北京大学

Introduction to Roundoff Error

CHEN YIHANG
1700010780

Supervised by
Prof. ZHANG PINGWEN

February 19, 2020

1. IEEE standard

A floating point number sysetem contains 4 integers:

- β: base.
- p: precision.
- $[L, U]$: exponent range, where L is the lower bound and U is the upper bound.

Any $x \in F$ has the following form:

$$x = \pm(d_0.d_1d_2\cdots d_{p-1})_\beta \times \beta^E \tag{1.1}$$

where $0 \le d_i \le \beta - 1$, for $i = 0, 1, \cdots, p-1$. $d_0.d_1d_2\cdots d_{p-1}$ is called mantissa and E is called exponent.

For a real number x, it can only be approximated through rounding rules in the floating number system.We denote the floating approximation as $fl(x)$.

The normalized version of floating-point system requires $d_0 = 1,\ 0 \le d_i < \beta$, for $i = 1, \cdots, p-1$ unless $x = 0$. Since the first term is determined, it gives an extra bit of precision so that the roundoff error is reduced. The effective number of normalized floating point is $2(\beta-1)\beta^{p-1}(U-L+1)+1$.

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for **normalized floating-point** computation which was established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). We can list the standard below

TYPES	β	p	$[L, U]$
Single precision	2	24	$[-126, 127]$
Double precision	2	53	$[-1022, 1023]$

Table 1.1: IEEE standard: settings

The above data is stored by binary representation:

TYPES	SIGN	BIASED EXPONENT	NORMALISED MANTISA	BIAS
Single precision	1(31st bit)	8(30-23 bits)	23(22-0 bits)	127
Double precision	1(63rd bit)	11(62-52 bits)	52(51-0 bits)	1023

Table 1.2: IEEE standard: storage

The effective range of single precision is $\pm 2^{-126}$ to $(2 - 2^{-23}) \times 2^{127}$, of the double precision is $\pm 2^{-1022}$ to $(2 - 2^{-52}) \times 2^{1023}$.

2. Roundoff error

Difference between the results produced by an algorithm when using exact arithmetic and when using finite precision rounded arithmic. One of the goals of numerical analysis is to estimate computation errors, which include both truncation errors and roundoff errors. There are mainly two sources of roundoff errors:

1. Computers have magnitude and precision limits on their ability to present numbers.
2. Certain numerical calculations are highly sensitive to roundoff errors.

2.1. Roundoff error caused by representation

There are two common rounding rules: round-by-chop and round-to-nearest. The IEEE standard uses round-to-nearest.

For $x = \pm(d_0.d_1d_2\cdots d_{p-1}d_p\cdots)_\beta \times \beta^E$,

1. Round-by-chop: $fl(x) = \pm(d_0.d_1d_2\cdots d_{p-1})_\beta \times \beta^E$.
2. Round-to-nearest: $fl(x)$ is set to the nearest floating number to x. In order to reduce biased rounding, when there is a tie, we set d_{p-1} to be even, e.g. $1.050 \approx 1.0,\ 1.051 \approx 1.1$.

If define the machine epsilon to be $\max_x \frac{x-fl(x)}{x}$, then the round-by-chop method's machine epsilon is β^{1-p} and round-to-nearest method's is $\frac{1}{2}\beta^{1-p}$.

2.2. Roundoff error caused by arithmetic

The procedure of floating-point arithmetic can be summerized as follows

- **Step 1:** Round x and y to their floating-point approximation $fl(x)$ and $fl(y)$.
- **Step 2:** Perform exact arithmetic between $fl(x)$ and $fl(y)$.
- **Step 3:** Round the result to its floating-point approximation.

Example 1. $0.1 + 0.2 \neq 0.3$ in Python

$$\begin{aligned} 0.1 &= (1.100110011001\cdots)_2 \times 2^{-4} \\ 0.2 &= (1.100110011001\cdots))_2 \times 2^{-3} \end{aligned} \tag{2.1}$$

Hence, by round-to-nearest approximation, we can get

$$\begin{aligned} fl(0.1) &= (1.(1001)^{12}1010)_2 \times 2^{-4} \\ fl(0.2) &= (1.(1001)^{12}1010)_2 \times 2^{-3} \end{aligned} \tag{2.2}$$

Adding them together, we can get $(1.(0011)^{12}0100)_2 \times 2^{-2}$ (after rounding), which is 0.30000000000000004 instead of 0.3.

