Floating-Point Arithmetic

ECS130 Winter 2017

January 27, 2017

Floating point numbers

- Floating-point representation of numbers (scientific notation) has four components, for example,

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

- Computers use binary (base 2) representation, each digit is the integer 0 or 1, e.g.

$$\begin{aligned} 10110 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 22_{\text{in base 10}} \end{aligned}$$

and

$$\begin{aligned} \frac{1}{10} &= \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \dots \\ &= \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \dots \\ &= 1.100110011\dots \times 2^{-4} \end{aligned}$$

Floating point numbers

- The representation of a floating point number:

$$x = \pm b_0.b_1 b_2 \cdots b_{p-1} \times 2^E$$

  where
  - It is *normalized*, i.e., $b_0 = 1$ (the hidden bit)
  - *Precision* $(= p)$ is the number of bits in the significand (mantissa) (including the hidden bit).
  - *Machine epsilon* $\epsilon_m = 2^{-(p-1)}$, the gap between the number 1 and the smallest floating point number that is greater than 1.
- Special numbers

$$0,\ -0,\ \mathsf{Inf} = \infty,\ -\mathsf{Inf} = -\infty,\ \mathsf{NaN} = \text{“Not a Number”}.$$

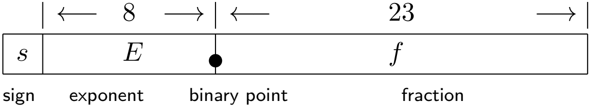
IEEE standard

- 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:
  1. consistent representation of floating-point numbers by all machines adopting the standard;
  2. correctly rounded floating-point operations, using various rounding modes;
  3. consistent treatment of exceptional situation such as division by zero.

Citation of 1989 Turing Award to William Kahan: "... *his devotion to providing systems that can be safely and robustly used for numerical computations has impacted the lives of virtually anyone who will use a computer in the future*"

IEEE single precision format

- **Single** format takes 32 bits (=4 bytes) long:



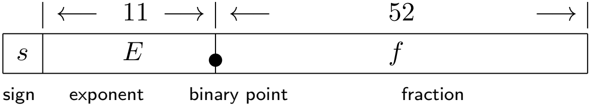
- It represents the number $(-1)^s \cdot (1.f) \times 2^{E-127}$
- The leading 1 in the fraction need not be stored explicitly since it is always 1 (*hidden bit*)
- Precision $p = 24$ and machine epsilon $\epsilon_m = 2^{-23} \approx 1.2 \times 10^{-7}$
- The "$E - 127$" in the exponent is to avoid the need for storage of a sign bit.
  $E_{\min} = (00000001)_2 = (1)_{10}$, $E_{\max} = (11111110)_2 = (254)_{10}$.
- The range of positive normalized numbers:

$$N_{\min} = 1.00\cdots0 \times 2^{E_{\min}-127} = 2^{-126} \approx 1.2 \times 10^{-38}$$
$$N_{\max} = 1.11\cdots1 \times 2^{E_{\max}-127} \approx 2^{128} \approx 3.4 \times 10^{38}.$$

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

IEEE double pecision format

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



- It represents the numer $(-1)^s \cdot (1.f) \times 2^{E-1023}$
- Precision $p = 53$ and machine epsilon $\epsilon_m = 2^{-52} \approx 2.2 \times 10^{-16}$
- The range of positive normalized numbers is from

$$
\begin{aligned}
N_{\min} &= 2^{-1022} \approx 2.2 \times 10^{-308} \\
N_{\max} &= 1.11\cdots1 \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}.
\end{aligned}
$$

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

Rounding modes

- Let a positive real number x be in the normalized range, i.e., $N_{\min} \leq x \leq N_{\max}$, and write 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-pont 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_- (also the next one that bigger than x) is

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

- If x is negative, the situtation is reversed.

rounding modes, cont'd

Four rounding modes:

1. *round down*: $\text{fl}(x) = x_-$
2. *round up*: $\text{fl}(x) = x_+$
3. *round towards zero*:
$$\text{fl}(x) = x_- \text{ of } x \geq 0$$
$$\text{fl}(x) = x_+ \text{ of } x \leq 0$$
4. *round to nearest* (*IEEE default rounding mode*):
$$\text{fl}(x) = x_- \quad \text{or} \quad x_+ \quad \text{whichever is nearer to } x.$$

Note: except that if $x > N_{\max}$, $\text{fl}(x) = \infty$, and if $x < -N_{\max}$, $\text{fl}(x) = -\infty$. 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.

