Floating point numbers

Background: Scientific notation, e.g.

Avogadro constant: $N_A = 6.022 \cdot 10^{23} \ \frac{1}{mol}$

Speed of light: $c = 3.00 \cdot 10^{8} \ \frac{m}{s}$

Advantages:
- Makes it easy to handle quantities of very different orders of magnitude
- "Number of significant digits" easy to trace through computation

Scientific notation can be thought of as "arbitrary precision floating point".

Floating point in hardware (IEEE 754)

$$\pm s \cdot 2^{e} = \pm \underbrace{d_0 . d_1 d_2 \cdots d_{p-1}}_{\text{significant (or mantissa)}} \cdot 2^{\overset{\text{exponent}}{\overset{\downarrow}{e}}}$$

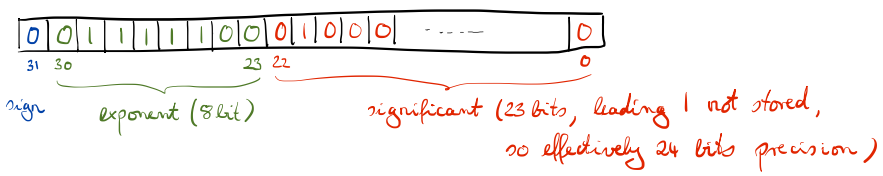
- significant has fixed length of p bits
- exponent has fixed number of bits s.t. $e_{min} \le e \le e_{max}$; store e + bias
- one sign bit
- if possible, "normalize" s.t. $d_0 = 1$, no need to store

E.g. layout for 32-bit IEEE floating point for

$$(0.15625)_{10} = 1.01 \cdot 2^{-3}$$

is computed as follows: Exponent bias per IEEE standard is 127,

so e + bias $= -3 + 127 = 124 = (01111100)_2$



Most common: "double precision" (64 bit) floating point with 11 bit exponent, effective 53 bit precision

Features of the standard:

- Floating point addition $\oplus$, multiplication $\odot$, subtraction $\ominus$, division $\oslash$ are exactly rounded, i.e.

$$fl(x) \oplus fl(y) = fl\left(fl(x) \cdot fl(y)\right)$$

where "fl" denotes "round to floating point"

- Two zeros, +0 and −0, which compare as equals
- NaN ("not a number"): propagate as NaN, except $NaN^0 = 1$
- $\pm$ inf ("infinities"): $1 \oslash +0 = +\text{inf}$, $1 \oslash -0 = -\text{inf}$, $0 \odot \text{inf} = NaN$
- Subnormal numbers $0.d_1 \cdots d_{p-1} \cdot 2^{e_{min}}$

  Needed to ensure that $x = y \Leftrightarrow x - y = 0$ in floating point.

  Expensive, even though handled in hardware on contemporary CPUs.
- Correct rounding (round to nearest, round to even in case of a tie) removes bias
- Lexical sorting is relatively easy

Error measures:

absolute error: If $fl(x) = 1.d_1 d_2 \cdots d_{p-1} \cdot 2^e$, then

$$|x - fl(x)| \leq 0.5\, 2^{1-p} \cdot 2^e$$

$$\equiv 0.5 \text{ ulp} \quad \text{"units in the last place"}$$

relative error: $$\frac{|x - fl(x)|}{|x|} \leq \frac{0.5\, 2^{1-p} \cdot 2^e}{2^e} \qquad (\text{as } |x| \geq 2^e)$$

$$= 0.5\, \varepsilon \qquad \text{where } \varepsilon = 2^{1-p} \text{ is the "machine epsilon"}$$

Remark: ε is the smallest positive floating point number such that $1 \oplus \varepsilon > 1$.

