Rounding errors

Example

Show demo: "Waiting for 1".

Determine the double-precision machine representation for 0.1

$\# \times 2$	Integer part	Fractional part
0.2	0	0.2
0.4	0	0.4
0.8	0	0.8
1.6	1	0.6
1.2	1	0.2
0.4	0	0.4
0.8	0	0.8
1.6	1	0.6
1.2	1	0.2

$$0.1 = (0.000110011\,\overline{0011}\ldots)_2$$
$$= (1.100110011\ldots)_2 \times 2^{-4}$$

$$s = 0$$

$$f = 100110011\ldots00110011010$$

$$m = -4$$

$$c = m + 1023 = 1019 = (01111111011)_2$$

$$0\ 01111111011\ \underbrace{10011\ldots0011\ldots0011010}_{(52\text{-bit})}$$

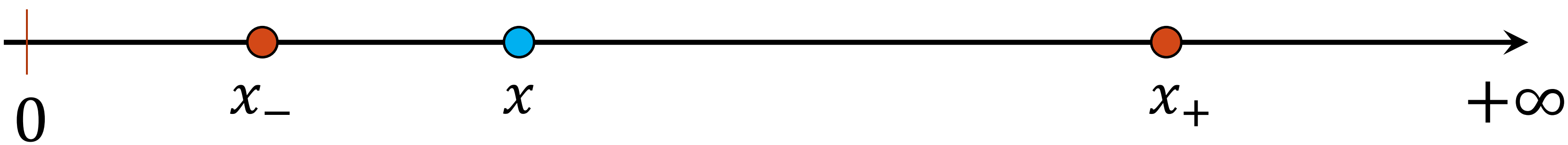
Roundoff error in its basic form!

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.

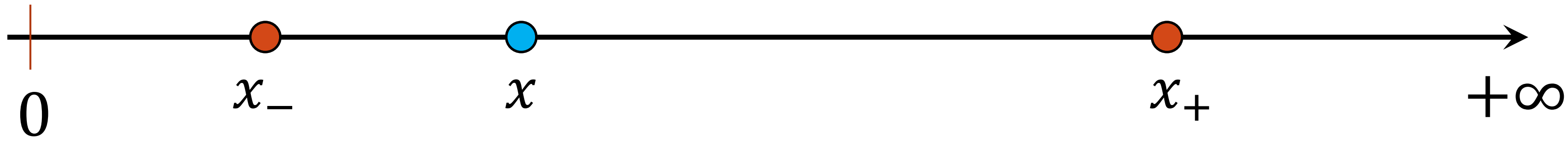


Without loss of generality, let's see what happens when trying to represent a positive machine floating point number:

Exact number: $x = 1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m$

$x_- = 1.b_1 b_2 b_3 \ldots b_n \times 2^m$ (rounding by chopping)

$$x_+ = 1.b_1 b_2 b_3 \ldots b_n \times 2^m + \underbrace{0.000 \ldots 01}_{\epsilon_m} \times 2^m$$



Exact number: $x = 1.b_1b_2b_3 \ldots b_n \ldots \times 2^m$

$$x_- = 1.b_1b_2b_3 \ldots b_n \times 2^m$$

$$x_+ = 1.b_1b_2b_3 \ldots b_n \times 2^m + \underbrace{0.000 \ldots 01}_{\epsilon_m} \times 2^m$$

Gap between x_+ and x_-: $|x_+ - x_-| = \epsilon_m \times 2^m$

Examples for single precision:

x_+ and x_- of the form $q \times 2^{-10}$: $|x_+ - x_-| = 2^{-33} \approx 10^{-10}$

x_+ and x_- of the form $q \times 2^{4}$: $|x_+ - x_-| = 2^{-19} \approx 2 \times 10^{-6}$

x_+ and x_- of the form $q \times 2^{20}$: $|x_+ - x_-| = 2^{-3} \approx 0.125$

x_+ and x_- of the form $q \times 2^{60}$: $|x_+ - x_-| = 2^{37} \approx 10^{11}$

The interval between successive floating point numbers is not uniform: the interval is smaller as the magnitude of the numbers themselves is smaller, and it is bigger as the numbers get bigger.

Gap between two successive machine floating point numbers

A "toy" number system can be represented as $x = \pm 1.b_1 b_2 \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

$(1.00)_2 \times 2^0 = 1$
$(1.01)_2 \times 2^0 = 1.25$
$(1.10)_2 \times 2^0 = 1.5$
$(1.11)_2 \times 2^0 = 1.75$

$(1.00)_2 \times 2^1 = 2$
$(1.01)_2 \times 2^1 = 2.5$
$(1.10)_2 \times 2^1 = 3.0$
$(1.11)_2 \times 2^1 = 3.5$

$(1.00)_2 \times 2^2 = 4.0$
$(1.01)_2 \times 2^2 = 5.0$
$(1.10)_2 \times 2^2 = 6.0$
$(1.11)_2 \times 2^2 = 7.0$

