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

   $$\underset{\text{sign}}{-} \quad \underset{\text{significand}}{3.1416} \times \underset{\text{base}}{10}^{\underset{\text{exponent}}{1}}$$

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

   $$x = \pm\, b_0.b_1b_2\cdots b_{p-1} \times 2^E. \tag{1}$$

   (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*, $\text{ulp}(x) = 2^{-(p-1)} \times 2^E = \epsilon \times 2^E$.

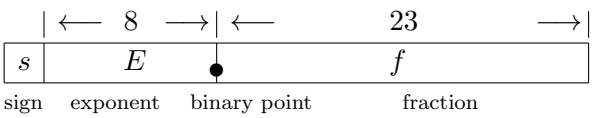
   - If $x > 0$, then $\text{ulp}(x)$ is the gap between x and the next larger floating-point number.
   - If $x < 0$, then $\text{ulp}(x)$ is the gap between x and the smaller floating-point number (larger in absolute value).

3. Special numbers: 0, -0, ∞, $-\infty$, 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:

   - 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 $E_{\min} = (00000001)_2 = (1)_{10}$, $E_{\max} = (11111110)_2 = (254)_{10}$.
- The range of positive normalized numbers is from

$$N_{\min} = 1.00\cdots0 \times 2^{E_{\min}-127} = 2^{-126} \approx 1.2 \times 10^{-38}$$

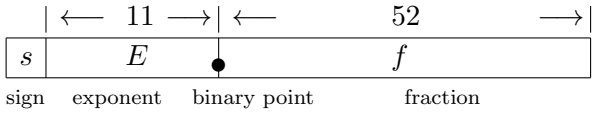
to

$$N_{\max} = 1.11\cdots1 \times 2^{E_{\max}-127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}.$$

- Special repsentations for 0, $\pm\infty$ and NaN:

zero	=	±	00000000	00000000000000000000000
$\pm\infty$	=	±	11111111	00000000000000000000000
NaN	=	±	11111111	otherwise

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



- It represents the numer

$$(-1)^s \cdot (1.f) \times 2^{E-1023}$$

- The range of positive normalized numbers is from

$$N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308}$$

to

$$N_{\max} = 1.11\cdots1 \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}.$$

- Special repsentations 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

Format	Precision p	Machine epsilon $\epsilon = 2^{-p-1}$
single	24	$\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
double	53	$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
extended	64	$\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$

9. Rounding

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

$$x = (1.b_1 b_2 \cdots b_{p-1} b_p b_{p+1} \ldots) \times 2^E,$$

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

$$x_- = 1.b_1 b_2 \cdots b_{p-1} \times 2^E,$$

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

$$x_+ = ((1.b_1 b_2 \cdots b_{p-1}) + (0.00 \cdots 01)) \times 2^E,$$

   therefore, also the next one that bigger than x.

   If x is negative, the situtation 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 $x \ge 0$; $\text{round}(x) = x_+$ of $x \le 0$;
   - *round to nearest*: $\text{round}(x) = x_-$ or x_+, whichever is nearer to x, except that if $x > N_{\max}$, $\text{round}(x) = \infty$, and if $x < -N_{\max}$, $\text{round}(x) = -\infty$. 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,

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

   and

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

   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

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

   where $|\delta| \le \frac{1}{2}\epsilon$ 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

Event	Example	Set result to
Invalid operation	$0/0$, $0 \times \infty$	NaN
Division by zero	Finite nonzero/0	$\pm\infty$
Overflow	$\lvert x\rvert > N_{\max}$	$\pm\infty$ or $\pm N_{\max}$
underflow	$x \neq 0, \lvert x\rvert < N_{\min}$	± 0, $\pm N_{\min}$ or subnormal
Inexact	whenever $\mathrm{fl}(x \circ y) \neq x \circ y$	correctly rounded value

