3 Aritmética Computacional

3.1 Introdução

Quando se utiliza um qualquer instrumento de trabalho para realizar uma tarefa deve-se ter um conhecimento profundo do seu modo de funcionamento, das suas capacidades e das suas limitações.

Sempre que uma calculadora ou um computador digital é utilizado para efectuar um cálculo numérico quase sempre ocorre um erro, o chamado erro de arredondamento. Este erro é inevitável porque a aritmética utilizada pela máquina envolve somente números que pertencem a um subconjunto F, que é finito, discreto e limitado, dos números reais. Este subconjunto é usado para representar todos os números reais. Por conseguinte existem reais que, não sendo exactamente representáveis, têm de ser aproximados por outros reais que pertencem a este subconjunto.

Os erros de arredondamento podem ter efeitos colaterais importantes. Normalmente, os erros de arredondamento provocam efeitos menos nefastos do que os chamados “erros colaterais” das guerras modernas, todavia as suas consequências podem ser suficientemente sérias, pelo que não podem ser ignoradas. Em [30] encontram-se descritos quatro casos de efeitos colaterais

calamitosos que se tornaram mundialmente famosos e que tiveram origem em erros de arredondamento:

1. Em 4 de Junho de 1996 o foguetão Ariane 5 caiu após 36 segundos de voo. A queda ficou a dever-se a um erro de programação: ao converter um número fraccionário, representado no computador de bordo com 64 dígitos binários (bits), para um número inteiro representado com 16 bits, o sistema de voo do foguetão, controlado por computador, entrou em colapso. Os prejuízos foram estimados em várias centenas de milhões de euros.

2. Em 25 de Fevereiro de 1991, durante a guerra do Golfo, um anti-míssel Patriot lançado pelas tropas aliadas falhou a intercepção dum míssil vindo do lado de Saddam Hussein. Morreram 28 pessoas. O problema resultou da acumulação sucessiva de erros de arredondamento no cálculo do tempo necessário para a intercepção do míssil invasor.

3. Em 1982 a Bolsa de Valores de Vancouver instituiu um novo índice, inicializado com o valor nominal de 1 000 000. O índice era recalculado e actualizado no final de cada transacção. Após 22 meses, o índice caiu para 524 881. A causa dessa desvalorização resultou de se terem efectuado truncaturas em cada transacção registada, em lugar de arredondamentos. O valor arredondado correcto daria 1 098 892.

4. Na Alemanha, um partido com menos de 5% de votos não elege nenhum deputado. Na *land* Schleswig-Holstein, num certo período eleitoral, um determinado partido foi dado como tendo obtido 5% dos votos, elegendo assim um deputado. Depois de anunciados os resultados, veio a verificar-se que esse partido na realidade apenas

> tinha obtido uma percentagem de 4.97%. O deputado eleito deixou de o ser. O fracasso ficou a dever-se ao facto do resultado da votação ter sido arredondado para 2 dígitos. Reposta a legalidade, o maior partido da região acabou por ter a maioria absoluta no Parlamento com a vantagem de um deputado.

Como refere M. Graça ([30]),

> *claro que nenhuma tragédia equivalente às descritas acima deverá ocorrer em resultado do uso de máquinas de calcular pelos alunos. No entanto, para se evitar que os alunos utilizem incorrectamente as máquinas de calcular, os professores deverão alertá-los para as suas limitações.*

É pois indispensável compreender a noção de número, os vários tipos de números, as diferentes formas de representação (sobretudo as que são usadas nas calculadoras e computadores), as operações admissíveis, os erros cometidos e os seus efeitos nos resultados ([50] pp.1-33). Por último será abordado o sistema de ponto flutuante das calculadoras gráficas.

3.2 Representação de números inteiros

Todos nós estamos familiarizados com a representação de inteiros no sistema decimal. A representação de um número inteiro na base decimal consiste numa sequência de algarismos, em que cada um possui um valor que depende da respectiva posição na representação. Assim por exemplo,

$$234 = 2\times10^2 + 3\times10^1 + 4\times10^0 .$$

Trata-se portanto de uma representação posicional.

Utilizando o símbolo d_i para denotar o algarismo ou dígito decimal colocado na posição i a contar da direita, um inteiro *N* com $n+1$ dígitos possui a seguinte representação decimal:

$$N = *(d_n d_{n-1} ... d_1 d_0) = *(d_n \times 10^n + d_{n-1} \times 10^{n-1} + ... + d_1 \times 10^1 + d_0 \times 10^0) \quad \textbf{(3.1)}$$

com $* \in \{+,-\}$ e $0 \le d_i \le 9$, $i = 0,1,...,n$ e $d_n \neq 0$.

Generalizando esta ideia a uma base b diferente de 10, em que $b \ge 2$ e inteiro, vem que um número inteiro $N \neq 0$ terá uma representação na forma:

$$N = *(d_n d_{n-1} ... d_1 d_0)_b = *(d_n \times b^n + d_{n-1} \times b^{n-1} + ... + d_1 \times b^1 + d_0 \times b^0) \quad \textbf{(3.2)}$$

com $* \in \{+,-\}$ e $0 \le d_i < b$, $i = 0,1,...,n$ e $d_n \neq 0$.

Assim, fixada a base, qualquer inteiro ficará completamente definido pelo **sinal** e pela **sequência de dígitos** $d_n,...,d_0$. A representação de um número natural N numa base b é única, isto é, se $N = \sum_{i=0}^{n} d_i b^i = \sum_{i=0}^{m} d'_i b^i$ então $n = m$ e $d_i = d'_i$ para $i = 0,...,n$.

O sistema binário adquiriu uma importância especial com o advento dos computadores digitais. Estes utilizam, para armazenar informação, dispositivos físicos que podem assumir de modo estável dois estados distintos. De facto, um interruptor pode estar ligado ou desligado, uma lâmpada pode estar acesa ou apagada, uma corrente eléctrica pode magnetizar um núcleo num sentido ou noutro, etc.. Os computadores têm impulsos enviados pelas suas componentes electrónicas. O estado de impulso é ON ou OFF. Se identificarmos esses estados com os dígitos 0 e 1, obtemos de imediato uma correspondência entre os estados do computador e os números representados na forma binária. É usual designar por "bit" (*binary digit*) o elemento de memória básico que assume os dois estados que se associam aos dígitos 0 e 1.

Se usarmos qualquer outra base para representar os números, então cada um dos dígitos que representam o número nessa base terá de ser codificado na forma binária. Se a base b for uma potência de 2, essa codificação é muito simples. Por exemplo, para $b = 8 = 2^3$, vão ser precisos 3 bits para representar cada um dos números 0, 1, ..., 7:

Representação decimal	Representação binária	Representação decimal	Representação binária
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Tabela 3.1

Com *k* bits obtemos 2^k configurações. Deste modo, a representação dos dez dígitos decimais requer 4 bits[4], pois $2^3 = 8$ e $2^4 = 16$, ou seja, 3 bits são insuficientes, já que apenas permitem 8 configurações, e 4 bits permitem 16 configurações, o que é demais. Este facto significa que a representação decimal desperdiça bits e é, por conseguinte, menos económica do que uma base que seja potência de 2. Aliada a esta situação, acresce ainda o facto de que a aritmética decimal é de implementação difícil em computador. Por todas estas razões, a base 10 é muito pouco utilizada na representação de números em computador, excepto quando se torna absolutamente necessário que a representação e a aritmética sejam integralmente decimais.

Em síntese, fixada a base, um número ficará totalmente determinado pelo seu sinal e pela sequência de dígitos. Se representarmos o sinal + por 0 e o sinal – por 1, o número aparece-nos apenas como uma sequência de dígitos.

Como todas as máquinas, o computador é uma ferramenta com capacidade finita. Por conseguinte, o número total de bits que o computador utiliza na representação de números é necessariamente *finito*. Assim, existe apenas um número finito de inteiros exactamente representáveis num computador.

[4] Para representar um número com *n* decimais são necessários $(\log_2 10)n \approx 3.3n$ bits.

3.3 Representação de números reais

Na base decimal a notação 123.75 é interpretada da seguinte forma:

$$123.75 = 123 + .75$$

onde $123 = 1\times10^2 + 2\times10^1 + 3\times10^0$ e $.75 = 7\times10^{-1} + 5\times10^{-2}$. A expansão binária equivalente é dada por

$$123.75 = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-2} = (1111011.11)_2 .$$

A expansão deste número nas duas bases é finita. No entanto, esta situação nem sempre ocorre. Por exemplo, o número $1/10$, que tem obviamente uma representação decimal finita $(0.1)_{10}$, não possui uma representação binária finita. De facto,

$$\frac{1}{10} = (0.0001100110011...)_2 = \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \frac{1}{256} + \frac{1}{512} + \frac{0}{1024} + ...$$

Note-se que esta representação apesar de não ser finita, repete-se. A fracção $1/3$, por exemplo, não possui uma expansão finita, binária ou decimal. Todos os números racionais admitem, em qualquer base, uma expansão finita ou uma expansão que se repete (no caso da base decimal, todo o racional pode ser escrito através de uma dízima finita ou infinita periódica). Por outro lado, os números irracionais admitem sempre expansões que não se repetem. Por exemplo[5],

$$\sqrt{2} = (1.41421356237...)_{10} = (1.0110101000001000011...)_2$$
$$\pi = (3.14159265358979...)_{10} = (11.00100100001111111...)_2$$
$$e = (2.71828182845905...)_{10} = (10.10110111111000010...)_2$$

De um modo geral, podemos dizer que a representação de um número real x na base 10 é:

[5] Expansões obtidas através do software *Mathematica*.

$$x = (d_n d_{n-1} \ldots d_0 . d_{-1} d_{-2} \ldots d_{-k} \ldots)_{10} = \\ = d_n \times 10^n + d_{n-1} \times 10^{n-1} + \ldots + d_0 + d_{-1} \times 10^{-1} + \ldots + d_{-k} \times 10^{-k} + \ldots \quad \textbf{(3.3)}$$

Analogamente, para um número real x na base b temos:

$$x = (d_n d_{n-1} \ldots d_0 . d_{-1} d_{-2} \ldots d_{-k} \ldots)_b = \\ = d_n \times b^n + d_{n-1} \times b^{n-1} + \ldots + d_0 + d_{-1} \times b^{-1} + \ldots + d_{-k} \times b^{-k} \ldots \quad \textbf{(3.4)}$$

Os dígitos $(d_n d_{n-1} \ldots d_0)_b$ constituem a parte inteira e os dígitos $(d_{-1} d_{-2} \ldots d_{-k} \ldots)_b$, a parte fraccionária da representação do número real x na base b.

