Video 1: Rounding errors

A number system can be represented as $x = \pm 1.b_1 b_2 b_3 b_4 \times 2^m$ for $m \in [-6,6]$ and $b_i \in \{0,1\}$.

$n = 4$

Let's say you want to represent the decimal number 19.625 using the binary number system above. Can you represent this number exactly?

$(19.625)_{10} = (10011.101)_2 = (1.0011101)_2 \times 2^4$

either

$1.0011 \times 2^4 = 19$

$1.0100 \times 2^4 = 20$

} this process is called rounding

and these are different rounding rules

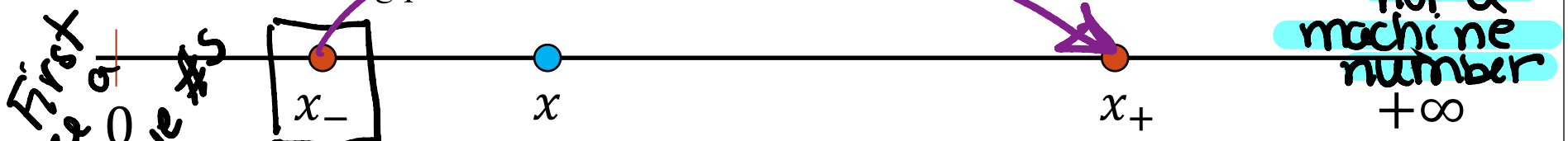
Machine floating point number

- Not all real numbers can be exactly represented as a machine floating-point number.
- Consider a real number in the normalized floating-point form:

$$x = \pm 1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m$$

$b_{n+1} b_{n+2} \ldots$ → cannot represent exactly using FP → not a machine number

- The real number x will be approximated by either x_- or x_+, the nearest two machine floating point numbers.



First take a look at positive #s

$x_- = 1.b_1 b_2 b_3 \ldots b_n \times 2^m$ (nearest machine number "smaller" than x)

$x_+ = x_- + \underbrace{0.000\ldots01}_{2^{-n}} \times 2^m$ → nearest machine number "greater" than x