$(1.00)_2 \times 2^3 = 8.0$
$(1.01)_2 \times 2^3 = 10.0$
$(1.10)_2 \times 2^3 = 12.0$
$(1.11)_2 \times 2^3 = 14.0$

$(1.00)_2 \times 2^4 = 16.0$
$(1.01)_2 \times 2^4 = 20.0$
$(1.10)_2 \times 2^4 = 24.0$
$(1.11)_2 \times 2^4 = 28.0$

$(1.00)_2 \times 2^{-1} = 0.5$
$(1.01)_2 \times 2^{-1} = 0.625$
$(1.10)_2 \times 2^{-1} = 0.75$
$(1.11)_2 \times 2^{-1} = 0.875$

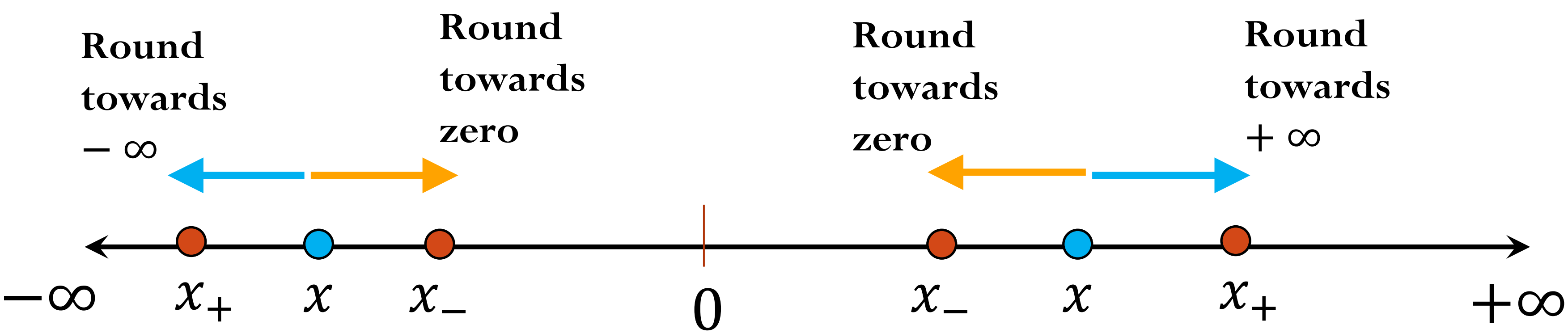
$(1.00)_2 \times 2^{-2} = 0.25$
$(1.01)_2 \times 2^{-2} = 0.3125$
$(1.10)_2 \times 2^{-2} = 0.375$
$(1.11)_2 \times 2^{-2} = 0.4375$

$(1.00)_2 \times 2^{-3} = 0.125$
$(1.01)_2 \times 2^{-3} = 0.15625$
$(1.10)_2 \times 2^{-3} = 0.1875$
$(1.11)_2 \times 2^{-3} = 0.21875$

$(1.00)_2 \times 2^{-4} = 0.0625$
$(1.01)_2 \times 2^{-4} = 0.078125$
$(1.10)_2 \times 2^{-4} = 0.09375$
$(1.11)_2 \times 2^{-4} = 0.109375$

Rounding

The process of replacing x by a nearby machine number is called rounding, and the error involved is called **roundoff error.**



Round by chopping: $fl(x) = x_-$

	x is positive number	x is negative number
Round up (ceil)	$fl(x) = x_+$ Rounding towards $+\infty$	$fl(x) = x_-$ Rounding towards zero
Round down (floor)	$fl(x) = x_-$ Rounding towards zero	$fl(x) = x_+$ Rounding towards $-\infty$

Round to nearest: either round up or round down, whichever is closer

Rounding (roundoff) errors

Consider rounding by chopping:

- **Absolute error:**

$$|\mathrm{fl}(x) - x| \leq |x_+ - x_-| = \epsilon_m \times 2^m$$

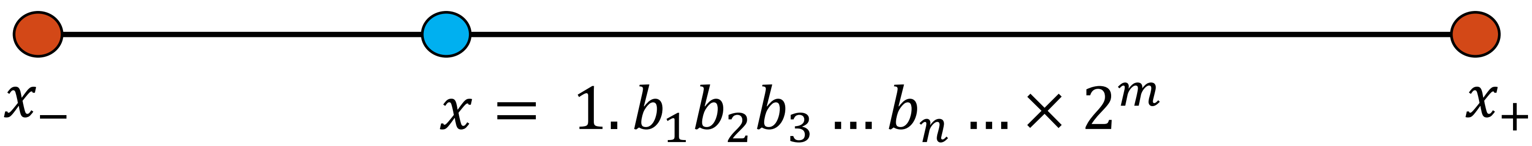
$$|\mathrm{fl}(x) - x| \leq \epsilon_m \times 2^m$$

- **Relative error:**

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leq \frac{\epsilon_m \times 2^m}{1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m}$$

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

Rounding (roundoff) errors



$$\frac{|\tilde{x} - x|}{|x|} \leq 2^{-23} \approx 1.2 \times 10^{-7}$$

Single precision: Floating-point math consistently introduces relative errors of about 10^{-7}. Hence, single precision gives you about 7 (decimal) accurate digits.

$$\frac{|\tilde{x} - x|}{|x|} \leq 2^{-52} \approx 2.2 \times 10^{-16}$$

Double precision: Floating-point math consistently introduces relative errors of about 10^{-16}. Hence, double precision gives you about 16 (decimal) accurate digits.

