Rounding errors
Example

Show demo: “Waiting for 1”.

Determine the double-precision machine representation for 0.1

\[ 0.1 = (0.0001100111011 \ldots)_2 = (1.1001100111011 \ldots)_2 \times 2^{-4} \]

- **s** = 0
- **f** = 1001100111011 \ldots 00110011010
- **m** = -4
- **c** = m + 1023 = 1019 = (01111111011)_2

\[ 0 \ 01111111011 \ 10011 \ldots \ 0011 \ldots \ 0011010 \]

(52-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)

\[ x_+ = 1.\, b_1\, b_2\, b_3 \ldots b_n \times 2^m + 0.000 \ldots 01 \times 2^m \]
Exact number: \( x = 1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m \)

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

\( x_+ = 1.b_1 b_2 b_3 \ldots b_n \times 2^m + 0.000 \ldots 01 \times 2^m \)

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 $b_i \in \{0,1\}$.

<table>
<thead>
<tr>
<th>Number</th>
<th>Value</th>
<th>Number</th>
<th>Value</th>
<th>Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1.00)_2 \times 2^0$</td>
<td>1</td>
<td>$(1.00)_2 \times 2^1$</td>
<td>2</td>
<td>$(1.00)_2 \times 2^2$</td>
<td>4.0</td>
</tr>
<tr>
<td>$(1.01)_2 \times 2^0$</td>
<td>1.25</td>
<td>$(1.01)_2 \times 2^1$</td>
<td>2.5</td>
<td>$(1.01)_2 \times 2^2$</td>
<td>5.0</td>
</tr>
<tr>
<td>$(1.10)_2 \times 2^0$</td>
<td>1.5</td>
<td>$(1.10)_2 \times 2^1$</td>
<td>3.0</td>
<td>$(1.10)_2 \times 2^2$</td>
<td>6.0</td>
</tr>
<tr>
<td>$(1.11)_2 \times 2^0$</td>
<td>1.75</td>
<td>$(1.11)_2 \times 2^1$</td>
<td>3.5</td>
<td>$(1.11)_2 \times 2^2$</td>
<td>7.0</td>
</tr>
<tr>
<td>$(1.00)_2 \times 2^3$</td>
<td>8.0</td>
<td>$(1.00)_2 \times 2^4$</td>
<td>16.0</td>
<td>$(1.00)_2 \times 2^{-1}$</td>
<td>0.5</td>
</tr>
<tr>
<td>$(1.01)_2 \times 2^3$</td>
<td>10.0</td>
<td>$(1.01)_2 \times 2^4$</td>
<td>20.0</td>
<td>$(1.01)_2 \times 2^{-1}$</td>
<td>0.625</td>
</tr>
<tr>
<td>$(1.10)_2 \times 2^3$</td>
<td>12.0</td>
<td>$(1.10)_2 \times 2^4$</td>
<td>24.0</td>
<td>$(1.10)_2 \times 2^{-1}$</td>
<td>0.75</td>
</tr>
<tr>
<td>$(1.11)_2 \times 2^3$</td>
<td>14.0</td>
<td>$(1.11)_2 \times 2^4$</td>
<td>28.0</td>
<td>$(1.11)_2 \times 2^{-1}$</td>
<td>0.875</td>
</tr>
<tr>
<td>$(1.00)_2 \times 2^{-2}$</td>
<td>0.25</td>
<td>$(1.00)_2 \times 2^{-3}$</td>
<td>0.125</td>
<td>$(1.00)_2 \times 2^{-4}$</td>
<td>0.0625</td>
</tr>
<tr>
<td>$(1.01)_2 \times 2^{-2}$</td>
<td>0.3125</td>
<td>$(1.01)_2 \times 2^{-3}$</td>
<td>0.15625</td>
<td>$(1.01)_2 \times 2^{-4}$</td>
<td>0.078125</td>
</tr>
<tr>
<td>$(1.10)_2 \times 2^{-2}$</td>
<td>0.375</td>
<td>$(1.10)_2 \times 2^{-3}$</td>
<td>0.1875</td>
<td>$(1.10)_2 \times 2^{-4}$</td>
<td>0.09375</td>
</tr>
<tr>
<td>$(1.11)_2 \times 2^{-2}$</td>
<td>0.4375</td>
<td>$(1.11)_2 \times 2^{-3}$</td>
<td>0.21875</td>
<td>$(1.11)_2 \times 2^{-4}$</td>
<td>0.109375</td>
</tr>
</tbody>
</table>
Rounding

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

Round towards $-\infty$  
Round towards zero  
Round towards $+\infty$

Round by chopping: $\lfloor x \rfloor = x_-$

<table>
<thead>
<tr>
<th></th>
<th>$x$ is positive number</th>
<th>$x$ is negative number</th>
</tr>
</thead>
</table>
| Round up (ceil) | $\lfloor x \rfloor = x_+$  
Rounding towards $+\infty$ | $\lfloor x \rfloor = x_-$  
Rounding towards zero |
| Round down (floor) | $\lfloor x \rfloor = x_-$  
Rounding towards zero | $\lfloor x \rfloor = x_+$  
Rounding towards $-\infty$ |

