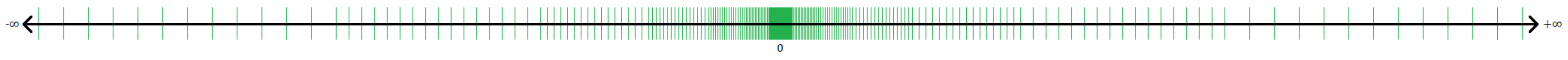
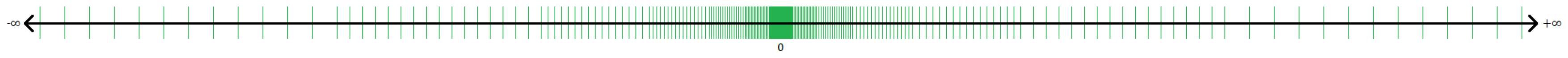
Computer Arithmetic
Number Representation & Errors



Scientific & Exponential Notation

- Distance from Earth to Proxima Centauri
  - 4.2 light years = 24,698,100,000,000 miles
  - 111100011101001110000000011011111010011100011.0000
- Diameter of an average atom 0.25 nanometers
  - 0.000000025 meters
  - 0.0000000000000000000000000000000001011
- NEED A BETTER WAY!
  - Very large
  - Very small

Floating Point Representation for Real Number



Floating Point Representation $m.b^{e}$

m: mantissa or significand
b: base (2, 8, 10, 16)
e: exponent (always an integer)

We can represent very large real numbers

Consider 1/34 = 0.029411765

Assume 4 decimal places is allowed

$1/34 \approx 0.0294 \times 10^{+0}$ $1/34 \approx 0.2941 \times 10^{-1}$

By normalizing the mantissa, we increase precision because leading bit is always 1 so no need to store it

$\frac{1}{b} \leq m < 1$

Floating Point Representation

Real number set $\mathbb{R}$ is infinite, continuous, and unbounded
But computers have a finite precision (i.e., 32 bit, 64 bit)

Floating point representation is based on exponential (scientific) notation

$$\pm(S) \times 10^E, where\ 1 \leq S < 10, E\ (exponent\ is\ an\ integer)$$

2.345×10^2 (decimal system)

On computers, a binary system is used to represent real numbers

$$\pm(S) \times 2^E, where\ 1 \leq S < 2$$

S is the *significand* or *mantissa*

$$S = \pm(1 + f)$$

$$f = \sum_{i=1}^{p} b_i 2^{-i}$$

where p is the precision, b is the binary digit $\in \{0,1\}$

$$71 = (1.000111)_2 2^6 \qquad \left(1 + 0 \cdot \frac{1}{2^1} + 0 \cdot \frac{1}{2^2} + 0 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} + 1 \cdot \frac{1}{2^5} + 1 \cdot \frac{1}{2^6}\right) \times 2^6$$

$$f = \sum_{i=1}^{p} b_i 2^{-i} = 2^{-p} \sum_{i=1}^{p} b_i 2^{p-i} = 2^{-p} z \qquad where\ z \in \{0,1,\ldots,2^p - 1\}$$

substitute into $\pm(1+f) \times 2^E$

There are 2^p evenly spaced numbers in the interval $[2^E, 2^{E+1})$

We observe that the smallest number larger than 1.0 is $1 + 2^{-p}$

The precision *p* sets the machine epsilon $\epsilon_m = 2^{-p}$

Exercise: Show that the distance between floating point numbers 2^E and 2^{E+1} is $2^E \epsilon_m$

Define a rounding function $fl(x)$ which is the mapping of a real number x to the nearest member of $\mathbb{F}$ (round(2.3)=2.0, round(2.7)=3.0)

$$|fl(x) - x| \leq \tfrac{1}{2} 2^{n-p}$$

Note: ½ factor is due to the rounding function

$$\frac{|fl(x) - x|}{|x|} \leq \frac{2^{n-p-1}}{2^n} \leq \frac{1}{2}\epsilon_m$$

Toy problem

Assume a 7-bit system. Use first bit for the sign of the number, next three digits for the sign and magnitude of the exponents, and the last three digits for the mantissa

