| \leq & \left|\tfrac{2x^2}{x^2-y^2}\right|_M |\delta_x| + \left|\tfrac{2y^2}{x^2-y^2}\right|_M |\delta_y| + \\ & + \left|\tfrac{x^2}{x^2-y^2}\right|_M |\delta_{arr1}| + \left|\tfrac{y^2}{x^2-y^2}\right|_M |\delta_{arr2}| + |\delta_{arr3}| \end{aligned}$$

Para o algoritmo B, temos:

$$\begin{aligned} |\delta_{z_1}| &\leq \left|\tfrac{x}{x+y}\right|_M |\delta_x| + \left|\tfrac{y}{x+y}\right|_M |\delta_y| + |\delta_{arr1}| \\ |\delta_{z_2}| &\leq \left|\tfrac{x}{x-y}\right|_M |\delta_x| + \left|\tfrac{y}{x-y}\right|_M |\delta_y| + |\delta_{arr2}| \\ |\delta_{z_3}| &\leq |\delta_{z_1}| + |\delta_{z_2}| + |\delta_{arr3}| \end{aligned}$$

E substituindo, temos:

$$\begin{aligned} |\delta_{z_3}| \leq & \left|\tfrac{2x^2}{x^2-y^2}\right|_M |\delta_x| + \left|\tfrac{2y^2}{x^2-y^2}\right|_M |\delta_y| + \\ & + |\delta_{arr1}| + |\delta_{arr2}| + |\delta_{arr3}| \end{aligned}$$

Notemos que a parte relativa ao condicionamento do problema, $\left|\tfrac{2x^2}{x^2-y^2}\right|_M |\delta_x| + \left|\tfrac{2y^2}{x^2-y^2}\right|_M |\delta_y|$, é a mesma para os dois algoritmos (como seria de esperar), mas os termos respeitantes aos erros de arredondamento diferem. Em particular, no algoritmo A, são amplificados pelos factores $\left|\tfrac{x^2}{x^2-y^2}\right|_M$ e $\left|\tfrac{y^2}{x^2-y^2}\right|_M$. Para valores de x e y próximos, isto justifica a diferença de desempenho dos dois algoritmos.

3 Bibliografia:

[1] Accuracy and stability of numerical algorithms; Higham, N. J.; SIAM; 2002

[2] Numerical Analysis - A mathematical introduction; Schatzam, M.; Oxford University Press; 2002

[3] Métodos Numéricos; Valença, M.; Imprensa Nacional; 1988