Errors on Computation: FP Numbers

Overflow, Underflow, EPS de la maquina, etc.

William Oquendo

Credits: ICTP (Stefano Cozzini); Computational Physics - Landau and Paez

Bits, Bytes, ...

- BIT: 1 o 0, true or false, yes or no
  0 0
  1 1
  01 1
  10 2
  11 3
  1101 ???
  Solo hay 10 tipos de personas: Las que entienden binario y las que no.
- If we are given N bits, the first one is used for the sign, and the remainder for representing the number: 2^{N-1}
- BYTE: 8 bits, 256 values
- 2^10 = 1,024 = 1 Kbyte (small difference with 1000 !!!)

FP Number representation

- Fixed notation:

$$x_{\text{fix}} = \text{sign} \times (\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_0 2^0 + \cdots + \alpha_{-m} 2^{-m}).$$

Advantage: All numbers have the same error, 2^{-m-1}
Disadvantage: Small numbers have LARGE relative errors
Applications: Bussiness.

- Floating point notation: (Scientific notation!)

$$x_{\text{float}} = (-1)^s \times \text{mantissa} \times 2^{\text{expfld - bias}}.$$

Example: Single precision 32 bits number, 8 bits fot the exponent [0, 255], negative exponents are represented with bias equal to -127. Of the remainder bits, one is used for the sign and the others for the mantissa

Number representation

- Floating (continued ...):
  Single precision (4 byte): 6-7 decimal places of precision, 1 part in 2^23, 10^{-44} < single precision < 10^{38}

- Mantissa:

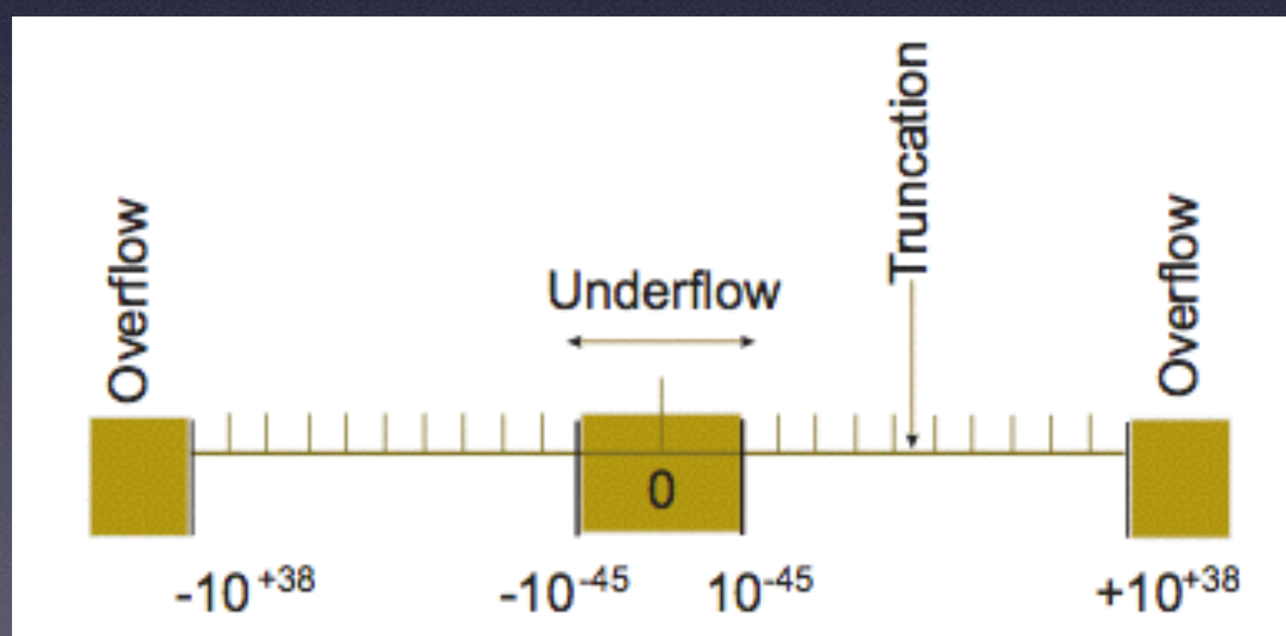
$$\text{mantissa} = m_1 \times 2^{-1} + m_2 \times 2^{-2} + \cdots + m_{23} \times 2^{-23},$$

- Double precision: 64 bits (8 bytes), 11 bits for the exponent, and 52 for the mantissa, 16 decimal places of precision (1 in 2^52), and range in

$$10^{-322} \leq \text{double precision} \leq 10^{308}$$

FP Number Range

Format	# bits	#significand bits	macheps	#exponent bit
Single	32	23+1	2^{-24} (~10^{-7})	8
Double	64	52+1	2^{-53} (~10^{-16})	11
Double Extended (*80 bits on all Intel machines*)	>=80	>=64	<=2^{-64}(~10^{-19})	>=15



FP Number Density

With 32 bits, there are $2^{32}-1$, or about 4 billion, different bit patterns.