3.3.1 Notação científica de números reais

Em determinadas aplicações científicas há necessidade de recorrer a números "muito grandes" e a números "muito pequenos". A forma de ultrapassar as dificuldades inerentes à representação destes números, é utilizando a chamada **notação científica (ou exponencial)**. Esta notação consiste em exprimir um número real x na forma

$$x = \pm m \times b^e \quad \textbf{(3.5)}$$

em que m é um número real não negativo designado por **mantissa**, $b \geq 2$ é um inteiro positivo designado por **base** e e é um número inteiro designado por **expoente**. Fixada a base b, esta representação não é única. Assim para ultrapassar esta ambiguidade, é usual optar por uma mantissa que satisfaça a seguinte convenção:

$$\begin{cases} m = 0 & se \quad x = 0 \\ 1 \leq m < b & se \quad x \neq 0 \end{cases} . \quad \textbf{(3.6)}$$

Nesta situação diz-se que se trata de uma representação **normalizada**. De notar que neste tipo de representação, o primeiro dígito da mantissa de um

número diferente de zero é sempre diferente de zero. Por exemplo, a representação normalizada em base 10 de 1975.05 é 1.97505×10^{3} e a do número 0.000197505 é 1.97505×10^{-4}. É sempre possível satisfazer a condição $1\le m<10$, uma vez que m pode ser obtido de x através de multiplicações ou divisões sucessivas por 10, adequando o respectivo expoente e.

A adopção desta convenção não elimina, no entanto, todas as ambiguidades. Assim, para o número zero continuam a ser possíveis infinitas representações, todas com mantissa $m=0$ e expoente e arbitrário. Uma outra situação em que a representação também não é única é o caso de números cuja mantissa tem infinitos dígitos repetindo-se periodicamente. Por exemplo, 9.999999... representa, pela expressão (3.3) e pela fórmula da soma das progressões geométricas, o número:

$$x=9.9999...=\sum_{k=0}^{\infty}9\times10^{-k}=9\times\frac{1}{1-0.1}=10\ .$$

Assim, consideraremos 9.9999... e 1.0×10^{1} como duas representações do mesmo número.

3.3.2 Representação em sistema de ponto (ou vírgula) flutuante

É claro que a notação científica apresentada anteriormente não pode ser implementada em computador, pois para abranger todos os números reais, a mantissa e o expoente exigiriam um número infinito de dígitos. Por conseguinte, a notação científica é alterada no sentido de apenas se utilizar um número finito p de dígitos para a mantissa e um número finito q de dígitos para o expoente, obtendo-se assim a chamada representação em **ponto flutuante**, que se baseia na notação científica.

Neste estudo usaremos a notação $F(b,p,q)$, para denominar o **sistema de representação em ponto flutuante de base b**, cuja mantissa "ocupa" p dígitos (base *b*) e cujo expoente pode utilizar no máximo q dígitos (base *b*);

fixados b, p e q, denotaremos simplesmente por F o conjunto dos números que têm representação exacta neste sistema. Assim,

$$\text{se} \quad x \in F \quad \text{então} \quad x = \pm m \times b^e, \qquad \textbf{(3.7)}$$

onde $1 \leq m < b$, se o número for normalizado e e é um número inteiro.

A partir da definição de F, é imediato que todos os seus números são apenas alguns dos racionais. Assim sendo tem-se que F é um subconjunto próprio de *IR*. Este subconjunto F é **finito** por construção, uma vez que apenas um número finito de números racionais podem ser representados exactamente. Como consequência imediata tem-se que F, ao contrário de IR, é **discreto** e **limitado**.

Para a representação de números no computador, já foi referido que é preferível utilizar a base binária em vez da base decimal. Assim, os números normalizados são representados da seguinte forma:

$$x = \pm m \times 2^e, \text{ onde } 1 \leq m < 2. \qquad \textbf{(3.8)}$$

Consequentemente, a expansão binária da mantissa é:

$$m = \left(b_0.b_1b_2b_3...b_{p-1}\right)_2 \text{ com } b_0 = 1. \qquad \textbf{(3.9)}$$

Por exemplo, o número $\frac{13}{2}$ é representado como:

$$\frac{13}{2} = \left(1.101\right)_2 \times 2^2.$$

Logo, para os números diferentes de zero, e como b_0 é 1, podemos escrever que um número x está normalizado se:

$$x = \pm m \times 2^e, \text{ com } m = \left(1.b_1b_2...b_{p-1}\right)_2.$$

Portanto, para representar um número neste sistema divide-se a sequência de bits em três campos: um para o sinal, outro para o expoente e e um outro para a mantissa m, respectivamente.

Uma sequência de 32 bits, pode ser dividida nos seguintes campos: 1 bit para o sinal, 8 bits para o expoente e 23 bits para a mantissa. O bit do sinal é 0

para os números positivos e 1 para os números negativos. Os 23 bits da mantissa são utilizados depois do ponto binário para representar a expansão binária de m, isto é, $b_1, b_2, ..., b_{23}$. É claro que não é necessário guardar b_0, visto que sabemos que o seu valor é 1 (diz-se que b_0 é um "bit implícito").

De acordo com o que foi referido, o número $\frac{13}{2}$ é representado como:

0	*ebits (2)*	10100000000000000000000

e o número $71 = (1.000111)_2 \times 2^6$ é expresso como:

0	*ebits (6)*	00011100000000000000000

Para facilitar a representação, os bits do expoente não são apresentados para já, explicitamente, mas através da expressão *"ebits (e)".*

Neste sistema, se x é exactamente uma potência de 2, então a mantissa é o número 1.0, uma vez que os bits da parte fraccional são todos iguais a zero (recordemos que $b_0 = 1$ não é armazenado de forma explícita). Por exemplo, $1 = (1.000...)_2 \times 2^0$ é expresso como

0	*ebits (0)*	00000000000000000000000

e o número $1024 = (1.000...)_2 \times 2^{10}$ é representado como

0	*ebits (10)*	00000000000000000000000

Nesta representação, não é possível normalizar o número zero, uma vez que todos os seus bits teriam de ser zeros. De facto, uma sequência de bits nesta representação significaria 1.0 e não 0.0, uma vez que o bit b_0 está implícito. Para ultrapassar esta dificuldade existem dois processos. O primeiro, que foi usado pela maioria dos sistemas de ponto flutuante até cerca de 1975, não pressupunha a existência do bit implícito e considerava que na representação de um número diferente de zero, o bit b_0 teria de ser

armazenado explicitamente, em vez de ser sempre igual a 1. Neste processo, o número zero podia ser representado por uma mantissa de bits todos iguais a zero. O segundo processo (utilizado na norma IEEE 754 de que falaremos mais à frente) consiste em usar uma sequência especial de bits para o campo do expoente para assinalar o facto do número ser zero.

3.3.3 Precisão e epsilon da máquina

A ***precisão*** de um sistema de ponto flutuante é o número de bits significativos (incluindo o bit implícito) usados para a representação da mantissa ([49]). Denotando a precisão por *p*, tem-se no sistema atrás descrito que $p = 24$ (23 bits da parte fraccional da mantissa e 1 bit implícito).

Qualquer número normalizado em ponto flutuante com precisão *p*, pode ser expresso como

$$x = \pm\left(1.b_1 b_2 ... b_{p-2} b_{p-1}\right)_2 \times 2^e . \qquad \textbf{(3.10)}$$

O menor número $x \in F$ que é maior do que 1 é

$$\left(1.00...01\right)_2 = 1 + 2^{-(p-1)}$$

e a diferença entre estes dois números

$$\varepsilon = \left(0.00...01\right)_2 = 2^{-(p-1)} \qquad \textbf{(3.11)}$$

designa-se por ***epsilon da máquina***. Esta quantidade que, como se pode ver, depende da base e do número de algarismos das mantissas, é da maior importância na análise de erros de arredondamento, como veremos mais adiante.

Mais geralmente, para um número em ponto flutuante $x \in F$ dado por (3.10) define-se

$$ulp(x) = (0.00...01)_2 \times 2^e = 2^{-(p-1)} \times 2^e = \varepsilon \times 2^e . \qquad \textbf{(3.12)}$$

Ulp é a abreviatura para *unit in the last place*. Se $x > 0$, então $ulp(x)$ é a distância entre x e o número que lhe sucede em F; se $x < 0$, então $ulp(x)$ é a distância entre x e o número que o antecede em F.

Uma aproximação para o *epsilon da máquina* de um sistema de vírgula flutuante de base b pode ser calculada usando o seguinte algoritmo e assumindo que o modo de arredondamento é para o mais próximo:

$\varepsilon \longleftarrow 1$

repetir

$\varepsilon \longleftarrow \varepsilon / b$

$\delta \longleftarrow 1 + \varepsilon$

até $(\delta = 1)$.

A interpretação deste algoritmo é a seguinte: se x é uma potência negativa de b tal que $x < \varepsilon$ então $1 + x$ dá 1.

O conhecimento de ε do sistema computacional ou máquina de calcular é fundamental. De facto, se considerarmos, por exemplo, a equação $1 + x = 1$, esta admite muitas soluções em aritmética de ponto flutuante e não apenas $x = 0$.

3.3.4 Overflow e underflow

Consideremos $x \in F$ tal que

$$x = \pm(b_0 . b_1 b_2 ... b_{p-1})_2 \times 2^e$$

onde *p* é a precisão, com $b_0 = 1$ e $e_{\min} \le e \le e_{\max}$ para números normalizados e com $b_0 = 0$ e $e = e_{\min}$ para números desnormalizados. Designemos por $N_{\max}$ o maior número normalizado e por $N_{\min}$ o menor número positivo normalizado.