Smallest positive number: 0 1 1 1 1 0 0 : 0.0625 $\quad +m.\,b^{-e}$

m is normalized $\frac{1}{b} \leq m < 1$ so smallest m in binary is 100

Next largest number: increment mantissa 0 1 1 1 1 0 1 : 0.078125

Next floating-point number 0 1 1 1 1 1 0 : 0.093750

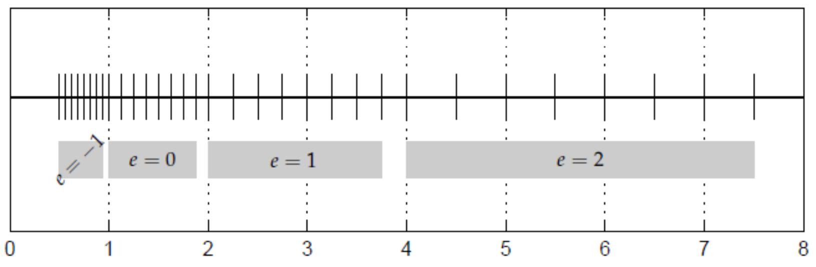
Next floating-point number 0 1 1 1 1 1 1 : 0.109375

Decrease exponent to keep increasing the number toward +∞

0 1 1 0 1 0 1 : 0.125000

0 1 1 0 1 1 0 : 0.156250

0 1 1 0 1 1 1 : 0.187500



Largest positive number in the toy problem: 0 0 1 1 1 1 1 : 7.0

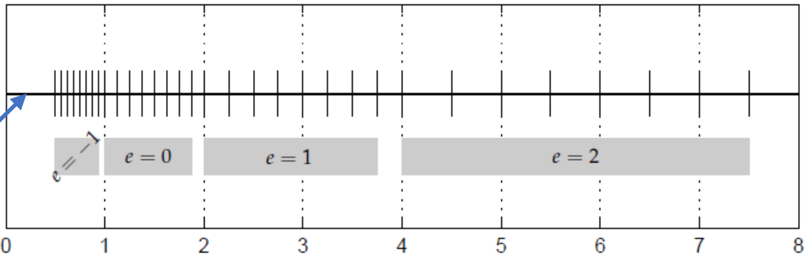
Overflow error: attempt to represent large numbers outside the acceptable range

Underflow error: attempt to represent small numbers outside the acceptable range

We can have smaller numbers than the smallest number in the IEEE 754 standard but with less precision

Subnormals & Gradual underflow

Why the gap larger?



When we normalized the mantissa, we did not store the leading bit 1 because it is always 1 why waste a bit. However, if we let it to be zero, we can squeeze in more floating-point numbers, but with reduced precision

Subnormals do NOT make machine epsilon ϵ_m smaller

Exceptional Values

$$\frac{1}{0} \rightarrow inf$$

$$\frac{0}{0}, 0 * inf, inf/inf \rightarrow NaN\ (not\ a\ number)$$

Support for Inf, NaN depends on the language, compiler, run-time system

Precision & Significant Digits

- Consider Planck's constant given by 6.626068×10^{-34} m² kg/s. If we change the last digit by 1, then the relative change is

$$\frac{0.000001 \times 10^{-34}}{6.626068 \times 10^{-34}} \approx 1.51 \times 10^{-7}$$

- The relative error is about 10^{-7}, so we can say that the original number had 7 significant digits or precision of the floating-point number is 7 decimal digits.
- More generally,

$$digits = \log_{10}\left|\frac{\tilde{x} - x}{x}\right|$$

- Precision does NOT mean accuracy
  - Any wrong answer can be represented with any given precision

Accuracy

Absolute accuracy: $|\tilde{x} - x|$

Relative accuracy: $\frac{|\tilde{x}-x|}{|x|}$

Advice: In your publications, technical reports be consistent with your precision!

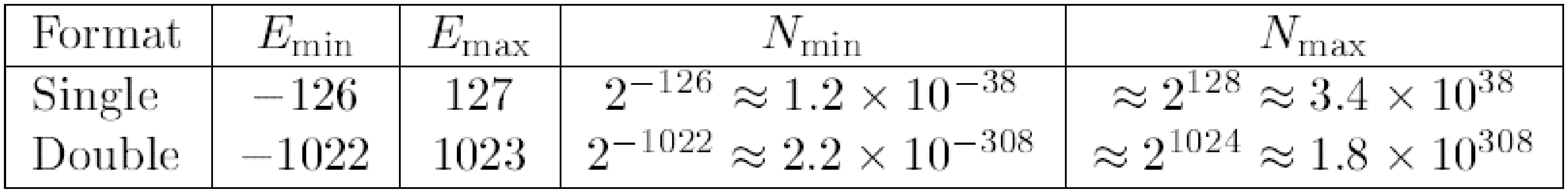
Bad example:

