Unit 1:

Floating-Point Numbers

How does the inaccuracy of computer arithmetic affect the way we do computations?

FP-01: FP Problem

Goal: To see that computation on a computer can be inaccurate, even if the math is correct.

Things don't always add up!

https://cs370forensic.blogspot.ca/

Floating-Point Blues

Suppose we need to compute the integral

$$I_n = \int_0^1 \frac{x^n}{x+\alpha}\,dx$$

For a given real number α and integer n, $n \geq 0$.
This is tough to do, except for this trick...

$$I_n = \int_0^1 \frac{x^n}{x+\alpha}\,dx$$

$$= \int_0^1 \frac{x^n + x^{n-1}\alpha - x^{n-1}\alpha}{x+\alpha}\,dx$$

$$= \int_0^1 x^{n-1}\frac{x+\alpha}{x+\alpha} - \alpha\frac{x^{n-1}}{x+\alpha}\,dx$$

$$= \int_0^1 x^{n-1}\,dx - \alpha\int_0^1 \frac{x^{n-1}}{x+\alpha}\,dx$$

$$= \frac{1}{n} - \alpha I_{n-1} \quad \text{WOW!}$$

Thus, $I_n = \frac{1}{n} - \alpha I_{n-1}$ (recurrence relation)
Notice that I_0 is easy

$$I = \int_0^1 1 \ldots$$

$$I_0 = \int_0^1 \frac{1}{x+\alpha}\,dx = \ln(x+\alpha)\Big|_0^1 = \ln(1+\alpha) - \ln\alpha = \ln\frac{1+\alpha}{\alpha}$$

Cool! Let's try it out.
Create a Matlab script (text file with extension .m).

Comments →
```
% Try alpha values of 0.5 and 2.
```
Initialize params →
```
alpha = 0.5;
N = 100;
```
I_0 →
```
I = log((1+alpha) / alpha);
```
$I_n = \frac{1}{n} - \alpha I_{n-1}$
```
for n = 1:N
   I = 1/n - alpha * I;
end
```
Print result →
```
disp(['Answer: ' num2str(I)]);
```

For $\alpha = 0.5$ ⇒ answer = 0.0066444

For $\alpha = 2$ ⇒ answer = 6.058×10^{12}

Hmmm... seems strange.
Observation: If $0 \le x \le 1$ and $\alpha > 1$, then $\frac{x^n}{x+\alpha} \le x^n$

Hence, $I_n = \int_0^1 \frac{x^n}{x+\alpha}\,dx \le \int_0^1 x^n\,dx = \frac{1}{n+1}$

So, for $\alpha = 2$, we should get $I_{100} \le \frac{1}{101}$.

Note: Arithmetic on a computer uses truncated numbers.
Thus, we can have a small error in every number.

$T^{(comp)} = T^{(exact)}$ + ε

instead I_0

and $I_n^{(comp)} = I_n^{(exact)} + e_n$ ← tiny error; ← error at step n

Using our recurrence relation,

$$I_n^{(exact)} = \frac{1}{n} - \alpha I_{n-1}^{(exact)} \quad \text{(mathematical)}$$

$$I_n^{(comp)} \approx \frac{1}{n} - \alpha I_{n-1}^{(comp)} \quad \text{(computational)}$$

Then, $e_n = I_n^{(comp)} - I_n^{(exact)}$

$$= \left(\frac{1}{n} - \alpha I_{n-1}^{(comp)}\right) - \left(\frac{1}{n} - \alpha I_{n-1}^{(exact)}\right)$$

$$= -\alpha\left(I_{n-1}^{(comp)} - I_{n-1}^{(exact)}\right)$$

$$e_n = -\alpha e_{n-1}$$

That is, $e_n = \alpha^2 e_{n-2}$

$$= \alpha^3 e_{n-3}$$

$$= \vdots$$

$$= \alpha^n e_0$$

$$\Rightarrow |e_n| = |\alpha|^n |e_0|$$

If $|\alpha| < 1 \Rightarrow |e_n| \to 0$ as $n \to \infty$ (Good)

If $|\alpha| > 1 \Rightarrow |e_n| \to \infty$ as $n \to \infty$ (Bad)

So there seems to be a build-up of round-off errors, but only

So there seems to be a build-up of round-off errors, but only when $|\alpha| > 1$.