Dizemos que ocorre *overflow* sempre que um cálculo produz um número $x > N_{\max}$. *Underflow* ocorre quando um cálculo produz $x < N_{\min}$ e neste caso, a máquina apresenta este número como zero. No entanto, nas máquinas que adoptem a norma IEEE 754, existe igualmente o *underflow gradual*, o qual estudaremos posteriormente.

A grandeza dos números que produzem *overflow* e *underflow* depende da máquina com que trabalhamos, como ilustra a tabela 3.2 ([39] p. 5).

Máquina	Underflow	Overflow
DEC PDP-11, VAX, formatos F e D	$2^{-128} \approx 2.9\times10^{-39}$	$2^{127} \approx 1.7\times10^{38}$
DEC PDP-10; Honeywell 600, 6000; Univac 110x simples; IBM 709X, 704X	$2^{-129} \approx 1.5\times10^{-39}$	$2^{127} \approx 1.7\times10^{38}$
Burroughs 6X00 simples	$8^{-51} \approx 8.8\times10^{-47}$	$8^{76} \approx 4.3\times10^{68}$
H-P 3000	$2^{-256} \approx 8.6\times10^{-78}$	$2^{256} \approx 1.2\times10^{77}$
IBM 360, 370; Amdahl1; DG Eclipse M/600; …	$16^{-65} \approx 5.4\times10^{-79}$	$16^{63} \approx 7.2\times10^{75}$
Maioria das calculadoras manuais	10^{-99}	10^{100}
CDC 6X00, 7X00, Cyber	$2^{-976} \approx 1.5\times10^{-294}$	$2^{1070} \approx 1.3\times10^{322}$
DEC VAX formato G ; UNIVAC, 110X duplo	$2^{-1024} \approx 5.6\times10^{-309}$	$2^{1023} \approx 9\times10^{307}$
HP 85	10^{-499}	10^{500}
Cray I	$2^{-8192} \approx 9.2\times10^{-2467}$	$2^{8192} \approx 1.1\times10^{2466}$
DEC VAX formato H	$2^{-16384} \approx 8.4\times10^{-4933}$	$2^{16383} \approx 5.9\times10^{4931}$
Burroughs 6X00 duplo	$8^{-32755} \approx 1.9\times10^{-29581}$	$8^{32780} \approx 1.9\times10^{29603}$
Proposta da norma IEEE: Intel i8087;Motorola 6839		
• Simples	$2^{-126} \approx 1.2\times10^{-38}$	$2^{127} \approx 1.7\times10^{38}$
• Duplo	$2^{-1022} \approx 2.2\times10^{-308}$	$2^{1023} \approx 9\times10^{307}$
• Duplo - estendido	$2^{-16382} \approx 3.4\times10^{-4932}$	$2^{16383} \approx 5.9\times10^{4931}$

Tabela 3.2 – *Underflow* e *Overflow* de algumas máquinas

3.4 A Norma IEEE 754 [6]

Os fabricantes de computadores têm adoptado, na construção das suas máquinas, diferentes sistemas de ponto flutuante diferindo na base, nos números de dígitos da mantissa e do expoente, nas regras de arredondamento, etc. Esta variedade leva, por exemplo, a que um mesmo procedimento possa ter resultados diferentes, consoante a máquina em que eles foram executados. Esta situação levou em 1985 à publicação da norma IEEE 754, cujo objectivo primordial consistia na uniformização dos vários sistemas existentes.

Esta aritmética prevê três aspectos fundamentais ([49]):

- uma representação consistente dos números no sistema de ponto flutuante em todas as máquinas que a adoptem;
- os resultados das operações em ponto flutuante são correctamente arredondados utilizando diversos modos de arredondamento;
- um tratamento consistente de situações excepcionais como por exemplo a divisão por zero.

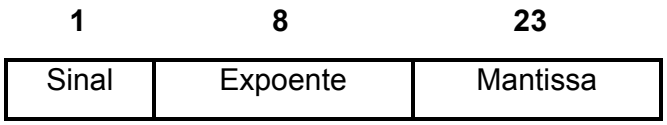
Nos formatos básicos da norma IEEE, o bit principal b_0 de um número normalizado é implícito, tal como já foi descrito anteriormente. Por conseguinte, é necessária uma representação especial para o armazenamento do número zero. No entanto, o zero não é o único número para o qual a norma tem uma representação especial – outro "número especial", não usado nas máquinas mais antigas, é o ∞. A existência deste "número", permite que se possa dividir um número por zero, obtendo como resultado matemático ∞, no lugar de uma mensagem de *overflow*. Assim sendo, uma questão se coloca de imediato: e então em relação ao $-\infty$? Posteriormente trataremos deste caso. Para já, note-se que enquanto -0 e $+0$ são duas diferentes representações para o mesmo número, $-\infty$ e $+\infty$ representam dois "números diferentes". Outra

[6] IEEE é a sigla do *Institute of Electrical and Electronic Engineers*, uma associação profissional dos Estados Unidos da América.

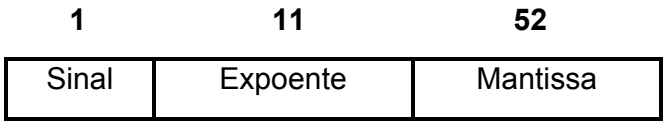
representação especial é *NaN* (abreviatura de *Not a Number*), que não é de modo algum um número.

A norma IEEE 754 define dois formatos básicos para os números em ponto flutuante: o formato simples, com 32 bits, e o formato duplo, com 64 bits.

Simples:



Duplo:



A base da representação utilizada é a binária e o primeiro bit é usado para o sinal da mantissa (0 para os números positivos e 1 para os negativos).

As representações dos números no ***formato simples*** estão sintetizadas na tabela 3.3:

$\pm$	$a_1a_2a_3...a_8$	$b_1b_2b_3...b_{23}$

Se os bits do expoente $a_1a_2a_3...a_8$ são	Então o seu valor numérico é
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1b_2b_3...b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{-125}$
$(00000011)_2 = (3)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{-124}$
$\downarrow$	$\downarrow$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{1}$
$\downarrow$	$\downarrow$
$(11111100)_2 = (252)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{125}$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1b_2b_3...b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ se $b_1 = ... = b_{23} = 0$, ou NaN

Tabela 3.3 – Formato simples da norma IEEE

A primeira linha desta tabela mostra que a representação do zero requer uma sequência especial de zeros para o campo do expoente, assim como uma sequência de bits para a parte fraccional, isto é,

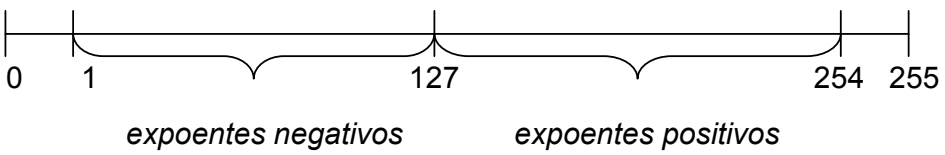
±	00000000	00000000000000000000000

Nenhuma outra linha nesta tabela pode ser usada para representar o número zero, uma vez que todas as linhas exceptuando a primeira e a última representam números normalizados com o bit inicial igual a 1 (este é o tal bit que está implícito).

Todas as linhas da tabela 3.3, excepto a primeira e a última, referem-se a números normalizados, ou seja, todos os números em ponto flutuante que de algum modo não são "especiais". A mantissa dispõe de 23 bits e é normalizada, ou seja, o primeiro bit da mantissa é sempre 1 e portanto, uma vez que é conhecido não é necessário armazená-lo. Assim sendo, o primeiro bit da mantissa é um bit implícito, o que significa que na realidade a mantissa possui $p = 24$ dígitos.

A última linha desta tabela mostra que uma sequência de bits do expoente composta só por 1s é uma configuração especial para representar $\pm\infty$ ou NaN, dependendo da sequência de bits da parte fraccional.

No formato simples o menor expoente é 00000001, correspondendo a 2^{-126}, e o maior é 11111110, correspondendo a 2^{127}, uma vez que para se obter o verdadeiro, subtrai-se 127 ao expoente armazenado. Por esta razão, o expoente diz-se *enviesado*. De facto, tem-se no formato simples 8 bits para o expoente. O menor expoente é 00000001 = 1 e o maior expoente é 11111110 = $2^8 - 2 = 254$.



Assim, sendo o expoente correspondente à sequência $(a_1 a_2 a_3 ... a_8)$, excepções à parte, o verdadeiro expoente é dado por:

$$e = E - \frac{2^8 - 2}{2} = E - \left(2^7 - 1\right) = E - 127\,, \qquad \textbf{(3.13)}$$

onde E é o expoente armazenado.

Note-se que, a partir da definição, se pode concluir que a gama de expoentes não é simétrica em torno da origem.

Por exemplo, o número $1 = (1.000...0)_2 \times 2^0$ é representado como:

0	01111111	00000000000000000000000

127 e portanto, $e = 127 - 127 = 0$.

O número $\frac{11}{2} = (1.011)_2 \times 2^2$ é representado como:

0	10000001	01100000000000000000000

O número $\frac{1}{10} = (1.10011001100...)_2 \times 2^{-4}$ não tem uma expressão binária finita. Se optarmos então por proceder a uma truncatura de modo a ocupar o campo da mantissa, vemos que $1/10$ é dado por (veremos posteriormente os outros modos de arredondamento previstos na norma IEEE):

0	01111011	10011001100110011001100

É por esta razão que num sistema computacional que utilize base 2, o somatório $\sum_{i=1}^{100} 0.1$ não dá exactamente 10.