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 τ_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

$$\begin{aligned}
\mathrm{fl}(\hat{x}+\hat{y}) &= (\hat{x}+\hat{y})(1+\delta), \qquad |\delta| \leq \frac{1}{2}\epsilon \\
&= x(1+\tau_1)(1+\delta) + y(1+\tau_2)(1+\delta) \\
&= x + y + x(\tau_1 + \delta + O(\tau\epsilon)) + y(\tau_2 + \delta + O(\tau\epsilon)) \\
&= (x+y)\left(1 + \frac{x}{x+y}(\tau_1 + \delta + O(\tau\epsilon)) + \frac{y}{x+y}(\tau_2 + \delta + O(\tau\epsilon))\right) \\
&\equiv (x+y)(1+\hat{\delta}),
\end{aligned}$$

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 $\mathrm{fl}(\hat{x}+\hat{y})$ 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 $\mathrm{fl}(\hat{x}+\hat{y})$ 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, $\mathrm{fl}(\hat{x}+\hat{y})$ 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

n	$\mathrm{fl}(\sqrt{n+1})$	$\mathrm{fl}(\sqrt{n})$	$\mathrm{fl}(\mathrm{fl}(\sqrt{n+1}) - \mathrm{fl}(\sqrt{n})$
1.00e+10	1.00000000004999994e+05	1.00000000000000000e+05	4.99999441672116518e-06
1.00e+11	3.16227766018419061e+05	3.16227766016837908e+05	1.58115290105342865e-06
1.00e+12	1.00000000000050000e+06	1.00000000000000000e+06	5.00003807246685028e-07
1.00e+13	3.16227766016853740e+06	3.16227766016837955e+06	1.57859176397323608e-07
1.00e+14	1.00000000000000503e+07	1.00000000000000000e+07	5.02914190292358398e-08
1.00e+15	3.16227766016838104e+07	3.16227766016837917e+07	1.86264514923095703e-08
1.00e+16	1.00000000000000000e+08	1.00000000000000000e+08	0.00000000000000000e+00

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

n	$\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n}))$
1.00e+10	4.99999999875000e-06
1.00e+11	1.58113883008023 7e-06
1.00e+12	4.99999999998749e-07
1.00e+13	1.58113883008415 0e-07
1.00e+14	4.99999999999987e-08
1.00e+15	1.58113883008418 9e-08
1.00e+16	5.00000000000000e-09

In fact, one can show that $\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n})) = (\sqrt{n+1}-\sqrt{n})(1+\delta)$, where $|\delta| \le 5\epsilon + O(\epsilon^2)$ (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

$$0 \le f(x) < 1/2 \quad \text{for all } x \neq 0.$$

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:

$$\begin{aligned} \mathrm{fl}(\hat{x} \times \hat{y}) &= (\hat{x} \times \hat{y})(1+\delta) = xy(1+\tau_1)(1+\tau_2)(1+\delta) \equiv xy(1+\hat{\delta}_\times), \\ \mathrm{fl}(\hat{x}/\hat{y}) &= (\hat{x}/\hat{y})(1+\delta) = (x/y)(1+\tau_1)(1+\tau_2)^{-1}(1+\delta) \equiv xy(1+\hat{\delta}_\div), \end{aligned}$$

where

$$\hat{\delta}_\times = \tau_1 + \tau_2 + \delta + O(\tau\epsilon), \quad \hat{\delta}_\div = \tau_1 - \tau_2 + \delta + O(\tau\epsilon).$$

Thus

$$\begin{aligned} |\hat{\delta}_\times| &\le 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon) \\ |\hat{\delta}_\div| &\le 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon). \end{aligned}$$

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, $y \in \mathcal{R}^3$

   $$x^T y \stackrel{\text{def}}{=} x_1y_1 + x_2y_2 + x_3y_3,$$

   assuming already x_i's and y_j's are floating-point numbers. It is likely that $\mathrm{fl}(x \cdot y)$ is computed in the following order.