On standard 64-bit floating point, $\varepsilon \approx 2.22 \cdot 10^{-16}$

$$\Rightarrow \tfrac{1}{2}\varepsilon \approx 1.11 \cdot 10^{-16}$$

Error propagation:

(a) $$fl(x) \odot fl(y) = fl(x) \cdot fl(y)\,(1 + \delta_1) \qquad \text{with } |\delta_1| \leq \frac{\varepsilon}{2}$$

$$= x(1+\delta_2)\cdot y(1+\delta_3)\,(1+\delta_1) \quad \text{with } |\delta_2|, |\delta_3| \le \tfrac{\varepsilon}{2}$$

$$= x\cdot y\,\big(1 + (\delta_1+\delta_2+\delta_3) + \text{terms bounded by } \varepsilon^2\big)$$

$$= x\cdot y\,(1+\delta) \quad \text{with } |\delta| \le \tfrac{3}{2}\varepsilon$$

⇒ Multiplication of floats is always well-behaved ("numerically stable")

(b) $$fl(x) \oplus fl(y) = \big(fl(x)+fl(y)\big)(1+\delta_1) \quad \text{with } |\delta_1| \le \tfrac{\varepsilon}{2}$$

$$= \big(x(1+\delta_2) + y(1+\delta_3)\big)(1+\delta_1) \quad \text{with } |\delta_2|, |\delta_3| \le \tfrac{\varepsilon}{2}$$

$$= \big(x+y+x\delta_2+y\delta_3\big)(1+\delta_1)$$

$$= x+y+(x+y)\delta_1 + x\delta_2 + y\delta_3 + \text{quadratic terms}$$

$$= (x+y)\Big(1+\underbrace{\delta_1 + \tfrac{x}{x+y}\delta_2 + \tfrac{y}{x+y}\delta_3}_{=:\delta}\Big) + \dots$$

⇒ Addition of same-signed numbers is well-behaved: $|\delta| \le \tfrac{3}{2}\varepsilon$)

Addition of different-signed numbers can have huge amplification of relative error. Worst case: $x \approx -y$ ▽

(c) Suppose x, y, z are already floating point values. Then

$$(x \oplus y) \oplus z = \big((x+y)(1+\delta_1) + z\big)(1+\delta_2) \qquad |\delta_1|, |\delta_2| \le \tfrac{\varepsilon}{2}$$

$$= \big(x+y+z + (x+y)\delta_1\big)(1+\delta_2)$$

$$= x+y+z + (x+y+z)\delta_2 + (x+y)\delta_1 + \text{quadratic terms}$$

$$= (x+y+z)\Big(1+\delta_2 + \tfrac{x+y}{x+y+z}\delta_1\Big) + \dots$$

- If $z \approx -(x+y)$, catastrophic loss of accuracy may occur, as above.
- For same-signed numbers, repeated addition adds at most $\frac{\varepsilon}{2}$ error per term. But it helps to add smallest numbers first!

Problematic example: $$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

(The $\frac{1}{h}$ converts big relative errors to big absolute errors!)

Mitigation:

① Rule of thumb: never test floating-point numbers for equality

E.g. `a = 0.15 + 0.15`

`b = 0.1 + 0.2`

(i) `a == b` gives False in Python. Not a good idea.

(ii) `abs(a-b) <= tol` Better, but absolute tolerance is wrong concept

(iii) `abs(a-b)/abs(a) <= tol` Usually works, but there are edge cases

(iv)
```
def almost-equal(a,b):
    if a == b:
        return True                    (Handles infinities!)
    else if a==0 or b==0 or abs(a) + abs(b) < min_normal:
        return abs(a-b) < tol * min_normal
    else return abs(a-b)/min(abs(a)+abs(b), max_float) < tol
```

② Rewrite formulas (often, but not always possible):

(i) $a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n = a_0 + x(a_1 + x(a_2 + \dots (a_n x)))$ (Horner's scheme)

(ii) $x_\pm = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ (quadratic formula)

Problematic case: $4ac \ll b^2$. Rewrite, for $b > 0$,

$$x_+ = \frac{\left(-b + \sqrt{b^2-4ac}\right)\left(-b - \sqrt{b^2-4ac}\right)}{\left(-b - \sqrt{b^2-4ac}\right)2a}$$

$$= \frac{b^2 - (b^2 - 4ac)}{\left(-b - \sqrt{b^2-4ac}\right)2a} = \frac{2c}{b + \sqrt{b^2-4ac}}$$

No loss of significant digits!