What is numerical analysis?

Numerical Analysis is the study of algorithms for solving mathematical problems. Central question becomes:

- how to construct an approximate solution?
- how accurate the ~~stat~~ approximation is?
- how much does it cost?

(Ex) Evaluating a polynomial

$$P(x) = 2x^4 + 3x^3 - 3x^2 + 5x - 1$$

Suppose that coefficients of $P(x)$ are stored, and suppose we seek $P(\frac{1}{2})$. What is the minimal cost that needs to be paid to get $P(\frac{1}{2})$?

- straightforward (A)

$$P(\tfrac{1}{2}) = 2\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + 3\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} - 3\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + 5\cdot\tfrac{1}{2} - 1$$

- each arithmetic operation such as $+, -, *, /$ takes time to execute, we call it flop = floating point operation
- count flops in (A): 14 flops = 10 mult, 4 add/sub
- a more clever way is to reuse some of the work, eg.

precompute $t_1 = 1/2$, $t_2 = t_1\cdot\frac{1}{2}$, $t_3 = t_2\cdot\frac{1}{2}$, $t_4 = t_3\cdot\frac{1}{2}$

$$P(\tfrac{1}{2}) = 2\cdot t_4 + 3\cdot t_3 - 3\cdot t_2 + 5\cdot t_1 - 1$$

7 mult, 4 add. = 11 flops

– <u>Nested multiplication (Horner's method)</u>

$$P(x) = -1 + 5x - 3x^2 + 3x^3 + 2x^4 = -1 + x\cdot(5 - 3\cdot x + 3\cdot x^2 + 2\cdot x^3) =$$

$$= -1 + x\cdot(5 + x\cdot(-3 + 3\cdot x + 2\cdot x^2)) =$$

$$= -1 + x\cdot(5 + x\cdot(-3 + x\cdot(3 + 2\cdot x)))$$

$t_1 = 3 + 2\cdot x$ 1 mult, 1 add

$t_2 = -3 + x\cdot t_1$

$t_3 = 5 + x\cdot t_2$

$p(\frac{1}{2}) = -1 + x\cdot t_3$

4 mult, 4 additions = 8 flops.

<u>Moral</u>: A carefully designed algorithm can give a substantial performance boost!

To represent floating point numbers on a computer we will need to understand <u>binary numbers</u> first

– Decimal numbers are converted to binary numbers to be stored in memory.

(Ex) <u>Binary number</u>: $\ldots b_2 b_1 b_0 . b_{-1} b_{-2} \ldots$

$$\ldots + b_2\cdot 2^2 + b_1\cdot 2^1 + b_0\cdot 2^0 + b_{-1}\cdot 2^{-1} + b_{-2}\cdot 2^{-2} + \ldots$$

* $(100.)_2 = 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 = 4$

* $(0.11)_2 = 1\cdot 2^{-1} + 1\cdot 2^{-2} = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$

