Lecture Notes for Math 5630/6630 **Fall 2024**

Note 2: Floating Point Numbers

Tags: `math.na` *Date: 08/27/2024*

Disclaimer: *This lecture note is for math 5630/6630 class only.*

If we evaluate the boolean expression `0.1 + 0.2 == 0.3` using Python or MATLAB, we will find the computer returns a `false` or a logical zero. This means Python or MATLAB treats numbers differently.

1 Numbers in Computer

It is well known that there are infinitely many real numbers inside any interval.

Prove by contradiction. If there are only a finite number of them inside I, denote the set as $\mathcal{A}$, we can order them as $a_1 < a_2 < \cdots < a_n$, then $\frac{a_1+a_2}{2} \in (a_1, a_2)$ is a real number but not a member of $\mathcal{A}$.

The computer can only store a finite number of numbers with a finite storage device and limited runtime memory. An efficient way to store the numbers is important.

1.1 Representation of Real Numbers

A straightforward way to store a real number is to save its "digits". For instance, a number 3.14159 can be stored by saving the digits $3, 1, 4, 1, 5, 9$ and the position of the decimal point.

$$3.14159 = 3.14159 \times 10^{0}.$$

Saving just digits and the decimal point position is not enough, because we cannot distinguish -3.14159 and 3.14159. Therefore, we at least need the sign, the digits, and the decimal point to uniquely determine a number.

It is still not enough. Let's consider two real numbers $9.9999999999\cdots \times 10^{-1}$ and 1×10^{0}. These two numbers are identical, but the first one needs infinitely many digits.

To show these two are identical, one can write the first one as the limit

$$\sum_{k=1}^{\infty} 9 \times 10^{-k} = 9 \times \frac{10^{-1}}{1 - 10^{-1}} = 1.$$

To eliminate the ambiguity caused by using infinitely many digits, we need an extra assumption that there are **infinitely many unused** "digits".

1.2 Binary Representation

A more computer-friendly way to represent numbers is using binary representation. The binary states can be easily implemented through circuit gates, which provide hardware acceleration.

In binary representation, a real number can be written as

$$\pm\, d_0.d_1d_2d_3 \cdots d_{t-1} \cdots \times 2^e$$

and $d_i \in \{0,1\}$. The leading symbol $+$ or $-$ is the sign. The digits $\{d_i\}_{i\geq 0}$ are called the **mantissa**

$$d_0.d_1d_2d_3 \cdots d_{t-1} \cdots = d_0 + \frac{d_1}{2} + \frac{d_2}{2^2} + \cdots + \frac{d_{t-1}}{2^{t-1}} + \cdots$$

The integer e is the **exponent**. Using a finite number of bits for mantissa and exponent parts brings us the binary floating point system.

2 Floating Point System

A general floating point system $\mathbb{F}(\beta, t, L, U)$ is determined by 4 values, where

β is the base.

t is the number of digits.

L is the lower bound of the exponent e.

U is the upper bound of the exponent e.

An element in the floating point system can be written as

$$\pm\left(\frac{d_0}{\beta^0} + \frac{d_1}{\beta^1} + \cdots \frac{d_{t-1}}{\beta^{t-1}}\right) \times \beta^e \tag{0.1}$$

The digits $d_i \in \{0, 1, \cdots, \beta - 1\}$.

Notice that such representation will introduce some ambiguity. For instance,

$$1.25 = +(1 + \frac{0}{2} + \frac{1}{2^2}) \times 2^0 \quad \text{(normalized)},$$
$$1.25 = +(0 + \frac{1}{2} + \frac{0}{2^2} + \frac{1}{2^3}) \times 2^1 \quad \text{(denormalized)}.$$

We call the first representation ($d_0 \neq 0$) normalized and the second ($d_0 = 0$) denormalized.

