

CS450: Numerical Analysis

Howard Hao Yang

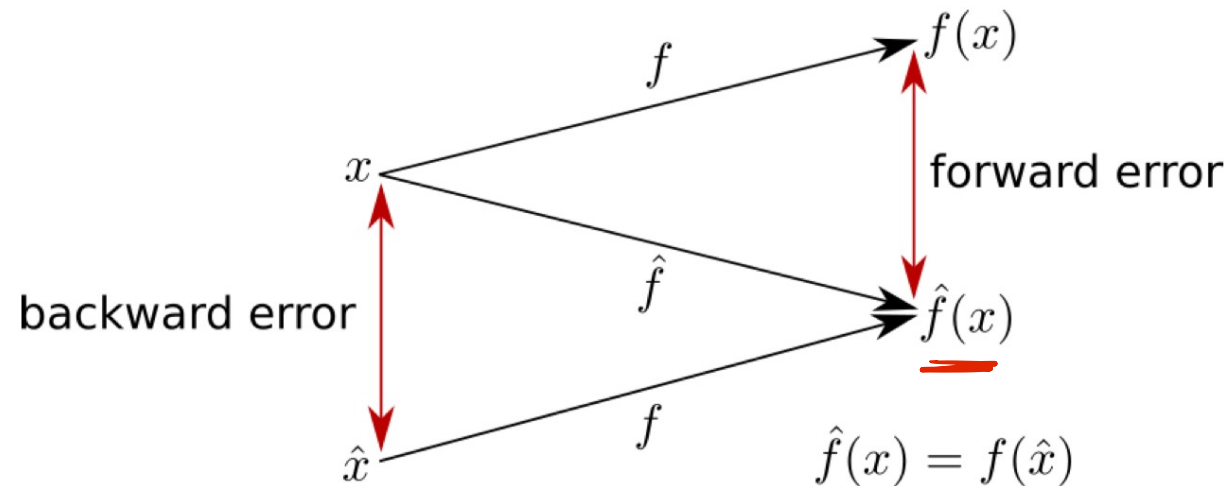
Assistant Professor, ZJU-UIUC Institute

27/09/2023

Recap: Approximation and Errors

- Forward and backward error
 - Suppose we want to compute $y = f(x)$, where $f: R \rightarrow R$, but obtain approximate value \hat{y}
 - Forward error: $\Delta y = \hat{y} - y$
 - Backward error: $\Delta x = \hat{x} - x$, where $f(\hat{x}) = \hat{y}$

$$\frac{\text{fwd-err}}{\text{bck-err}} = \text{cond. \#}$$



Recap: Conditioning

- Absolute condition number
 - The absolute condition number is a **property of the problem**, measuring its sensitivity to perturbations in input
 - Absolute condition number, defined by the ratio of absolute errors at output and input

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|\Delta f\|}{\|\Delta x\|} = |f'(x)|$$

$C_{rel} = \frac{f}{b}$

- Relative condition number, defined by the ratio of relative errors at output and input

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|}$$

$f(x) > 0$
 $\sin(x) \leftarrow ?$
 $x = \pi$

Condition Number

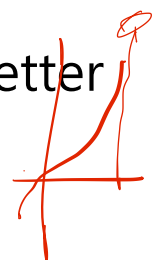
- What if we cannot directly obtain the value of conditioning number?
 - Estimate/bound it
 - Example: evaluate the conditioning number of $f(x) = e^{\sqrt{x}}$ at $x = 2$

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{|x \cdot (e^{\sqrt{x}})'|}{|e^{\sqrt{x}}|} = \frac{1}{2} \sqrt{x} = \frac{1}{2} \sqrt{2} \quad \text{or} \quad \frac{1}{2} \times 2 = 1$$

0.1% error \rightarrow input

\rightarrow 0.1% output

Refreshing Questions

- True or false?
 - A problem is ill-conditioned if its solution is highly sensitive to small changes in the problem data.
 - Using higher-precision arithmetic will make an ill-conditioned problem better conditioned. *→ example: evaluate $\tan(\frac{\pi}{2} - 0.0000001)$* 
 - The conditioning of a problem depends on the algorithm used to solve it.
 - If a computational problem has a condition number of 1, is this good or bad? Why?