Decimal to binary conversion

- notation $(53.7)_{10} = 53.7$

Ex: break into integer and fractional part,

$53.7 = 53 + 0.7$

int: divide 53 successively by 2 and record remainder,

$53/2 = 26$ rem $1 = b_0$

$26/2 = 13$ rem $0 = b_1$

$13/2 = 6$ rem $1 = b_2$

$6/2 = 3$ rem $0 = b_3$

$3/2 = 1$ rem $1 = b_4$

$1/2 = 0$ rem $1 = b_5$

$(110101)_2 = 1\cdot2^5 + 1\cdot2^4 + 1\cdot2^2 + 1\cdot2^0 = (53)_{10}$

32 16 4 1

Lecture 2

frac part

$(0.7)_{10}$,

$0.7 \times 2 = 1 + 0.4$

$0.4 \times 2 = 0 + 0.8$

$0.8 \times 2 = 1 + 0.6$

$0.6 \times 2 = 1 + 0.2$

$0.2 \times 2 = 0 + 0.4$

repeat

$0.4 \times 2 = 0 + 0.8$

$0.8 \times 2 = 1 + 0.6$

$0.6 \times 2 = 1 + 0.2$

repeat

$\ldots$

$(0.7)_{10} = (0.1\overline{0110})_2$

Combine the two,

$$(53.7)_{10} = (53)_{10} + (0.7)_{10} = (110101.1\overline{0110})_2$$

Binary to decimal

- integer part is clear
- if fractional part is finite, same otherwise,

let $(0.\overline{1011})_2 = x = 0.1011\overline{1011}$

$$2^4 \cdot x = (1011.\overline{1011})_2 = (1011)_2 + x$$

$$16x = (2^3 + 2^1 + 2^0) + x = 11 + x$$

$$15x = 11, \quad \boxed{x = 11/15}$$

~~EEEE~~

- double precision floating point number is a standard way to represent real numbers on a computer (along with the eg single).

precision	sign	exponent	mantissa
single	1	8	23
double	1	11	52
long double	1	15	64

(double)

S – sign bit

$e_{10}, e_9, e_8, \ldots, e_0$ – exponent, ~~is~~

$b_1, b_2, b_3, \ldots, b_{52}$ – mantissa

an auxiliary number is formed $F = (e_{10} e_9 \ldots e_0)_2 = e_{10} \cdot 2^{10} + \ldots + e_0 \cdot 2^0$

$$0 \le F \le 2047$$

$\max F = (1111 \ldots 1)_2 = 2^{11} + 2^{10} + \ldots$, note.

$(11 \ldots 1)_2 + (1)_2 = 2^{11} = 2048$

$(11 \ldots 1)_2 = 2048 - 1 = \underline{2047}$

The value that the 64 bits represent depends on what F is.

Normalized #s

$1 \le F \le 2046$, then

$$V = (-1)^s (1.b_1 b_2 \ldots b_{52})_2 \times 2^{F-1023}$$

$$(1.b_1 b_2 \ldots b_{52})_2 = 1 + b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \ldots + b_{52} \cdot 2^{-52}$$

- largest normalized # is $V_{max,norm} = (1.11 \ldots 1)_2 \times 2^{1023} =$

$$= \sum_{n=0}^{52} 2^{-n} \cdot 2^{1023} = \frac{1 - 2^{-53}}{1 - \frac{1}{2}} \cdot 2^{1023} \simeq 1.8 \times 10^{308}$$

- smallest norm # is

$$V_{min,norm} = (1.00 \ldots 0)_2 \cdot 2^{-1022} \simeq 2.2 \times 10^{-308}$$

Unnormalized #s

if $F = 0$

$$V = (-1)^s (0.b_1 b_2 \ldots b_{52})_2 \times 2^{-1022}$$

$$V_{min,unnorm} = (0.00 \ldots 01)_2 \times 2^{-1022} = 2^{-52} \cdot 2^{-1022} \simeq 4.9 \times 10^{-324}$$

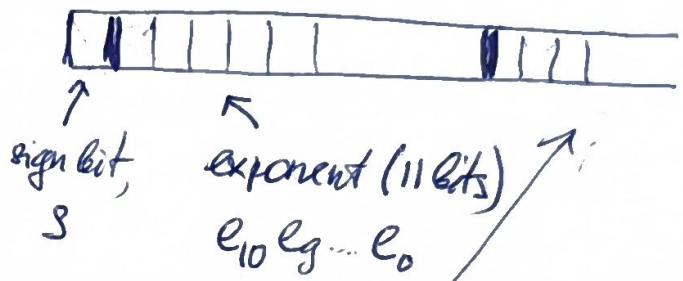
If $F = 2047$ and mantissa is non zero, then