Another example:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Suppose we use only 5 digits of accuracy.

$$e^{-5.5} = 1 - 5.5 + 15.125 - 27.729 + \cdots \quad (25 \text{ terms})$$

$$= 0.0026363$$

Mathematically, it's equivalent to

$$\frac{1}{e^{5.5}} = \frac{1}{1 + 5.5 + 15.125 + 27.729 + \cdots}$$

$$= \boxed{0.0040865}$$

It's not just what you compute, but how you compute it.

<u>Take-Home Message</u>

We follow some basic rules when doing arithmetic and mathematics. For example:

1) $(a+b)+c = a+(b+c)$
2) $a+e = a \Rightarrow e = 0$
3) $\frac{a+b}{c} = \frac{a}{c} + \frac{b}{c}$
4) Correct mathematical algorithms produce correct answers.

Once you do arithmetic using floating-point numbers, none of the above are true.

FP-02: Floating-Point Numbers

Goal: To learn how computers represent real numbers.

Patriot Missile Disaster

https://www.ima.umn.edu/~arnold/disasters/patriot.html

Computer Arithmetic

A computer has two basic strategies for representing numbers:

1) Fixed-point (for integers)
2) Floating-point (for real numbers)

Normal Form of Number

eg. $12.34 = 0.1234 \times 10^2$ ← exponent

mantissa, base

Any number can be represented by a (possibly infinite) expansion base β in the normalized form

$$\pm 0.d_1 d_2 \ldots \times \beta^p$$ ← integer exponent

digits base-β

i.e. $d_k \in \{0, 1, \ldots, \beta-1\}$

$d_1 \neq 0$ (normalized form)

Examples:

1) $\beta = 10 \quad x = \pi$

$x = 0.31415926535897932384626433832 \ldots \times 10^1$

2) $\beta = 2 \quad x = 9 + \frac{1}{3}$

$9_{10} = 8_{10} + 1_{10} = 1000_2 + 1_2 = 1001_2$

How do we represent $\frac{1}{3}$ in binary?

$0.\dot{3} \times 2 = 0.\dot{6} = \boxed{0} + 0.\dot{6}$

$0.\dot{6} \times 2 = 1.\dot{3} = \boxed{1} + 0.\dot{3}$

$0.\dot{3} \times 2 = 0.\dot{6} = \boxed{0} + 0.\dot{6}$

$\boxed{1}$

$\hookrightarrow 0.0101\overline{01}\ldots_2$

$\therefore \left(9 + \frac{1}{3}\right)_{10} = 1001.0101\overline{01}\ldots_2$

$= 0.10010101\ldots_2 \times 2^4$

A floating-point number system has to limit:

1) Density: it keeps only a finite number of digits (t) in the mantissa.
2) Range: finite number of integers for the exponent (p)

i.e. $L \le p \le U$

Thus, a floating-point number system (FPNS) can be characterized by four values, (t, β, L, U), so that any non-zero values have the form,

$$\pm 0.d_1 d_2 \cdots d_t \times \beta^p$$

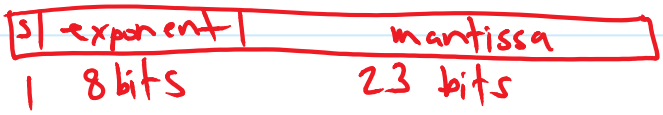
where $d_1 \neq 0$, $d_i \in \{0, 1, \ldots, \beta - 1\}$ $i = 1, \ldots, t$,

$L \le p \le U$

Example:

IEEE Single Precision

$\beta = 2 \quad t = 23 \quad L = -126 \quad U = 127$



IEEE Double Precision

$\beta = 2 \quad t = 52 \quad L = -1022 \quad U = 1023$

$$\pm 1 . d_1 d_2 d_3 \_ \_ \_ \times 2^{D}$$

FP-03: Finite Set of Floating-Point Numbers

Goal: To learn how we represent an uncountably infinite set of numbers in a finite-state machine.

