MA 8019: Numerical Analysis I
Computer Arithmetic

Suh-Yuh Yang (楊肅煜)

Department of Mathematics, National Central University
Jhongli District, Taoyuan City 320317, Taiwan
E-mail: syyang@math.ncu.edu.tw
http://www.math.ncu.edu.tw/∼syyang/

First version: May 03, 2018   Last updated: October 17, 2023

Continuous versus discrete

- The set of real numbers $\mathbb{R}$ includes:
  (1) the set of rational numbers $\mathbb{Q} = \{\frac{q}{p} : p \neq 0, q \textit{ are integers}\}$:
  e.g., $1.1, 3.14, 2/3, -3/7, \cdots$
  (2) the set of irrational numbers $\mathbb{Q}^c = \mathbb{R} \setminus \mathbb{Q}$:
  e.g., $-\pi = -3.14159265358979...$, $e = 2.718281828...$,
  $\sqrt{2} = 1.4142...$

  The real numbers are "continuous"!

- Computer numbers:
  (1) integers: 0, +1, -1, $\cdots$
  (2) non-integers (floating-point numbers): $x_1 x_2 ... x_n . y_1 y_2 ... y_m$,
  where both m and n are finite.

  The computer numbers are "finite and discrete"!

Number systems: computer versus user

- *The decimal number system: base* $= 10$

  e.g., $427.325 := (427.325)_{10} = 4 \times 10^2 + 2 \times 10^1 + 7 \times 10^0 + 3 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$

- *The binary number system: base* $= 2$

  e.g., $(1001.11101)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} = (9.90625)_{10}$

- **Notation:**

  $\beta > 1$ *integer,* $(N)_\beta$ *denotes a number system with base* β*,*
  digits $0, 1, 2, \cdots, \beta - 1$, and a sign ($+$ or $-$) affixed to it.

  e.g., $(1001.11101)_2 = (9.90625)_{10}$

Number systems: computer versus user (cont'd)

- Most computers deal with real numbers in the binary number system!

$$\text{computer (binary)} \underset{\textit{conversion}}{\overset{\textit{conversion}}{\rightleftarrows}} \text{user (decimal)}$$

$\Longrightarrow$ *roundoff error!*

- For example, $\frac{1}{10} = (0.00011001100110011...)_2$

  If we read 0.1 into a 32-bit computer and then print it out to 40 decimal places, we obtain:

  0.10000 00014 90116 11938 47656 25000 00000 00000

Questions

- How to represent the numbers in a computer?
- How to perform the basic operations $+$, $-$, $\times$, $/$?
- What is the error?

Numbers on a computer

Suppose that our computer can store only six digits, with five digits after the decimal point, i.e., X.XXXXX

- ordinary number system
  (1) smallest (positive) number: 0.00001 ($= 10^{-5}$)
  (2) largest number: 9.99999 ($\approx 10^{1}$)
  (3) range of the system $\approx 10^{-5} \sim 10^{1}$
- A "better" number system: let us allocate two digits for the "power of ten", (assuming we know how to do the signs without using any of the digits)
  (1) smallest (positive) number: 0.001×10^{-99}
  (2) largest number: $9.999 \times 10^{99} (\approx 10^{100})$
  (3) range of the system $\approx 10^{-102} \sim 10^{100}$
- **Good:** the "better" system has a much bigger range, which has a lot more numbers that one can use.

  Bad: the "better" system has only 4 digits of accuracy.

Examples (cont'd)

1. $\pi = 3.14159265358979...$
   - in the ordinary system, $\pi = 3.14159$
   - in the "better" system, $\pi = 3.142 \times 10^0$
2. $n = 1234567$
   - in the ordinary system, n = "overflow"
   - in the "better" system, $n = 1.235 \times 10^6$
   - relative error = $\left| \dfrac{1234567 - 1.235 \times 10^6}{1234567} \right| \approx 10^{-4}$
3. $A = 0.000001$
   - in the ordinary system, A = "underflow" (set to zero)
   - in the "better" system, $A = 0.100 \times 10^{-5}$

What is the best way to represent numbers on a computer?