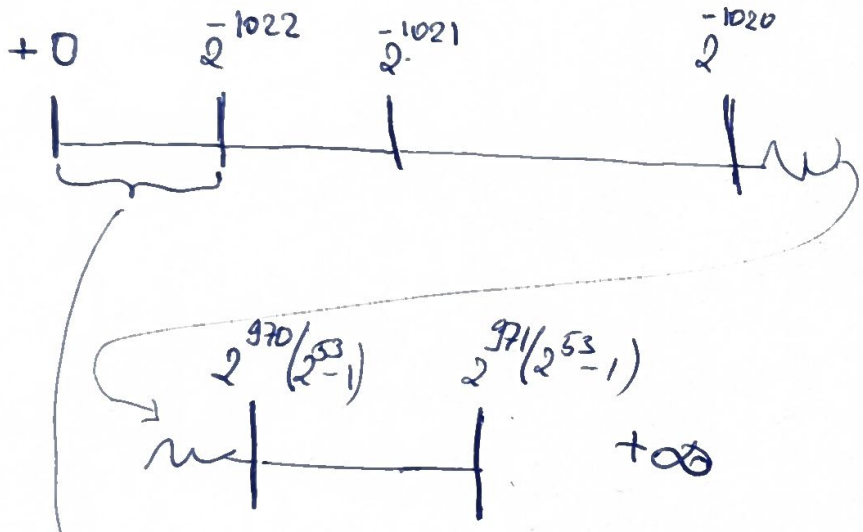
$V = NaN$ – produced by ops whose result is undefined,

e.g. division by zero.

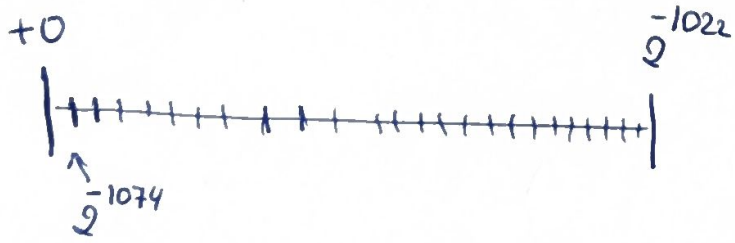
If $F = 2047$, mantissa $= 0$, then $V = (-1)^s \infty$

Illustration of a double precision # in memory





subnormal #s



(Ex) (Matlab Illustration)

- open matlab command line, try

>> $x = 2^{(-1074)}$

>> $x/2$

(you will get 0 as output!)

Sometimes, this is called underflow

(lecture 5)

Machine precision (machine epsilon) denoted by ε_{mach}.

ε_{mach} is the smallest positive number such that

$1 + \varepsilon_{mach} \neq 1$ (where + is a computer implementation of addition)

Ex Floating point implementation of $(9.4)_{10}$

$(9.4)_{10} = (1001.\overline{0110})_2$ ← show this

(*) $(1001.\overline{0110})_2 = +1.\underbrace{001\,0110\,0110\,\ldots\,0110\,0110}_{52\text{ bits}} \times 2^3$

- so far this is exact expansion. However, for storage in memory the infinite expansion must be truncated (and possibly modified)

IEEE rounding to nearest rule

(i) If the 53rd bit in (*) is 0, then round down (truncate)

(ii) If the 53rd bit in (*) is 1, then consider two cases:

a) if at least one bit after and including 54 is 1, then round up (add 1 to 52nd bit)

b) if the 54th and all trailing bits are 0, then add 1 to the 52nd bit if and only if the 52nd bit is 1.

- so

fl(9.4) mantissa is

$$\boxed{001\,\underbrace{0110\;0110\;\ldots\;0110}\;1}$$

52nd bit according to rounding rule (iia).