Computer Arithmetic

- When conducting computations, we encounter the following

$$2 + 2 = 4, 4 \times 8 = 32, (\sqrt{3})^2 = 3 \dots$$

- Would it hold?: $2 + 2 = 4$
- Would it hold?: $4 \times 8 = 32$
- Would it hold?: $(\sqrt{3})^2 = 3$

\downarrow
 $\approx 1 \times 1^2$

Floating Point Numbers

$$1200 = 1.2 \times 10^3$$

- Scientific notation, format like the following

$$\underbrace{2.103 \times 10^7}_{\text{Scientific notation}} \rightarrow 21,030,000$$

- Why we need this?
- Scientific-notation provides a unique representation of any real number for a given amount of 'precision' (number of significant digits)
- Useful for computer representation & storage
- **Floating point** numbers are a computational realization of scientific notation

Floating Point Numbers (cont'd)



1.010111
 $\beta = 2$
 $1 + \frac{0}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5}$



- Floating point number system is characterized by **four integers**

- Base/radix: $\beta = 2$, **binary** 64 bits
- Precision: p 16
- Exponent range: $[L, U]$

6-bits
 1-bit: sign
 5-bits: $\beta = 10$ $\begin{cases} 2 = 4 \\ 3 \text{ bit} \rightarrow P \\ 2 \text{ bit} \rightarrow E \end{cases}$
 P, E
 $d_0.d_1d_2 \times 10^4$
 9.99×10^4
 4 bit $\rightarrow P$ 1 bit $\rightarrow E$

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Sign, exponent, and mantissa are stored in separate fixed-width fields

$d_0.d_1d_2d_3 \times 10^2$
 9.999×10^2

Floating Point Numbers (cont'd)

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p-1$, and $L \leq E \leq U$

- Sign, exponent, and mantissa are stored in separate fixed-width fields
- A 64-bit (binary digit $\beta = 2$) representation $x = (-1)^s 2^{E-1023} (1 + m)$
 - The first bit is used for **sign** indication $s \in \{0, 1\}$
 - Followed by 11-bit **exponent** E
 - And a 52-bit **mantissa** $m = d_0.d_1d_2 \cdots d_{p-1}$

$$1 + 11 + 52 = 64$$

- Example:

- The left most bit: $s = 0$

- The value is $x = (-1)^0 \cdot 2^{2017-2013} \left(1 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\right)^8 + \left(\frac{1}{2}\right)^{12} \right) =$

27.56640625

Typical Floating-Point Systems

- Parameters for typical floating-point systems

System	β	p	L	U
IEEE SP	2	24	-126	127
IEEE DP	2	53	-1,022	1,023
Cray	2	48	-16,383	16,384
HP calculator	10	12	-499	499
IBM mainframe	16	6	-64	63

- Most modern computers use binary ($\beta = 2$) arithmetic
- IEEE floating-point systems are now almost universal in digital computers

Normalization

$$0.104 \times 10^2 \rightarrow 1.04 \times 10^1$$

- Number x is represented as

$$x = \pm \left(\underline{d_0} + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p-1$, and $L \leq E \leq U$

- Floating-point system is normalized if the leading digit d_0 is always non-zero unless number represented is zero ($d_0 \neq 0$)
- In normalized systems, the mantissa $m = \underline{d_0}.d_1d_2 \cdots d_{p-1}$ always satisfies $1 \leq m < \beta$ (why?)
 $(\beta-1) + 0.x \dots < 1 + \beta-1 = \beta$
 e.g., $\beta=10, [59.9999 \dots] < 10$
- Why we do normalization?

Properties

β

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E \quad \triangleleft \text{FP system}$$

$L \leq E \leq U$

where $0 \leq d_i \leq \beta - 1, i = 0, \dots, p - 1$, and $L \leq E \leq U$

- How many numbers can be represented by this system?
- The number of normalized floating-point numbers is

$$2(\beta - 1)\beta^{p-1}(\underline{U - L + 1}) + 1$$

- Two choices of sign
- $\beta - 1$ choices for leading digit, β choices for the $p - 1$ remaining digits
- $U - L + 1$ possible values for the exponent

Properties (cont'd)

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Largest and smallest representable numbers are
 - Underflow level (smallest positive normalized number): $\text{UFL} = \beta^L$
 - Overflow level (largest floating-point number): $\text{OFL} = \beta^{U+1}(1 - \beta^{-p})$

