CSI-401 Numerical methods for digital computers **Fall 2018**

Lecture 2: Error and computer arithmetic

Lecturer: Baojian Zhou *Last Update: 09-17-2018*

Abstract

In this lecture, we discuss how the errors could occur during a numerical calculation process. The definitions and examples are mainly from [KW85]. Some related notations are listed in Table 1.

Symbols	Meaning
x	any real number
x_T	the true value of x, i.e., $x = x_T$.
x_A	the approximated value of x because of rounding or errors, i.e., $x_T \approx x_A$
$\bar{x}$	the significand of x using floating-point arithmetic
e	the exponent of x using floating-point arithmetic
b	the base of x using floating-point arithmetic
$\mathrm{fl}(x)$	the machine number of x

Table 2.1: Caption

2.1 Basic concepts

Definition 2.1 (Floating-point arithmetic) *For any nonzero real number x, we can present it as the following*

$$x = \sigma \cdot \bar{x} \cdot b^e, \ \text{such that } \sigma \in \{+1, -1\}, 1 \leq \bar{x} < b, e \in \mathbb{Z}, \tag{2.1}$$

where σ is the ***sign*** *of x, $\bar{x}$ is the* ***significand(mantissa)****, and b is the* ***base****, e is the* ***exponent****. If $b = 2$, Equation (2.1) is called binary floating-point arithmetic. If $b = 10$, Equation (2.1) is called decimal floating-point arithmetic.*

Let us consider two examples. Let $x = (2018.375)_{10}$, its binary and decimal floating-point representation are $(x)_{10} = (2018.375)_{10} = (+1) \cdot (2.018375)_{10} \cdot (10^3)$, $(x)_2 = (11111100010.011)_2 = (+1) \cdot (1.1111100010011)_2 \cdot (2^{10})$. How to use floating-point arithmetic in a digital computer ? There is a popular standard called IEEE 754. We define it in the following.

Definition 2.2 (IEEE floating-point formats)

$$\text{single precision format:} \quad \underbrace{b_1}_{\sigma} \underbrace{b_2, b_3, \ldots, b_9}_{E} \underbrace{b_{10}, b_{11}, \ldots, b_{32}}_{\bar{x}}$$

$$\text{double precision format:} \quad \underbrace{b_1}_{\sigma} \underbrace{b_2, b_3, \ldots, b_{12}}_{E} \underbrace{b_{13}, b_{14}, \ldots, b_{64}}_{\bar{x}}$$

Instead of storing e, we store $E = e + 127$. We present $x = 0$ as $E = 0, \sigma = b_1 = 0, b_{10}b_{11} \ldots, b_{32} = (00 \ldots 0)_2$. We present $x = \pm\infty$ when $b_2 b_3 \ldots b_9 = (11111111)$ and $b_{10}b_{11} \ldots, b_{32} = (00 \ldots 0)_2$, $x = NaN$

when $b_2 b_3 \dots b_9 = (11111111)$ and $b_{10} b_{11} \dots, b_{32} \neq 0$. Question 2: how to represent ∞, NaN(Not a number) in double precision?

machine number: when a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$. In most of cases, we cannot exactly convert x into a machine number,i.e., $x! = fl(x)$. For example, $x = (0.1)_{10} \approx 0.0001\overline{1100}$

machine epsilon: The machine epsilon for any particular floating-point format is the difference between 1 and the next larger number that can be stored in that format. So machine epsilon is to measure how accurately a number can be stored in the floating-point representation. In IEEE single precision format, the machine epsilon is $2^{-23} \approx (1.19 \cdot 10^{-7})_{10}$. Therefore, single precision can be used to store approximately 7 decimal digits for a decimal number. For double precision, $2^{-52} = 2.22 \cdot 10^{-16}$.

precision: The number of allowable digits in $\bar{x}$ is called precision. For example, in IEEE 754 single precision, we have 24 precision. For double, we have 53.

Largest integer M: For single precision $M = 2^{24} = 16777216$, for double precision, $M = 2^{53} \approx 9.0 \cdot 10^{15}$.