Floating point arithmetic

Arithmetic operations $+, -, *, /$ have implementations in IEEE arithmetic, if we denote $\oplus, \ominus, \circledast, \oslash$ these implementations, then we can expect

$$x \oplus y = (x+y)(1+\mu)$$
$$x \ominus y = (x-y)(1+\lambda)$$
$$x \circledast y = (x \cdot y)(1+\delta)$$
$$x \oslash y = (x/y)(1+\alpha)$$

where $|\mu|, |\lambda|, |\delta|, |\alpha|$ are all $\leq \varepsilon_{mach}$

Loss of significance

Errors Suppose x_A is an approximation to a true number $x_T \in \mathbb{R}$

absolute error $= x_T - x_A$, $\quad \text{abserr}(x_A) = |x_T - x_A| \geq 0$

relative error $= \dfrac{x_T - x_A}{x_T}$, $\quad \text{relerr}(x_A) = \dfrac{|x_T - x_A|}{|x_T|} \geq 0.$

- naturally relative error is not defined if $x_T = 0$.

Significant digits in x_A relative to x_T is the number of leading digits in x_A that match x_T

Example

$$x_T = (d_N \dots d_0 . \overset{1}{d_{-1}} \overset{N+1}{d_{-2}} \overset{N+2}{\dots} \overset{N+3}{d_{-(p-N-1)}} \overset{p}{d_{-(p-N)}} \overset{p+1}{\dots})$$

$$|x_T - x_A| = (0 \dots 0 . 0 \; 0 \dots 0 \; a_{-(p-N)} \dots)$$

If $a_{-(p-N)} \leq 5$, then x_A has p significant digits relative to x_T, otherwise x_A has $p-1$ significant digits.

Ex (A) $x_T = \pi = 3.14159265\dots$ and $x_A = \frac{22}{7} = 3.1428571\dots$

$$x_T = (3.14159265\dots)$$

$$|x_T - x_A| = (\underbrace{0.00126}_{3 \text{ digits}}448\dots)$$

≤ 5, then $p = 3$ significant digits

(B) $x_T = \frac{7}{9} = 0.7777\dots$ and $x_A = 0.7777$

$$x_T = (0.777777\dots)$$

$$|x_T - x_A| = (0.\underbrace{0000}_{4 \text{ d}}77\dots)$$

> 5, then $p = 4 - 1 = 3$ significant digits

Rough estimate, if x_A has p significant digits relative to x_T, then $\text{relerr}(x_A) = \frac{|x_T - x_A|}{|x_T|} \sim 10^{-p}$

Guideline to assigning significant digits to a number x_T "by itself". Sometimes the true number x_T is not known, then the rule is:

(i) all nonzero digits are considered significant, e.g.

143.2 has 4 significant digits.
3.1843 has 5 significant digits.

(ii) zeros which lie between non-zero digits are significant, eg.

1001.3 has 5 significant digits.

(iii) zeros which come before the first nonzero digit are not significant, so

0.001 has 1 significant digit,
0.0143 has 3 significant digits.

Loss of significance

Consider $x = 123.01$ and $y = 123.02$ each known to 5 significant digits. These could be approximations to $x_T = 123.01289\ldots$ and $y_T = 123.02175\ldots$

The true difference is $y_T - x_T = 0.00885\ldots$ — is a # with infinitely many significant digits, however $y - x = 0.01$ has only one significant digit

Moral: subtracting nearly equal numbers is prone to loss of significance

An example that illustrates the issue:

(Ex) consider quadratic equation,

$$2.1x^2 - 4.5x + 10^{-11} = 0$$

use Matlab/Octave/Python to find roots,

```
>> a = 2.1; b = -4.5; c = 10^-11;
>> roots([a b c])   <- solves quadratic eqn with coeff a, b, c.
   2.142857142854921e+00
   2.222222222224527e-12
```

→ compare this to directly applying quadratic formula.

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = 2.142857142854921e+00 \leftarrow \text{good.}$$

$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = 2.222137817668790e-12 \leftarrow \text{four significant digits left, why?}$$

loss of significance due to subtraction of nearly equal numbers occurred here!

- Verify $-b$ and $\sqrt{b^2 - 4ac}$ are nearly the same
- How to fix this? Avoid subtraction, recall that

$$r_1 \cdot r_2 = c,$$

so we can use accurate root r_1, and find $r_2 = \frac{c}{r_1}$ to avoid precision loss.