Vejamos outro exemplo. No caso do número *B*:

0	10000100	10110011000001000000000

o bit mais à esquerda indica que o número é positivo, os 8 bits seguintes representam na base 10 o número 132:

$$1\times 2^7+0\times 2^6+0\times 2^5+0\times 2^4+0\times 2^3+1\times 2^2+0\times 2^1+0\times 2^0=132$$

E portanto o expoente é $132-127=5$. A fracção é

$$1.10110011000001000000000=1+2^{-1}+2^{-3}+2^{-4}+2^{-7}+2^{-8}+2^{-14}.$$

Assim sendo, estamos perante o número

$$B=\left(1+2^{-1}+2^{-3}+2^{-4}+2^{-7}+2^{-8}+2^{-14}\right)\times 2^5\cong 54.376953125.$$

O número que antecede *B* é

0	10000100	10110011000000111111111

ou seja,

$$A=\left(1+2^{-1}+2^{-3}+2^{-4}+2^{-7}+2^{-8}+2^{-15}+2^{-16}+2^{-17}+2^{-18}+2^{-19}+2^{-20}+2^{-21}+2^{-22}+2^{-23}\right)\times 2^5$$
$$\cong 54.376949310303$$

Por outro lado, o número que sucede a *B* é

0	10000100	10110011000001000000001

ou seja,

$$C=\left(1+2^{-1}+2^{-3}+2^{-4}+2^{-7}+2^{-8}+2^{-14}+2^{-23}\right)\times 2^5\cong 54.376956939697$$

Tem-se que

$$C-B=ulp(B)=\varepsilon\times 2^5=B-A\ ,$$

ou seja, as distâncias de *A* e *C* a *B* são iguais.

A distância entre números consecutivos de F decresce à medida que os números se aproximam de zero. No entanto, o zero não é ponto de

acumulação de F, uma vez que o conjunto F é finito. Esta situação ocorre quer no formato simples quer no duplo.

Em suma, a norma IEEE permite representar em *formato simples* números na forma

$$x = (-1)^s .2^{E-127}\left(b_0 .b_1 ... b_{p-1}\right), \qquad \textbf{(3.14)}$$

em que $s \in \{0,1\}$, $0 < E < 255$ é o expoente e os b_k são bits, isto é, $b_k = \{0,1\}$ com $b_0 = 1$ se o número for normalizado e $b_0 = 0$ se o não for.

A representação do número zero (ver a primeira linha da tabela 3.3), requer bits todos nulos no campo do expoente assim como para o campo da fracção, isto é,

±	00000000	00000000000000000000000

cujo valor é pois: $\pm (0.b_1 b_2 b_3 ... b_{23})_2 \times 2^{-126}$.

O menor número positivo normalizado que pode ser representado neste sistema é

0	00000001	00000000000000000000000

ou seja,

$$N_{\min} = (1.000...0)_2 \times 2^{-126} = 2^{-126} \approx 1.2 \times 10^{-38} \qquad \textbf{(3.15)}$$

O maior número normalizado (equivalentemente o maior número finito) que pode ser representado neste sistema é

0	11111110	11111111111111111111111

ou seja,

$$N_{\max} = (1.111...1)_2 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38} \qquad \textbf{(3.16)}$$

Consideremos novamente a primeira linha da tabela 3.3. Apesar de neste formato, o menor número normalizado ser 2^{-126}, é possível usar uma combinação de uma sequência de bits iguais a zero para o expoente e uma sequência de bits diferentes de zero para a parte fraccional, que permite representar números menores (estes são designados em [49] por *subnormais*). Por exemplo, $2^{-127} = (0.1)_2 \times 2^{-126}$, é representado como

0	00000000	10000000000000000000000

enquanto que $2^{-149} = (0.000...001)_2 \times 2^{-126}$ (com 22 bits iguais a zero depois do ponto binário) é armazenado como

0	00000000	00000000000000000000001

É claro que neste formato, este é o menor número positivo que poderá ser guardado (2^{-150} já aparece como zero).

Pelo que já foi referido, estes números não podem ser normalizados, uma vez que a normalização iria necessitar de um expoente que não pertence ao intervalo permitido. No entanto, a possibilidade de representação deste tipo de números permite a existência do designado *underflow gradual* ([49] pp. 44-45). Vejamos um exemplo. Consideremos os números A e B tais que

$$A = (1.01000000000000000000000)_2 \times 2^{-126} \quad \text{e}$$

$$B = (1.00000000000000000000000)_2 \times 2^{-126} = N_{\min}.$$

Então,

$$A - B = (0.01000000000000000000000)_2 \times 2^{-126}.$$

Normalizando este resultado, temos que:

$$A - B = (1.00000000000000000000000)_2 \times 2^{-128},$$

que é um número menor do que $N_{\min}$. Sem o *underflow gradual*, o resultado produzido é o número zero (este método é designado por *underflow súbito* e é ainda utilizado por alguns fabricantes de máquinas, nomeadamente no caso das máquinas de calcular gráficas que serão utilizadas neste estudo). Com o

underflow gradual, o resultado 2^{-128} pode ser armazenado exactamente com uma representação "desnormalizada"

0	00000000	01000000000000000000000

A utilização deste método foi e ainda é, a parte mais controversa da norma IEEE, com apoiantes e oponentes que apresentam as diversas razões para justificar as suas opiniões. Apesar da introdução deste método poder encarecer o hardware, as vantagens do ponto de vista numérico prevalecem sobre os aspectos económicos.

O ***formato duplo*** tem uma estrutura semelhante à do formato simples mas utiliza 64 bits: 1 para o sinal, 11 para o expoente e 52 para a mantissa (ver tabela 3.4). Assim, o expoente pode variar entre –1022 e +1023, e a mantissa, por via do bit implícito, dispõe de 53 bits, ou seja, $p = 53$.

$\pm$	$a_1a_2a_3...a_{11}$	$b_1b_2b_3...b_{52}$

Se os bits do expoente $a_1a_2a_3...a_{11}$ são	Então o seu valor numérico é
$(00000000000)_2 = (0)_{10}$	$\pm(0.b_1b_2b_3...b_{52})_2 \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$ $(00000000010)_2 = (2)_{10}$ $(00000000011)_2 = (3)_{10}$ $\downarrow$	$\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{-1022}$ $\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{-1021}$ $\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{-1020}$ $\downarrow$
$(01111111111)_2 = (1023)_{10}$ $(10000000000)_2 = (1024)_{10}$ $\downarrow$	$\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{0}$ $\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{1}$ $\downarrow$
$(11111111100)_2 = (2044)_{10}$ $(11111111101)_2 = (2045)_{10}$ $(11111111110)_2 = (2046)_{10}$	$\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{1021}$ $\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{1022}$ $\pm(1.b_1b_2b_3...b_{52})_2 \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$\pm\infty$ se $b_1 = ... = b_{52} = 0$, ou *NaN*

Tabela 3.4 – Formato duplo da norma IEEE

Os números com uma expansão binária infinita, como por exemplo, $1/10$ ou π, são agora obviamente, representados mais precisamente com o formato duplo do que com o simples.

O menor número positivo normalizado que pode ser representado no formato duplo é

$$N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308} \qquad \textbf{(3.17)}$$

e o maior é:

$$N_{\max} = (2 - 2^{-52}) \times 2^{1023} \approx 1.8 \times 10^{308}. \qquad \textbf{(3.18)}$$

Uma síntese dos limites dos expoentes e os valores dos números normalizados menores e maiores dados em (3.15), (3.16), (3.17) e (3.18), encontram-se na seguinte tabela:

Formato	$e_{\min}$	$e_{\max}$	$N_{\min}$	$N_{\max}$
Simples	-126	127	$2^{-126} \approx 1.2 \times 10^{-38}$	$2^{128} \approx 3.4 \times 10^{38}$
Duplo	-1022	1023	$2^{-1022} \approx 2.2 \times 10^{-308}$	$2^{1024} \approx 1.8 \times 10^{308}$

Tabela 3.5 – Limites do formato IEEE de ponto flutuante

A ***precisão*** de um sistema de ponto flutuante é, como já foi referido anteriormente, o número de bits significativos (incluindo o bit implícito). Tem-se que no formato simples $p = 24$ e no duplo $p = 53$. Qualquer número normalizado em ponto flutuante com precisão *p*, pode ser expresso como

$$x = \pm \left(1.b_1 b_2 \ldots b_{p-2} b_{p-1}\right)_2 \times 2^e. \qquad \textbf{(3.19)}$$

No formato simples, o primeiro número maior do que 1 é $1 + 2^{-23}$, enquanto que o primeiro número maior do que 1 no formato duplo é $1 + 2^{-52}$.

A precisão $p = 24$ no formato simples corresponde, aproximadamente, a sete algarismos decimais significativos. Por outro lado, no formato duplo, a precisão $p = 53$, corresponde, aproximadamente, a dezasseis algarismos decimais significativos (mais pormenores em [49] pp. 23-24). Estas observações encontram-se sumariadas na seguinte tabela:

Formato	Precisão	Épsilon da máquina
Simples	$p = 24$	$\varepsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
Duplo	$p = 53$	$\varepsilon = 2^{-52} \approx 2.2 \times 10^{-16}$

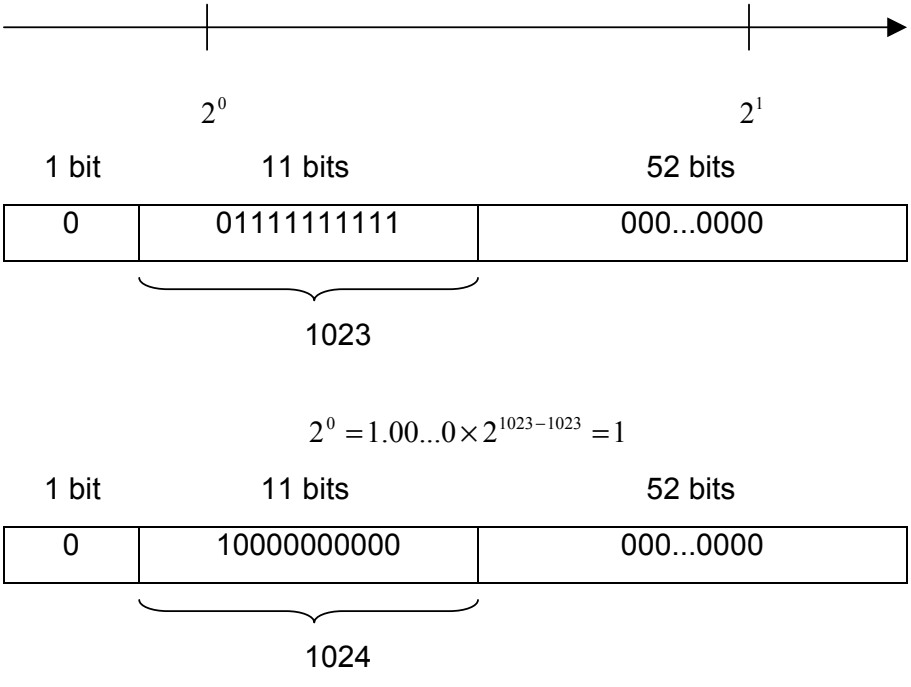
Tabela 3.6 – Precisão dos formatos de ponto flutuante IEEE

Para se perceber melhor que a distribuição dos números de F não é uniforme apresentamos o seguinte resultado:

Proposição:

Para valores de *k* inteiros dentro de certos limites existem 2^{p-1} números de F no intervalo $]2^k, 2^{k+1}]$ e o espaçamento entre cada dois pontos consecutivos é $2^k.2^{-(p-1)} = 2^{k-p+1}$.