To be precise, the floating point system $\mathbb{F}$ is the following set

$$\mathbb{F}(\beta, t, L, U) = \{x \in \mathbb{R} \mid x = \pm \left(\frac{d_0}{\beta^0} + \frac{d_1}{\beta^1} + \cdots \frac{d_{t-1}}{\beta^{t-1}} \right) \times \beta^e, d_0 \neq 0, L \leq e \leq U\} \cup \{0\}. \tag{0.2}$$

Example 2.1. *The largest element in $\mathbb{F}(\beta, t, L, U)$ is $x_{max} = (1 - \beta^{-t}) \times \beta^{U+1}$. The smallest positive number is $x_{min} = \beta^L$. Therefore*

$$\mathbb{F}(\beta, t, L, U) \subset \mathbb{I} := \{x \in \mathbb{R} \mid x_{min} \leq |x| \leq x_{max}\} \cup \{0\} \tag{0.3}$$

There are $(U - L + 1) \times 2 \times (\beta - 1) \times \beta^{t-1} + 1$ floating point numbers in total.

In the following, we can see the distribution of the floating point numbers is equidistant on each interval $[\beta^e, \beta^{e+1}]$.

Theorem 2.2. *For any $L \leq e \leq U$, the distribution of the floating point number system $\mathbb{F}(\beta, t, L, U)$ on the interval $[\beta^e, \beta^{e+1}]$ is equidistant with distances of length β^{e-t+1}.*

This is because for any $x \in \mathbb{F}(\beta, t, L, U) \cap [\beta^e, \beta^{e+1}]$, it is given by the form

$$x = (\frac{d_0}{\beta^0} + \cdots \frac{d_{t-1}}{\beta^{t-1}}) \times \beta^e, \quad d_0 \geq 1. \tag{0.4}$$

The mantissa is equidistantly distributed with distance $\beta^{-(t-1)}$, therefore the floating point numbers are equidistantly distributed with distance $\beta^{-(t-1)+e}$.

2.1 Machine Precision

To understand the approximation to real numbers by the floating point number system $\mathbb{F}(\beta, t, L, U)$, it is important to consider the maximal relative distance between the numbers in $\mathbb{F}(\beta, t, L, U)$ and their respective closest element in the superset $\mathbb{I} := \{x \in \mathbb{R} \mid x_{min} \leq |x| \leq x_{max}\} \cup \{0\}$, which is the following quantity:

$$\max_{x \in \mathbb{I}, x \neq 0} \min_{z \in \mathbb{F}(\beta, t, L, U)} \frac{|z - x|}{|x|}$$

Theorem 2.3.
$$\max_{x\in\mathbb{I},x\neq 0}\min_{z\in\mathbb{F}(\beta,t,L,U)}\frac{|z-x|}{|x|}\leq\frac{1}{2}\beta^{1-t}.$$
The number $\eta := \frac{1}{2}\beta^{1-t}$ *is also called the rounding unit or machine precision.*

For $x\in[x_{min},x_{max}]$, we can write in β-base number
$$x=(d_0.d_1d_2\cdots d_{t-1}d_t\cdots)\times\beta^e\in[\beta^e,\beta^{e+1}]. \tag{0.5}$$
Since the floating point numbers are equidistantly distributed on $[\beta^e,\beta^{e+1}]$, we can find $z^*\in\mathbb{F}(\beta,t,L,U)$ such that
$$|z^*-x|\leq\frac{1}{2}\beta^{e-t+1}. \tag{0.6}$$
Then,
$$\frac{|z^*-x|}{|x|}\leq\frac{1}{2}\frac{\beta^{e-t+1}}{\beta^e}=\frac{1}{2}\beta^{1-t}. \tag{0.7}$$

2.2 Chopping & Rounding

There are two usual ways to approximate a real number $x\in\mathbb{I}$ by the floating point system. Writing $x=\pm(d_0.d_1d_2\cdots d_{t-1}d_t\cdots)\times\beta^e$, we use `fl` to represent the approximation to x.

1. Chopping: Ignore the digits $d_t,d_{t+1},\cdots$ and only keep the first t digits, we denote
$$\mathtt{fl}_c(x)=\pm(d_0.d_1d_2\cdots d_{t-1})\times\beta^e. \tag{0.8}$$

2. Rounding (to nearest even): the digit d_t will be chosen to make the floating point number as close to x as possible.
$$\mathtt{fl}_r(x)=\begin{cases}\pm(d_0.d_1d_2\cdots d_{t-1})\times\beta^e, & d_t<\beta/2\\ \pm(d_0.d_1d_2\cdots d_{t-1}+\beta^{1-t})\times\beta^e, & d_t>\beta/2.\end{cases}$$
In the special case that $d_t=\frac{\beta}{2}$, it rounds to the nearest **even** number. There are several other rounding strategies, see Rounding - Wikipedia.