Rounding error

- When the *round to nearest* is in effect,

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

- Therefore, we have

$$\text{relerr} = \begin{cases} \frac{1}{2} \cdot 2^{1-24} = 2^{-24} \approx 5.96 \cdot 10^{-8}, & \text{single precision} \\ \\ \frac{1}{2} \cdot 2^{-52} \approx 1.11 \times 10^{-16}, & \text{double precision.} \end{cases}$$

Floating-point arithmetic

- IEEE rules for correctly rounded floating-point operations:

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

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

  where

$$|\delta| \le \frac{1}{2}\epsilon_m$$

  for the round to nearest,

- IEEE standard also requires that correctly rounded remainder and square root operations be provided.

Floating-point arithmetic, cont'd

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

Floating-point arithmetic error

- 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, and
- **Question: how do the four basic arithmetic operations behave?**

Floating-point arithmetic error: $+, -$

Addition and subtraction:

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

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

$$|\hat{\delta}| \le \frac{|x|+|y|}{|x+y|}\left(\tau + \frac{1}{2}\epsilon_m + O(\tau\epsilon_m)\right).$$

Floating-point arithmetic error: $+, -$

Three possible cases:

1. If x and y have the same sign, i.e., $xy > 0$, then $|x + y| = |x| + |y|$; this implies
$$|\hat{\delta}| \leq \tau + \frac{1}{2}\epsilon_m + O(\tau\epsilon_m) \ll 1.$$
Thus $\mathrm{fl}(\hat{x} + \hat{y})$ approximates $x + y$ well.
2. 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.
This is so called **catastrophic cancellation**, it causes relative errors or uncertainties already presented in $\hat{x}$ and $\hat{y}$ to be magnified.
3. In general, if $(|x| + |y|)/|x + y|$ is not too big, $\mathrm{fl}(\hat{x} + \hat{y})$ provides a good approximation to $x + y$.

Catastrophic cancellation: example 1

- Computing $\sqrt{x+1} - \sqrt{x}$ straightforward causes substantial loss of significant digits for large n

x	$\mathrm{fl}(\sqrt{x+1})$	$\mathrm{fl}(\sqrt{x})$	$\mathrm{fl}(\mathrm{fl}(\sqrt{x+1}) - \mathrm{fl}(\sqrt{x})$
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: example 1

- *Catastrophic cancellation can sometimes be avoided if a formula is properly reformulated.*
- For example, one can compute $\sqrt{x+1} - \sqrt{x}$ almost to full precision by using the equality

$$\sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1} + \sqrt{x}}.$$

Consequently, the computed results are

n	$\mathrm{fl}(1/(\sqrt{n+1} + \sqrt{n}))$
1.00e+10	4.999999998750000e-06
1.00e+11	1.581138830080237e-06
1.00e+12	4.999999999998749e-07
1.00e+13	1.581138830084150e-07
1.00e+14	4.999999999999987e-08
1.00e+15	1.581138830084189e-08
1.00e+16	5.000000000000000e-09

Catastrophic cancellation: example 2

- Consider the function
$$h(x) = \frac{1 - \cos x}{x^2}$$
Note that $0 \leq f(x) < 1/2 \quad$ for all $x \neq 0$.
- Let $x = 1.2 \times 10^{-8}$, then the computed
$$\mathrm{fl}(h(x)) = 0.770988...$$
is completely wrong!
- Alternatively, the function can be re-written as
$$h(x) = \left(\frac{\sin(x/2)}{x/2}\right)^2.$$
- Consequently, for $x = 1.2 \times 10^{-8}$, then the computed function $\mathrm{fl}(h(x)) = 0.499999... < 1/2$ is fine!