- These can represent 4 billion integers *or* 4 billion reals.
- But there are an infinite number of reals, and the IEEE format can only represent *some* of the ones from about -2^{128} to $+2^{128}$.
- Represent same number of values between 2^n and 2^{n+1} as 2^{n+1} and 2^{n+2}

- Same number of bits to represent all numbers : the smaller the number, the greater the density of representable numbers.
- Example: There are 8388607 single precision numbers between 1.0 and 2.0, while there are only about 8191 between 1023.0 and 1024.0
- The larger the number, the smaller the number of fp numbers to use, then the larger the truncation error.

Misconceptions

- FP arithmetic is not well defined: false, IEEE 754 standarizes.
- 15 decimal digits are enough for my simple 3-decimal digits calculation: false, 14 significant digits can be destroyed in a single operation.
- I can always cast a float to integer: be careful!
- Addition is associative: false! x + (y + z) != (x + y) + z , with x = -1.5e38, y = 1.5e38, z = 1.
- Everything can be represented: 1/3 is not, 0.01 is not, 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 != 1.0
- The compiler takes care: NO! -ffloat-store (and related) for gnu compilers; -mp for intel compiler; and -Kieee for PGI

Affects real-life

- During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and tens of peoples were killed.
- A later study determined that the problem was caused by the inaccuracy of the binary representation of 0.10.
  - The Patriot incremented a counter once every 0.10 seconds.
  - It multiplied the counter value by 0.10 to compute the actual time.
- However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- This doesn't seem like much, but after 100 hours the time ends up being off by 0.34 seconds—enough time for a Scud to travel 500 meters!

Kind of errors

- Prob. of an error:

$$\text{start} \to U_1 \to U_2 \to \ldots \to U_n \to \text{end},$$

- Blunders: Typographical, wrong program, etc
- Random errors: Electronics, alien invasion, etc.
- Approximation: (mathematical series truncation)
- Roundoff: Truncation of a number in the computer representation

Substractive cancellation

$$
\begin{aligned}
a = b - c \quad \Rightarrow \quad a_c &= b_c - c_c, \\
a_c &= b(1 + \epsilon_b) - c(1 + \epsilon_c), \\
\Rightarrow \quad \frac{a_c}{a} &= 1 + \epsilon_b \frac{b}{a} - \frac{c}{a}\epsilon_c.
\end{aligned}
$$

$$
\begin{aligned}
\frac{a_c}{a} &= 1 + \epsilon_a, \\
\epsilon_a &\simeq \frac{b}{a}(\epsilon_b - \epsilon_c).
\end{aligned}
$$

Substractive cancellation: Example

$$S_N^{(1)} = \sum_{n=1}^{2N} (-1)^n \frac{n}{n+1}.$$

$$S_N^{(2)} = -\sum_{n=1}^{N} \frac{2n-1}{2n} + \sum_{n=1}^{N} \frac{2n}{2n+1}.$$

$$S_N^{(3)} = \sum_{n=1}^{N} \frac{1}{2n(2n+1)}.$$

- Escribir un programa que calcula cada suma como funcion de N.
- Suponer que S3 es exacta. Hacer una tabla que compare a S1 y a S2 con S3 de forma relativa: (S1 - S3)/S3 en funcion de N
- Dibujar y analizar.

Significant figures: Example

$$S^{(\mathrm{up})} = \sum_{n=1}^{N} \frac{1}{n},$$

$$S^{(\mathrm{down})} = \sum_{n=N}^{1} \frac{1}{n}.$$

- Escribir un programa que calcula cada suma como funcion de N.
- Hacer una tabla de la diferencia relativa dividida entre la suma relativa como funcion de N.
- Dibujar y analizar.

Multiplicative errors

$$a = b \times c \quad \Rightarrow \quad a_c = b_c \times c_c,$$

$$\Rightarrow \quad \frac{a_c}{a} = \frac{(1+\epsilon_b)(1+\epsilon_c)}{(1+\epsilon_a)} \simeq 1 + \epsilon_b + \epsilon_c.$$

Rounding errors modeled as random walk

$$R \approx \sqrt{N} r.$$

$$\epsilon_{\text{ro}} \approx \sqrt{N} \epsilon_m.$$

- Sometimes errors are not random but coherent!!!

Errors in algorithms

$$\begin{aligned}\epsilon_{\text{tot}} &= \epsilon_{\text{aprx}} + \epsilon_{\text{ro}}, \\ &\simeq \frac{\alpha}{N^{\beta}} + \sqrt{N}\epsilon_m.\end{aligned}$$

$$\frac{d\epsilon_{tot}}{dN} = -\frac{\alpha\beta}{N^{\beta+1}} + \frac{\epsilon_m}{2\sqrt{N}} = 0$$

$$N^* = \left(\frac{2\alpha\beta}{\epsilon_m}\right)^{\frac{2}{2\beta+1}}$$

Errors in algorithms

$$N^* = \left(\frac{2\alpha\beta}{\epsilon_m}\right)^{\frac{2}{2\beta+1}}$$

β	N float $\epsilon_m = 10^{-6}$	N double $\epsilon_m = 10^{-16}$	float ϵ_{min}	double ϵ_{min}
2	437	4373448	2.6E-05	2.68E-15
4	34	5705	6.7E-06	8.5E-15

$\alpha = 1$