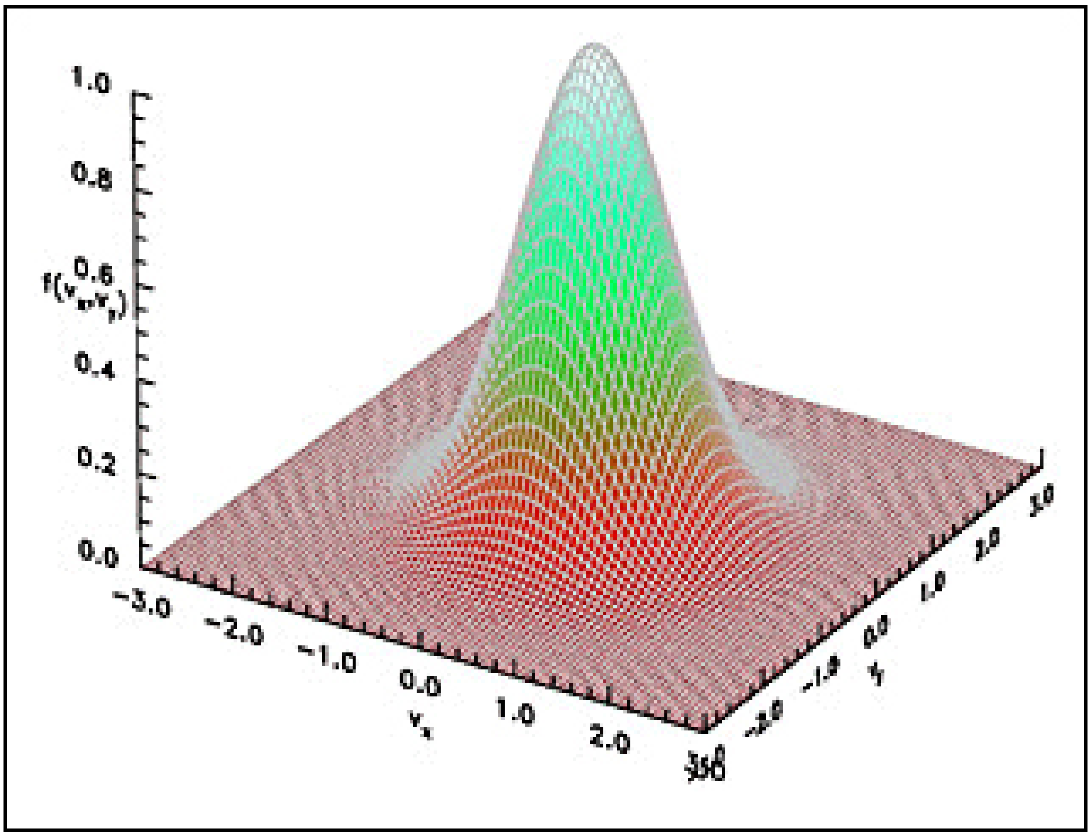
CSIE5123: Numerical Method

Vinsong

February 24, 2026

Abstract

The lecture note of 2026 Spring Numerical Method by professor 林智仁.



Contents

Chapter 1

Floating-point systems

Lecture 1

2025.12.09

1.1 Floating-point basics

This chapter is mainly based on the science of floating-point arithmetics which is based on the IEEE standard 754.

> **Remark.** The floating-point number system here introduction is to help us redesign a better floating-point system while we are doing deep learning implementation.

1.1.1 Why learning floating-point operations?

> **Example.** A one-variable problem
>
> $$\min_x f(x) \quad \text{where } x \geq 0$$

In the normal program, we should set an upper bound of x, or x may be wrongly increased to ∞. We have to find the largest representable number in the computer. Or how to define the "infinity" in the computer?

