Monahan Chapter 2

Representing numbers on a computer.

Our mathematical number systems are infinite

Computers are finite.

* Computers do not represent numbers beyond a certain magnitude
  ⟶ overflow / underflow errors
* There are gaps between representable numbers
  ⟶ rounding errors

Floating point number systems

Digital computers use a binary number system

Let's ignore that for a moment, describe a decimal system

This is a decimal representation of a number

6743.127

What does it mean?

$6 \cdot 1000 + 7 \cdot 100 + 4 \cdot 10 + 3 \cdot 1 + 1 \cdot 0.1 + 2 \cdot 0.01 + 7 \cdot 0.001$

$6 \times 10^{3} + 7 \times 10^{2} + 4 \times 10^{1} + 3 \times 10^{0} + 1 \times 10^{-1} + 2 \times 10^{-2} + 7 \times 10^{-3}$

To store this number, we would need to write down the 8 symbols (7 digits + •) in order.

In "scientific" notation

6.743127×10^{3}

The stuff in orange tells us the 7 digits, and the exponent tells us how to shift the decimal.

* this system has a "floating" decimal point.

One more thing: need to store the sign if we want negatives

In total, we need 3 quantities

S = sign (positive or negative)

E = exponent (where to put the decimal)

F = fraction (string of digits)

$6743.127 \rightarrow (S, E, F) \rightarrow (+, 3, 6743127)$

Why not just store "6743.127" and be done with it?

This could be wasteful for very small or large numbers

$(+, 18, 4362)$ vs "+436200000000000000."

$(-, -15, 314)$ vs "-.000000000000000314"

Floating point system balances the tradeoff between representing:

* a wide range of numbers $\longrightarrow$ to avoid overflow/underflow errors
* a fine grid of numbers $\longrightarrow$ to avoid rounding errors

<u>Review of binary numbers</u>

works the same way as decimals, but replace 10 with 2

$$1011.101_{two}$$

$$= (1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3})_{ten}$$

$$= (8 + 2 + 1 + 0.5 + 0.125)_{ten} = (11.625)_{ten}$$

The IEEE double precision floating point system.

We store S = sign with 1 binary digit

E = exponent with 11 digits

F = fraction with 52 digits

$$\text{number} = (-1)^S (1.F)(2)^{E-1023}$$

Example in binary:

$$(1, 101000\cdots0, 10000000001)$$

$$= (-1)^1 (1 + 1/2 + 1/8) \times (2)^{1025-1023}$$

$$= -(1.625) \times (2)^2$$

$$= -6.5$$

Consequence: fractions of powers of 2 can get exact representations

Condition of a task + stability of an algorithm

Most numerical tasks (problems) can be abstracted as:

$$\text{output} = f(\text{input})$$

The **condition** of the task is its sensitivity to perturbations of the input:

$$\text{condition} = \frac{|f(\text{input} + \delta) - \text{output}|}{\delta} \cdot \frac{\text{input}}{\text{output}}$$

An algorithm is a specific method for calculating $f(\text{input})$

$$\text{algorithm } f^*(\text{input}) \approx f(\text{input})$$

The **stability** of an algorithm measures discrepancies between

$$f^*(\text{input} + \delta) - f(\text{input})$$