I: Floating-point numbers and representations

1. Floating-point representation of numbers (scientific notation) has four components: sign, significand (mantissa), base and exponent

For example

\[-3.1416 \times 10^1\]

\[\uparrow \quad \uparrow \quad \uparrow \]

\[\text{sign} \quad \text{significand} \quad \text{base} \]

2. Computers use binary representation of numbers. The floating-point representation of a nonzero binary number \(x\) is of the form

\[x = \pm b_0.b_1b_2\cdots b_{p−1} \times 2^{E}.\]

(a) It is normalized, i.e., \(b_0 = 1\) (the hidden bit)
(b) Precision \((= p)\) is the number of bits in the significand (mantissa) (including the hidden bit).
(c) Machine epsilon \(\epsilon = 2^{−(p−1)}\), the gap between the number 1 and the smallest floating-point number that is greater than 1.
(d) The unit in the last place, \(ulp(x) = 2^{−(p−1)} \times 2^E = \epsilon \times 2^E\).
   • If \(x > 0\), then \(ulp(x)\) is the gap between \(x\) and the next larger floating-point number.
   • If \(x < 0\), then \(ulp(x)\) is the gap between \(x\) and the smaller floating-point number (larger in absolute value).

3. Special numbers: 0, −0, ∞, −∞, NaN(=“Not a Number”).

4. All computers designed since 1985 use the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) represent each number as a binary number and use binary arithmetic. Essentials of the IEEE standard:
   • consistent representation of floating-point numbers by all machines adopting the standard;
   • correctly rounded floating-point operations, using various rounding modes;
   • consistent treatment of exceptional situation such as division by zero.

5. IEEE single format takes 32 bits (=4 bytes) long:

<table>
<thead>
<tr>
<th>(s)</th>
<th>(E)</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign</td>
<td>exponent</td>
<td>binary point</td>
</tr>
</tbody>
</table>

- It represents the number

\[(-1)^s \cdot (1.f) \times 2^{E-127}\]
• The leading 1 in the fraction need not be stored explicitly, because it is always 1. This hidden bit accounts for the “1.” here.
• The “$E - 127$” in the exponent is to avoid the need for storage of a sign bit. $E$ is a normalized number, and $E_{\text{min}} = (00000001)_2 = (1)_{10}$, $E_{\text{max}} = (11111110)_2 = (254)_{10}$.
• The range of positive normalized numbers is from
$$N_{\text{min}} = 1.00 \cdots 0 \times 2^{E_{\text{min}} - 127} = 2^{-126} \approx 1.2 \times 10^{-38}$$

\[\text{to} \]
$$N_{\text{max}} = 1.11 \cdots 1 \times 2^{E_{\text{max}} - 127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}.$$ • Special representations for 0, $\pm \infty$ and NaN:

zero  =  ± 00000000 00000000000000000000000
±∞    =  ± 11111111 00000000000000000000000
NaN   =  ± 11111111 otherwise

6. IEEE **double** format takes 64 bits (= 8 bytes) long:

\[\begin{array}{|c|c|c|c|}
\hline
s & E & f & \\
\hline
\end{array}\]

• It represents the number
$$(-1)^s \cdot (1.f) \times 2^{E - 1023}$$

• The range of positive normalized numbers is from
$$N_{\text{min}} = 2^{-1022} \approx 2.2 \times 10^{-308}$$

\[\text{to} \]
$$N_{\text{max}} = 1.11 \cdots 1 \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}.$$ • Special representations for 0, $\pm \infty$ and NaN.

7. IEEE **extended** format, with at least 15 bits available for the exponent and at least 63 bits for the fractional part of the significant. (Pentium has 80-bit extended format)

8. Precision and machine epsilon of the IEEE formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Precision $p$</th>
<th>Machine epsilon $\epsilon = 2^{-p-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>24</td>
<td>$\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$</td>
</tr>
<tr>
<td>double</td>
<td>53</td>
<td>$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$</td>
</tr>
<tr>
<td>extended</td>
<td>64</td>
<td>$\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$</td>
</tr>
</tbody>
</table>
9. Rounding

Let a positive real number $x$ be in the normalized range, i.e., $N_{\text{min}} \leq x \leq N_{\text{max}}$, and be written in the normalized form

$$x = (1.b_1b_2\cdots b_{p-1}b_p b_{p+1}\ldots) \times 2^E,$$

Then the closest floating-point number less than or equal to $x$ is

$$x_- = 1.b_1b_2\cdots b_{p-1} \times 2^E,$$

i.e., $x_-$ is obtained by truncating. The next floating-point number bigger than $x_-$ is

$$x_+ = ((1.b_1b_2\cdots b_{p-1}) + (0.00\cdots 01)) \times 2^E,$$

therefore, also the next one that bigger than $x$.