Rounding and Chopping: Suppose the floating-point representation contain n binary digits, if a number x has a significant $\bar{x}$ that requires more than n binary bits, then the computer can not store x exactly. There are two ways to store an approximated version, say fl(x), the one is called **chopping**, the other one is called **rounding**. By the definition of a machine number, we can write any number x as $x \approx fl(x) = x \cdot (1 + \epsilon)$. If chopping is used, then $-2^{-n+1} \leq \epsilon \leq 0$. If rounding is used, then $-2^{-n} \leq \epsilon \leq 2^{-n}$. Why the error of chopping is always non-positive ? Is chopping always better than rounding, or the other way ? In practice, How to get higher precision of a real number ? Some tricks can be used in Matlab/Python/C/C++, 1.0D.

–	single precision	double precision
chopping	$-2^{-23} \leq \epsilon \leq 0$	$-2^{-24} \leq \epsilon \leq 2^{-24}$
rounding	$-2^{-52} \leq \epsilon \leq 0$	$-2^{-53} \leq \epsilon \leq 2^{-53}$

Table 2.2: Two different approximation strategies for single and double precision

2.2 Errors

We introduce two concepts to measure the errors occured in the computer. Given a number x, denote x_T as the true value, x_A as the approximated value, we define the following two errors

$$\text{absolute error:} \quad Err(x_A) = x_T - x_A. \tag{2.2}$$

$$\text{relative error:} \quad Rel(x_A) = \frac{Err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}. \tag{2.3}$$

Let $x_T = \pi = 3.1415926535\cdots$ and $x_A = \frac{22}{7} = 3.1428571$, then $Err(x_A = \frac{22}{7}) = \pi - \frac{22}{7} \approx -0.00126\cdots$, $Rel(x_A = \frac{22}{7}) = \frac{\pi - 22/7}{\pi} \approx -0.000402$. Suppose the exact distance between NYC and Albany $x_T = 152mi$, and the Google maps gives me an estimated distance $x_A = 151mi$. Then $Err(x_A) = 1mi$, $Rel(x_A) = 1mi/152mi \approx 0.66\%$. Suppose the exact distance between UAlbany to Crossgates is $x_T = 2mi$, and the Google maps gives me an estimated distance $x_A = 1mi$. Then $Err(x_A) = 1mi$, $Rel(x_A) = 1mi/2mi = 50\%$. We can see that the absolute errors are same. However, the first estimation is much better than the second one, in this case, the relative error is more valuable.

Significant digits: We say x_A has m significant digits with respect to x_T if the magnitude of $Err(x_A)$ is ≤ 0.5 units in the m^{st} digit, beginning with the first nonzero digit in x_T. In other words, given $Rel(x_A)$, find the largest integer m such that $Rel(x_A) \leq 0.5 \cdot 10^{-m}$. You can use the following formula

$$\left|\frac{x_T - x_A}{x_T}\right| \leq 0.5 \cdot 10^{-m} \tag{2.4}$$

then x_A has m significant digits with respect to x_T. If the relative error is greater than 0.5, state that the approximation does not have any significant digits (i.e., the number significant digits is 0). **Example:** $x_A = 3.14$ is to estimate $x_T = \pi$. $Rel(x_A) = |3.14 - \pi|/|\pi| \approx 0.00051 \leq 0.005 = 0.5 \cdot 10^{-2}$, and therefore it has 2 significant digits. Another example, $x_A = 240, x_T = 243.3275$, then $Rel(x_A) = |240 - 243.32753|/|243.32753| \approx 0.014 \leq 0.05 = 0.5 \cdot 10^{-1}$, and therefore it has 1 significant digit. Two examples are from [DH00].

