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: 510,072,000 km²
in scientific notation: $5.10072 \times 10^8$ km²
with fewer precision: 5.101 \times 10^8 km²

Planck constant: 6.626 \times 10^{-34} \text{ m}^2 \text{ kg/s}
always one digit precision before the decimal.
A Floating Point Number System is characterized by 4 integers:

- $\beta$ - base (radix)
- $p$ - precision
- $[L, U]$ - exponent range.

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

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

where $0 \leq d_i \leq \beta - 1$, $d_i \in \mathbb{N}$, $L \leq E \leq U$.

For example, the surface area of earth $S$ is $5.101 \times 10^8$ km$^2$.

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

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

$E$ is called the **exponent**.

For example, IEEE Single Precision System:

- $\beta = 2$
- $p = 24$
- $L = -126$
- $U = 127$

The numbers $-126$ and $127$ are represented with

- 33 bits
- 32 bits

So we need

- 24 bits for mantissa
- 8 bits for exponent

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.

\[ 2^{128} \text{ too large, } 2^{128} > U \]

\[ 2^{127} - 1 + \sum \text{ mantissa} \]

\[
\begin{array}{cccc}
\text{d}_{0} & \text{d}_{1} & \ldots & \text{d}_{p-1} \\
1 & 1 & \ldots & 1 \\
\end{array}
\]

\[ E = 127 \]

\[
(1 + \frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^{127}}) \times 2^{127}
\]

**Normalization:**

A Floating-Point System is normalized if the leading digit do not unless the number represented is 0.

- There is a unique representation for each nonzero value.

- If \( B = 2 \), we don't have to store the first digit.

Q: How many floating point number are in a floating point system \( F(B, p, L, U) \)

\[
2(B-1) \beta^{p-1} \cdot (U-L+1) + 1
\]

- **Sign**
- **Mantissa**
- **Exponents**
- **Zero**
not all real numbers are exactly representable in our floating-point system.

Numbers that are exactly representable are called machine numbers.

Rounding: \( f_l(x) \approx \) approximate \( x \) EIR by a "nearby" machine number

Rounding Rule: 1. Chop. Truncate the base \( \beta \) expansion of \( x \) after \( (p-1) \) digits \( \approx "Round \ toward \ zero" \)

2. Round to nearest. Find \( f_l(x) \) nearest to \( x \).

eg: \( x = 5.10072 \times 10^8 \) Km²

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

Chop: \( 5.100 \times 10^8 \)

Nearest: \( 5.101 \times 10^8 \)

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

\[ 1 \times \beta^L = \frac{\beta^L}{\beta^{u+1}} \leftarrow \text{underflow level (UFL)} \]

\[ (1-\beta^{-p}) \beta^U \leftarrow \text{overflow level (OFL)} \]

Def: The machine precision characterizes the accuracy of the floating point system.
Can be defined in a few ways:
- Max. relative error in representing a nonzero real number \( x \) with \( \text{fl}(x) \)
- Smallest \( \epsilon \) s.t. \( \text{fl}(1+\epsilon) > 1 \).

If we use rounding by chopping — nearest —

\[
5.1009999 \\rightarrow 5.100 \\
\text{in } F(p=10, p=4)
\]

we lose all of this.

\( 10^{-3} \)

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 \( E = \_ \).

These new floating point numbers are called subnormal or denormalized F.P. numbers.

This augment system exhibit gradual underflow.
Floating Point Arithmetic.

Floating Point Arithmetics is inexact.

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

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

\[ x + y = \frac{1.924}{6.357} \times 10^1 \]

In \( \mathbb{F}(\beta=10, \ p=4) \)

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)

\[ \sum_{n=1}^{\infty} \frac{1}{n} \]

should diverge, but converges in F.P. arithmetic.
Example 4.3: \( \text{in } F(\beta = 10, \ p = 4) \)

\[
\begin{align*}
n & = 1.924 \times 10^2 \\m & = 6.357 \times 10^{-4} \\n + m & = n.
\end{align*}
\]

Extension of 4.3:
You can imagine \( 1 + \epsilon \) where \( 1 + \epsilon = 1 \)

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

\[\text{but } \Rightarrow 1 + (\epsilon + \epsilon) > 1.\]

\[\Rightarrow (1 + \epsilon) + \epsilon \neq 1 + (\epsilon + \epsilon)\]

\[\Rightarrow \text{Floating point addition is not associative}\]

\[\text{(a + b) + c} \neq a + (b + c)\]

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:

\[\text{fl}(x) \text{ flop. fl}(y) = \text{fl}(x \text{ op } y)\]
Cancellation: subtracting two p-digit numbers with similar sign & magnitude yield result with much fewer than p digits:

\[ 1.92403 \times 10^2 \quad (p = 6) \beta = 10 \]

\[- 1.92275 \times 10^2 \]

\[ \frac{1.28000 \times 10^{-1}}{\text{3 significant digits!}} \]

eg// \( \epsilon \text{ where } 0 < \epsilon \leq \epsilon_{\text{mach}} \text{ machine precision } \approx \frac{\beta^{1-p}}{\beta^{1-p}_{\text{near}}} \)

\[ (1 + \epsilon) - (1 - \epsilon) = 1 - 1 = 0. \]

A 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.

\[ \sum_{i=1}^{N} (x_i - \bar{x})^2 = \sum_{i=1}^{N} x_i^2 - N \bar{x}^2. \]

\[ \text{large #} \]

\[ \text{large #} \]

\[ \text{difference is small compared to the 2 values} \]

\[ \text{data point} \]

\[ \text{Mean} \]
Catastrophic cancellation

\[ e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \]

For \( x < 0 \) this gives disastrous results because of cancellation.

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

\[ e^{-40} = 1 + (-40) + \frac{(-40)^2}{2!} + \frac{(-40)^3}{3!} + \ldots \]

Each term is large, and adjacent terms have opposite signs, leading to large errors.