Example 2.4. *For instance, if we take the floating point system* $\mathbb{F}(10,3,-5,5)$, *then*

	chopping	*rounding*
1.237	1.23	1.24
1.232	1.23	1.23
−1.235	−1.23	−1.24
−1.245	−1.24	−1.24

The relative approximation error of rounding is bounded by η according to the same idea behind Theorem **??**.

Theorem 2.5. *With rounding, for any $x \in \mathbb{I}, x \neq 0$, the relative approximation error*

$$\frac{|\mathtt{fl}_r(x) - x|}{|x|} \leq \eta,$$

We can similarly obtain the relative error for chopping.

Theorem 2.6. *With chopping, for any $x \in \mathbb{I}, x \neq 0$, the relative approximation error*

$$\frac{|\mathtt{fl}_c(x) - x|}{|x|} \leq 2\eta.$$

For convenience, we will use rounding as the default approximation method and denote $\mathtt{fl}(x) = \mathtt{fl}_r(x)$ as the rounded (to nearest) floating point number for $x \in \mathbb{I}$.

Remark 2.7. *On modern computers with binary representations $\beta = 2$, the definition of machine precision has two versions.*

The above formal definition appears mostly in the research literature and numerical packages (LAPACK). In modern programming languages like `Python`, MATLAB, C++, the machine precision is defined by 2^{1-t} instead. The meaning is the difference between one and the next floating point number, it is also known as `ulp` (unit in the last place).

In other words, two versions of machine precision are corresponding to different rounding strategies. For the former, the rounding strategy is to round to the nearest floating point number, while for the latter, the rounding strategy is to round by chop.

In practice, it is not necessary to distinguish the two versions of machine precision, since the difference is only a factor of 2.

It is clear that the rounding function `fl` is monotone and idempotent, which means

Theorem 2.8. *If $x \leq y$, then $\mathtt{fl}(x) \leq \mathtt{fl}(y)$.*

Theorem 2.9. *If $z \in \mathbb{F}(\beta, t, L, U)$, then $\mathtt{fl}(z) = z$.*

Corollary 2.10. *As a direct result of Theorem **??**, for any $x \in \mathbb{I}$, we can write*

$$\mathtt{fl}(x) = x(1 + \delta)$$

with $|\delta| \leq \eta$.

Remark 2.11. *Most computers implement the IEEE 754 2008 standard. The single precision is known as* $\mathbb{F}(2, 24, -126, 127)$, *its* $\eta = 2^{-24}$ *and the double precision is known as* $\mathbb{F}(2, 53, -1022, 1023)$, *its* $\eta = 2^{-53}$. *The single precision (32 bit) layout is like*

$$sign \mid e_7 e_6 e_5 \cdots e_0 \mid d_1 d_2 \cdots d_{23}$$

representing the number

$$\pm(1.d_1 d_2 \cdots d_{23}) \times 2^{\mathrm{e}}$$

where the exponent $\mathrm{e} = (e_0 e_1 \cdots e_7)_2 - 127$.

Notice that 8 bits actually can have $2^8 = 256$ *different values, but the IEEE754 standard reserves two special values (all zeros and all ones):* $\mathrm{e} = -127$ *and* $\mathrm{e} = 128$.

`Inf` *is the representation of* $\pm\inf$, *when the exponent bits are all ones and mantissa bits are all zeros, that means* $\mathrm{e} = 128$ *and* $d_i = 0$ *for all* i.

`NaN` *is the representation of Not a Number, the exponent bits are all ones and the mantissa bits are not all zeros, that means* $\mathrm{e} = 128$ *but not all* $d_i = 0$.

`subnormal` *numbers occur when the exponent bits are all zeros and the leading mantissa bit becomes zero, that means* $\mathrm{e} = -127$ *and it uses another formula to compute numbers*

$$\pm(0.d_1 d_2 \cdots d_{23}) \times 2^{-126}$$

to "smoothly" transit from $x_{min} = 2^{-126}$ *to numbers below that.*