Floating point numbers

Previously: working with rational numbers can easily produce proof certificates.

Problem: for realistic applications integers become too large ⇒ memory problems.

Fix for usual computations?

→ floating point arithmetic

→ Problem: simple computations are not exact

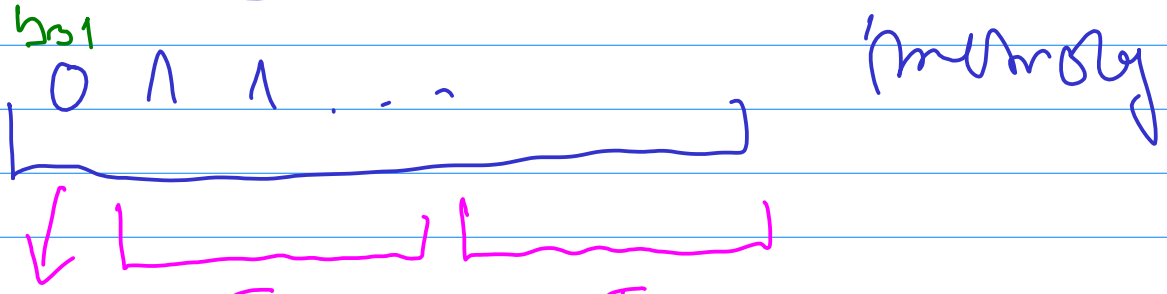
$0.1 + 0.2 == 0.3 \rightarrow$ False

Fixed number of digits → system is too limited

$0.b_1 \ldots b_k$ in binary $\in [0, 1]$

Overcome this limitation: add an exponent

binary number



Sign exponent fractional binary number

$$x = (-1)^{b_{31}} \; 2^{(E-127)} \cdot 1.\overbrace{b_{22} b_{21} \ldots b_0}^{F}$$

↓

increases the range of available numbers

Limitation in precision: only a number of significant digits are available in the computation

float 64 $\simeq$ 16 digits

123.1234567890 1234567890

used part — ignored

Math computations

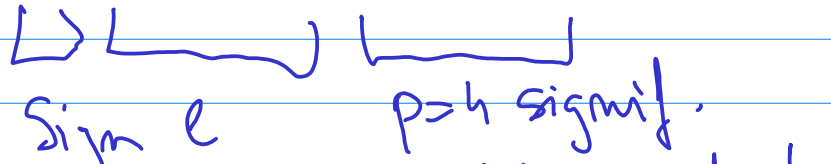
multiplications are handled ok

additions, subtractions → problems when there is a large difference in scale.

$10^{16} + 1 \overset{\text{float64}}{=} 10^{16}$

Reason: need to align the integer and fract. part first.

Toy format

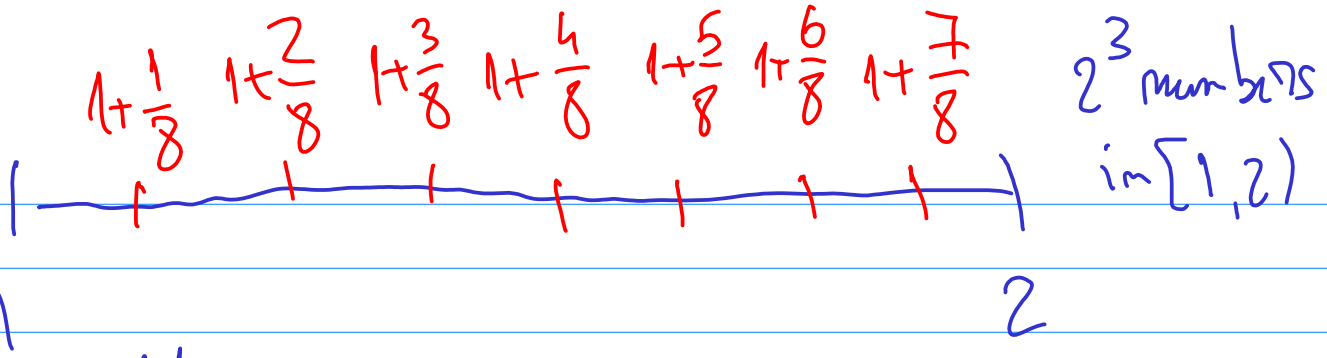


$$x = (1.b_1 b_2 b_3)\, 2^{\ell}$$

All such numbers in $[1, 2)$

$1.b_1 b_2 b_3$

$\frac{1}{2} \quad \frac{1}{4} \quad \frac{1}{8}$



Any floating point number system contains finitely many numbers.

