

**MAP3122: Métodos numéricos e aplicações**  
**Quadrimestral 2022**

Antoine Laurain

**Aritmética de ponto flutuante**

# Introdução

- Aritmética no computador  $\neq$  aritmética tradicional
- Números como  $\sqrt{3}$  ou  $\pi$ , que têm uma infinidade de dígitos, não podem ser representados exatamente no computador  
→ aproximações que geram erros de arredondamento.

**Exemplo 1:** Seja  $f(x) = \frac{1 - \cos(2x)}{2x^2}$ , deveríamos ter  $f(10^{-9}) \approx 1$ . Usando Matlab ou Python obtemos  $f(10^{-9}) = 0$  → errado!

Usando trigonometria, podemos escrever  $f(x)$  como  $f(x) = \left(\frac{\sin(x)}{x}\right)^2$ . Usando esta expressão, Python fornece  $f(10^{-9}) = 1$  → o resultado numérico depende da expressão!

**Exemplo 2:** Para todos  $k \in \mathbb{R}$  temos  $(345 + 10^k) - 10^k = 345$ . Usando Python, obtemos

$$(345 + 10^{15}) - 10^{15} = 345$$

$$(345 + 10^{16}) - 10^{16} = 344$$

$$(345 + 10^{17}) - 10^{17} = 352$$

$$(345 + 10^{18}) - 10^{18} = 384$$

$$(345 + 10^{19}) - 10^{19} = 0.$$

A explicação deste fenômeno é que para  $k$  grande,  $(345 + 10^k)$  não está representado exatamente no computador.

**Exemplo 3:** Usando Python calculamos

$$1234 \times (0,1 + 0,1 + 0,1 - 0,3)^{1/10} = 29,22 \dots,$$

mas o valor exato é 0!

→ o computador não consegue representar 0,1 e 0,3 exatamente, por isso ele calcula  $0,1 + 0,1 + 0,1 - 0,3 = 5,55 \times 10^{-17}$ . Este erro pequeno é amplificado extremamente, quando a décima raiz é calculada.

**Questões:**

- Como o computador armazena números?
- Como o computador faz operações básicas?
- Como os erros de arredondamento são amplificadas?

# Números com ponto flutuante

Computadores usam números com ponto flutuante da forma:

$$x = (-1)^{s_{10}} 2^{c_{10}-1023} (1 + f_{10}),$$

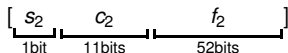
$s_{10}$ ,  $c_{10}$ ,  $f_{10}$  são números em base 10:

- $s_{10}$  = indicador de sinal (1 bit)
- $c_{10}$  = característica (11 bit)
- $f_{10}$  = mantissa (52 bit)
- total: 64 bits usados

## Observações:

- Essa representação permite representar ambos números grandes e pequenos.
- A representação é única pois  $1 < 1 + f_{10} < 2$ .
- $c_{10}$  satisfaz  $-1023 < c_{10} - 1023 < 1024$ , isto é  $0 < c_{10} < 2047$ .
- 52 dígitos binários  $\Leftrightarrow$  16 a 17 dígitos decimais.

O computador armazena  $s_{10}$ ,  $c_{10}$ ,  $f_{10}$  usando o número binário:



# Números com ponto flutuante

## Observações:

- Além destes números, têm dois valores  $+0$  e  $-0$  para zero, duas infinidades  $+\infty$  e  $-\infty$ , e *NaN* (not-a-number).
- Chama-se IEEE-standard 754 (de 1985) para aritmética de ponto flutuante (em dupla precisão).
- IEEE = Institute of Electrical and Electronics Engineers.

**Exemplo:**  $s_2 = 0$ ,  $c_2 = 1 \underbrace{0 \dots 0}_{8 \text{ vezes}} 11$ ,  $f_2 = 101110010001 \underbrace{0 \dots 0}_{40 \text{ vezes}}$

- A correspondência entre  $c_2$  (na base 2) e  $c_{10}$  (na base 10) é:

$$c_{10} = 1 \times 2^{10} + 0 \times 2^9 + \dots + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 1024 + 2 + 1 = 1027$$

- A correspondência entre  $f_2$  (na base 2) e  $f_{10}$  (na base 10) é:

$$f_{10} = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} + \dots + 1 \times 2^{-8} + \dots + 1 \times 2^{-12} + \underbrace{0 + \dots + 0}_{40 \text{ vezes}}$$

- Assim  $[s_2 \ c_2 \ f_2]$  representa o número decimal

$$\begin{aligned} & (-1)^{s_{10}} 2^{c_{10} - 1023} (1 + f_{10}) \\ & = (-1)^0 \times 2^{1027 - 1023} \left( 1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096} \right) = 27,566406256 \end{aligned}$$

# Números com ponto flutuante

- Se um número  $x$  não é exatamente da forma  $(-1)^{s_{10}} 2^{c_{10}-1023} (1 + f_{10})$ , ele precisa ser arredondado.
- O epsilon da máquina  $eps$  é o menor número que somado a 1 produza resultado diferente de 1, depois do arredondamento.
- Em Python com dupla precisão,  $eps = 2^{-52} \approx 2,22 \times 10^{-16}$   
→ então números com até 15 decimais podem ser armazenados exatamente.
- menor número positivo que pode ser representado:

$$x_{min} = 2^{-1023} \times (1 + \underbrace{2^{-52}}_{f_2=00\dots001}) \approx 10^{-308}$$