Two Limitations of Floating-Point Representation

Effect of Finite Precision (mantissa)

We're forced to round off.

eg. $\pi \approx (-1)^0 \cdot (3.14) \cdot 10^0$ for $t=3$, $\beta=10$

We represent the approximated value for x as $fl(x)$.

eg. $fl(\pi) = 3.14$

The difference is called the "round-off error".

$$|\pi - fl(\pi)| = 0.0015926\ldots$$

In general, for FPNS (t, β, L, U),

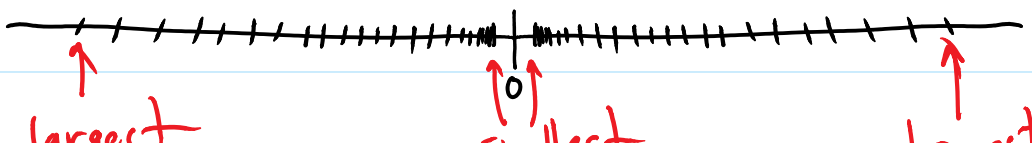
$$x = 0.d_1 d_2 \cdots d_t d_{t+1} \cdots \times \beta^p$$

$d_{t+1} < \frac{\beta}{2}$ $\qquad$ $d_{t+1} \geq \frac{\beta}{2}$

$fl(x) = 0.d_1 d_2 \cdots d_t \times \beta^p$ $\qquad$ $fl(x) = 0.d_1 d_2 \cdots (d_t+1) \times \beta^p$

Effect of Limited Exponent

The range of floating-point numbers is large, but still finite.



largest 0 smallest largest

For example, the largest value in the number system

$(\beta = 10, t = 8, L = -35, U = 35)$ is

$9.99999999 \times 10^{35}$

Anything larger cannot be represented, and causes an "overflow" i.e. $p > U$

```
>> 2^1023
>> 2^1024
```

Anything too small is called "underflow" and gets rounded to zero. i.e. $p < L$

```
>> realmin    = 2^-1022
>> realmin/2
>> realmin/2^52   ✓
>> realmin/2^53   = 0
```

Exception Handling

What do these return?

```
>> 0/0
```

NaN "Not a Number"

```
>> 1/0
```

Inf

```
>> -1/0
```

-Inf

```
>> 1/0 - 1/0
```
NaN

FP-04: Error of Floating-Point Representation

Goal: To see a useful way to track the round-off error through a computation.

Error of Floating-Point Representation

Let $\bar{x}$ be an approximation to x i.e. $\bar{x} = fl(x)$

Absolute Error

Abs. Err. $= |x - \bar{x}|$ eg. $x = 100$ $\bar{x} = 101$

Abs. Err. $= 1$

Relative Error

Rel. Err. $= \dfrac{|x - \bar{x}|}{|\bar{x}|}$ eg. Rel. Err. $= \frac{1}{100} = 0.01$ or 1%

> The relative error of $fl(x)$ for x is bounded for all x in the exponent range.

The maximum relative error is called "machine epsilon", E.

i.e. $\dfrac{fl(x) - x}{x} = \delta$ where $|\delta| \le E$

$\Rightarrow fl(x) = x(1+\delta)$

Definition of Machine Epsilon

E is defined to be the smallest number such that $fl(1+E) > 1$.

For (t, β, L, U) ...