2^{-n} ↓ last bit

ϵ_m

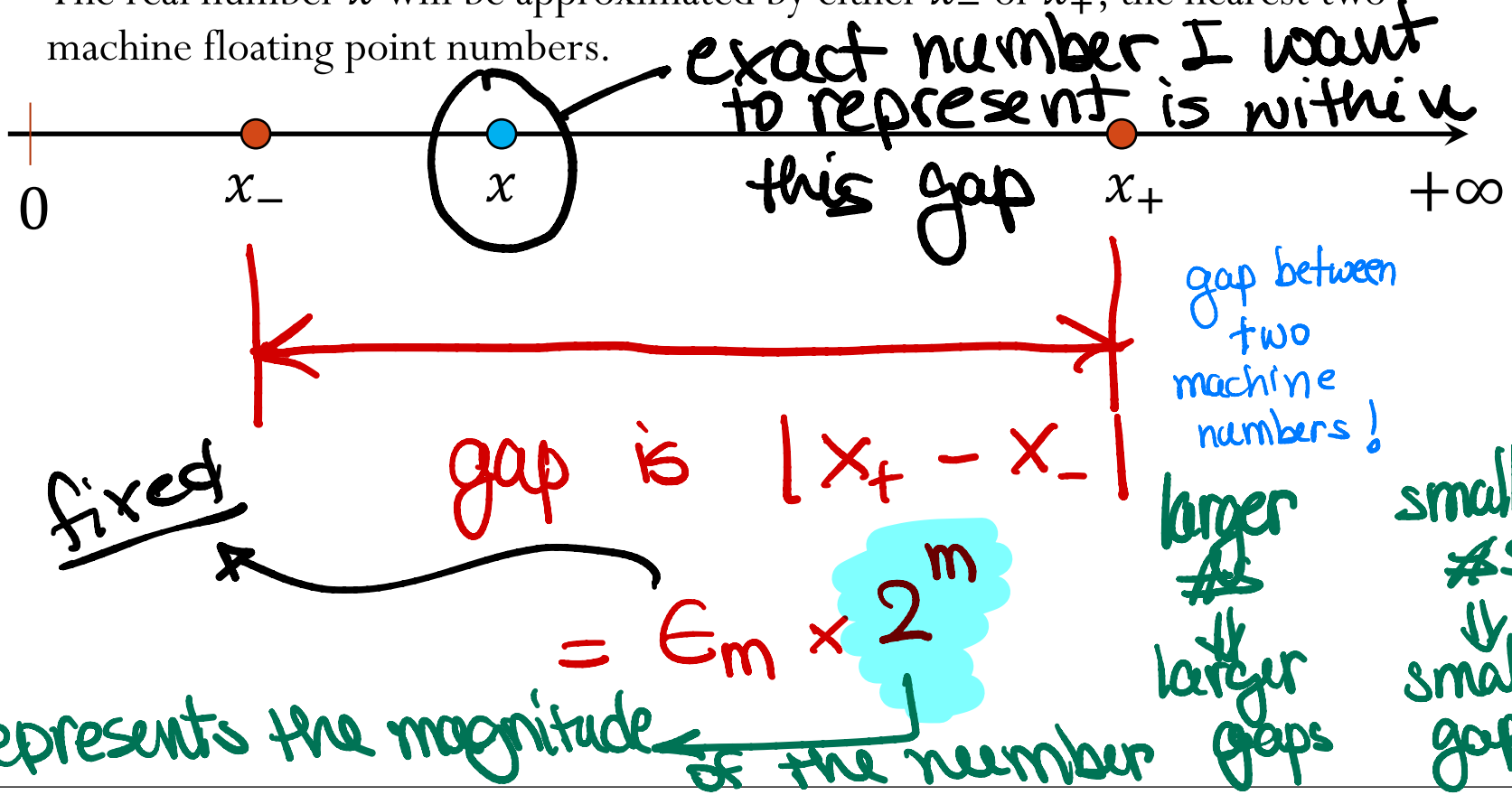
$x_+ = x_- + \epsilon_m \times 2^m$

Machine floating point number

- Not all real numbers can be exactly represented as a machine floating-point number.
- Consider a real number in the normalized floating-point form:

$$x = \pm 1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m$$

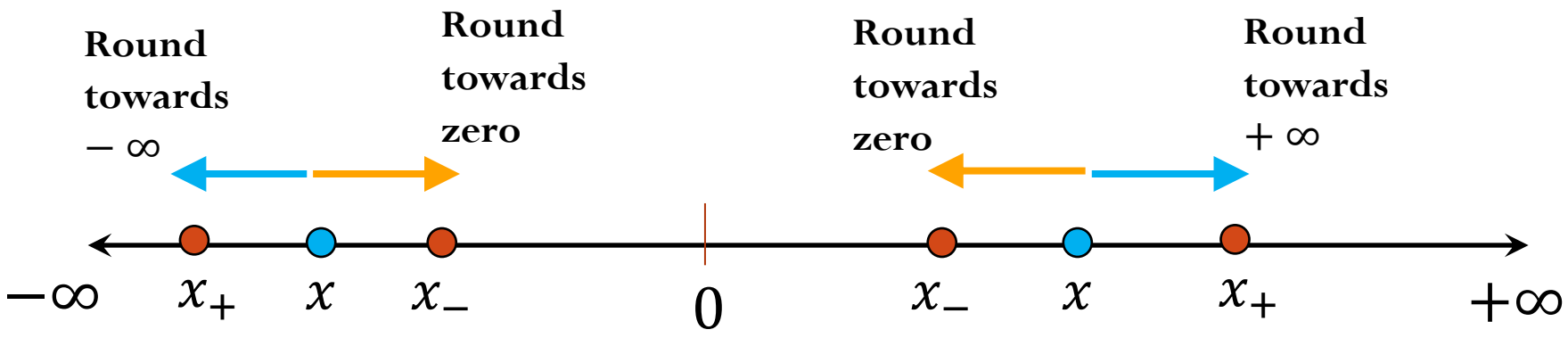
- The real number x will be approximated by either x_- or x_+, the nearest two machine floating point numbers.



Rounding

$x \longrightarrow fl(x)$

The process of replacing x by a nearby machine number is called rounding, and the error involved is called **roundoff error.** $= |fl(x) - x|$