Consideremos, sem perda de generalização, o formato duplo (64 bits), onde $p = 53$. Vamos então verificar que existem 2^{52} números de F no intervalo]1,2].



Quais os números entre 1 e 2?

$A_1 = 1.00...01 \times 2^0 = 1 + 2^{-52}$ $A_2 = 1.00...10 \times 2^0 = 1 + 2^{-51}$

$A_3 = 1.00...11 \times 2^0 = 1 + 2^{-51} + 2^{-52}$ $A_4 = 1.00...100 \times 2^0 = 1 + 2^{-50}$

$A_5 = 1.00...101 \times 2^0 = 1 + 2^{-50} + 2^{-52}$ $A_6 = 1.00...110 \times 2^0 = 1 + 2^{-50} + 2^{-51}$

…

$A_{2^{52}-1} = 1.11...10 \times 2^0 = 1 + 2^{-1} + 2^{-2} + ... + 2^{-51} = 1 + \left(1 - 2^{-51}\right) = 2 - 2^{-51}$

$A_{2^{52}} = 1.11...11 \times 2^0 = 2 - 2^{-52}$

Existem portanto 2^{52} números de F no intervalo $]2^0, 2^1]$. O espaçamento entre eles é 2^{-52}:

$A_1 - 1 = 2^{-52}$

$A_2 - A_1 = 1 + 2^{-51} - 1 - 2^{-52} = 2^{-51} - 2^{-52} = 2^{-52}(2 - 1) = 2^{-52}$

…

$A_{2^{52}} - A_{2^{52}-1} = 2 - 2^{-52} - 2 + 2^{-51} = 2^{-52}(-1 + 2) = 2^{-52}$

$2 - A_{2^{52}} = 2^{-52}$

No intervalo $]2,4]$ existem igualmente 2^{52} números de F e o espaçamento entre eles é 2^{-51}:

$B_1 = 1.00...01 \times 2^1 = (1 + 2^{-52}) \times 2 = 2 + 2^{-51}$

$B_2 = 1.00...10 \times 2^1 = (1 + 2^{-51}) \times 2 = 2 + 2^{-50}$

…

$B_{2^{52}-1} = 1.11...10 \times 2^1 = (2 - 2^{-51}) \times 2 = 4 - 2^{-50}$

$B_{2^{52}} = 1.11...11 \times 2^1 = (2 - 2^{-52}) \times 2 = 4 - 2^{-51}$

$B_1 - 2 = 2 + 2^{-51} - 2 = 2^{-51}$

$B_2 - B_1 = 2 + 2^{-50} - 2 - 2^{-51} = 2^{-51}(-1 + 2) = 2^{-51}$

…

$B_{2^{52}} - B_{2^{52}-1} = 4 - 2^{-51} - 4 + 2^{-50} = 2^{-51}(-1 + 2) = 2^{-51}$

$4 - B_{2^{52}} = 2^{-51}$

Seja k inteiro. Temos por conseguinte o intervalo $]2^k,2^{k+1}]$, onde $2^k = 1.00...0 \times 2^k$.

$N_1 = 1.00...01 \times 2^K = (1+2^{-52}) \times 2^k = 2^k + 2^{k-52}$

$N_2 = 1.00...10 \times 2^k = (1+2^{-51}) \times 2^k = 2^k + 2^{k-51}$

$N_3 = 1.00...11 \times 2^k = (1+2^{-51}+2^{-52}) \times 2^k = 2^k + 2^{k-51} + 2^{k-52}$

…

$N_{2^{52}-1} = 1.11...10 \times 2^k = (2-2^{-51}) \times 2^k = 2^{k+1} - 2^{k-51}$

$N_{2^{52}} = 1.11...11 \times 2^k = (2-2^{-52}) \times 2^k = 2^{k+1} - 2^{k-52}$

Existem portanto 2^{52} números de F no intervalo $]2^k,2^{k+1}]$. O espaçamento entre eles é:

$N_1 - 2^k = 2^k + 2^{k-52} - 2^k = 2^{k-52}$

$N_2 - N_1 = 2^k + 2^{k-51} - 2^k - 2^{k-52} = 2^{k-52}(2-1) = 2^{k-52}$

…

$N_{2^{52}} - N_{2^{52}-1} = 2^{k+1} - 2^{k-52} - 2^{k+1} + 2^{k-51} = 2^{k-52}(-1+2) = 2^{k-52}$

$2^{k+1} - N_{2^{52}} = 2^{k+1} - 2^{k+1} + 2^{k-52} = 2^{k-52}$

Ou seja, o espaçamento é $\varepsilon = 2^{k-52}$. É evidente que pelo facto de existirem 2^{52} números de F no intervalo $]2^k,2^{k+1}]$, segue imediatamente que o espaçamento entre cada dois dos seus pontos é igual a:

$$\frac{2^{k+1}-2^k}{2^{52}} = \frac{2^k.2-2^k}{2^{52}} = \frac{2^k(2-1)}{2^{52}} = 2^k.2^{-52} = 2^{k-52} \ .\blacksquare$$

3.5 Modos de arredondamento

Como verificámos anteriormente, a representação de números em ponto flutuante só permite a representação exacta de alguns números reais (todos racionais). Assim, dado um número real x, qual o número em $F(b,p,q)$, que denotaremos por $fl(x)$, $\tilde{x}$ ou $round(x)$, que o representa? Se x tiver representação exacta em F, então tem-se que $fl(x) = x$. Caso contrário, existem em alguns sistemas de ponto flutuante (nomeadamente nas máquinas

gráficas utilizadas neste estudo) dois modos para determinar $fl(x)$: o corte (ou truncatura) e o arredondamento. Assumindo que x não leva a *overflow* ou a *underflow*, vejamos em que consiste cada um destes processos:

- **Corte ou Truncatura (T):** desprezam-se simplesmente os dígitos do número real x que ultrapassam o número de dígitos permitidos para a mantissa.

- **Arredondamento (A):** o número real x é representado pelo número de F que lhe está mais próximo em valor absoluto, ou seja,

$$|fl(x)-x| \leq |g-x|, \forall g \in F.$$

Vejamos um exemplo. Pretendemos encontrar as representações do número π no sistema $F(10,5,2)$, utilizando as duas técnicas anteriores:

- $fl(\pi) = 3.1415926535 = 3.1416 \times 10^0$ ⟶ **Arredondamento**
- $fl(\pi) = 3.1415926535 = 3.1415 \times 10^0$ ⟶ **Truncatura.**

Se quiséssemos representar o número 3.1415 no sistema $F(10,4,2)$, a regra atrás enunciada leva a uma ambiguidade, uma vez que 3.1415 é o ponto médio de $[3.1410,3.1420]$. Para resolver esta situação, aplica-se uma técnica designada por **arredondamento simétrico**, e que consiste em ([49] pp.10-11):

- se b e $\frac{b}{2}$ forem pares, arredondar de modo a que o último dígito fique ímpar;
- se b for par mas $\frac{b}{2}$ for ímpar (como é o caso de quando $b=2$ ou $b=10$), arredondar de modo a que o último dígito fique par (no caso de $b=2$, corresponde a escolher o número cujo último bit é igual a zero);
- se b for ímpar, optar por uma das regras anteriores, já que parece não haver vantagens ou desvantagens determinantes em qualquer delas.

Assim, no exemplo anterior tem-se que

$$fl(\pi) = 3.1415926535 = 3.142 \times 10^{0}.$$

No que diz respeito à aritmética IEEE 754, vejamos como estão definidos os diversos tipos de arredondamento ([49] pp. 25-29). Já vimos anteriormente que os números neste sistema de ponto flutuante podem ser expressos na forma

$$x = \pm\left(b_0.b_1b_2...b_{p-1}\right)_2 \times 2^e,$$

onde *p* é a precisão do sistema em ponto flutuante, com $b_0 = 1$ e $e_{\min} \le e \le e_{\max}$ para números normalizados e com $b_0 = 0$ e $e = e_{\min}$ para números desnormalizados. Recordemos que $N_{\max}$ designa o maior número normalizado e que $N_{\min}$ é o menor número positivo normalizado. Existem também dois números infinitos em ponto flutuante, $\pm\infty$. Dizemos que um número real x está no intervalo normalizado do sistema de ponto flutuante se $N_{\min} \le |x| \le N_{\max}$. Os números ± 0 e $\pm\infty$ e os números desnormalizados não se encontram no intervalo normalizado do sistema de ponto flutuante.