$1 \quad 2 \quad 3 \quad \dots \quad t$

$1.00 \dots 0 \times \beta^0$

$+ \; 0.00 \cdots 0 \left(\frac{\beta}{2}\right) \times \beta^0 \le E$

$$\frac{+\ 0.00\cdots0\left(\frac{\beta}{2}\right)\times\beta^{0} \le E}{1.00\cdots01\ \times\ \beta^{0}}$$

$$E = \frac{\beta}{2}\times\beta^{-t} = \frac{1}{2}\beta^{1-t}$$

Example: IEEE double precision

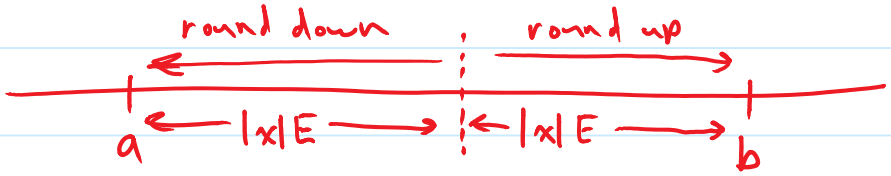
$$\beta = 2 \qquad t = 52$$

$$E = \frac{1}{2}\times 2^{1-52} = 2^{-52} \approx 10^{-16}$$

Distribution of Floating-Point Numbers

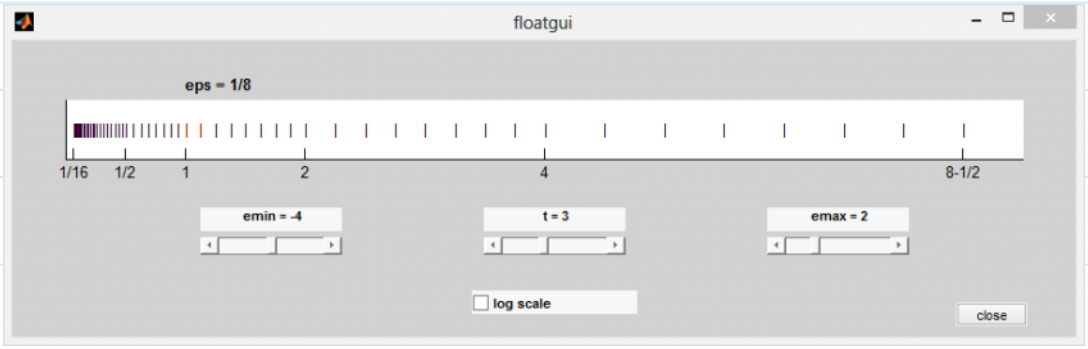
Since relative error is bounded,

$$\frac{|x - fl(x)|}{|x|} < E \quad \Rightarrow \quad \underbrace{|x - fl(x)|}_{\text{abs. error}} < |x|E$$



Thus, numbers of magnitude x are spaced approx. $2|x|E$ apart.

>> floatgui



Floating-Point Arithmetic

The result of an arithmetic operation may need to be rounded to represent it as a floating-point number.

Let $\mathcal{F}$ = set of FP #s

Assume $x, y \in \mathcal{F}$

$$fl(x+y) = (x+y)(1+\delta) = x \oplus y$$

($\oplus$: FP sum)

Error Analysis of $(a \oplus b) \oplus c$

$$\text{Rel Err} = \frac{|(a \oplus b) \oplus c - (a+b+c)|}{|a+b+c|}$$

$$= \frac{|(a+b)(1+\delta_1) \oplus c - (a+b+c)|}{|a+b+c|}$$

$$= \frac{|[(a+b)(1+\delta_1)+c](1+\delta_2) - (a+b+c)|}{|a+b+c|}$$

$$= \cdots$$

$$= \frac{|(a+b)\delta_1(1+\delta_2) + (a+b+c)\delta_2|}{|a+b+c|}$$

triangle $\leq \frac{|a+b|}{|a+b+c|}|\delta_1(1+\delta_2)| + |\delta_2|$

$|X+Y| <= |X|+|Y|$

triangle inequality $\leq \frac{|a+b|}{|a+b+c|}|\delta_1(1+\delta_2)| + |\delta_2|$

'cuz $|\delta| < E$ $\leq \frac{|a+b|}{|a+b+c|}E(1+E) + E$

FP-05: Round-Off Error 2

Goal: See some examples of round-off error yielding poor results.

