Announcements:

* Assignment 1 is due Monday 9 pm
* MarkUs link is posted on the course website.

Last Class

* Conditioning / sensitivity
* stability
* Error: Absolute, Relative, Forward, Backward.

Today

Floating Point Numbers.
- ↳ Representing real numbers
- ↳ Floating point arithmetic

Floating-Point Numbers

Problem: We need to store real numbers (continuous) using hardware that is discrete.
Ideally, each real number will take up the same amount of storage space.

Solution: Floating-Point numbers!
↳ which are like scientific notation.

surface area of Earth: 510072000 km^2

in scientific notation: $5.10072 \times 10^8 \text{ km}^2$

with fewer precision: $5.101 \times 10^8 \text{ km}^2$

Planck constant : $6.\underbrace{626}_{\text{precision}} \times \underbrace{10}_{\text{base}}^{-34} \text{ m}^2\text{kg/s}$

always one digit before the decimal.

exponent denotes the position of decimal

A Floating Point Number System is characterized by 4 integers:

β – base (radix) ← eg// base 10 for scientific notation; base 2 on most computers.

p – precision

$[L, U]$ – exponent range.

In this system, a real number x is represented as:

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

where $0 \le d_i \le \beta - 1$ $\quad d_i \in \mathbb{N}$

$L \le E \le U$.

eg// surface area of earth $\quad$ ~~0.5101×10^9~~ $\quad 5.101 \times 10^8$ km^2.

$$\left(5 + \frac{1}{10} + \frac{0}{10^2} + \frac{1}{10^3} \right) 10^8$$

The digits $d_0 d_1 \cdots d_{p-1}$ is called the mantissa.

$d_1 \cdots d_{p-1}$ " " " fraction.

E " " " exponent.

eg// IEEE Single Precision System:

$\beta = 2$.

$p = 24$

$L = -126$

$U = 127$

# bits we need		
mantissa	24	23
exponent	8	8
sign	1	1
	33	32

There are 254 different numbers between -126 and 127 (inclusive).

$127 - (-126) + 1 = 254$

Q: What is the largest number we can store using IEEE SP.

$$2^{128} \quad \text{— too large,} \quad 128 > U$$

$$2^{127} - 1 + \Sigma \text{ mantissa} \quad \text{— ?}$$

d_0	d_1	....	d_{p-1}
1	1	....	1

E: $\boxed{127}$ $= \cancel{2^{128} - 2^{128-24}}$

$$\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{23}}\right) \times 2^{127}$$

Normalization:

A Floating-Point System is normalized if (we enforce the additional rule that) the leading digit $d_0 \neq 0$ unless the number represented is 0.

- There is a unique representation for each nonzero value.

  5.101×10^8 ✓

  0.5101×10^9 ✗

- If $\beta = 2$, we don't have to store the first digit.

Q: ~~What~~ How many floating point number are in a normalized floating point system $F(\beta, p, L, U)$

$$\underbrace{2}_{\text{sign}}\underbrace{(\beta-1)\,\beta^{p-1}}_{\text{mantissa}}\underbrace{(U-L+1)}_{\text{exponents}}\underbrace{+1}_{\text{zero}}$$

not all real numbers are exactly representable in our floating-point system.

Numbers that are exactly representable are called <u>machine numbers</u>.

<u>Rounding</u>: $fl(x)$ ~ approximate $x \in \mathbb{R}$ by a "nearby" machine number

→ rounding error is introduced here.

Rounding Rule: 1. Chop. truncate the base β expansion of x after $(p-1)$ digits ~ "Round toward zero"

2. Round to nearest.

find $fl(x)$ <u>nearest</u> to x.

eg// $x = 5.10072 \times 10^8 km^2$

want to represent x in $F(\beta=10, p=4, L=-5000, U=5000)$

<u>Chop</u>: 5.100×10^8

<u>Nearest</u>: 5.101×10^8

<u>Q</u>: What is the smallest normalized floating-point number in $F(\beta, p, L, U)$:

$1 \times \beta^L = \underline{\beta^L}$ ← underflow level (UFL)

... largest: $\underline{(1-\beta^{-p})\beta^{U+1}}$ ← overflow level (OFL)

~~Def~~: The <u>machine precision</u> characterizes the accuracy of the floating point system.