Example 2. Subtractive cancellation For $|\epsilon| < \epsilon_m$, then $fl((1+\epsilon) - (1-\epsilon)) = fl(1+\epsilon) - fl(1-\epsilon) = 1 - 1 = 0$.

The error introduced by arithmetic can accumulate in an unstable[1] or ill-conditioned[2] problem, such as Runge phenomenon.[3]

3. Other floating point system

3.1. Arbitary-precision arithmetic

In computer science, arbitrary-precision arithmetic, also called bignum arithmetic, multiple-precision arithmetic, or sometimes infinite-precision arithmetic, indicates that calculations are performed on numbers whose digits of precision are limited only by the available memory of the host system. This contrasts with the faster fixed-precision arithmetic found in most arithmetic logic unit (ALU) hardware, which typically offers between 8 and 64 bits of precision. Arbitrary-precision arithmetic is considerably slower than arithmetic using numbers that fit entirely within processor registers, since the latter are usually implemented in hardware arithmetic whereas the former must be implemented in software.

Applications: Arbitrary precision is used in applications where the speed of arithmetic is not a limiting factor, or where precise results with very large numbers are required, such as public-key cryptography, computation of fundamental mathematical constants, computation of fractal images such as Mandelbrot set.

[1] Usually in a sequential calculation.

[2] A problem is well-conditioned if small relative changes in input result in small relative changes in the solution. Otherwise. the problem is called ill-conditioned

[3] Since high order polynomials are sentitive to roundoff error.

3.2. Extended precision

We use long double in C as an example. On the x86 architecture, most C compilers implement long double as the 80-bit extended precision type supported by x86 hardware (sometimes stored as 12 or 16 bytes to maintain data structure alignment), as specified in the C99 / C11 standards. [4]

This 80-bit format uses one bit for the sign of the significand, 15 bits for the exponent field and 64 bits for the significand. The exponent field is biased by 16383. Note that it is an unnormalized floating-point sysetem.

TYPES	SIGN	BIASED EXPONENT	INTERGER PART	FRACTION	BIAS
Long double	1(79st bit)	15(64-78 bits)	1(63 bit)	63(0-62 bit)	16383

Table 3.1: x86 long double

4. Disasters caused by errors

- **The Patriot Missile failure** in Dharan, Saudi Arabia, on February 25, 1991 which resulted in 28 deaths, is ultimately attributable to poor handling of rounding errors.[Defense, 1992]
  It turns out that the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors. Specifically, the time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds. This calculation was performed using a 24 bit fixed point register. In particular, the value 1/10 was chopped at 24 bits after the radix point. In other words, the binary expansion of 1/10 is 0.0001100110011001100110011001100.... Now the 24 bit register in the Patriot stored instead 0.000110011001100110011001100 introducing an error of 0.000000000000000000000011001100... binary, or about 0.000000095 decimal. The Patriot battery had been up around 100 hours. Multiplying by the number of tenths of a second in 100 hours gives $0.000000095 \times 100 \times 60 \times 60 \times 10 = 0.34$.) A Scud travels at about 1,676 meters per second, and so travels more than half a kilometer in this time. This was far enough that the incoming Scud was outside the "range gate" that the Patriot tracked.
- **The explosion of the Ariane 5 rocket** just after lift-off on its maiden voyage off French Guiana, on June 4, 1996, was ultimately the consequence of a simple overflow. [Lions et al., 1996]
  The failure of the Ariane 501 was caused by the complete loss of guidance and altitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.
  The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

References

[Defense, 1992] Defense, P. M. (1992). Software problem led to system failure at dhahran, saudi arabia. *US GAO Reports, report no. GAO/IMTEC-92-26*.

[Lions et al., 1996] Lions, J.-L., Luebeck, L., Fauquembergue, J.-L., Kahn, G., Kubbat, W., Levedag, S., Mazzini, L., Merle, D., and O'Halloran, C. (1996). Ariane 5 flight 501 failure report by the inquiry board.

[4] An exception is Microsoft Visual C++ for x86, which makes long double a synonym for double. The Intel C++ compiler on Microsoft Windows supports extended precision, but requires the /Qlong-double switch for long double to correspond to the hardware's extended precision format.