Answer: *binary + floating-point system.*

- Decimal representation is convenient for people, but not for computers.
- *Binary representation is much more useful on computers.*

  The basic unit in a binary representation is called a bit.
- A bit can be viewed as a physical entity that is either off or on.

Some terminologies

- Bits are organized in groups of 8, each called a byte.
  A byte can represent any of $256 = 2^8$ different bitstrings (0-255 integers, 256 characters, 256 colors, ...).
- A word is four consecutive bytes; i.e., 32 bits.
- A double word is eight consecutive bytes; i.e., 64 bits.
- A kilobyte (KB) is $1024 = 2^{10}$ bytes (kilo $\approx 10^3$).
- A megabyte (MB) is 1024 KB = 2^{20} bytes (mega $\approx 10^6$).
- A gigabyte (GB) is 1024 MB = 2^{30} bytes (giga $\approx 10^9$).
- A terabyte (TB) is 1024 GB = 2^{40} bytes (tera $\approx 10^{12}$).
- A petabyte (PB) is 1024 TB = 2^{50} bytes (peta $\approx 10^{15}$).

Large Hadron Collider (大型强子對撞機): produces 15PB data/per year.

Binary system

Two types of binary system can be designed:

- *Fixed-point system is very limited in its range.* For example, in a 32-bit system, 1 bit for the sign, 15 bits for the number before the binary point, 16 bits for the number after the binary point, range of the system (positive number) $\approx 2^{-16} \sim 2^{15}$.
- *Floating-point system:* Consider the normalized scientific notation for decimal number system:

$$\begin{aligned} 732.5051 &= 0.7325051 \times 10^3, \\ -0.005612 &= -0.5612 \times 10^{-2}. \end{aligned}$$

  The decimal point floats to the position immediately before the first nonzero digit. *In general, a nonzero real number x can be represented in the form:*

$$x = \pm r \times 10^n, \quad \text{where } \frac{1}{10} \le r < 1 \text{ and } n \in \mathbb{Z}.$$

Floating-point system

- *Floating-point system:* $\pm f \times \beta^e$

  (1) f: mantissa part (fraction) that contains the significant figures of the number;
  (2) e: exponent (the scale of the number);
  (3) β: the base of the number system.

- *A nonzero floating-point number* $a = \pm f \times \beta^e$ *is said to be normalized if*

$$\beta^{-1} \le f < 1.$$

  For example, if $\beta = 10$, then $0.1 \le f < 1$.

  Then f can be written as $0.x_1x_2x_3...$ and $x_1 \neq 0$,

  e.g., $0.002597 = 0.2597 \times 10^{-2}$.

- *Some bases:*

  (1) $\beta = 2$, binary, most computers;
  (2) $\beta = 10$, decimal, most calculators;
  (3) $\beta = 16$, hexadecimal, IBM mainframes.

IEEE standard 32-bit binary systems

- Published in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).
- Based on the work of William Kahan (1933 –) of UC-Berkeley. Kahan received the 1989 Turing Award for this work.

http://www.cs.berkeley.edu/~wkahan/

- The essentials of the standard include
  (1) consistent representation of floating-point numbers by all computers adopting the standard;
  (2) correctly rounded floating point numbers;
  (3) consistent treatment of exceptional situations such as division by zero.

Single precision format: hypothetical computer `Marc-32`

- A single precision floating-point word

$$x = \boxed{s} \boxed{a_1 a_2 a_3 \cdots a_8} \boxed{b_1 b_2 b_3 \cdots b_{23}}$$

(1) 1 bit for the sign of the fraction: s (0 for $+$ and 1 for $-$)
(2) 8 bits for the biased exponent: e
$0 < e < (11111111)_2 = 2^8 - 1 = 255$
$e = 0$ and $e = 255$ are reserved for special cases such as ± 0, $\pm\infty$ and `NaN` (not a number).
(3) 23 bits for the fraction: f

- The bias on the exponent is

$$127 = 2^0 + 2^1 + 2^2 + \cdots + 2^6 = (01111111)_2$$

The actual exponent $m = e - 127 \quad (\Rightarrow -126 \leq m \leq 127)$

- The actual fraction is $q = (1.f)_2 \quad (\Rightarrow 1 \leq q < 2)$.
- *The nonzero normalized binary floating-point number (machine number) is:* $x = (-1)^s q \times 2^m$