This system "misses" almost all real numbers.

→ All floating point numbers have the form $\frac{n}{2^k}$, $n \in \mathbb{Z}$

Binary exp of $\frac{1}{10} = 0.(\quad)$ periodic exp.

In double precision

$$x = (-1)^s \, 2^{E-1023} \left(1 + \frac{F}{2^{52}}\right)$$

$1 \le E \le 2046 \quad 0 \le F < 2^{52}$

How many floating numbers are there between $[1, 2)$

$S = 0$ +

$E = 1023$

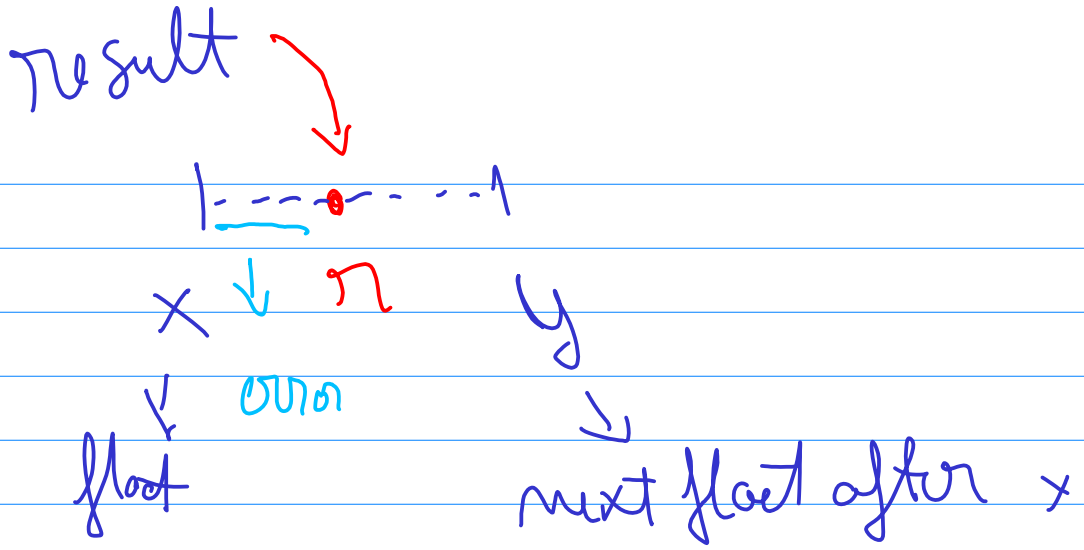
as many numbers as (F) $\sim 2^{52}$

How about in $[2, 4)$?

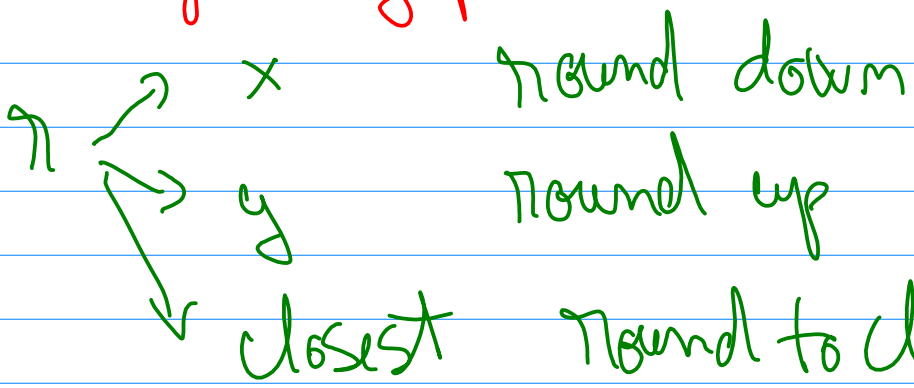
The same, but $E = 1024$

We still have as many numbers as in $[1, 2)$, but the space between them is twice larger.