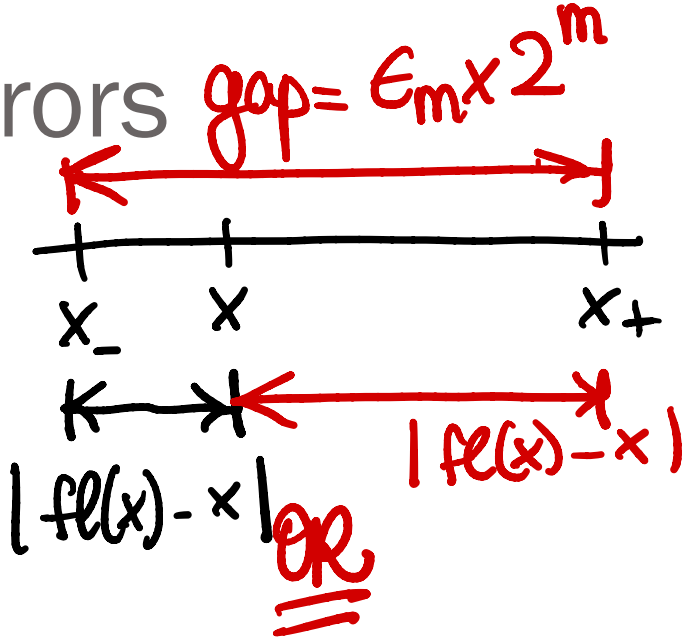
Round by chopping:

	x is positive number	x is negative number
Round up (ceil)	round	
Round down (floor)		

Round to nearest:

Rounding (roundoff) errors

Consider rounding by chopping:



- **Absolute error:**

$$|fl(x) - x| \leq \epsilon_m \times 2^m$$

- **Relative error:**

$$\frac{|fl(x) - x|}{|x|} \leq \frac{\epsilon_m \times 2^m}{|1.b_1 b_2 \cdots b_n \times 2^m|} = \underbrace{\frac{\epsilon_m}{1.b_1 b_2 \cdots b_n}}_{[1,2)}$$

$$\frac{|fl(x) - x|}{|x|} \leq \epsilon_m$$

Relative error due to rounding is always smaller than machine epsilon

Rounding (roundoff) errors

$$\frac{|fl(x) - x|}{|x|} \leq 2^{-23} \approx 1.2 \times 10^{-7} \leq 5 \times 10^{-7}$$

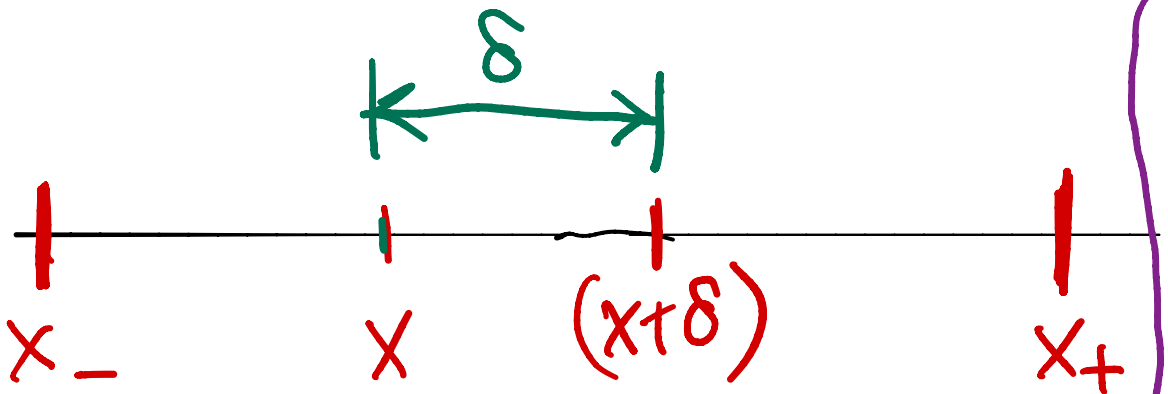
$e_r \leq 5 \times 10^{-7} \longrightarrow$ $fl(x)$ has 7 decimal accurate digits

$$\frac{|fl(x) - x|}{|x|} \leq 2^{-52} \approx 2.2 \times 10^{-16} \leq 5 \times 10^{-16}$$

$e_r \leq 5 \times 10^{-16} \longrightarrow$ $fl(x)$ has 16 decimal accurate digits

Video 2: Gap between machine numbers

Gap between two machine numbers



δ = gap

$\delta = \epsilon_m \times 2^m$

$x + \delta$

$= q \times 2^m + \epsilon_m \times 2^m$

$= (q + \epsilon_m) \times 2^m$

$q + \epsilon_m \neq q$

$fl(x) = \hat{x} = \begin{cases} x_+ \\ x_- \end{cases}$

$fl(x+\delta) = \hat{x} = fl(x)$

What is the smallest δ such that

$fl(x+\delta) = fl(x) \Rightarrow \delta <$ gap !