Iclicker question

Assume you are working with IEEE single-precision numbers. Find the smallest number a that satisfies

$$2^8 + a \neq 2^8$$

A) 2^{-1074}
B) 2^{-1022}
C) 2^{-52}
D) 2^{-15}
E) 2^{-8}

Demo

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)

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

s	c	f

- 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

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**

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

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

Floating point arithmetic

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

- **Example 2: Require rounding**

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

$$c = a + b = (10.101)_2 \times 2^0 \approx (1.010)_2 \times 2^1$$

- **Example 3:**

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

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

Floating point arithmetic

Consider a number system such that $x = \pm 1.b_1b_2b_3b_4 \times 2^m$ 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: $c = (1.????)_2 \times 2^{-3}$

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

Loss of significance

Assume $a \gg b$. For example

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

In Single Precision (without loss of generality):

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

$$1.a_1a_2a_3a_4a_5a_6a_7a_8a_9 \ldots a_{22}a_{23} \times 2^0$$
$$+\ 0.0000001b_1b_2b_3b_4b_5 \ldots b_{14}b_{15} \times 2^0$$

In this example, the result $fl(a+b)$ includes 15 bits of precision from $fl(b)$. Lost precision!

Cancellation

$$a = 1.a_1a_2a_3a_4a_5a_6 \dots a_n \dots \times 2^{m1}$$
$$b = 1.b_1b_2b_3b_4b_5b_6 \dots b_n \dots \times 2^{m2}$$

Suppose $a \approx b$ and single precision (without loss of generality)

$$a = 1.a_1a_2a_3a_4a_5a_6 \dots a_{20}a_{21}10\boxed{a_{24}a_{25}a_{26}a_{27}} \dots \times 2^m$$

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

Lost due to rounding

$$fl(b-a) = 0.0000 \dots 0001 \times 2^m = 1.??????? \dots ?? \times 2^{-n+m}$$

$$fl(b-a) = 1.\underbrace{000 \dots 00}\times 2^{-n+m}$$

Not significant bits (precision lost, not due to $fl(b-a)$ but due to rounding of a, b from the beginning

Example of cancellation:

Suppose $a = 1.1011 a_5 a_6 a_7 \dots \times 2^1$

$b = 1.1010 b_5 b_6 b_7 \dots \times 2^1$

using machine where $n = 4$ $\Rightarrow$ $a = 1.1011 \times 2^1$
$b = 1.1010 \times 2^1$

$a - b \Rightarrow$

$$\begin{array}{r} 1.1011\ a_5\ a_6\ a_7 \dots \times 2^1 \\ \ominus\ 1.1010\ b_5\ b_6\ b_7 \dots \times 2^1 \\ \hline 0.0001 \times 2^1 \end{array}$$

machine resulting with cancellation

1.0000×2^{-3}

not significant digits

when done by "hand"

$1.c_1 c_2 c_3 c_4 \times 2^{-3}$

significant digits from $a_5 a_6 a_7 a_8$
$\ominus\ b_5 b_6 b_7 b_8$

Loss of Significance

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.

For example, if $x = 10^{-3}$ and we use five-decimal-digit arithmetic

$$f(10^{-3}) = \sqrt{(10^{-3})^2 + 1} - 1 = 0$$

How can we fix this issue?

Loss of Significance

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

Evaluate now the function for $x = 10^{-3}$ using five-decimal-digit arithmetic

$$f(10^{-3}) = \frac{(10^{-3})^2}{\sqrt{(10^{-3})^2+1}+1} = \frac{10^{-6}}{2}$$

Example:

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?

Using five decimal digits of accuracy, the numbers are rounded as:

$$fl(x) = 0.37214 \text{ and } fl(y) = 0.37202$$

Then the subtraction is computed:

$$fl(x) - fl(y) = 0.37214 - 0.37202 = 0.00012$$

The result of the operation is: $fl(x - y) = 1.20000 \times 10^{-2}$ (the last digits are filled with spurious zeros)

The relative error between the exact and computer solutions is given by

$$\frac{|(x - y) - fl(x - y)|}{|(x - y)|} = \frac{0.0001234322 - 0.00012}{0.000123432} = \frac{0.0000034322}{0.000123432} \approx 3 \times 10^{-2}$$

Note that the magnitude of the error due to the subtraction is large when compared with the relative error due to the rounding

$$\frac{|x - fl(x)|}{|x|} \approx 1.3 \times 10^{-5}$$