> **Example.** A ten-variable problem
>
> $$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{where } x_i \geq 0, i = 1, 2, \ldots, 10$$

We want to know how many are zeros, we may use

```c
for (int i = 0; i < 10; i++)
    if (x[i] == 0) count++;
    // count the number of zeros
```

But people say that don't do the comparison of floating-point due to the precision issue.

```c
double epsilon = 1.0e-12;
for (int i = 0; i < 10; i++)
    if (x[i] <= epsilon) count++;
```

Which is better? How to chose "`epsilon`"? Can't do the comparison of floating-point? We need to understand the floating-point representation.

1.1.2 Floating-point Format

We know `float` (single precision): 4 bytes and `double` (double precision) 8 bytes in C/C++.

Note. We usually spare the bits for deeplearning system to save memory and speed up the training process.

Definition 1.1.1 (format)**.** A floating-point system requires

- a base β
- precision p
- significand (mantissa) $d.d \dots d$

Example.

$$\begin{aligned} 0.1 &= 1.00 \times 10^{-1} && (\beta = 10, p = 3) \\ &\approx 1.1001 \times 2^{-4} && (\beta = 2, p = 5) \end{aligned}$$

exponent: -1 and -4; significand: 1.00 and 1.1001.

We let $e_{\max}$ to be the largest exponent and $e_{\min}$ to be the smallest exponent. Then, β^p is the possible significand, and $e_{\max} - e_{\min} + 1$ is the possible exponent.

$$\lceil \log_2(e_{\max} - e_{\min} + 1) \rceil + \lceil \log_2(\beta^p) \rceil + 1$$

bits for storing a floating-point number. 1 bit for the sign.

In the pratical it is more complicated.

Definition 1.1.2 (Normalized floating-point number)**.** A normalized floating-point number is a number whose significand is in the form of $1.d_1 d_2 \dots d_{p-1}$.

Example. 0.1 is not; 1.00×2^{-3} is

Now the issue will become represent of zero. We natually use $1.0 \times \beta^{e_{\min}-1}$ to represent zero, but it is not allow in computer system.

Remark. We usually reserve some special numbers to represent 0, ∞, and NaN.

1.1.3 Relative error and Ulps

Example. When $\beta = 10, p = 3$, for 3.14159

We represent it as 3.14×10^0, and the error is $|3.14159 - 3.14| = 0.00159 = 0.159 \times 10^{-2}$ i.e.

- 10^{-2} is the unit of the last place.
- `ulps`: 0.159 is the unit in the last place

The absolute error is 0.159 ulps.

Definition 1.1.3 (ulps)**.** The unit in the last place (ulps) is the unit value of the least significant digit in a floating-point number.

We also use "relative error" to measure the error. The relative error is $0.00159/3.14159 \approx 0.0005$. For a number $d.d \ldots d \times \beta^e$, the largest error cause by rounding is

$$0.\underbrace{0\ldots0}_{p-1}\beta' \times \beta^e \quad \beta' = \beta/2$$

So, Error $= \frac{\beta}{2} \times \beta^{-p} \times \beta^e$ and

$$1 \times \beta^e \leq \text{original number} < \beta \times \beta^e$$

The relative error will between

$$\frac{\frac{\beta}{2} \times \beta^{-p} \times \beta^e}{\beta^e} \text{ and } \frac{\frac{\beta}{2} \times \beta^{-p} \times \beta^e}{\beta^{e+1}}$$

So,

$$\text{relative error} \leq \frac{\beta}{2} \times \beta^{-p} \tag{1.1.1}$$

Definition 1.1.4 (machine epsilon)**.** The machine epsilon is the upper bound of the relative error of a floating-point number $\epsilon = \frac{\beta}{2} \times \beta^{-p} = \beta^{1-p}/2$

Example. $x = 12.35 \;\Rightarrow\; \tilde{x} = 1.24 \times 10^1 \quad (p = 3,\; \beta = 10)$

- 1 ulps $= 0.01 \times 10^1 = 0.1$, $\epsilon = 1/2 \cdot 10^{-2} = 0.005$
- error $= 0.05 = 0.005 \times 10^1 = 0.5$ ulps
- relative error $= 0.05/12.35 \approx 0.004 = 0.8\epsilon$
- $8x = 98.8,\; 8\tilde{x} = 9.92 \times 10^1$ error $= 4.0$ ulps, relative error $= 0.04 = 0.8\epsilon$

Note. ulps and ϵ is used interchangeably usually.