   $$\mathrm{fl}(x^T y) = \mathrm{fl}\big(\,\mathrm{fl}(\mathrm{fl}(x_1y_1) + \mathrm{fl}(x_2y_2)) + \mathrm{fl}(x_3y_3)\,\big).$$

   Adopting the floating-point arithmetic model, we have

   $$\begin{aligned}
   \mathrm{fl}(x^T y) &= \mathrm{fl}\big(\,\mathrm{fl}(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2)) + x_3y_3(1+\epsilon_3)\,\big) \\
   &= \mathrm{fl}\big(\,(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2))(1+\delta_1) + x_3y_3(1+\epsilon_3)\,\big) \\
   &= \big(\,(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2))(1+\delta_1) + x_3y_3(1+\epsilon_3)\,\big)(1+\delta_2) \\
   &= x_1y_1(1+\epsilon_1)(1+\delta_1)(1+\delta_2) + x_2y_2(1+\epsilon_2)(1+\delta_1)(1+\delta_2) \\
   &\quad + x_3y_3(1+\epsilon_3)(1+\delta_2),
   \end{aligned}$$

   where $|\epsilon_i| \le \frac{1}{2}\epsilon$ and $|\delta_j| \le \frac{1}{2}\epsilon$.

   Now there are two ways to interpret the errors in the computed $\mathrm{fl}(x^T y)$:

   (a) We have

   $$\mathrm{fl}(x^T y) = x^T y + E,$$

   where $E = x_1y_1(\epsilon_1 + \delta_1 + \delta_2) + x_2y_2(\epsilon_2 + \delta_1 + \delta_2) + x_3y_3(\epsilon_3 + \delta_2) + O(\epsilon^2)$. It implies that

   $$|E| \le \frac{1}{2}\epsilon(3|x_1y_1| + 3|x_2y_2| + 2|x_3y_3|) + O(\epsilon^2) \le \frac{3}{2}\epsilon \cdot |x|^T|y| + O(\epsilon^2).$$

   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

   $$\mathrm{fl}(x^T y) = \hat{x}^T \hat{y} = (x + \Delta x)^T (y + \Delta y),$$

   where[1]

   $$\begin{array}{lllllllll}
   \hat{x}_1 &=& x_1(1+\epsilon_1), & \hat{y}_1 &=& y_1(1+\delta_1)(1+\delta_2) &\equiv& y_1(1+\hat{\delta}_1), \\
   \hat{x}_2 &=& x_2(1+\epsilon_2), & \hat{y}_2 &=& y_2(1+\delta_1)(1+\delta_2) &\equiv& y_2(1+\hat{\delta}_2), \\
   \hat{x}_3 &=& x_3(1+\epsilon_3), & \hat{y}_3 &=& y_3(1+\delta_2) &\equiv& y_3(1+\hat{\delta}_3).
   \end{array}$$

   It can be seen that $|\hat{\delta}_1| = |\hat{\delta}_2| \le \epsilon + O(\epsilon^2)$ and $|\hat{\delta}_3| \le \frac{1}{2}\epsilon$. This says the computed value $\mathrm{fl}(x^T y)$ 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.

   D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 18(1):5–48, 1991.

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

   M. Overton. Numerical computing with IEEE floating-point arithemetic. SIAM, Philadelphia, 2001.

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

   - `http://bit.ly/kacov18`

4. Websites for discussion of numerical disasters:

   - T. Huckle, Collection of software bugs
     http://www5.in.tum.de/∼huckle/bugse.html
     Recent book: "Bits and Bugs: A Scientific and Historical Review of Software Failures in Computational Science" by T. Huckle and T. Neckel, SIAM, March 2019.
   - K. Vuik, Some disasters caused by numerical errors
     http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html
   - D. Arnold, Some disasters attributable to bad numerical computing
     http://www.ima.umn.edu/∼arnold/disasters/disasters.html