sources of error: There are 5 mains types of errors: (1) modeling errors, one simple model for population growth is given by $N(t) = N_0 e^{kt}$, $N(t)$ equals the population at time t, and N_0 and k are positive constants; (2) mistakes(blunders). Blunders are still often programming errors; (3) physical measurement errors, the radius of an electron is given by $(2.81777 + \epsilon) \cdot 10^{-13}$ cm, where $|\epsilon| \leq 0.00011$(uncertainty principle); (4) rounding/chopping errors. This is the main source of many problems; (5) mathematical approximation errors. This is the main source of error with which we deal in this course. For example, Taylor polynomial approximation $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \approx 1 + x + \frac{1}{2}x^2$. Another example is that $\int_0^1 f(x)dx \approx \frac{1}{n}\sum_{j=1}^{n} f(\frac{j}{n})$.

loss-of-significance errors give an example and show how to avoid them. Define $f(x) = x(\sqrt{x+1} - \sqrt{x})$. If we consider evaluating it on a 6-digit decimal calculator which uses rounded arithmetic. The values of $f(x = 10000) = 49.9988 \approx 50.0$, $f(x = 100000) = 158.113 \approx 100$. Try to reformulate it so as to avoid the loss-of-significance error. $f(x) = \frac{x}{\sqrt{x+1}+\sqrt{x}}$. When two numbers are nearly equal and we subtract them, then we suffer a loss of significance error in the calculation. Some other examples are $f(x) = \frac{1-cos(x)}{x^2}$. Whenever a sum is being formed in which the final answer is much smaller than some of the terms being combined, then a loss of significance error is occurring.

Noise in function evaluation: Whenever a function $f(x)$ is evaluated, there are arithmetic operations carried out which involve rounding or chopping errors. This means that what the computer eventually returns as an answer contains noise. This noise is generally "random" and small. But it can affect the accuracy of other calculations which depend on $f(x)$. Consider $f(x) = (x-1)^3 = -1 + x(3 + x(-3 + x))$, it has only one root $x = 1$.

Underflow and overflow errors: Consider $f(x) = x^{10}$, for the single precision, $-0.00161 < x < 0.000161$, it will underflow. $|x| > 7131.6$ it will overflow.

2.3 Propagation of error

errors in arithmetic operations: Consider the following operations:

$$E = (x_T \omega y_T) - (x_A \omega y_A),$$

where ϵ denotes arithmetic operation such as $+, -, \times, /$. Let w^* denote the same arithmetic operation as it is actually carried out in the computer. We want to obtain $x_T \omega y_T$, but we actually obtain $x_A \omega^* y_A$. The error in $x_A \omega^* y_A$ is given by

$$x_T \omega y_T - x_A \omega^* y_A = \underbrace{\left[x_T \omega y_T - x_A \omega y_A\right]}_{\text{propagated error}} + \underbrace{\left[x_A \omega y_A - x_A \omega^* y_A\right]}_{\text{noise}} \tag{2.5}$$

To simplify, we usually assume $x_A \omega y_A = fl(x_A \omega y_A)$. This means that the quantity $x_A \omega y_A$ is computed exactly and is then rounded or chopped to fit the answer into the floating point representation of the machine.

Let us consider multiplication, i.e. $\omega = *$,

$$Rel(x_A y_A) = \frac{x_T y_T - x_A y_A}{x_T y_T} \tag{2.6}$$

Assume $x_T = x_A + \xi, y_T = y_A + \eta$, then we have

$$\begin{aligned} Rel(x_A y_A) &= \frac{x_T y_T - x_A y_A}{x_T y_T} \\ &= \frac{x_T y_T - (x_T - \xi)(y_T - \eta))}{x_T y_T} \\ &= \frac{x_T \eta + y_T \xi - \xi\eta}{x_T y_T} = Rel(x_A) + Rel(y_A) - Rel(x_A) \cdot Rel(y_A). \end{aligned}$$

Since we usually have $|Rel(x_A)| \ll 1, |Rel(y_A)| \ll 1$, then we have

$$Rel(x_A y_A) \approx Rel(x_A) + Rel(y_A) \tag{2.7}$$

Thus small relative errors in the arguments x_A and y_A leads to a small relative error in the product $x_A y_A$. Also, note that there is some cancellation if these relative errors are of opposite sign.