An example

$$x = \boxed{0} \boxed{0000\ 1110} \boxed{1010\ 0000\ 0000\ 0000\ 0000\ 000}$$

- Mantissa is $(1.1010\cdots0)_2 = (2^0 + 2^{-1} + 2^{-3})_{10} = (1 + 0.5 + 0.125) = 1.625$
- Exponent is $00001110 - 01111111 = -01110001 = -(2^0 + 2^4 + 2^5 + 2^6)_{10} = -(113)_{10}$

$$\begin{array}{r} 01111111 \\ 00001110 \\ \hline 01110001 \end{array}$$

- *Finally, the number is* $x = 1.625 \times 2^{-113}$

Summary ($s = 0$)

- Single precision has roughly 7 digits of decimal accuracy and the range is $2^{-126} \sim (2^0 + 2^{-1} + \cdots + 2^{-23})2^{127} = (2 - 2^{-23})2^{127}$ $\approx 1.1754944 \times 10^{-38} \sim 3.4028235 \times 10^{38}$

  $2^{-23} \approx 1.1920929 \times 10^{-07}$

  $2^{-24} \approx 5.9604645 \times 10^{-08}$

- IEEE double precision (64-bit) system:

  (1) one bit for the sign of the fraction
  (2) 11 bits for the biased exponent
  (3) 52 bits for the fraction

  It has roughly 15 digits of decimal and range is $\approx 10^{-307} \sim 10^{307}$

Some limitations of the floating-point system

- The range of the fraction is limited (round-off error).
- The range of the exponent is limited (overflow, underflow).
  (1) “overflow” is a fatal error (program stops).
  (2) “underflow” is often the same as “set to zero.”

Some properties of the floating-point system:

- The floating-point system is a small subset of the real number system.
- The floating-point numbers are not equally spaced on the real line (see page 21 below).

How to get around the limitation?

Example: calculating the vector norm $\|x\|$?

Let $x = (a, b)^\top$, $c = \|x\| = \sqrt{a^2 + b^2}$. Let us take a toy floating-point system: $\beta = 10$, two digits for the exponent, and $a = 10^{60}$, $b = 1.0$. Then $a^2 = (10^{60})^2 = 10^{120}$ overflow, program stops, can't obtain c.

A trick: use a mathematically equivalent form of c

$$c = s\sqrt{\left(\frac{a}{s}\right)^2 + \left(\frac{b}{s}\right)^2},$$

where $s = \max\{|a|, |b|\}$. In this case, $s = 10^{60}$. Then

$\left(\frac{a}{s}\right)^2 = 1.$ $\left(\frac{b}{s}\right)^2 = \left(\frac{1}{10^{60}}\right)^2$ *(underflow, set to zero).*

$c \approx s\sqrt{1+0} = s = 10^{60}$.

Mathematically equivalent forms are often not numerically equivalent!

Nearby machine numbers

Given a positive real number x by

$$\begin{aligned} x &= q \times 2^m, \quad 1 \le q < 2, \quad -126 \le m \le 127, \\ &= (1.a_1a_2\cdots a_{23}a_{24}a_{25}\cdots)_2 \times 2^m, \end{aligned}$$

each a_i is either 0 or 1, we have two nearby machine numbers

$$\begin{aligned} x_- &= (1.a_1a_2\cdots a_{23})_2 \times 2^m \qquad (\textit{chopping}), \\ x_+ &= ((1.a_1a_2\cdots a_{23})_2 + 2^{-23}) \times 2^m \qquad (\textit{rounding up}). \end{aligned}$$

Then $x_- \le x \le x_+$. The closer of x_- and x_+ is chosen to represent x in the computer, denoted by $fl(x)$ (machine number).

---

chopping: 無條件捨棄
rounding up: 無條件進位
rounding off: 有捨有入(例如十進位時的四捨五入)

Nearby machine numbers (cont'd)

- If $fl(x) = x_-$, then we have

$$|x - x_-| \le \frac{1}{2}|x_+ - x_-| = \frac{1}{2}2^{m-23} = 2^{m-24}.$$

The relative error is $\left|\dfrac{x - x_-}{x}\right| \le \dfrac{2^{m-24}}{q \times 2^m} = \dfrac{1}{q}2^{-24} \le 2^{-24}$.