- maior número positivo que pode ser representado:

$$x_{max} = 2^{1024} \times (1 + \underbrace{1 - 2^{-52}}_{f_2=11\dots11}) \approx 10^{308}$$

# Erros de arredondamento

- Para simplificar a apresentação, vamos usar uma forma normalizada de ponto flutuante decimal:  $\pm 0, d_1 d_2 \dots d_k \times 10^n$  com  $1 \leq d_1 \leq 9$ ,  $0 \leq d_i \leq 9$  para  $i = 2, 3, \dots, k$ . (precisamos de  $d_1 \geq 1$  por causa da unicidade da representação).
- Qualquer número real positivo  $y$  pode ser representado por:

$$y = 0, \underbrace{d_1 d_2 \dots d_k d_{k+1} \dots}_{\text{infinitude de dígitos}} \times 10^n.$$

- Definimos  $fl(y)$  = ponto flutuante correspondente a  $y$ .
- Existem 2 métodos para definir  $fl(y)$ :
  1. Truncamento:  $fl(y) = 0, d_1 d_2 \dots d_k \times 10^n$ , isto é, truncamos todos os dígitos de  $y$  a partir de  $d_{k+1}$ .
  2. Arredondamento para baixo:  $fl(y) = 0, d_1 d_2 \dots d_k \times 10^n$  se  $d_{k+1} < 5$ , ou arredondamento para cima:  $fl(y) = 0, d_1 d_2 \dots (d_k + 1) \times 10^n$  se  $d_{k+1} \geq 5$ .

**Exemplo:**  $\pi = 0, 314159265 \dots \times 10^1$

$$fl(\pi) = 0, 31415 \times 10^1 \text{ (truncamento)}$$

$$\text{ou } fl(\pi) = 0, 31416 \times 10^1 \text{ (arredondamento para cima)}$$

- Erro absoluto =  $|x - fl(x)|$
- Erro relativo =  $\frac{|x - fl(x)|}{|x|}$  para  $x \neq 0$ .

# Aritmética computacional

As operações  $\oplus$ ,  $\ominus$ ,  $\otimes$ ,  $\oslash$  são as 4 operações básicas da aritmética computacional:

$$x \oplus y = fl(fl(x) + fl(y))$$

$$x \ominus y = fl(fl(x) - fl(y))$$

$$x \otimes y = fl(fl(x) \times fl(y))$$

$$x \oslash y = fl(fl(x) \div fl(y))$$

**Exemplo:**  $x = 5/7$ ,  $y = 1/3$ , truncamento em 5 dígitos. Estes erros são razoáveis pois a precisão depois da operação é aproximadamente  $10^{-5}$  (5 dígitos), isto é, da mesma ordem que o truncamento.

Aritmética tradicional	Aritmética ponto flutuante	Erro absoluto	Erro relativo
$x + y = \frac{22}{21}$	$x \oplus y = 0,10476 \times 10^1$	$0,190 \times 10^{-4}$	$0,182 \times 10^{-4}$
$x - y = \frac{8}{21}$	$x \ominus y = 0,38095 \times 10^0$	$0,238 \times 10^{-5}$	$0,625 \times 10^{-5}$

**Exemplo:** as operações  $\oplus$ ,  $\ominus$  não são associativas nem distributivas. Por exemplo, usando arredondamento em 3 dígitos:

$$(4,26 \oplus 9,24) \oplus 5,04 = 13,5 \oplus 5,04 = 18,5$$

$$4,26 \oplus (9,24 \oplus 5,04) = 4,26 \oplus 14,3 = 18,6$$



# Aritmética computacional

**Definição:** Dizemos que  $p^*$  aproxima  $p$  com  $t$  algarismos significativos se  $t \geq 0$  é o maior inteiro tal que

$$\frac{|p - p^*|}{|p|} \leq 5 \times 10^{-t}.$$

**Exemplo (perda de significância):**

Valores exatos	Valores com arredondamento para 4 dígitos
$x_1 = 0,10024$	$\tilde{x}_1 = 0,1002$
$x_2 = 0,10011$	$\tilde{x}_2 = 0,1001$
$x_1 - x_2 = 0,13 \times 10^{-3}$	$\tilde{x}_1 - \tilde{x}_2 = 0,1 \times 10^{-3}$

A diferença  $\tilde{x}_1 - \tilde{x}_2$  é calculada exatamente. No entanto, há uma forte amplificação do erro relativo, pois

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|} \approx 4 \times 10^{-4}, \quad \frac{|\tilde{x}_2 - x_2|}{|x_2|} \approx 1 \times 10^{-4}$$

mas

$$\frac{|(\tilde{x}_1 - \tilde{x}_2) - (x_1 - x_2)|}{|x_1 - x_2|} \approx 2 \times 10^{-1}.$$

Então  $\tilde{x}_1$  aproxima  $x_1$  e  $\tilde{x}_2$  aproxima  $x_2$  ambos com  $t = 4$  algarismos significativos, mas  $(\tilde{x}_1 - \tilde{x}_2)$  aproxima  $(x_1 - x_2)$  com apenas  $t = 1$  algarismo significativo. Este fenômeno chama-se perda de significância.

→ melhor evitar a subtração de dois números aproximadamente iguais.

→ este número pode ser amplificado depois pela multiplicação por um número grande.

# Conclusão

- A aritmética computacional não é exata.
- Pequenos erros vão se adicionando e podem levar a erros significativos, quando temos uma grande quantidade de operações.
- Métodos numéricos que são matematicamente equivalentes podem levar a resultados diferentes.  
—→ escolher o método que produz menos erros de arredondamento, se for possível.