1.2 Guard digits

In some situation, the order of floating-point computation and rounding may not have a huge effect on the final result. So, usually we choose to

$$\boxed{\text{Round}} \quad \Rightarrow \quad \boxed{\text{Compute}}$$

to let the computation to be more efficient. But in some cases, the error may be huge.

Example.

$$x = 10.1$$
$$y = 9.93$$
$$x - y = 0.17$$

- Round and then compute:

$$10.1 - 9.93 = 1.01 \times 10^1 - 0.99 \times 10^1 = 0.02 \times 10^1 = 2.00 \times 10^{-1}$$

  - error $= 2.00 \times 10^{-1} - 0.17 = 0.03$
  - ulps $= 0.01 \times 10^{-1} = 10^{-3}$
  - error $= 30$ ulps
  - relative error

$$= \frac{0.03}{0.17} = \frac{3}{17}$$

- Compute and then round:

$$10.1 - 9.93 = 0.17 \approx 1.7 \times 10^{-1}$$

  - error $= 1.7 \times 10^{-1} - 0.17 = 0$

So, we have to know how big will the error be if we choose to round first then compute.

Theorem 1.2.1. Using p precision with base β for $x - y$, the relative error can be as large as $\beta - 1$

Proof. Let

$$x = 1.0\dots0,\; y = 0.\underbrace{\eta\dots\eta}_{p\text{ digits}}, \quad \eta = \beta - 1$$

The correct solution is $x - y = \beta^{-p}$. Under some rounding scheme, we have the solution

$$1.0\dots0 - 0.\underbrace{\eta\dots\eta}_{p-1\text{ digits}} = \beta^{-p+1}$$

Note. Above is a bad rounding just ignore the last η

Relative error is

$$\frac{|\beta^{-p+1} - \beta^{-p}|}{\beta^{-p}} = \beta - 1$$

■

Example. $(p = 3,\ \beta = 10)\ x = 1.00,\ y = 0.999,\ x - y = 0.001 = 10^{-3}$

The compute solution is $1.00 \times 10^0 - 0.99 \times 10^0 = 0.01 \times 10^0 = 0.01$. The relative error is

$$\frac{|0.01 - 0.001|}{0.001} = 9$$

Such a huge error occurs when the two numbers are close to each other. So, we need to use guard digits to store the intermediate result to avoid such a huge error.

Definition 1.2.1 (guard digits). Guard digits are extra digits used in the intermediate steps of a calculation to reduce the effect of rounding errors and improve the accuracy of the final result.

In here, the guard digits is p increased by 1 in the device for addition and subtraction

For the above example, we have the computation become

$$1.010 \times 10^1 - 0.993 \times 10^1 = 0.017 \times 10^1 = 0.17$$

note that $0.17 = 1.70 \times 10^{-1}$ can be stored in the system with $p = 3$ and $\beta = 10$. The relative error becomes 0.

Note. We only allocate more digit precision for the calculation device (i.e. the substraction process) to store the intermediate result.

Example. $(p = 3,\ \beta = 10)$ $x = 110,\ \ y = 8.59, x - y = 101.41$

$$\begin{aligned} 110 - 8.59 &= 1.100 \times 10^2 - 0.085 \times 10^2 \\ &= 1.015 \times 10^2 \approx 1.01 \times 10^2 \end{aligned}$$

Relative error is

$$\frac{|1.01 \times 10^2 - 101.41|}{101.41} \approx 0.003$$

$\epsilon = \frac{1}{2} \cdot 10^{-2} = 0.005$.