Round to nearest: either round up or round down, whichever is closer
Rounding (roundoff) errors

Consider rounding by chopping:

• Absolute error:

\[ |\text{fl}(x) - x| \leq |x_+ - x_-| = \epsilon_m \times 2^m \]

\[ |\text{fl}(x) - x| \leq \epsilon_m \times 2^m \]

• Relative error:

\[ \frac{|\text{fl}(x) - x|}{|x|} \leq \frac{\epsilon_m \times 2^m}{1 \cdot b_1 b_2 b_3 \ldots b_n \ldots \times 2^m} \]

\[ \frac{|\text{fl}(x) - x|}{|x|} \leq \epsilon_m \]
Rounding (roundoff) errors

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

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

\[ x = (-1)^s \ 1 \cdot f \times 2^m = [s \ c \ f] \]

- 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

\[ a = (1.101)_2 \times 2^1 \]
\[ b = (1.001)_2 \times 2^1 \]
\[ c = a + b = (10.110)_2 \times 2^1 = (1.011)_2 \times 2^2 \]
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:**
  \[
  a = (1.100)_2 \times 2^1 \\
  b = (1.100)_2 \times 2^{-1} \\
  c = a + b = (1.100)_2 \times 2^1 + (0.011)_2 \times 2^1 = (1.111)_2 \times 2^1 
  \]
Not necessarily associative:
For some $x, y, z$ the result below is possible:

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

Not necessarily distributive:
For some $x, y, z$ the result below is possible:

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

Not necessarily cumulative:
Repeatedly adding a very small number to a large number may do nothing

Demo: FP-arithmetic
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:**

\[
\begin{align*}
    a &= (1.1011)_2 \times 2^1 \\
    b &= (1.1010)_2 \times 2^1 \\
    c &= a - b = (0.0001)_2 \times 2^1
\end{align*}
\]

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**.
Cancellation

\[
a = 1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_n \ldots \times 2^{m_1}
\]
\[
b = 1. b_1 b_2 b_3 b_4 b_5 b_6 \ldots b_n \ldots \times 2^{m_2}
\]

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

\[
a = 1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_{20} a_{21} 10 a_{24} a_{25} a_{26} a_{27} \ldots \times 2^m
\]
\[
b = 1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_{20} a_{21} 11 b_{24} b_{25} b_{26} b_{27} \ldots \times 2^m
\]

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

\[
fl(b - a) = 1.000 \ldots 00 \times 2^{-n+m}
\]

Lost due to rounding

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 \( a = 1.1011a_5a_6a_7... \times 2^4 \)
\( b = 1.1010b_5b_6b_7... \times 2^4 \)

Using machine where \( n = 4 \) \( \Rightarrow a = 1.1011 \times 2^4 \)
\( b = 1.1010 \times 2^4 \)

\( a - b \Rightarrow 1.1011a_5a_6a_7... \times 2^4 \)
\( \underline{1.1010} b_5b_6b_7... \times 2^4 \)

\( 0.0001 \times 2^4 \)

Machine resulting with cancellation

1.0000 \( \times 2^{-3} \)

When done by "hand"

1. \( C_1C_2C_3C_4 \times 2^{-3} \)  

Significant digits from \( a_5a_6a_7a_8 \)

\( \underline{\oplus} b_5b_6b_7b_8 \)

Not significant digits
Cancellation

\[ a = 1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_n \ldots \times 2^{m_1} \]
\[ b = 1. b_1 b_2 b_3 b_4 b_5 b_6 \ldots b_n \ldots \times 2^{m_2} \]

For example, assume single precision and \( m_1 = m_2 + 18 \) (without loss of generality), i.e. \( a \gg b \)

\[
fl(a) = 1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_{22} a_{23} \times 2^{m+18}
\]
\[
fl(b) = 1. b_1 b_2 b_3 b_4 b_5 b_6 \ldots b_{22} b_{23} \times 2^m
\]

\[
1. a_1 a_2 a_3 a_4 a_5 a_6 \ldots a_{22} a_{23} \times 2^{m+18} + 0.0000 \ldots 001 b_1 b_2 b_3 b_4 b_5 \times 2^{m+18}
\]

In this example, the result \( fl(a + b) \) only included 6 bits of precision from \( fl(b) \). Lost precision!
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 \( f(x) = \frac{x^2}{\sqrt{x^2 + 1} - 1} \) (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:
\[
\text{fl}(x) = 0.37214 \quad \text{and} \quad \text{fl}(y) = 0.37202
\]

Then the subtraction is computed:
\[
\text{fl}(x) - \text{fl}(y) = 0.37214 - 0.37202 = 0.00012
\]

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

The relative error between the exact and computer solutions is given by
\[
\frac{|(x - y) - \text{fl}(x - y)|}{|x - y|} = \frac{0.0001234322 - 0.00012}{0.000123432} = \frac{0.0000034322}{0.000123432} \approx 3 \times 10^{-2}
\]

Note that the magnitude of the error due to the subtraction is large when compared with the relative error due to the rounding
\[
\frac{|x - \text{fl}(x)|}{|x|} \approx 1.3 \times 10^{-5}
\]