Gaps in the floating point number system $\Rightarrow$ we need a procedure to "get back" to a floating point number if we fall between numbers



$r \to$ floating point number:



Rounding produces an error.

$r \mapsto fl(r)$ (rounding to nearest)

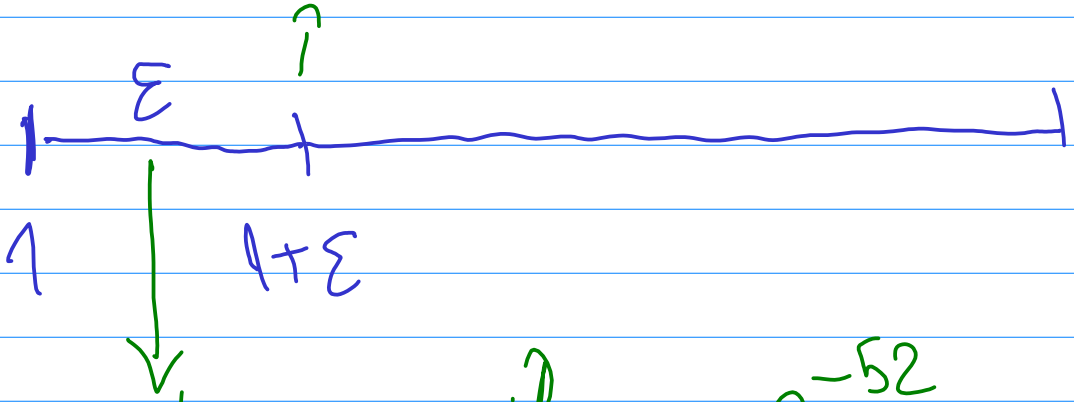
Result: $fl(z) = z + \underbrace{z \cdot \delta}_{\text{error}} = z(1+\delta)$

$|\delta| \leq u = 2^{-p}$

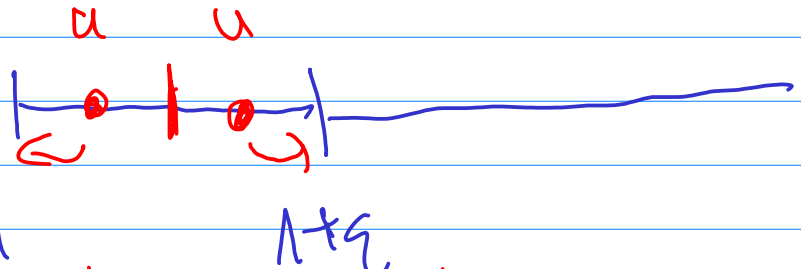
p number of binary digits in fractional part

float 64: $u = 2^{-53}$

Interpret: first binary number in float64 above 1



machine epsilon $= 2^{-52}$ → $u = \frac{\varepsilon}{2}$



Rounding to nearest gives error $\frac{\varepsilon}{2}$

$$fl(z) - z = z \cdot \delta$$

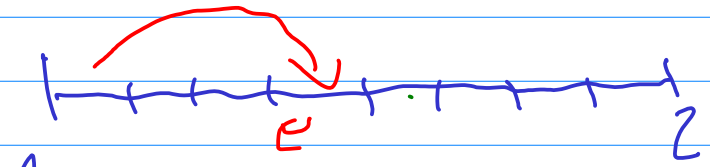
$$|fl(z) - z| = |z| \cdot |\delta|$$ ← abs error