Suponhamos que o número real $x \notin F$. Então pelo menos uma (ou talvez as duas) das seguintes situações é verdadeira:

- x encontra-se fora do intervalo normalizado (o seu valor absoluto é maior do que $N_{\max}$ ou menor do que $N_{\min}$). Por exemplo, no formato simples os números 2^{130} e 2^{-130} não pertencem ao intervalo normalizado (ver tabela 3.5);
- a expansão binária de x requer mais de p bits para representar exactamente o número, ou seja, a precisão do sistema de ponto flutuante é insuficiente para representar x exactamente. Por exemplo, no formato simples, o número $1 + 2^{-25} = \left(1.00000000000000000000001\right)_2$ requer mais bits do que aqueles que são permitidos.

Em qualquer dos casos, é necessário aproximar x por um número que pertença ao sistema.

Definamos x_- como sendo o número de F mais próximo de x tal que $x_- \leq x$ e x_+ como o número de F mais próximo de x tal que $x_+ \geq x$. Se o número mais próximo é o zero, escolhemos para sinal do zero o sinal de x.

Consideremos um número positivo x pertencente ao intervalo normalizado e escrevamos x na sua forma normalizada:

$$x = \left(1.b_1 b_2 ... b_{p-1} b_p b_{p+1} ...\right)_2 \times 2^e . \qquad \textbf{(3.20)}$$

Tem-se que o número mais próximo de F que é menor ou igual a x é:

$$x_- = \left(1.b_1 b_2 ... b_{p-1}\right)_2 \times 2^e ,$$

ou seja, x_- é obtido por truncatura da expansão binária da mantissa, desprezando b_p, b_{p+1}. Se $x \notin F$ pelo menos um dos bits desprezados na sua expansão é diferente de zero, então,

$$x_+ = \left(\left(1.b_1 b_2 ... b_{p-1}\right)_2 + \left(0.00...01\right)_2\right) \times 2^e ,$$

é o próximo número de F maior do que x_- e, por conseguinte também o próximo número maior do que x (o qual deverá situar-se entre x_- e x_+). Neste caso, o "1" no incremento está na posição $(p-1)$ depois do ponto binário, e portanto a amplitude do intervalo $[x_-, x_+]$ é

$$2^{-(p-1)} \times 2^e . \qquad \textbf{(3.21)}$$

Note-se que este valor é o mesmo de $ulp(x)$ definido em (3.12). Encontrar a expansão binária de x_+ é um pouco mais complicado, uma vez que um bit tem de ser adicionado à última posição da parte fraccional de x_-.

Se x é maior do que $N_{\max}$, então $x_- = N_{\max}$ e $x_+ = \infty$. Se x é positivo mas menor do que $N_{\min}$, então x_- é um número desnormalizado ou zero e x_+ é um número desnormalizado ou $N_{\min}$. No caso de x ser um número negativo, x_+ e x_- são determinados de forma análoga.

Estamos agora em condições de enunciar os modos de arredondamento previstos na aritmética IEEE. A primeira regra prevista na norma IEEE é a de que se o resultado de uma determinada operação tiver representação exacta,

então o resultado dessa operação deverá ser exactamente esse valor. Ou seja, se x tiver representação exacta no sistema de ponto flutuante, então tem-se que $round(x) = x$. Caso contrário, a norma estabelece quatro modos diferentes de arredondamento que poderão ser implementados para determinar $round(x)$:

- **arredondamento por defeito**, ou seja, o resultado é sempre arredondado para o número representável imediatamente abaixo, isto é, $round(x) = x_-$;
- **arredondamento por excesso**, ou seja, o resultado é sempre arredondado para o número representável imediatamente acima, isto é, $round(x) = x_+$;
- **truncatura**, isto é, $round(x) = x_-$ se $x \geq 0$ e $round(x) = x_+$ se $x \leq 0$;
- **arredondamento simétrico, ou para o mais próximo** (este é o mais favorável do ponto de vista numérico e também o mais usado na prática – é o modo por defeito nos computadores que utilizem a norma IEEE): $round(x)$ é x_- ou x_+, o que estiver mais próximo de x, excepto se $x > N_{\max}$, então $round(x) = \infty$, e se $x < -N_{\max}$, então $round(x) = -\infty$. No caso de empate, ou seja, x_- e x_+ estão à mesma distância de x, arredondar de modo a que o último bit seja igual a zero.

Se x é um número positivo, então x_- encontra-se entre zero e x e, portanto, o arredondamento por defeito e a truncatura obtêm o mesmo $round(x)$. No caso de x ser um número negativo, então x_+ encontra-se entre zero e x e, por conseguinte, o arredondamento por excesso e a truncatura obtêm o mesmo $round(x)$.

Se $x > N_{\max}$ e o modo de arredondamento é o simétrico, então $round(x) = \infty$, o qual não é, mais próximo de x do que $N_{\max}$.

3.5.1 Erro absoluto e erro relativo de arredondamento

O passo seguinte consiste em analisar os erros de representação, ou seja, determinar a discrepância entre a representação $\tilde{x} = round(x)$ do número real x num sistema de ponto flutuante e o número real x.

Seja

$$x = m \times b^e \text{ e } \tilde{x} = round(x) = \tilde{m} \times b^e . \quad \textbf{(3.22)}$$

Define-se ***erro absoluto de arredondamento*** **A** por $A = \left| round(x) - x \right|$, ou seja,

$$A = \left| round(x) - x \right| = \left| \tilde{m} \times b^e - m \times b^e \right| = \left| (\tilde{m} - m) \times b^e \right| = \left| \tilde{m} - m \right| \times b^e . \quad \textbf{(3.23)}$$

Este valor depende da precisão e do modo de arredondamento utilizado.

Suponhamos que x é da forma dada em (3.20), mas não pertence ao sistema de ponto flutuante. Então, é imediato que o erro absoluto de arredondamento associado a x é menor do que a amplitude entre x_- e x_+, independentemente do modo de arredondamento. Por conseguinte, de (3.21) tem-se que:

$$A = \left| round(x) - x \right| < 2^{-(p-1)} \times 2^e . \quad \textbf{(3.24)}$$

Informalmente podemos dizer que o erro absoluto de arredondamento é menor do que *ulp*, significando $ulp(x_-)$ se $x > 0$ e $ulp(x_+)$ se $x < 0$. Quando o modo de arredondamento utilizado é o simétrico, então tem-se

$$A = \left| round(x) - x \right| \leq 2^{-p} \times 2^e . \quad \textbf{(3.25)}$$

Vejamos, por exemplo, qual o erro absoluto de arredondamento no formato simples do número $1/10$, para os quatro modos diferentes.

Já vimos que o número $\frac{1}{10} = (1.10011001100...)_2 \times 2^{-4}$ não tem uma expressão binária finita. Então, utilizando o modo de:

- **arredondamento por defeito** ou a **truncatura** tem-se que:

0	01111011	10011001100110011001100

E portanto, $A = |round(x) - x| = (1.10011001100...)_2 \times 2^{-28} < 2^{-23} \times 2^{-4} = 2^{-27}$.

- **arredondamento por excesso** ou o **arredondamento simétrico** tem-se que:

0	01111011	10011001100110011001101

E portanto, $A = |round(x) - x| = (1.0011001100...)_2 \times 2^{-29} \leq 2^{-24} \times 2^{-4} = 2^{-28}$.

O ***erro relativo de arredondamento*** associado ao número x diferente de zero é definido por:

$$R = |\delta| \text{, onde } \delta = \frac{round(x)}{x} - 1 = \frac{round(x) - x}{x}. \qquad \textbf{(3.26)}$$

Assumindo novamente que x é um número normalizado, de (3.20) sabemos que

$$|x| > 2^e,$$

e portanto, para todos os modos de arredondamento, o erro relativo de arredondamento possui o seguinte majorante:

$$R = |\delta| = \frac{|round(x) - x|}{|x|} < \frac{2^{-(p-1)} \times 2^e}{2^e} = 2^{-(p-1)} = \varepsilon, \qquad \textbf{(3.27)}$$

utilizando o majorante do erro absoluto (3.24) e a definição de ε em (3.11). No caso de se considerar o arredondamento simétrico temos, utilizando (3.25), que:

$$R = |\delta| \leq \frac{2^{-p} \times 2^e}{2^e} = 2^{-p} = \frac{1}{2}\varepsilon. \qquad \textbf{(3.28)}$$

Ao número $\mu = 2^{-p} = \frac{1}{2}\varepsilon$ (no caso do arredondamento simétrico) ou $\mu = 2^{-(p-1)} = \varepsilon$ (no caso da truncatura) chamamos ***unidade de erro de arredondamento*** da máquina. A unidade de erro de arredondamento é

portanto, um majorante para o erro relativo de arredondamento de qualquer número.

De (3.26) tem-se que

$$round(x) = x(1+\delta), \text{ com } |\delta| \leq \mu .$$

Combinando este resultado com (3.27) e (3.28), completa-se a demonstração do seguinte resultado ([49] p. 29):

Teorema 3.1 – Seja x um número real qualquer no intervalo normalizado de um sistema binário de ponto flutuante com precisão *p*. Então:

$$round(x) = x(1+\delta),$$

para algum δ satisfazendo $|\delta| < \varepsilon$, onde ε, precisão da máquina, é o valor entre 1 e o próximo número de ponto flutuante maior, isto é,

$$\varepsilon = 2^{-(p-1)}.$$

Além disso, se o modo de arredondamento é o simétrico, tem-se que:

$$|\delta| \leq \frac{1}{2}\varepsilon = 2^{-p}. \ \blacksquare$$

Este teorema é muito importante porque mostra que, independentemente de como x é armazenado e apresentado, podemos pensar neste valor não como "exacto" mas como "exacto com um factor" de $1+\varepsilon$. Portanto, por exemplo, no formato simples os números são apresentados com um factor de $1+10^{-7}$, ou seja, são aproximados com sete dígitos decimais significativos (consultar a tabela 3.6).