propagated error in function evaluation: Suppose we are evaluating a function $f(x)$ in the machine. Then the result is generally not $f(x)$, but rather an approximate of it which we denote by $\tilde{f}(x)$. Now suppose that we have a number $x_A \approx x_T$. We want to calculate $f(x_T)$, but instead we evaluate $\tilde{f}(x_A)$. What can we say about the error in this latter computed quantity?

$$f(x_T) - \tilde{f}(x_A) = \underbrace{\Big[f(x_T) - f(x_A)\Big]}_{\text{propagated error}} + \underbrace{\Big[f(x_A) - \tilde{f}(x_A)\Big]}_{\text{noise}} \tag{2.8}$$

If the function $f(x)$ is differentiable, then we can use the "mean-value theorem" to write

$$f(x_T) - f(x_A) = f'(\eta)(x_T - x_A) \tag{2.9}$$

for some $\eta \in [x_T, x_A]$ (without loss of generality, we assume $x_T \le x_A$). Since x_T and x_A are close to each other, we can say η is close to either of them, let us assume $\eta = x_T$ and

$$f(x_T) - f(x_A) \approx f'(x_T)(x_T - x_A) \tag{2.10}$$

Furthermore, if $f(x_T) \neq 0$ and $x_T \neq 0$, we have

$$\begin{aligned} \frac{f(x_T) - f(x_A)}{f(x_T)} &\approx x_T \cdot \frac{f'(x_T)}{f(x_T)} \frac{(x_T - x_A)}{x_T} \\ Rel(f(x_A)) &\approx \underbrace{x_T \cdot \frac{f'(x_T)}{f(x_T)}}_{\text{condition number}} \; Rel(x_A). \end{aligned}$$

The $Rel(f(x_A))$ is dependent on the condition number and $Rel(x_A)$. For example, $f(x) = b^x$, where b is a positive real number. Then we have $Rel(f(x_A)) \approx x_T \log(b) \cdot Rel(x_A)$. Note that $K = 10^4$ and $Rel(x_A) = 10^{-7}$. This is a large decrease in accuracy.

2.4 Summation

Consider the following summation procedure

$$S = a_1 + a_2 + \cdots + a_n = \sum_{i=1}^{n} a_i, \tag{2.11}$$

where each a_i is a floating-point number. There are $n-1$ additions, each of which will probably involve a rounding or chopping error. Define $S_2 = fl(a_1 + a_2) = (a_1 + a_2) \cdot (1 + \epsilon_2)$ and $S_n = fl(a_n + S_{n-1}) \cdot (1 + \epsilon_n)$. Then the error is esimated by

$$S - S_n \approx -a_1(\epsilon_2 + \cdots + \epsilon_n) - a_2(\epsilon_2 + \cdots + \epsilon_n) - a_3(\epsilon_3 + \cdots + \epsilon_n) - \ldots - a_n(\epsilon_n) \qquad (2.12)$$

If we can arrange the numbers $|a_1| \leq |a_2| \leq |a_3| \leq \cdots \leq |a_n|$, then the accumulated error will be reduced. Consider the following

$$S = \sum_{i=1}^{n} a_i b_i, \qquad (2.13)$$

where a_i and b_i are all machine numbers. Imagine calculating this in single precision arithmetic. Then there will be approximately n rounding errors from the multiplications and $n-1$ rounding errors from the additions. To cut this, imagine extending each of the numbers a_i, b_i to double precision by appending sufficient zeros to them. Then multiply and add them in double precision; and when completed, round the answer back to single precision. This replaces $2n-1$ single precision rounding errors with 1 such rounding error, a significant improvement.

References

[KW85] Atkinson, Kendall E., and Weimin Han. , Elementary numerical analysis. *New York et al.: Wiley,* , 1985.

[DH00] Douglas Wilhelm Harder, M.Math. Numerical Analysis for Engineering, University of Waterloo *Department of Electrical and Computer Engineering*