Honors Numerical Analysis

Jan 26, 2022

Course email / Google Group: hna22@nyu.edu

Today: IEEE Arithmetic

Computers store information - namely numbers - in binary format:

$$2 = (10)_2$$

$$7 = \underbrace{(111)_2}$$ $= (000111)_2$

$$= 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$= 0 \cdot 2^5 + \ldots$$

Fractional numbers:

$$2.5 = (10.1)_2$$

$$2.25 = \underbrace{(10.01)}_{1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}}$$

Each 0 or 1 is a bit

Fixed representation: XXXX.XXXX ← Not how computers store info anymore

Floating point representation: $\pm M \times 2^E$, $M \in [1,2)$
(scientific / engineering notation)

M ↑ Mantissa, E ↑ Exponent

Example $23 = (10111)_2$

$= (1.0111 \times 2^4)$

Every multiplication by 2 moves decimal over to the right by 1 place.

$\frac{1}{8} = (0.001)_2$

$= (1.0 \times 2^{-3})$

How many bits does a computer use?

"single precision" : 32 bits

1 bit for sign

8 bits for exponent

23 bits for mantissa

Ex: $5.5 = 1.011 \times 2^2$

$= (\underbrace{0}_{\text{sign}} \mid \underbrace{E = 00000010}_{\text{exponent}} \mid \underbrace{M = 10110 \cdots 0}_{\text{mantissa}})$

always 1 - could ignore, except for the number 0. (hidden bit representation).

Def A floating point number is one that can be represented exactly using the above structure.

"double precision" : 64 bits

1 bit for sign

11 bits for exponent

52 bits for mantissa $\quad 1.\underbrace{\cdots}_{52 \text{ bits}}$

These representations were standardized in the 1980s by the IEEE. (Int. of Electrical and Electronics Engineers)
(IEEE 754)

In Julia : $x = 7.0$

typeof(x) $\to$ Float64 (i.e. double precision)

$x = 7$

typeof(x) $\to$ Int64 (i.e. 64 bit = 8 byte integer).

To represent negative exponents E, the assumption is that E is offset by 1023

$$\Rightarrow \quad x = \underbrace{\pm}_{1\text{bit}} \underbrace{M}_{} \times 2^{E-1023}$$

$M \to \underbrace{1.0\ldots0}_{52 \text{ bits}}$ $\qquad E \to \underbrace{1010\cdots0}_{11 \text{ bits.}}$

In Julia:
bitstring(x) returns IEEE representation

Inf, ∞, ± 0

$E = (01111111111)_2 \Rightarrow 2^0$

$E = (10000000000)_2 \Rightarrow 2^{1024-1023} = 2^1$

The point of IEEE 754 is to have consistent rules so that code is portable!

Special values:

$Inf = 1/0 \Rightarrow M=0,\ E=1\cdots1$

$\pm 0 \Rightarrow M=0,\ E=0\cdots0$

$NaN = \sqrt{-1} \Rightarrow M\neq0,\ E=1\cdots1$

Rounding

Example: $\frac{1}{10} = (0.1)_{10}$

$$= \underbrace{(1.100\overline{1100})_2 \times 2^{-4}}$$

this must be rounded and stored as a floating point number.

There are 4 options: up, down, to-zero, nearest $\rightarrow$ default

Details are not important, just know that it's a standard specified by IEEE.

The important idea is the error made in rounding:

<u>Absolute</u> rounding error $= |\text{round}(x) - x|$

<u>Relative</u> rounding error $= \dfrac{|\text{round}(x) - x|}{|x|}$ } similar to percentage error

Most important rule/consequence of the standard:

IEEE says that it must be the case that:

$$\text{round}(\underbrace{a+b}_{\text{exact addition}}) = \text{round}(a) \oplus \text{round}(b)$$

computer addition

$$= (a+b)(1+\delta)$$

where $|\delta| < \epsilon$, with ϵ = machine <u>precision</u>

Machine precision is the distance between 1 and <u>the next</u> floating point number:

$$\epsilon = \underbrace{(0|01\cdots1|0\cdots00)_2}_{(1.0)_{fp}} - \underbrace{(0|01\cdots1|0\cdots01)}_{(1.0\cdots01)_2 = 1+2^{-52}}$$

$\Rightarrow \epsilon = 2^{-52}$ on double precision

How to calculate this? $\Rightarrow$ <u>Julia demo</u>

The IEEE standards enforce rules on the relative accuracy of $+ \times - \div$:

$$\text{round}(a+b) = (a+b)(1+\delta)$$

$$\Rightarrow \quad \frac{|\text{round}(a+b) - (a+b)|}{|a+b|} \leq \epsilon$$

In terms of absolute accuracy:

$$
\begin{aligned}
|\text{round}(a+b) - (a+b)| &= |a+b|\,|\delta| \\
&\leq (|a| + |b|)\,\epsilon \\
&\leq 2\max(|a|,|b|)\,\epsilon
\end{aligned}
$$

Julia comments:

- problems installing?
  julialang.org
- make plot of something
- formatted output