Using the FPNS $(t, \beta, L, U) = (3, 10, -20, 20)$
Evaluate the true relative error, and the upper bound for each set of numbers {a,b,c}.

Example 1

$a = 5670 \quad b = 7890 \quad c = 123$

True value = 13683

Approx. value (rounding)

$$= (a \oplus b) \oplus c$$
$$= (5670 \oplus 7890) \oplus 123$$
$$= fl(13560) \oplus 123$$
$$= 13600 \oplus 123$$
$$= fl(13723) = 13700$$

$$\text{Actual Rel Err} = \frac{|13700 - 13683|}{|13683|} = 0.001242 = 0.12\%$$

Upper Bound

$$\text{Rel Err} \leq \frac{13560}{13683}\,(\approx 1)\; E(1+E) + E$$
$$\approx E(1+E) + E = 2E + E^2 \quad (E^2 \text{ very small})$$

In our case, $E = 10^{-2} = 0.01$ E=1/2 * β x ^(1-t)=1/2 x 10^-2

Thus, Rel Err $\leq 2E = 0.02$ ✓ 2E=0.01

Example 2

$a = 5670 \quad b = 7890 \quad c = 10^8$

True value = 100,013,560

Approx value = $(5670 \oplus 7890) \oplus 10^8$

$= fl(13560) \oplus 10^8$

$= 13600 \oplus 10^8$

$= fl(100,013,600)$

$= 100,000,000 = 10^8$

(Actual) Rel Err $= \frac{13560}{100,013,560} \approx 1.36 \times 10^{-4}$

(Bound) Rel Err $\leq \frac{13560}{100,013,560}$ (≈ 0) $E(1+E) + E \approx E = 0.01$ ✓

Example 3

$a = 5670 \quad b = 7890 \quad c = -13500$

True value = 60

Approx value = $13600 \oplus (-13500)$

Approx value $= 13600 \oplus (-13500)$

$= fl(100) = 100$

(Actual) Rel Err $= \frac{|160-100|}{|160|} = \frac{40}{60} = 0.6 \approx 67\%$

(Bound) Rel Err $\leq \frac{13560}{60} E(1+E) + E$

$= 226E(1+E) + E$

$= 2.29 = 229\%$ ✓

Cancellation Error

What we are observing is called cancellation error. It results from round-off error when you are subtracting two large values that have almost the same magnitude.

Patriot Missile (revisitted)

Time was computed by the number of 0.1-second clock ticks.

Time $= k \otimes fl(0.1)$

(k: clock ticks)

$= k \otimes 0.1(1+\delta_1)$

$= 0.1k(1+\delta_1)(1+\delta_2)$

$$\text{Abs-Err.} = |0.1k - 0.1k(1+\delta_1)(1+\delta_2)|$$

$$= |0.1k(1 - 1 - \delta_1 - \delta_2 - \delta_1\delta_2)|$$

$$= |0.1k|\,|\delta_1 + \delta_2 + \delta_1\delta_2|$$

$$\leq 0.1k\,(2E + \underbrace{E^2}_{\text{small}})$$

In this FPNS $(t=20,\ \beta=2)$

$$\Rightarrow E = \frac{1}{2}2^{1-20} = 20^{-20} \approx 9.537 \times 10^{-7}$$

After 100 hours, $k = 10 \times 60 \times 60 \times 100 = 0.36 \times 10^7$

$$\text{Abs Err} \leq 0.1\,(3.6 \times 10^6)(1.907 \times 10^{-6})$$

$$= 0.68664 \text{ seconds}$$