$$\frac{|fl(z) - z|}{|z|} = |\delta| \overset{\text{thm}}{\leq} u$$

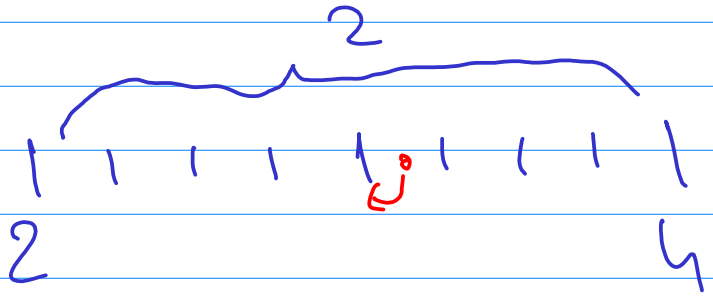
↑ relative error

Rounding to nearest produces relative errors smaller than $\boxed{\frac{\varepsilon}{2} = 2^{-p}}$

3 bit example



Error: $\frac{1}{8} \cdot \frac{1}{2} \simeq \boxed{\frac{1}{16}}$



Abs error: $\frac{2}{8} \frac{1}{2} \simeq \frac{1}{8}$

Rel error: $\frac{1}{8} \cdot \frac{1}{2} = \boxed{\frac{1}{16}}$

Product of 3 numbers

$$fl(fl(fl(ab)c)) = fl(ab(1+\delta_1)c)$$
$$= abc(1+\delta_1)(1+\delta_2)$$
$$= abc(1+\delta_1+\delta_2+\delta_1\delta_2)$$

Rel error : $|\delta_1 + \delta_2 + \delta_1\delta_2|$

$\leq u + u + u^2$

$= 2u + u^2$

Repeating rounding ACCUMULATES errors

Multiple operations involving rounding then the relative error $\simeq u \times$ (Nb of operations)

Conditioning, stability, cancellation

Task: $y = f(x)$

x desired input

stored in the machine

$\tilde{x}$ stored input

$f(\tilde{x})$

$\hat{y}$ algorithm output.

Total error

$$\hat{y} - \underbrace{f(x)}_{\text{Real answer}} = \underbrace{\hat{y} - f(\tilde{x})}_{\text{eval error computation}} + \underbrace{f(\tilde{x}) - f(x)}_{\text{effect of changing input math problem concerning } f}$$

→ A better formula can improve the first term.

Forward error: $|\hat{y} - f(x)|$

Backward error: for what Δx

$$\hat{y} = f(x + \Delta x)$$

Example approx square root.

$a = 2$, $\sqrt{a} \approx \hat{y} = 1.4142$

$$\Delta a = \hat{y}^2 - a = - \ldots$$

$\Delta a + a = \hat{y}^2 \Rightarrow \hat{y} = \sqrt{a + \Delta a}$

↙ backward error.

Forward error: $\sqrt{2} - 1.4142 = \dots$

Conditioning: how outputs change under input perturbations?

- ill conditioned if small perturbations in x lead to large pert in $f(x)$
- well cond: small pert. in $x \rightarrow$ small pert in $f(x)$

Stability: alg. vars.

Example $x = 1 + 10^{-20}$

$f(x) = x - 1$

Stored $x \Rightarrow \tilde{x} = 1$

Computed $f(\tilde{x}) = f(1) = 0$

Error: $10^{-20} = f(x)$ but $f(\tilde{x}) = 0$

Rel error: $\left|\frac{f(x)-f(\tilde{x})}{f(x)}\right| = \frac{10^{-20}}{10^{-20}} = 1$

↓

"large"

Quick illustration: different formula → better result.

$$\sqrt{3}-\sqrt{2} = \frac{(\sqrt{3})^2-(\sqrt{2})^2}{\sqrt{3}+\sqrt{2}} = \frac{1}{\sqrt{3}+\sqrt{2}}$$

$$\sqrt{1+10^{-6}}-\sqrt{1} = \frac{1+10^{-6}-1}{\sqrt{1+10^{-6}}+1} = \frac{10^{-6}}{\sqrt{1+10^{-6}}+1}$$