If $x$ is negative, the situation is reversed.

Correctly rounding modes:

- **round down**: $\text{round}(x) = x_-;$
- **round up**: $\text{round}(x) = x_+;$
- **round towards zero**: $\text{round}(x) = x_-$ of $x \geq 0$; $\text{round}(x) = x_+$ of $x \leq 0$;
- **round to nearest**: $\text{round}(x) = x_-$ or $x_+$, whichever is nearer to $x$, except that if $x > N_{\text{max}}$, $\text{round}(x) = \infty$, and if $x < -N_{\text{max}}$, $\text{round}(x) = -\infty$. In the case of tie, i.e., $x_-$ and $x_+$ are the same distance from $x$, the one with its least significant bit equal to zero is chosen.

When the **round to nearest** (IEEE default rounding mode) is in effect,

$$\text{abserr}(x) = |\text{round}(x) - x| \leq \frac{1}{2} \text{ulp}(x)$$

and

$$\text{relerr}(x) = \frac{|\text{round}(x) - x|}{|x|} \leq \frac{1}{2} \epsilon.$$

Therefore, we have

\[
\text{the max. rel. representation error} = \begin{cases} 
\frac{1}{2} \cdot 2^{1-24} = 2^{-24} \approx 5.96 \cdot 10^{-8} \\
\frac{1}{2} \cdot 2^{-52} \approx 1.11 \times 10^{-16}. 
\end{cases}
\]

II: Floating point arithmetic

1. IEEE rules for correctly rounded floating-point operations:

   if $x$ and $y$ are correctly rounded floating-point numbers, then

   $$\text{fl}(x + y) = \text{round}(x + y) = (x + y)(1 + \delta)$$
   $$\text{fl}(x - y) = \text{round}(x - y) = (x - y)(1 + \delta)$$
   $$\text{fl}(x \times y) = \text{round}(x \times y) = (x \times y)(1 + \delta)$$
   $$\text{fl}(x/y) = \text{round}(x/y) = (x/y)(1 + \delta)$$

   where $|\delta| \leq \frac{1}{2} \epsilon$ for the round to nearest.

   IEEE standard also requires that correctly rounded remainder and square root operations be provided.
### 2. IEEE standard response to exceptions

<table>
<thead>
<tr>
<th>Event</th>
<th>Example</th>
<th>Set result to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid operation</td>
<td>0/0, 0 × ∞</td>
<td>NaN</td>
</tr>
<tr>
<td>Division by zero</td>
<td>Finite nonzero/0</td>
<td>±∞ or ±N(_{\text{max}})</td>
</tr>
<tr>
<td>Overflow</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td>underflow</td>
<td>(x \neq 0,</td>
<td>x</td>
</tr>
<tr>
<td>Inexact</td>
<td>whenever fl((x \circ y)) \≠ x \circ y</td>
<td>correctly rounded value</td>
</tr>
</tbody>
</table>

### 3. Let \(\hat{x}\) and \(\hat{y}\) be the floating-point numbers and that

\[
\hat{x} = x(1 + \tau_1) \quad \text{and} \quad \hat{y} = y(1 + \tau_2), \quad \text{for } |\tau_i| \leq \tau \ll 1
\]

where \(\tau_i\) could be the relative errors in the process of “collecting/getting” the data from the original source or the previous operations.

Question: how do the four basic arithmetic operations behave?

### 4. Addition and subtraction

\[
\begin{align*}
\text{fl}(\hat{x} + \hat{y}) &= (\hat{x} + \hat{y})(1 + \hat{\delta}), & |\hat{\delta}| &\leq \frac{1}{2} \epsilon \\
&= x(1 + \tau_1)(1 + \delta) + y(1 + \tau_2)(1 + \delta) \\
&= x + y + x(\tau_1 + \delta + O(\tau \epsilon)) + y(\tau_2 + \delta + O(\tau \epsilon)) \\
&= (x + y) \left(1 + \frac{x}{x+y}(\tau_1 + \delta + O(\tau \epsilon)) + \frac{y}{x+y}(\tau_2 + \delta + O(\tau \epsilon))\right) \\
&= (x + y)(1 + \hat{\delta}),
\end{align*}
\]

where \(\hat{\delta}\) can be bounded as follows:

\[
|\hat{\delta}| \leq \frac{|x| + |y|}{|x + y|} \left(\tau + \frac{1}{2} \epsilon + O(\tau \epsilon)\right).
\]