In practice (Rule of Thumb)

~~Show Ipython notebook demos~~

Binary base $x = q \times 2^m$

$fl(x+\delta) = fl(x)$

$\delta < \epsilon_m 2^m$

Example

$x = 2^8$

$\delta < 2^{-23} 2^8 = 2^{-15}$ $\boxed{\delta < 2^{-15}}$

if $\delta < 2^{-15} \Rightarrow fl(x+\delta) = fl(x)$

otherwise $fl(x+\delta) \neq fl(x)$

Decimal base

$x = q \times 10^m$

Example $x = 4.5 \times 10^4$

Double Precision

$\delta < 10^{-16} \times 10^9$

$\boxed{\delta < 10^{-12}}$

Video 2: Arithmetic with machine numbers

Mathematical properties of FP operations

Not necessarily associative:

For some x , y, z the result below is possible:

$$(x + y) + z \neq x + (y + z)$$

```
In [5]: (np.pi+1e100)-1e100
Out[5]: 0.0

In [6]: (np.pi)+(1e100-1e100)
Out[6]: 3.141592653589793

In [7]: b = 1e80
        a = 1e2
        print(a + (b - b) )
        print((a + b) - b )

100.0
0.0
```

Not necessarily distributive:

For some x , y, z the result below is possible:

$$z\,(x + y) \neq z\,x + z\,y$$

```
In [3]: print(100*(0.1 + 0.2))
        print(100*0.1 + 100*0.2)

30.000000000000004
30.0

In [4]: 100*(0.1 + 0.2) == 100*0.1 + 100*0.2
Out[4]: False
```

Not necessarily cumulative:

Repeatedly adding a very small number to a large number may do nothing

Floating point arithmetic (basic idea)

$$x = (-1)^{s}\ 1.f \times 2^{m}$$

- First compute the exact result
- Then round the result to make it fit into the desired precision

- $x + y = fl(x + y)$

- $x \times y = fl(x \times y)$

Floating point arithmetic

$n=3$
$p=4$

Consider a number system such that $x = \pm 1.b_1 b_2 b_3 \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

Rough algorithm for addition and subtraction:

1. Bring both numbers onto a common exponent
2. Do "grade-school" operation
3. Round result

- **Example 1: No rounding needed**

$$a = (1.101)_2 \times 2^1$$
$$b = (1.001)_2 \times 2^1$$

$$1.101 \times 2^1$$
$$1.001 \times 2^1$$
$$10.110 \times 2^1$$

$$c = a + b = (10.110)_2 \times 2^1 = (1.011)_2 \times 2^2$$

round

normalize

Floating point arithmetic

Consider a number system such that $x = \pm 1.\underbrace{b_1 b_2 b_3}_{n=3} \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

- **Example 2: Require rounding**

$$a = (1.101)_2 \times 2^0$$
$$b = (1.000)_2 \times 2^0$$

$$c = a + b = (10.101)_2 \times 2^0 \xrightarrow{\text{normalize}} 1.0101 \times 2^0$$

$fl(a+b) = 1.010 \times 2^0$

nearby floating-point number of machine number

- **Example 3:**

$$a = (1.100)_2 \times 2^1$$
$$b = (1.100)_2 \times 2^{-1}$$

$$c = a + b = (1.100)_2 \times 2^1 + (0.011)_2 \times 2^1 = (1.111)_2 \times 2^1$$

$fl(a+b) = 1.111 \times 2^1$ (no rounding required!) ✓

Floating point arithmetic

Consider a number system such that $x = \pm 1.b_1 b_2 b_3 b_4 \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

$n = 4$

$p = 5$

- **Example 4:**

$$a = (1.1011)_2 \times 2^1$$
$$b = (1.1010)_2 \times 2^1$$

$$c = a - b = (0.0001)_2 \times 2^1$$

$fl(a-b) = 1.\underline{\quad ? \quad} \times 2^{-3}$

$fl(a-b) = 1.0000 \times 2^{-3}$

machine choice

(NOT SIGNIFICANT DIGITS!)

Floating point arithmetic

Consider a number system such that $x = \pm 1.b_1 b_2 b_3 b_4 \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

- **Example 4:**

$$a = (1.1011)_2 \times 2^1$$
$$b = (1.1010)_2 \times 2^1$$

$$c = a - b = (0.0001)_2 \times 2^1$$

Or after normalization: $c = (1.????)_2 \times 2^{-3}$

p = 5
↓
p = 1

- There is not data to indicate what the missing digits should be.
- Machine fills them with its best guess, which is often not good (usually what is called spurious zeros).
- Number of significant digits in the result is reduced.
- This phenomenon is called **Catastrophic Cancellation**.