Vejamos o seguinte exemplo. Os Babilónios recorriam às seguintes duas aproximações para $\sqrt{2}$: $(1.\underline{25})_{60}$ e $(1.\underline{24}\,\underline{51}\,\underline{10})_{60}$. Na base 10 tem-se que:

$$(1.\underline{25})_{60} = 1+\frac{25}{60} = 1+\frac{5}{12} = \frac{17}{12} = 1.41(6) \text{ e}$$

$$(1.\underline{24}\,\underline{51}\,\underline{10})_{60} = 1+\frac{24}{60}+\frac{51}{60^2}+\frac{10}{60^3} = \frac{30547}{21600} = 1.41421(296).$$

Determinando os erros absoluto e relativo cometidos em cada uma das aproximações temos:

$$A = \left|\frac{17}{12} - \sqrt{2}\right| = .0024531042936... \quad \text{e} \quad R = |\delta| = \left|\frac{\frac{17}{12} - \sqrt{2}}{\sqrt{2}}\right| = .0017346066809624...$$

Por outro lado,

$$A = \left|\frac{30547}{21600} - \sqrt{2}\right| = 5.994101 \times 10^{-7} \quad \text{e} \quad R = |\delta| = \left|\frac{\frac{30547}{21600} - \sqrt{2}}{\sqrt{2}}\right| = 4.23846946 \times 10^{-7}.$$

Em qualquer uma das aproximações tem-se que $R < A$ por ser $\sqrt{2} > 1$. ■

3.6 Sistema de ponto flutuante das calculadoras gráficas

3.6.1 Introdução

Quando se trabalha com uma calculadora há que ter em conta que esta está limitada ao sistema numérico de ponto flutuante que tem incorporado. O sistema de números com que a calculadora trabalha não é, como já foi referido, o conjunto dos números reais, nem sequer o dos racionais. As calculadoras gráficas utilizadas neste estudo suportam um conjunto de números da forma

$$\pm \quad , \qquad ... \qquad \times 10^{\pm} \quad ,$$

onde, obviamente, o número de dígitos na mantissa e no expoente são limitados. Para o expoente são reservados dois dígitos[7] (o que permite expoentes de –99 a 99), enquanto que o número de dígitos da mantissa varia de máquina para máquina. O expoente é ajustado de forma a que o primeiro dígito da mantissa (à esquerda da vírgula) seja diferente de zero. A base

[7] As máquinas de que temos conhecimento usam todas dois algarismos para os expoentes.

incorporada nas máquinas utilizadas neste estudo é a base 10. Ao contrário do que acontece na base 2, não é possível na base 10, assumir a existência de um algarismo implícito. É claro que nesta base, contrariamente ao que se passava na base 2, o número 0.1 possui uma representação exacta.

Consideremos pois $F(10,p,q)$, o **sistema de representação em ponto flutuante de base 10**, onde p é o número de dígitos da mantissa e cujo expoente pode utilizar no máximo q dígitos (base *10*). Este sistema é constituído por todos os números reais x normalizados da forma

$$x = \pm m \times 10^{e}, \qquad \textbf{(3.29)}$$

onde $1 \le m < 10$ e e é um número inteiro tal que $|e| \le 10^{q} - 1$ e ainda $x = 0$. É de notar que nesta máquina a gama de expoentes é simétrica relativamente à origem. No entanto, esta situação nem sempre sucede nos sistemas de ponto flutuante, como vimos anteriormente no caso da aritmética IEEE754.

Vejamos então em pormenor o sistema de ponto flutuante das máquinas que foram utilizadas neste trabalho, a Casio CFX – 9850 GB Plus e a Texas Instruments TI-83 Plus (para abreviar e em todas as referências a estas duas máquinas, diremos Casio CFX - 9850 e Texas TI – 83, respectivamente). A escolha destas máquinas deve-se ao facto destas serem as mais divulgadas e utilizadas nas escolas portuguesas.

3.6.2 Casio CFX – 9850 GB Plus

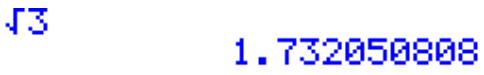
Esta máquina possui uma mantissa de 15 dígitos, apresentando os resultados no visor com uma mantissa de 10 dígitos. O expoente dispõe de dois dígitos, ou seja, $-99 \le e \le 99$. Assim sendo, temos que o sistema de ponto flutuante utilizado por esta calculadora é $F(10,15,2)$.

Vejamos por exemplo o caso de $\sqrt{3}$. Temos que

$$\sqrt{3} \cong 1.7320508075688772935 \ \ (20\ dígitos)^{8}.$$

[8] Este valor foi obtido através do software *Mathematica*.

Esta máquina apresenta $\sqrt{3}$ como



ou seja, com 10 dígitos. Para encontrar os outros cinco dígitos que internamente a máquina armazena, podemos proceder do seguinte modo: subtrai-se ao resultado obtido, o número que surge no visor com todos os algarismos excepto o último. Neste caso tem-se

ou seja, $\sqrt{3} = 1.73205080756887\ (15\ dígitos)$. Note-se que para o resultado apresentado no visor, é feito um arredondamento simétrico.

Quando se introduzem nesta máquina, a partir do teclado, números com mais de 10 dígitos, ela trunca todos os algarismos a partir do décimo primeiro, como podemos ver pelo seguinte exemplo.

Seja $A = 9.9999999999$, com uma mantissa de 11 dígitos e $B = 9.999999993$, com uma mantissa de 10 dígitos. Então o valor exacto de $A - B$ é dado por:

$$A - B = 9.9999999999 - 9.999999993 = 6.9 \times 10^{-9},$$

mas o resultado na máquina é

(a máquina truncou o último algarismo de A).

Segundo o manual de utilização ([11]), esta calculadora possui quatro formatos de apresentação dos números:

O modo ***Fix*** permite especificar um número fixo de casas decimais. A opção ***Sci*** permite especificar o número de algarismos significativos e o modo ***Eng*** representa os números na notação de engenharia. Por exemplo

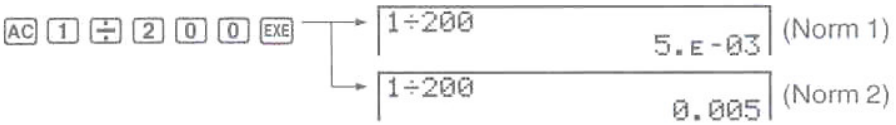
```
10^3                1.k
3×10^6              3.M
7×10^9              7.G
```

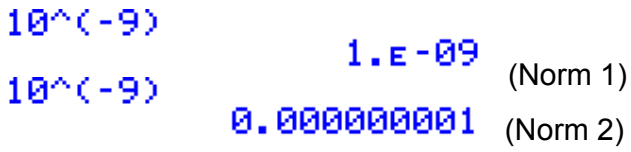
onde o símbolo k (kilo), representa 10^3, M (mega) 10^6 e G (giga) representa 10^9.

O modo ***Norm*** permite representar os números em notação científica (o manual chama-lhe formato exponencial). Como já foi mencionado, esta calculadora apresenta os números com uma mantissa de 10 dígitos. Sempre que estes números ultrapassem este limite, eles são automaticamente convertidos e visualizados num dos seguintes formatos (a escolha de um destes formatos é realizada pelo utilizador): *norm 1* ou *norm 2*[9]. A única diferença entre estes dois formatos reside na apresentação dos números que pertençam ao intervalo $]10^{-9},10^{-2}[$ e $]-10^{-2},-10^{-9}[$. Assim, no caso de *norm 1*, qualquer número x tal que $|x|<10^{-2}$ é visualizado na notação científica. Por outro lado, no formato *norm 2*, os números só são apresentados na notação científica se $|x|<10^{-9}$.

Vejamos os seguintes exemplos:

[9] Refere o manual, que todos os exemplos apresentados encontram-se no formato *Norm1*.

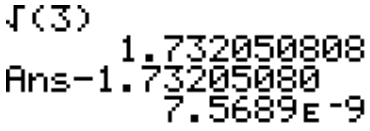




3.6.3 Texas Instruments TI-83 Plus

Esta calculadora possui uma mantissa de 14 dígitos, apresentando os resultados no visor com uma mantissa de 10 dígitos. O expoente dispõe de dois dígitos, ou seja, $-99 \le e \le 99$. Assim sendo, temos que o sistema de ponto flutuante utilizado por esta máquina é $F(10,14,2)$.

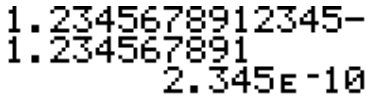
Vejamos novamente o exemplo de $\sqrt{3}$. Temos nesta máquina que:



Ou seja, esta máquina apresenta $\sqrt{3}$ como $1.732050808\ (10\ dígitos)$ enquanto que internamente considera $\sqrt{3} = 1.7320508075689\ (14\ dígitos)$. Analogamente ao que se passa com a Casio CFX - 9850, esta máquina também apresenta o valor de $\sqrt{3}$ arredondado simetricamente.

Ao contrário do que sucede com a Casio CFX - 9850, no que diz respeito aos números introduzidos, a partir do teclado, com mais de 10 dígitos, eles são guardados até ao 14º dígito como podemos ver pelo seguinte exemplo.

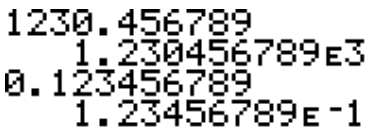
Sejam $A = 1.2345678912345$ com 14 dígitos e $B = 1.234567891$ com uma mantissa de 10 dígitos. Então o valor exacto de $A - B$ é dado correctamente pela máquina



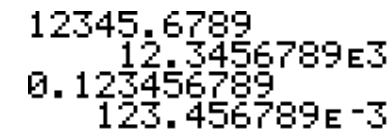
De acordo com o manual de utilização desta máquina ([67]), existem três modos de notação: ***Normal***, ***Sci*** e ***Eng***[10]. Estes modos só afectam a forma como uma resposta é apresentada no écran de visualização. É possível introduzir um número em qualquer um dos formatos.

O modo de notação ***Normal***, é a forma mais habitual em que expressamos os números, com dígitos à esquerda e à direita do ponto decimal (exemplo 1230.456789).