Exceptional Values

- IEEE floating-point standard provides special values to indicate two exceptional situations
- Largest and smallest representable numbers are
 - Inf, which stands for “infinity”, results from dividing a finite number by zero (e.g., $1/0$)
 - NaN, which stands for “not a number”, results from undefined or indetermined operations such as $0/0$, $0 * \text{Inf}$, or Inf/Inf
- Inf and NaN are implemented in IEEE arithmetic through special reserved values of exponent field

Rounding Error

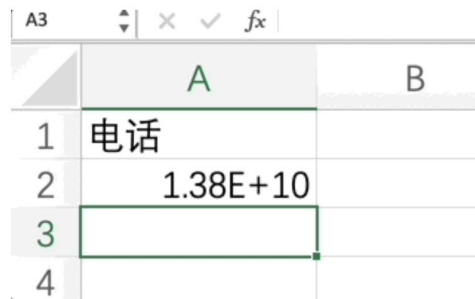
Excel害人事件大赏

Original 陈薛 果壳 2022-09-19 16:00 Posted on 北京

虽然看起来操作人员只是把数字和文字填进表格里，但Excel软件似乎总是有自己的想法。

自动把身份证号转换成科学计数法、把一大堆数字变成了日期.....更不要提复制粘贴表格这种高难度动作了。Excel的每一个错误结果仿佛都写满了“叛逆”。

你永远也不知道按下回车后，刚刚填进单元格里内容会变成什么样。



	A	B
1	电话	
2	1.38E+10	
3		
4		

Excel，谢谢你自作多情的聪明

在网上随便一搜，你能看到各种Excel答疑解惑和学习资料包。也许你曾经也觉得“不就是填个表吗”，但当你看过下面这些Excel害人事件之后，可能真的想去报个班了。

Rounding

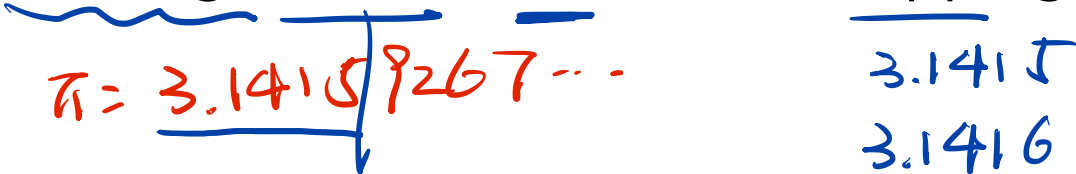
$$d_0.d_1 \dots d_{p-1} \times 10^E$$

- Not all real numbers can be represented exactly by floating-point systems
- The numbers **representable** are called **machine numbers**
- The **numbers not representable are approximated** by “nearby” floating-point numbers
- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero
 - Round to nearest: $\text{fl}(x)$ is nearest floating-point number to x
- Round to nearest is most accurate and is default in IEEE systems

1.414

Rounding (cont'd)

- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero
 - Round to nearest: $\text{fl}(x)$ is nearest floating-point number to x
- What is the five-digit value of π under chopping and round to nearest?



$\pi = 3.14159267\dots$

Chopping: 3.1415

Round to nearest: 3.1416
- The error that results from replacing a number with its floating-point form is called **round-off error** regardless of whether the rounding to nearest or chopping is applied

Example: Machine Precision

- Task: evaluate the following sum

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

- Ideally, what is the result?
- When doing computation...
- What happens?

```
In [2]: import numpy as np

In [*]: n = int(0)

float_type = np.float32

my_sum = float_type(0)

while True:
    n += 1
    last_sum = my_sum
    my_sum += float_type(1 / n)

    if n % 200000 == 0:
        print("1/n = %g, sum0 = %g"%(1.0/n, my_sum))
```

1/n = 5e-06, sum0 = 12.7828
1/n = 2.5e-06, sum0 = 13.4814
1/n = 1.66667e-06, sum0 = 13.8814
1/n = 1.25e-06, sum0 = 14.1666
1/n = 1e-06, sum0 = 14.3574
1/n = 8.33333e-07, sum0 = 14.5481
1/n = 7.14286e-07, sum0 = 14.7388
1/n = 6.25e-07, sum0 = 14.9296
1/n = 5.55556e-07, sum0 = 15.1203
1/n = 5e-07, sum0 = 15.311
1/n = 4.54545e-07, sum0 = 15.4037
1/n = 4.16667e-07, sum0 = 15.4037
1/n = 3.84615e-07, sum0 = 15.4037
1/n = 3.57143e-07, sum0 = 15.4037
1/n = 3.33333e-07, sum0 = 15.4037
1/n = 3.125e-07, sum0 = 15.4037
1/n = 2.94118e-07, sum0 = 15.4037
1/n = 2.77778e-07, sum0 = 15.4037
1/n = 2.63158e-07, sum0 = 15.4037
1/n = 2.5e-07, sum0 = 15.4037
1/n = 2.38095e-07, sum0 = 15.4037
1/n = 2.27273e-07, sum0 = 15.4037
1/n = 2.17391e-07, sum0 = 15.4037
1/n = 2.08333e-07, sum0 = 15.4037