- If $fl(x) = x_+$, then we have

$$|x - x_+| \le \frac{1}{2}|x_+ - x_-| = \frac{1}{2}2^{m-23} = 2^{m-24}.$$

The relative error is $\left|\dfrac{x - x_+}{x}\right| \le \dfrac{2^{m-24}}{q \times 2^m} = \dfrac{1}{q}2^{-24} \le 2^{-24}$.

- *For both cases, we have* $\left|\dfrac{x - fl(x)}{x}\right| \le 2^{-24}$.

Nearby machine numbers (cont'd)

- *Letting* $\delta = \frac{fl(x) - x}{x}$, *then we have* $fl(x) = x(1 + \delta)$ *and* $|\delta| \leq 2^{-24}$, *where the number* 2^{-24} *is called the unit roundoff error* (單位捨入誤差).

- *Machine epsilon* (ε): the smallest positive floating-point number ε such that $1 + \varepsilon > 1$.

  In general, for number system with base β, $fl(x) = x(1 + \delta)$, where $|\delta| \leq \gamma\varepsilon$ and γ is not too large (For `Marc-32`, the machine epsilon, $\varepsilon = 2^{-23}$, is twice of the unit roundoff error, $\gamma = 1/2$).

Machine numbers

Suppose that $x = q \times 2^m$, a positive nonzero machine number.

Then the next (larger) machine number on the right is
$x_r = (q + 2^{-23}) \times 2^m$.

The previous (smaller) machine number on the left is
$x_\ell = (q - 2^{-23}) \times 2^m$.

We have

$$x_r - x = x - x_\ell = 2^{m-23} \implies \frac{x_r - x}{x} = \frac{x - x_\ell}{x} = \frac{1}{q} \times 2^{-23}.$$

Since $1 \leq q < 2$, we have

$$2^{-24} < \frac{x_r - x}{x} = \frac{x - x_\ell}{x} \leq 2^{-23}.$$

Hence, *the relative spacing* between machine numbers x and x_r, or x and x_ℓ is approximately a constant value, 2^{-23}.

Floating-point operations: $+, -, \times, \div$

1. Let the symbol $\odot$ stand for any one of the arithmetic operations $+, -, \times$ or $\div$. For `Marc-32`, we have

$$fl(x \odot y) = (x \odot y)(1+\delta), \quad |\delta| \leq 2^{-24},$$

if x and y are machine numbers;

$$fl(fl(x) \odot fl(y)) = (x(1+\delta_1) \odot y(1+\delta_2))(1+\delta_3), \quad |\delta_i| \leq 2^{-24},$$

if x and y are not machine numbers.

2. **Floating-point error analysis:** Suppose that x, y and z are machine numbers in `Marc-32`. We want to compute $x(y+z)$. Then we have

$$\begin{aligned} fl(x(y+z)) &= (x fl(y+z))(1+\delta_1) \qquad |\delta_1| \leq 2^{-24} \\ &= (x(y+z)(1+\delta_2))(1+\delta_1) \qquad |\delta_2| \leq 2^{-24} \\ &= x(y+z)(1+\delta_2+\delta_1+\delta_2\delta_1) \\ &\approx x(y+z)(1+\delta_1+\delta_2) \\ &:= x(y+z)(1+\delta_3) \qquad |\delta_3| \leq 2^{-23}. \end{aligned}$$

Conditioning of $f(x)$

- The words *condition or conditioning* are used to indicate how sensitive the solution of problem may be to small relative changes in the input data.
- In general, how do we calculate a function $f(x)$ for some $x \in \mathbb{R}$?
  (1) find an $x^* := fl(x)$ in the floating point system such that $x^* \approx x$.
  (2) compute $f(x^*)$.
- **Question:** How sensitive is $f(x)$ to the change of x to x^*?