Three possible cases:

(a) If \(x\) and \(y\) have the same sign, i.e., \(xy > 0\), then \(|x + y| = |x| + |y|\); this implies

\[
|\hat{\delta}| \leq \tau + \frac{1}{2} \epsilon + O(\tau \epsilon) \ll 1.
\]

Thus \(\text{fl}(\hat{x} + \hat{y})\) approximates \(x + y\) well.

(b) If \(x \approx -y \Rightarrow |x + y| \approx 0\), then \((|x| + |y|)/|x + y| \gg 1\); this implies that \(|\hat{\delta}|\) could be nearly or much bigger than 1. Thus \(\text{fl}(\hat{x} + \hat{y})\) may turn out to have nothing to do with the true \(x + y\). This is so called catastrophic cancellation which happens when a floating-point number is subtracted from another nearly equal floating-point number. Cancellation causes relative errors or uncertainties already presented in \(\hat{x}\) and \(\hat{y}\) to be magnified.

(c) In general, if \((|x| + |y|)/|x + y|\) is not too big, \(\text{fl}(\hat{x} + \hat{y})\) provides a good approximation to \(x + y\).

### 5. Examples of catastrophic cancellation

**Example 1.** Computing \(\sqrt{n + 1} - \sqrt{n}\) straightforward causes substantial loss of significant digits for large \(n\).
Catastrophic cancellation can sometimes be avoided if a formula is properly reformulated. In the present case, one can compute \( \sqrt{n + 1} - \sqrt{n} \) almost to full precision by using the equality

\[
\sqrt{n + 1} - \sqrt{n} = \frac{1}{\sqrt{n + 1} + \sqrt{n}}.
\]

Consequently, the computed results are

\[
\begin{array}{c|c|c|c|c}
 n & \text{fl}(\sqrt{n + 1}) & \text{fl}(\sqrt{n}) & \text{fl}(\sqrt{n + 1} - \sqrt{n}) \\
\hline
1.00e+10 & 4.999941672116518e-06 & 4.9999999999999999e+05 & 4.9999999999999999e+05 \\
1.00e+11 & 1.5811529010534285e-06 & 1.0000000000000000e+06 & 1.0000000000000000e+06 \\
1.00e+12 & 5.0000380724668502e-07 & 1.0000000000000000e+07 & 1.0000000000000000e+07 \\
1.00e+13 & 1.5785917639732361e-07 & 3.1622776601683790e+07 & 3.1622776601683790e+07 \\
1.00e+14 & 5.0291419029235839e-08 & 1.0000000000000000e+08 & 1.0000000000000000e+08 \\
1.00e+15 & 1.8626451923095703e-08 & 3.1622776601683810e+08 & 3.1622776601683810e+08 \\
1.00e+16 & 0.0000000000000000e+00 & 1.0000000000000000e+09 & 1.0000000000000000e+09 \\
\end{array}
\]

In fact, one can show that \( \text{fl}(1/(\sqrt{n + 1} + \sqrt{n})) = (\sqrt{n + 1} - \sqrt{n})(1+\delta) \), where \( |\delta| \leq 5\epsilon + O(\epsilon^2) \) (try it!)

**EXAMPLE 2.** Consider the function

\[
f(x) = \frac{1 - \cos x}{x^2} = \frac{1}{2} \left( \frac{\sin(x/2)}{x/2} \right)^2.
\]

Note that

\[
0 \leq f(x) < 1/2 \quad \text{for all } x \neq 0.
\]

Compare the computed values for \( x = 1.2 \times 10^{-5} \) using the above two expressions (assume that the value of \( \cos x \) rounded to 10 significant figures).