Can be defined in a few ways:

- max. relative error in representing a nonzero real number $\underline{x}$ with $\underline{fl(x)}$
- Smallest ϵ s.t. $fl(1+\epsilon) > 1$.

If we use rounding by chopping — β^{1-p}

nearest — $\frac{1}{2}\beta^{1-p}$

$\underbrace{5.100\overline{9999}}_{\text{we lose all of this.}} \rightarrow 5.100$ (in $F(\beta=10, p=4)$)

10^{-3}

When we plot floating point (machine) numbers, there is a "gap" around 0. due to normalization.

To remove this gap, we'll relax normalization and allow leading zeros ($d_0 = 0$) when $E = L$

These new floating point numbers are called <u>Subnormal</u> or <u>denormalized</u> F.P. numbers.

This augment system exhibit <u>gradual underflow</u>.

Floating Point Arithmetic

→ Floating Point Arithmetics is inexact

Addition / Subtraction: shift the mantissa to match exponents before adding / subtracting

$x = 1.924 \times 10^2$
$y = 6.357 \times 10^{-1}$

in $F(\beta = 10, p = 4)$

$$
\begin{array}{rl}
x+y: & 1.924 \times 10^2 \\
& 6.357 \times 10^{-1} \\
\hline
& \boxed{1.930}357 \times 10^2
\end{array}
$$

we will lose this information due to rounding

Multiplication: product of two f.p. numbers with precision p will have $2p$ digits

Division: quotient can have many more digits
$1 \div 3$ has infinite # digits in base 10.

Possible Issues

1. Loss of accuracy.
2. Overflow – exponent > U
3. Underflow – exponent < L. (silently set to zero)

eg// $\sum_{n=1}^{\infty} \frac{1}{n}$ should diverge, but converges in F.P. arithmetic.

1. overflow
2. $\frac{1}{n}$ eventually underflow
3. $\frac{1}{n}$ becomes insignificant compared to the Σ so far

example of 3: in $F(\beta = 10,\ p = 4)$

(*) $x = 1.924 \times 10^2$

$y = 6.357 \times 10^{-4}$

$x + y = x.$

extension of (*):

You can imagine ϵ where $1 + \epsilon = 1$

So $(1+\epsilon) + \epsilon = 1.$

but $1 + (\epsilon + \epsilon) > 1.$

$\Rightarrow (1+\epsilon) + \epsilon \neq 1 + (\epsilon + \epsilon)$

$\Rightarrow$ Floating Point Addition is not associative

$$(a+b)+c \neq a+(b+c)$$

So the order that you add numbers matter.

Normal laws of arithmetic don't hold.

Ideally: a floating point operation flop (F.P. addition)
and its arbitrary precision op. (real addition)
should have:

$$fl(x) \text{ flop. } fl(y) = fl(x \text{ op } y)$$

Cancellation: subtracting two p-digit numbers with similar sign & magnitude yield result with much fewer than p digits:

$$\begin{array}{rl} & 1.92403 \times 10^2 \\ - & 1.92275 \times 10^2 \\ \hline & \underbrace{1.28}\,000 \times 10^{-1} \end{array} \quad (p=6,\ \beta=10)$$

3 significant digits!

eg// ϵ where $0 < \epsilon < \epsilon_{mach}$ ← machine precision $\begin{cases} \beta^{1-p} & \text{chop} \\ \frac{1}{2}\beta^{1-p} & \text{near} \end{cases}$

$$(1+\epsilon) - (1-\epsilon) = 1 - 1 = 0.$$

↳ slightly different issue than cancellation. but still shows order of operations matter

we don't want to compute small quantities as a difference of 2 large quantities.

eg// standard deviation, need to compute.

$$\sum_{i=1}^{N} (x_i - \bar{x})^2 = \underbrace{\underbrace{\sum_{i=1}^{N} x_i^2}_{\text{large \#}} - \underbrace{N\bar{x}^2}_{\text{large \#}}}_{\text{difference is small compared to the 2 values}}.$$

x_i: data point

$\bar{x}$: mean

Catestrophic cancellation

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

For $x < 0$ this gives distrastrous results because of cancellation.

For example, e^{-40} is small, but if we try to compute

$$e^{-40} = 1 + (-40) + \frac{(-40)^2}{2!} + \frac{(-40)^3}{3!} + \cdots$$

Each term is <u>large</u>, and adjacent terms have opposite signs, leading to large errors.