Machine Precision (cont'd)

- Definition 2: (Equivalently,) maximum relative error in representing real number x within range of floating-point system is

$$\left| \frac{\text{fl}(x) - x}{x} \right| \leq \epsilon_{\text{mach}}$$

- Definition (significant digit): The number \hat{x} is said to approximate x to t significant digits if t is the largest nonnegative integer for which

$$\left| \frac{\hat{x} - x}{x} \right| \leq 5 \times 10^{-t}$$

Machine Precision (cont'd)

- Given a number x represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

- The machine epsilon of such a system is
 - With rounding by chopping, $\epsilon_{\text{mach}} = \beta^{1-p}$
 - With rounding to nearest, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$
- For IEEE floating-point systems
 - In single precision: $\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$
 - In double precision: $\epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16}$

Machine Precision (cont'd)

- Exercise

Q1a-4⁵. In 1958 the Russians developed a ternary (base-3) computer called *Setun*, after the Setun River that flows near Moscow State University where it was built. In contrast with today's binary computers, this machine used "trits" (ternary bits) whose three possible states can be represented as $\{0, 1, 2\}$. Its floating-point number system was based on 27-trit numbers, with 9 trits reserved for the exponent and 18 for the mantissa. What was the value of machine epsilon ϵ_M for the *Setun*?

- (A) 3^{-19}
- (B) 3^{-18}
- (C) 3^{-9}
- (D) $\frac{1}{3} \cdot 2^{-18}$

Setun – Moscow State University



{ Source: JMS, plus info from <http://homepage.divms.uiowa.edu/~jones/ternary/numbers.shtml> }

Machine Precision (cont'd)

- Machine precision and underflow level
 - Machine epsilon is determined by number of digits in mantissa
 - Underflow level is determined by number of digits in exponent field
- In all practical floating-point systems

$$0 < \text{UFL} < \epsilon_{\text{mach}} < \text{OFL}$$

Floating-Point Arithmetic

- Rule: When adding two numbers, the exponents align before mantissa adds

$$2.414 \times 10^3 + 1.2 \times 10^2 = 2.414 \times 10^3 + 0.12 \times 10^3 = 2.534 \times 10^3$$

- Addition or subtraction: Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them
- Some familiar laws of real arithmetic are not necessarily valid in floating-point system, e.g., floating-point addition and multiplication are commutative but not associative
 - If ϵ is slightly smaller than the machine precision ϵ_{mach} , then $(1 + \epsilon) + \epsilon = 1$, but $1 + (\epsilon + \epsilon) > 1$

Floating-Point Arithmetic (cont'd)

- Example
- Suppose $x = 1.48234$ and $y = 1.48235$, then $x - y = -0.00001$; but...

Let's make two numbers with very similar magnitude:

```
x = 1.48234  
y = 1.48235
```

Now let's compute their difference in double precision:

```
x_dbl = np.float64(x)  
y_dbl = np.float64(y)  
diff_dbl = x_dbl - y_dbl  
  
print(repr(diff_dbl))
```

```
-1.00000000000065512e-05
```

Floating-Point Arithmetic (cont'd)

- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y add up

$$\kappa_{\text{add}} = \frac{|x+y - \text{fl}(x) - \text{fl}(y)|}{|x+y|}$$

- What happens if $y \approx -x$?

Floating-Point Arithmetic (cont'd)

- Cancellation often implies serious loss of information
- Operands are often uncertain due to rounding or other previous errors, so relative uncertainty in difference may be large
- It is generally bad idea to compute any small quantity as difference of large quantities, since rounding error is likely to dominate result

Rounding Error in Multiplication

- This operation, in general, is safe...
- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y multiply

$$\kappa_{mul} = \frac{|xy - \text{fl}(xy)|}{|xy|}$$

Exceptional and Subnormal Numbers

- Exceptional numbers
- Subnormal numbers