6. Multiplication and Division:

\[
\text{fl}(\hat{x} \times \hat{y}) = (\hat{x} \times \hat{y})(1 + \delta) = x y (1 + \tau_1)(1 + \tau_2)(1 + \delta) \equiv x y (1 + \hat{\delta}_x),
\]

\[
\text{fl}(\hat{x}/\hat{y}) = (\hat{x}/\hat{y})(1 + \delta) = (x/y)(1 + \tau_1)(1 + \tau_2)^{-1}(1 + \delta) \equiv x y (1 + \hat{\delta}_z),
\]

where

\[
\hat{\delta}_x = \tau_1 + \tau_2 + \delta + O(\tau \epsilon), \quad \hat{\delta}_z = \tau_1 - \tau_2 + \delta + O(\tau \epsilon).
\]

Thus

\[
|\hat{\delta}_x| \leq 2\tau + \frac{1}{2}\epsilon + O(\tau \epsilon),
\]

\[
|\hat{\delta}_z| \leq 2\tau + \frac{1}{2}\epsilon + O(\tau \epsilon).
\]

Therefore, multiplication and Division are very well-behaved!
III: Floating point error analysis

1. Forward and backward error analysis

We illustrate the basic idea through a simple example. Consider the computation of an inner product of two vector \( x, y \in \mathbb{R}^3 \)
\[
x^T y \overset{\text{def}}{=} x_1 y_1 + x_2 y_2 + x_3 y_3,
\]
assuming already \( x_i \)'s and \( y_j \)'s are floating-point numbers. It is likely that \( \text{fl}(x \cdot y) \) is computed in the following order.
\[
\text{fl}(x^T y) = \text{fl}\left( \text{fl}(x_1 y_1) + \text{fl}(x_2 y_2) + \text{fl}(x_3 y_3) \right).
\]
Adopting the floating-point arithmetic model, we have
\[
\text{fl}(x^T y) = x^T y + E,
\]
where \( E = x_1 y_1 (\epsilon_1 + \delta_1 + \delta_2) + x_2 y_2 (\epsilon_2 + \delta_1 + \delta_2) + x_3 y_3 (\epsilon_3 + \delta_2) + O(\epsilon^2) \). It implies that
\[
|E| \leq \frac{1}{2} \epsilon (3|x_1 y_1| + 3|x_2 y_2| + 2|x_3 y_3|) + O(\epsilon^2) \leq \frac{3}{2} \epsilon \cdot |x|^T |y| + O(\epsilon^2).
\]
This bound on \( E \) tells the worst case difference between the exact \( x^T y \) and its computed value. Such an error analysis is so-called Forward Error Analysis.

(b) We can also write
\[
\text{fl}(x^T y) = \hat{x}^T \hat{y} = (x + \Delta x)^T (y + \Delta y),
\]
where\(^1\)
\[
\begin{align*}
\hat{x}_1 &= x_1 (1 + \epsilon_1), & \hat{y}_1 &= y_1 (1 + \delta_1) (1 + \delta_2) \equiv y_1 (1 + \hat{\delta}_1), \\
\hat{x}_2 &= x_2 (1 + \epsilon_2), & \hat{y}_2 &= y_2 (1 + \delta_1) (1 + \delta_2) \equiv y_2 (1 + \hat{\delta}_2), \\
\hat{x}_3 &= x_3 (1 + \epsilon_3), & \hat{y}_3 &= y_3 (1 + \delta_2) \equiv y_3 (1 + \hat{\delta}_3).
\end{align*}
\]
It can be seen that \( |\hat{\delta}_1| = |\hat{\delta}_2| \leq \epsilon + O(\epsilon^2) \) and \( |\hat{\delta}_3| \leq \frac{1}{2} \epsilon \). This says the computed value \( \text{fl}(x^T y) \) is the exact inner product of a slightly perturbed \( \hat{x} \) and \( \hat{y} \). Such an error analysis is so-called Backward Error Analysis.

\(^1\)There are many ways to distribute factors \((1 + \epsilon_i)\) and \((1 + \delta_j)\) to \( x_i \) and \( y_j \). In this case it is even possible to make either \( \hat{x} \equiv x \) or \( \hat{y} \equiv y \).
V: Further reading

1. The following article based on lecture notes of Prof. W. Kahan of the University of California at Berkeley provides an excellent review of IEEE float point arithmetics.


2. The following book gives a broad overview of numerical computing, with special focus on the IEEE standard for binary floating-point arithmetic.


3. The following lecture by N. Higham presents the latest development on low precision and multiprecision arithmetic.


4. Websites for discussion of numerical disasters:

   • T. Huckle, Collection of software bugs
     [http://www5.in.tum.de/~huckle/bugse.html](http://www5.in.tum.de/~huckle/bugse.html)

   • K. Vuik, Some disasters caused by numerical errors
     [http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html](http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html)

   • D. Arnold, Some disasters attributable to bad numerical computing