O modo de notação ***Sci*** (científica) expressa números em duas partes. Os números significativos são apresentados com um dígito à esquerda do ponto decimal e a potência de 10 adequada é apresentada à direita de E. Vejamos o seguinte exemplo:



O modo de notação ***Eng*** (de engenharia) é semelhante à notação científica. No entanto, o número pode ter um, dois ou três dígitos antes do ponto decimal e o expoente é um múltiplo de três, como por exemplo

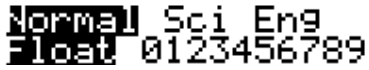


No caso de seleccionar a notação ***Normal*** mas não seja possível apresentar a resposta em 10 dígitos ou se o valor absoluto for inferior a 0,001,

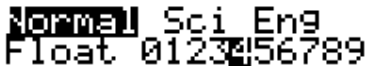
[10] Estes dois últimos são semelhantes aos já referidos para a máquina Casio CFX – 9850.

então esta máquina exprime a resposta em notação científica. Vejamos os exemplos

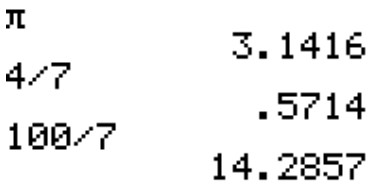
É possível ainda utilizar um modo decimal denominado *Float 0123456789*, que permite apresentar a resposta até 10 dígitos.



Esta opção aplica-se aos três modos de notação. Consideremos, por exemplo, a opção Float 4



todos os números são apresentados com 4 casas decimais. Por exemplo



3.6.4 *Overflow* e *underflow*

Como já foi mencionado, F é um conjunto limitado. Importa então, obter a resposta para as seguintes questões:

- Qual o maior número $N_{\max}$, em valor absoluto, que pode ser representado em cada uma das máquinas?
- E qual o menor número $N_{\min}$, em valor absoluto, que pode ser representado em cada uma das máquinas?

De um modo geral, em $F(10, p, q)$ e de acordo com (3.29) tem-se que

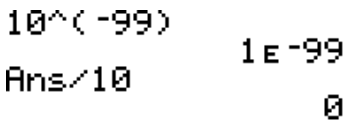
$$N_{\max} = \left(10 - 10^{-(p-1)}\right) \times 10^{e} \text{ e } N_{\min} = 10^{-e}.$$

Assim sendo, no caso das máquinas já mencionadas, a Casio CFX – 9850, como o sistema de representação em ponto flutuante utilizado é $F(10,15,2)$ então o maior número, em valor absoluto, representável será o $9.99999999999999 \times 10^{99}$, enquanto que o menor, em valor absoluto é 10^{-99}.

Por outro lado, na Texas TI - 83**,** cujo sistema de representação em ponto flutuante utilizado é $F(10,14,2)$, então o maior número, em valor absoluto, representável será o $9.9999999999999 \times 10^{99}$, enquanto que o menor, em valor absoluto é 10^{-99}.

A tentativa de representação de um número x tal que $|x| > N_{\max}$, conduz à condição de **"overflow"**, isto é, ultrapassagem do número de dígitos permitido para o expoente. Por outro lado, se tentarmos representar um número $x \neq 0$, tal que $|x| < N_{\min}$, somos conduzidos a uma situação de **"underflow"** e o número apresentado na máquina é o zero.

Vejamos o seguinte exemplo (obtido na Texas TI-83) onde ocorre *underflow*:



Consideremos agora um exemplo (realizado na Casio CFX – 9850) onde ocorre *overflow* ([53]):

Cálculo efectuado	Resultado obtido
69!	$1.711224524E+98$
70!	*overflow error* (Ma Error)[11]

Tabela 3.7

[11] Se a calculadora exceder os parâmetros de cálculo surge no visor uma mensagem de erro. Se qualquer resultado, quer intermédio ou final, ou qualquer valor em memória, exceder $\pm 9.99999999999999 \times 10^{99}$, a calculadora envia a mensagem de erro *Ma Error*. No caso da calculadora Texas Instruments TI-83 Plus, a mensagem de erro é *Overflow* (ver anexos A e B).

A representação exacta do número 69! envolve 99 dígitos[12]:
69!=171122452428141311372468338881272839092270544893520369393648040923257279754140647424000000000000000.
Por outro lado, a representação do número 70! envolve 101 dígitos o que requer um expoente igual a +100, ou seja, utilizando 3 dígitos:
70!=11978571669969891796072783721689098736458938142546425857555362864628009582789845319680000000000000000.

Na prática, pode ser necessário reformular certos cálculos no sentido de evitar a ocorrência destes problemas numéricos. Vejamos a seguinte situação ([50] p. 30). Pretende-se calcular a hipotenusa h de um triângulo rectângulo de catetos a e c, os quais assumem valores muito elevados mas inferiores ao limite de *overflow*. No entanto, existe a possibilidade de a^2 e c^2 produzirem *overflow*. Assim, para contornar este problema, se $a \geq c$, é preferível utilizar a expressão $h = a\sqrt{1+\left(\frac{c}{a}\right)^2}$, uma vez que o cálculo do valor desta expressão não produz *overflow*. ■

3.6.5 Precisão e epsilon da máquina

A ***precisão*** p de um sistema de ponto flutuante é, como já vimos, o número de algarismos significativos. Assim tem-se na **Casio CFX – 9850** que $p = 15$ e na **Texas TI - 83** $p = 14$.

Já referimos que qualquer número normalizado em ponto flutuante com precisão *p* nestas máquinas, pode ser expresso como:

$$x = \pm\left(d_0.d_1d_2...d_{p-2}d_{p-1}\right)_2 \times 10^e, \text{ onde } 1 \leq d_k < 10 \text{ e } -99 \leq e \leq 99.$$

O menor número $x \in F$ que é maior do que 1 é:

$$1 + 10^{-(p-1)}.$$

[12] Cálculo efectuado com o auxílio do software *Mathematica.*

O ***epsilon da máquina***, ou seja, a diferença entre 1 e o número que lhe sucede em F é, de acordo com o que já foi referido anteriormente, dado por

$$\varepsilon = 10^{-(p-1)}.$$

Assim sendo, seria de esperar que na máquina **Casio CFX – 9850** $\varepsilon = 10^{-14}$, enquanto que na **Texas TI-83** $\varepsilon = 10^{-13}$. No entanto, a realidade mostra-nos que na máquina **Casio CFX – 9850**: $\varepsilon = 8 \times 10^{-14}$ (resultado para o qual não obtivemos explicação), ao passo que na **Texas TI- 83** tem-se de facto que $\varepsilon = 10^{-13}$.

Em síntese,

Máquina	Precisão	Epsilon
Casio CFX – 9850 GB Plus	$p = 15$	$\varepsilon = 8 \times 10^{-14}$
Texas Instruments TI-83 Plus	$p = 14$	$\varepsilon = 10^{-13}$

Tabela 3.8 – Precisão e epsilon das máquinas

3.6.6 Modos de arredondamento

As calculadoras **Casio CFX – 9850** e **Texas TI-83**, utilizam dois modos de arredondamento:

- a truncatura;
- o arredondamento simétrico.

É de referir, uma vez que é de extrema importância, que estas duas máquinas não arredondam os números da mesma forma.

A truncatura é utilizada sempre que se introduz na calculadora, a partir da calculadora, um valor que ultrapasse o número de dígitos permitido. Já nos referimos a esta situação em 3.6.2 e 3.6.3. Na Casio CFX – 9850, se introduzirmos o número 123123456789, esta máquina apresenta o número $1.231234567E+11$. Assim, sempre que são considerados números com mais

de 10 dígitos, esta máquina efectua truncatura. Por outro lado, na calculadora Texas TI-83, se introduzirmos, a partir do teclado, um número com mais de 15 dígitos, esta máquina trunca o número a partir do 15º algarismo.

Até ao número de dígitos permitido, ambas as calculadoras procedem ao arredondamento simétrico. Por exemplo na Texas TI-83, temos que:

```
12345678987
  1.234567899E10
12345678932
  1.23456789 3E10
12345678965
  1.234567897E10
```

O arredondamento simétrico é também utilizado quando os resultados das operações excedem o número de dígitos permitidos para a visualização. Por exemplo, na Casio CFX – 9850, temos

```
√3+√5                    9÷7
          3.968118785              1.285714286
Ans-3.96811878           Ans-1.28571428
          5.06865E-09              5.71428E-09
```

Para ilustrar a diferença entre as máquinas que estamos a usar, consideremos o exemplo seguinte. Suponhamos que pretendíamos concluir que $\sin 45º = \cos 45º$ utilizando a calculadora, efectuando para isso a diferença. Os resultados obtidos com as duas máquinas encontram-se na seguinte tabela:

Cálculo efectuado	**Resultado obtido na Casio CFX – 9850**	**Resultado obtido na Texas TI-83**
$\sin 45º - \cos 45º$	1×10^{-15}	0

Tabela 3.9

Perante estes resultados, que conclusões podemos retirar? Porque é que os resultados são diferentes?

Sendo

$$\sin 45^\circ = \cos 45^\circ = \frac{\sqrt{2}}{2}$$

é claro que este valor não pode ser representado exactamente. E não há garantia de que a fórmula de cálculo para as funções seno e coseno produzam a mesma aproximação para cada valor do argumento. O que se pode esperar, em geral, é que a diferença em termos relativos, não seja muito superior (normalmente será inferior) ao epsilon da máquina.

Relativamente à Casio CFX – 9850, ela considera internamente $\sin 45^\circ = 0.707106781186548$, enquanto que $\cos 45^\circ = 0.707106781186547$. Portanto nesta máquina tem-se

$$\sin 45^\circ - \cos 45^\circ = 10^{-15} < 8 \times 10^{-14} = \varepsilon .$$

A Texas TI-83, considera $\sin 45^\circ = \cos 45^\circ = 0.70710678118655$ (14 casas decimais). O valor exacto de $\sin 45^\circ = \cos 45^\circ$, com 20 casas decimais[13] é dado por:

0.70710678118654752440. ■

[13] Valor obtido com o software *Mathematica.*