Theorem 1.2.2. Using $p + 1$ digits for $x - y \ \Rightarrow$ relative rounding error $< 2\epsilon$ (ϵ: machine epsilon)

Proof. We first make some assumption

- $x > y$
- $x = x_0 \,.\, x_1 x_2 \ldots x_{p-1} \times \beta^0$, the proof will be similar for not β^0

For the first two condition,

1. If $y = y_0 \,.\, y_1 y_2 \ldots y_{p-1}$ no error
2. If $y = 0.y_1 y_2 \ldots y_p \ \Rightarrow$ 1 guard digit, exact $x - y$ rounded to a closest number $\Rightarrow$ error $< \epsilon$

In general, $y = 0.0 \ldots 0\ y_{k+1} \ldots y_{k+p}$, we let $\bar{y}$: truncate y to $p + 1$ digits.

$$|y - \bar{y}| < (\beta - 1)(\beta^{-p-1} + \beta^{-p-2} + \cdots + \beta^{-p-k}) \tag{1.2.1}$$

After the truncation, we have to compute

$$x - \bar{y}$$

Then we rounded the result get

$$x - \bar{y} + \delta$$

Thus

$$\text{error} \leq 0.\ \underbrace{0 \ldots 0}_{p-1 \text{ digits}}\ (\beta/2) \quad \text{(worst case)}$$

Therefore,

$$|\delta| \leq (\beta/2) \cdot \beta^{-p} \tag{1.2.2}$$

Then the error is

$$(x - y) - (x - \bar{y} + \delta) = \bar{y} - y - \delta$$

Now, we consider three cases:

1° Case 1: $x - y \geq 1$, from (1.2.1) and (1.2.2), we have the relative error

$$\begin{aligned}
\frac{|\bar{y} - y - \delta|}{x - y} &\leq \frac{|\bar{y} - y - \delta|}{1} \\
&\leq \beta^{-p}[(\beta - 1)(\beta^{-1} + \beta^{-2} + \cdots + \beta^{-k}) + \beta/2] \\
&= \beta^{-p}[(\beta - 1)\beta^{-k}(1 + \cdots + \beta^{k-1}) + \beta/2] \\
&= \beta^{-p}[(\beta - 1)\beta^{-k} \cdot \frac{\beta^k - 1}{\beta - 1} + \beta/2] \\
&= \beta^{-p}[(1 - \beta^{-k}) + \beta/2] \\
&< \beta^{-p}[1 + \beta/2] \leq 2\epsilon
\end{aligned}$$

2° Case 2: $x - \bar{y} \leq 1$, enough digits to store $x - \bar{y}$, so the $\delta = 0$, and the relative error is

$$\frac{|\bar{y} - y|}{x - y}$$

The smallest $x - y$ is

$$1.0 - 0.0\ldots0\,\rho\ldots\rho > (\beta - 1)(\beta^{-1} + \beta^{-2} + \cdots + \beta^{-k})$$

we have k zeros, p ρ and $\rho = \beta - 1$, from (1.2.1), we have the relative error

$$\begin{aligned}
&\leq \frac{|\bar{y} - y|}{(\beta - 1)(\beta^{-1} + \beta^{-2} + \cdots + \beta^{-k})} \\
&< \frac{(\beta - 1)\beta^{-p}(\beta^{-1} + \beta^{-2} + \cdots + \beta^{-k})}{(\beta - 1)(\beta^{-1} + \beta^{-2} + \cdots + \beta^{-k})} = \beta^{-p} = 2\epsilon
\end{aligned}$$

3° Case 3: $x - y < 1$ but $x - \bar{y} > 1$
If $x - \bar{y} = 1.\underbrace{0\ldots1}_{p} \Rightarrow x - y \geq 1$: a contradiction.
Because

$$|y - \bar{y}| < \beta^{-p} = 0.\underbrace{0\ldots1}_{p}$$

then

$$x - y = \underbrace{(1 + \beta^{-p})}_{x - \bar{y}} - \underbrace{|y - \bar{y}|}_{<\beta^{-p}} > 1$$

Proof is completed. ■

Conclusion: we can add "some" (not only 1) guard digits to make the relative error small enough. Especially when the two numbers are close to each other. To add one bit on the adder is very cheap.