length = 1.34567889 m
width = 2.34 m

Double Precision

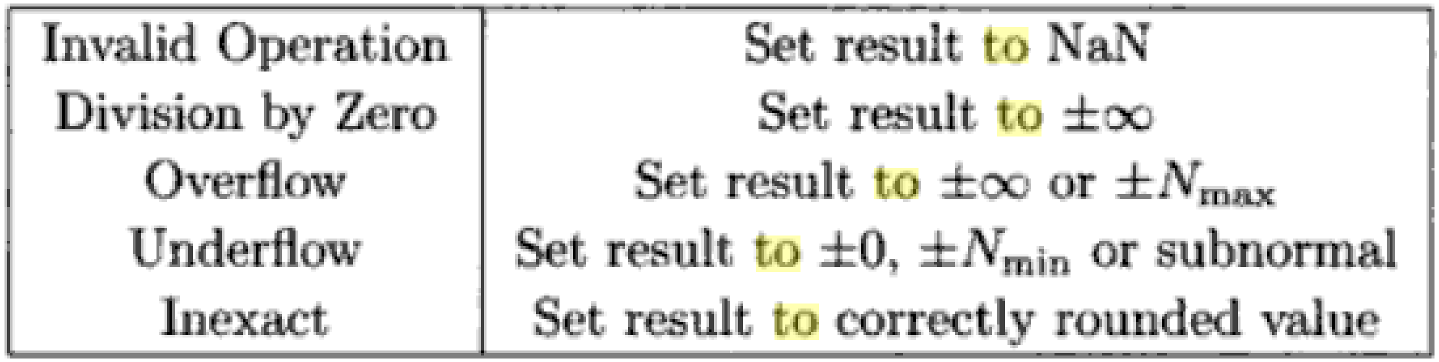
- IEEE standard 754 specifies how to store so-called double precision numbers
- 64 bits per number, d=52 digit mantissas, 11 digits for exponent e, and a sign bit.
  - 32 bits *d=23* binary digits for mantissa
- In this case, $\epsilon_M = 2^{-52} \approx 2.2 \times 10^{-16}$; this is about 16 digits
- Biggest number: $2^{1024} \approx 2 \times 10^{308}$
- If bigger, “overflow”
- Smallest number: $2^{-1022} \approx 2 \times 10^{-307}$
- If smaller, “underflow”

Range IEEE Floating Point Formats



Format	$E_{\min}$	$E_{\max}$	$N_{\min}$	$N_{\max}$
Single	-126	127	$2^{-126} \approx 1.2 \times 10^{-38}$	$\approx 2^{128} \approx 3.4 \times 10^{38}$
Double	-1022	1023	$2^{-1022} \approx 2.2 \times 10^{-308}$	$\approx 2^{1024} \approx 1.8 \times 10^{308}$

IEEE 754 Standard Response to Exceptions



Invalid Operation	Set result to NaN
Division by Zero	Set result to $\pm\infty$
Overflow	Set result to $\pm\infty$ or $\pm N_{\max}$
Underflow	Set result to ± 0, $\pm N_{\min}$ or subnormal
Inexact	Set result to correctly rounded value

Subtractive Cancellation

Consider two nearly equal large numbers *b* & *c*

Subscript _c is for computer representation of the number

$$a_c \approx b_c - c_c$$

$$a_c \approx b(1+\epsilon) - c(1+\epsilon)$$

$$a_c \approx b - c + b\epsilon - c\epsilon$$

$$\frac{a_c}{a} \approx 1 + \frac{b}{a}\epsilon - \frac{c}{a}\epsilon$$

$note\ b \approx c$

$$1 + \epsilon \approx 1 + \frac{b}{a}\epsilon$$

a is a small number whereas *b* is a large number

A different approach

- Putting the number x in the computer is $\mathrm{fl}(x) = x(1+\epsilon)$
- We can write that the computer implementation of

$$f(x) = x + 1 \text{ as } y = x(1+\epsilon) + 1$$

Then, the relative error becomes

$$\frac{|y - f(x)|}{|f(x)|} = \frac{|(x + \epsilon x + 1) - (x + 1)|}{|x + 1|} = \frac{|\epsilon x|}{|1 + x|}$$

- For x near -1, the relative error can become very large
- Say we have 5 digits and add -1.0012 to 1; then we get -1.2×10^{-3}
- Only two digits now are correct: subtractive cancellation!

Important source of error!

Condition Number

Condition number $\kappa_f(x)$ is the ratio of the relative error of the output to the relative error of the input.

It does not depend on the data or the computer.