Floating-point arithmetic error: $\times, /$

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 $\hat{\delta}_\times = \tau_1 + \tau_2 + \delta + O(\tau\epsilon_m)$

$\hat{\delta}_\div = \tau_1 - \tau_2 + \delta + O(\tau\epsilon_m)$.

Thus $|\hat{\delta}_\times| \leq 2\tau + \frac{1}{2}\epsilon_m + O(\tau\epsilon_m)$ and $|\hat{\delta}_\div| \leq 2\tau + \frac{1}{2}\epsilon_m + O(\tau\epsilon_m)$.

Multiplication and division are very well-behaved!

Reading

- Section 1.7 of *Numerical Computing with MATLAB* by C. Moler
- Websites discussions of numerical disasters:
  - T. Huckle, Collection of software bugs
    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
  - D. Arnold, Some disasters attributable to bad numerical computing
    http://www.ima.umn.edu/∼arnold/disasters/disasters.html
- In-depth material:
  D. Goldberg, **What every computer scientist should know about floating-point arithmetic**, *ACM Computing Survey, Vol.23(1), pp.5-48, 1991*

LU factorization – revisited: *the need of pivoting in LU factorization, numerically*

- LU factorization without pivoting

$$A = \begin{bmatrix} .0001 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix} = LU = \begin{bmatrix} 1 & \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ & u_{22} \end{bmatrix}$$

- In three decimal-digit floating-point arithmetic, we have

$$\begin{aligned} \widehat{L} &= \begin{bmatrix} 1 & 0 \\ \mathrm{fl}(1/10^{-4}) & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^4 & 1 \end{bmatrix}, \\ \widehat{U} &= \begin{bmatrix} 10^{-4} & 1 \\ & \mathrm{fl}(1 - 10^4 \cdot 1) \end{bmatrix} = \begin{bmatrix} 10^{-4} & 1 \\ & -10^4 \end{bmatrix}, \end{aligned}$$

- Check:

$$\widehat{L}\widehat{U} = \begin{bmatrix} 1 & 0 \\ 10^4 & 1 \end{bmatrix} \begin{bmatrix} 10^{-4} & 1 \\ & -10^4 \end{bmatrix} = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 0 \end{bmatrix} \not\approx A,$$

LU factorization – revisited: *the need of pivoting in LU factorization, numerically*

- Consider solving $Ax = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ for x using this LU factorization.
  - Solving

$$\widehat{L} y = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \Longrightarrow \quad \begin{array}{rcl} \widehat{y}_1 & = & \mathrm{fl}(1/1) = 1 \\ \widehat{y}_2 & = & \mathrm{fl}(2 - 10^4 \cdot 1) = -10^4. \end{array}$$

    note that the value 2 has been "lost" by subtracting 10^4 from it.
  - Solving

$$\widehat{U} x = \widehat{y} = \begin{bmatrix} 1 \\ -10^4 \end{bmatrix} \quad \Longrightarrow \quad \begin{array}{rcl} \widehat{x}_2 & = & \mathrm{fl}((-10^4)/(-10^4)) = 1 \\ \widehat{x}_1 & = & \mathrm{fl}((1-1)/10^{-4}) = 0, \end{array}$$

- $\widehat{x} = \begin{bmatrix} \widehat{x}_1 \\ \widehat{x}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is a completely erroneous solution to the correct answer $x \approx \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

LU factorization – revisited: *the need of pivoting in LU factorization, numerically*

- LU factorization with partial pivoting

$$PA = LU,$$

- By exchanging the order of the rows of A, i.e.,

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

Then for the LU factorization of PA is given by

$$\begin{aligned} \widehat{L} &= \begin{bmatrix} 1 & 0 \\ \mathrm{fl}(10^{-4}/1) & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{-4} & 1 \end{bmatrix}, \\ \widehat{U} &= \begin{bmatrix} 1 & 1 \\ & \mathrm{fl}(1 - 10^{-4} \cdot 1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ & 1 \end{bmatrix}. \end{aligned}$$

- The computed LU approximates A very accurately.
- As a result, the computed solution x is also perfect!