condition number of $f(x)$ at x

$$:= \max \left\{ \frac{\left| \frac{f(x)-f(x^*)}{f(x)} \right|}{\left| \frac{x-x^*}{x} \right|} \text{ for } |x - x^*| \text{ small} \right\}.$$

Some remarks

- The definition of condition number is difficult to use to see how good/bad a function is.
- If the function f is continuously differentiable, then we have an easier way to use approximation. By the Mean-Value Theorem,

$$f(x) - f(x^*) = f'(\xi)(x - x^*) \approx f'(x)(x - x^*), \quad \text{as } x^* \approx x,$$

  we have

$$\frac{\left|\frac{f(x)-f(x^*)}{f(x)}\right|}{\left|\frac{x-x^*}{x}\right|} = \left|\frac{f(x)-f(x^*)}{x-x^*}\frac{x}{f(x)}\right| \approx \left|\frac{f'(x)x}{f(x)}\right|,$$

  which is easier to compute.

Examples

- $f(x) = \sqrt{x}$

  $$\text{condition number} \approx \left| \frac{\frac{1}{2}\, x^{-1/2}}{x^{1/2}}\, x \right| \approx \frac{1}{2}.$$

  We say that $f(x)$ is well-conditioned for all $x > 0$.

- $f(x) = \dfrac{10}{(1 - x^2)}$

  $$\text{condition number} \approx \left| \frac{2x^2}{1 - x^2} \right|.$$

  We can find that the condition number is large for $|x| \approx 1$. Therefore, we claim that $f(x)$ is ill-conditioned for $|x| \approx 1$.

Remarks

- **The bad news**: If $f(x)$ is ill-conditioned, there is not much that we can do to accurately compute it, unless use high precision machines.
- **Question**: Can we always obtain a good answer if the function is well-conditioned?

  Answer: Yes! If you have a lot of experience.

An example

- Let $f(x) = \sqrt{x+1} - \sqrt{x}$. Then

$$\begin{aligned} f(12345) &= \sqrt{12346} - \sqrt{12345} \approx 0.111113 \cdot 10^3 - 0.111108 \cdot 10^3 \\ &= 0.005 = 0.5 \cdot 10^{-2}. \end{aligned}$$

  (Suppose computer can store only six digits after the decimal point!)

  Exact answer ≈ 0.0045.

  Relative error $\approx \dfrac{0.005 - 0.0045}{0.0045} \approx 11\%$.

  May be the function is ill-conditioned?

- $x = 12345$.

$$\textit{condition number of } f \textit{ at } x \approx \left| \frac{f'(x)}{f(x)} x \right| = \frac{1}{2} \left| \frac{x}{\sqrt{x+1}\sqrt{x}} \right|.$$

  When x is large, condition number $\approx 1/2$.

  This is a well-conditioned function for large x.

An example (cont'd)

Computing steps:

- Step 1: load $x_0 = 0.12345 \cdot 10^5$.
- Step 2: compute $x_1 = x_0 + 1$.
- Step 3: compute $x_2 = \sqrt{x_1}$.
- Step 4: compute $x_3 = \sqrt{x_0}$.
- Step 5: output $x_4 = x_2 - x_3$.

Reason

- The problem is Step 5.

  Let $g(t) := x_2 - t$, the condition number of $g(t)$ at t is approximately

  $$\left|\frac{g'(t)t}{g(t)}\right| = \left|\frac{t}{x_2 - t}\right|.$$

  Not well-conditioned if $t \approx x_2$

- **Note:** *to obtain a good result from a well-conditioned function, one has to design an algorithm so that every step is well-conditioned.*

How to avoid the bad steps?

1. **Answer:** *change the formula.*

   Example:

$$
\begin{aligned}
f(x) &= \sqrt{x+1} - \sqrt{x} = \frac{(\sqrt{x+1} - \sqrt{x})(\sqrt{x+1} + \sqrt{x})}{\sqrt{x+1} + \sqrt{x}} \\
&= \frac{1}{\sqrt{x+1} + \sqrt{x}}, \quad \text{for } x \gg 1.
\end{aligned}
$$

2. **Other techniques:** *use Taylor's expansion.*

   Example: $x - \sin(x) = x^3/3! - x^5/5! + \cdots, \quad \text{for } x \approx 0.$