Unit 1:

Floating-Point Numbers

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

L01: FP Problem

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

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

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

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)]);

```
% Try alpha values of 0.5 and 2.
alpha = 0.5;
N = 100;

I = log((1+alpha) / alpha);

for n = 1:N
   I = 1/n - alpha * I;
end

disp(['Answer: ' num2str(I)]);
```

For $\alpha = 0.5 \Rightarrow$ answer $= 0.006644$

For $\alpha = 2 \Rightarrow$ answer $= 6.058 \times 10^{12}$

Hmmm... seems strange.

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

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

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

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

Thus, $I_0^{(comp)} = I_0^{(exact)} + e_0$ ← tiny error

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

Using our recurrence relation,

Using our recurrence relation,

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

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

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 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.
Consider adding up these 4 binary numbers, but keeping only 4 significant digits.

Method 1

0.1111
0.0111
0.0011
0.0001

$0.1111 \oplus 0.0111 = 1.0110 = 0.1011 \times 10$

$0.0011 = 0.0001 \times 10$ (!)

$0.1011 \times 10 \oplus 0.0001 \times 10 = 0.1100 \times 10$

$0.0001 = 0.0000 \times 10$

$0.1100 \times 10 \oplus 0.0000 \times 10 = 0.1100 \times 10$

Answer = 1.1

Method 2

0.0001
0.0011
0.0111
0.1111

$0.0001 \oplus 0.0011 = 0.0100$

$0.0100 \oplus 0.0111 = 0.1011$ (!)

$0.1011 \oplus 0.1111 = 1.101$

Answer = 1.101

Take-Home Message

We follow some basic rules when doing arithemetic 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.

L02: 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.

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

$$\pm 0.\underbrace{d_1 d_2 \ldots}_{\text{digits base-}\beta} \times \beta^p \quad \leftarrow \text{integer exponent}$$

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

$d_1 \neq 0$ (normalized form)

Examples:

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

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

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

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, Example:

IEEE Single Precision

IEEE Double Precision

L03: 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 $t=2$, $\beta=10$

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

eg. $fl(\pi) = 3.14$

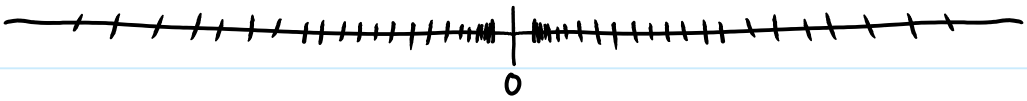
The difference is called the "round-off error".

$$|\pi - fl(\pi)| = 0.00159265\ldots$$

In general, for FPNS (t, β, L, U),

Effect of Limited Exponent

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



For example, the largest value in the number system

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

Anything larger cannot be represented, and causes an

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

Anything too small is called and gets rounded to zero.

```
>> realmin
>> realmin/2
>> realmin/2^52
>> realmin/2^53
```

Exception Handling

What do these return?

```
>> 0/0

>> 1/0

>> -1/0

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

L04: 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. =

eg. $x = 100$ $\bar{x} = 101$

Abs. Err. =

Relative Error

Rel. Err. =

eg. Rel. Err. =

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

The maximum relative error is called

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

Definition of Machine Epsilon

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

For (t, β, L, U) ...

Example: IEEE double precision

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

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

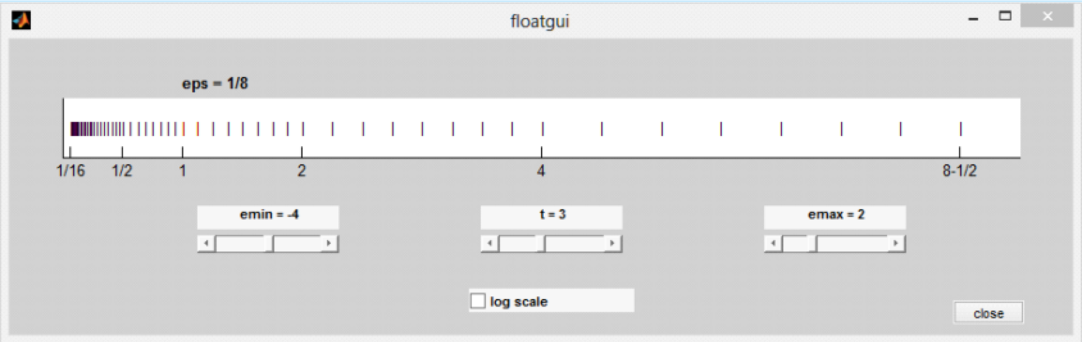
Distribution of Floating-Point Numbers

Since relative error is bounded,

$$\frac{|x - fl(x)|}{|x|} < E \quad \Rightarrow$$

Thus, numbers of magnitude are spaced approx. 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 $x, y \in \mathcal{F}$

FP sum

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

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

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

$=$

$=$

$=$

$=$

L05: Round-Off Error 2

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

Using the FPNS $(t, \beta, L, U) = (2, 10, -20, 20)$
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)

Actual Rel Err =

Upper Bound

Rel Err $\leq$

In our case, $E =$

Thus, Rel Err

Example 2

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

True value = 100,013,560

Approx value =

.

.

.

.

.

.

.

.

(Actual)
Rel Err =

(Bound)
Rel Err $\leq$

Example 3

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

True value = 60

Approx value =

(Actual)
Rel Err =

(Bound)
Rel Err $\leq$

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 =

Abs. Err. =

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

$\Rightarrow E =$

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

Abs Err $\leq$