Condition number indicates the magnification of errors in computation $f(x)$

We can measure how bad an operation or problem is with the *condition number*

$$\kappa_f(x) = \frac{\frac{|f(x) - f(\tilde{x})|}{|f(x)|}}{\underbrace{\frac{|x - \tilde{x}|}{|x|}}_{\epsilon}}$$

Let the exact number x become $\tilde{x} = \mathrm{fl}(x) = x(1 + \epsilon)$

- Then considering only changes due to x, one gets

$$\frac{|f(x) - f(x(1+\epsilon))|}{|\epsilon f(x)|}$$

- In the limit of small error $\epsilon \to 0$ (ideal computer)

$$\kappa(x) = \lim_{\epsilon \to 0}\left|\frac{f(x+\epsilon x) - f(x)}{\epsilon f(x)}\right| = \left|\lim_{\epsilon \to 0}\frac{f(x+\epsilon x) - f(x)}{\epsilon x}\cdot\frac{x}{f(x)}\right| = \left|\frac{x f'(x)}{f(x)}\right|$$

If x is perturbed by a small amount, the relative error of representing a function *f(x)* is magnified by its condition number

$$\left|\frac{f(x+\epsilon x)-f(x)}{f(x)}\right| \approx \kappa_f \epsilon$$

A problem is *ill-conditioned* when its condition number is large

Assume you have a function $h(x) = f(g(x))$

Composite condition number: $\kappa_h(x) = \kappa_f\big(g(x)\big) \cdot \kappa_g(x)$

- *Example*: consider $f(x) = x - c$
- Use

$$\kappa(x) = \left| \frac{x f'(x)}{f(x)} \right|$$

- Applying the formula,

$$\kappa(x) = \left| \frac{(x)(1)}{x - c} \right| = \left| \frac{x}{x - c} \right|$$

- The condition number is large when $x \approx c$; conditioning is poor there

- *Example*: Multiplication by constant c, $f(x) = cx.$

Then

$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{(x)(c)}{cx}\right| = 1.$$

No magnification of error!

- *Example*: $f(x) = \cos(x)$:

$$\kappa(x) = \left|\frac{(x)(-\sin x)}{\cos x}\right| = |x \tan x|.$$

The condition number is large when $x = a\pi/2$, where a is an odd integer

Approximations in Scientific Computing

- Example: Calculate the surface area of Earth
  - Approximate Earth as a sphere: Modeling error
    - $Area = 4\pi R^2$
  - $R \approx 6370\ km$: Measurement error
  - $\pi = 3.141592$: Truncation error
  - Rounding errors due finite precision of computer arithmetic

$$\text{Total error} = \hat{f}(\hat{x}) - f(x)$$

$$\text{Total error} = \underbrace{\left[\hat{f}(\hat{x}) - f(\hat{x})\right]}_{\text{Computational error}} + \underbrace{\left[f(\hat{x}) - f(x)\right]}_{\text{Propagated data error}}$$

Stability of Algorithms

Algorithms can also be a source of error

Ill-conditioning may not explain the source of error in a problem

When error in a problem exceeds what can be explained by ill-conditioning, we call the algorithm unstable

Stability does not guarantee accuracy!

Stability refers to the effects of computational error on the results

Conditioning refers to the effects of data error on the results

We get accurate results when we apply a stable algorithm to a well-conditioned problem

$$\begin{tikzcd}
x \arrow[r, "f"] \arrow[dr, "\hat{f}\ \text{(algorithm)}"] \arrow[d, leftrightarrow, "\text{backward error}"'] & y = f(x) \arrow[d, leftrightarrow, "\text{forward error}"] \\
\hat{x} \arrow[r, "f"'] & \hat{y} = \hat{f}(x) = f(\hat{x})
\end{tikzcd}$$

Relative Forward Error: algorithm $\tilde{f}(x)$ for problem $f(x)$ has forward error

$$\frac{\left|\tilde{f}(x) - f(x)\right|}{|f(x)|}$$

Relative Backward Error: Say we can find approximate input data such that

$$f(\tilde{x}) = \tilde{f}(x)$$

Then the backward error is

$$\frac{|\tilde{x} - x|}{|x|}$$

Condition Number
A different perspective

$$\kappa_f(x) = \frac{\dfrac{|f(x) - f(\tilde{x})|}{|f(x)|}}{\dfrac{|x - \tilde{x}|}{|x|}} = \frac{relative\ forward\ error}{relative\ backward\ error}$$

|relative forward error| $\leq \kappa(x) \times$ |relative backward error|

Condition number bounds the forward error