Loss of significance

Assume a and b are real numbers with $a \gg b$. For example

$$a = 1.a_1a_2a_3a_4a_5a_6 \dots a_n \dots \times 2^0$$
$$b = 1.b_1b_2b_3b_4b_5b_6 \dots b_n \dots \times 2^{-8}$$

In Single Precision (without loss of generality):

$$fl(a) = 1.a_1a_2a_3a_4a_5a_6 \dots a_{22}a_{23} \times 2^0$$
$$fl(b) = 1.b_1b_2b_3b_4b_5b_6 \dots b_{22}b_{23} \times 2^{-8}$$

} machine numbers in single precision

Compute $(a + b)$

$$1.a_1a_2a_3a_4a_5a_6a_7a_8a_9 \dots a_{22}a_{23} \times 2^0$$

$+$ $0.000000001\, b_1 \cdots b_{14}\, b_{15} \times 2^0$

$fl(a+b)$ includes only 15 sig. bit of b

Cancellation

→ when the digits / bits that are lost are the most significant ones.

Assume a and b are real numbers with $a \approx b$.

$$a = 1.a_1a_2a_3a_4a_5a_6 \ldots a_n \ldots \times 2^m$$
$$b = 1.b_1b_2b_3b_4b_5b_6 \ldots b_n \ldots \times 2^m$$

In single precision (without loss of generality), consider this example:

$$a = 1.a_1a_2a_3a_4a_5a_6 \ldots a_{20}a_{21}10a_{24}a_{25}a_{26}a_{27} \ldots \times 2^m$$
$$b = 1.a_1a_2a_3a_4a_5a_6 \ldots a_{20}a_{21}11b_{24}b_{25}b_{26}b_{27} \ldots \times 2^m$$

$$b - a = 0.0000 \ldots 0001 \times 2^m$$

$fl(b-a) = 1.\_\_\_\_\_ \times 2^{-23} \times 2^m$

$fl(b-a) = 1.000\ldots00 \times 2^{m-23}$ → those are not significant bits

Examples:

1) ***a* and *b* are real numbers with same order of magnitude** $(a \approx b)$. They have the following representation in a decimal floating point system with 16 decimal digits of accuracy:

$$fl(a) = 3004.45$$
$$fl(b) = 3004.46$$

→ 5 (out of the 16 initial accurate decimal digits) are lost in this computation

How many accurate digits does your answer have when you compute $b - a$?

3004.46
3004.45
0000.01

11 digits accurate

not significant digits !

only 11 accurate digits in the answer fl(b-a)

Loss of Significance

5 decimal digits accurate

How can we avoid this loss of significance? For example, consider the function $f(x) = \sqrt{x^2 + 1} - 1$

If we want to evaluate the function for values x near zero, there is a potential loss of significance in the subtraction.

$f(10^{-3}) = \sqrt{10^{-6} + 1} - 1$

$f(10^{-3}) = 1 - 1 = 0$

$$\begin{array}{r} 1.000000 \\ 0.000001 \\ \hline 1.000001 \end{array}$$

Rewrite the function to avoid subtractions of numbers with similar magnitude

Loss of Significance

$(a-b)(a+b) = a^2 - b^2$

Re-write the function as $f(x) = \dfrac{x^2}{\sqrt{x^2+1}-1}$ (no subtraction!)

$$f(x) = \left(\sqrt{x^2+1} - 1\right)\left(\frac{\sqrt{x^2+1}+1}{\sqrt{x^2+1}+1}\right) = \frac{x^2+1-1}{\sqrt{x^2+1}+1}$$

$$\boxed{f(x) = \frac{x^2}{\sqrt{x^2+1}+1}} \rightarrow \text{no subtraction!}$$

$$f(10^{-3}) = \frac{10^{-6}}{2}$$

Example:

exact real numbers

round-down

If $x = 0.3721448693$ and $y = 0.3720214371$ what is the relative error in the computation of $(x - y)$ in a computer with five decimal digits of accuracy?

$fl(x) = 0.37214$

$fl(y) = 0.37202$

error due to rounding

$$\left|\frac{fl(x) - x}{x}\right| = 1.308 \times 10^{-5}$$ ✓

$x - y = 0.0001234322$ → exact computation

$fl(x-y) = 0.00012$ → machine result

$$e_r = \left|\frac{(x-y) - fl(x-y)}{(x-y)}\right| \approx 3 \times 10^{-2}$$

→ this error is "large" compared to the roundoff error.

CANCELATION!