CS601: Software Development for Scientific Computing

Autumn 2023

Week2: Real Numbers, Programming Environment, ..

Recap: Toward Scientific Software

Physical process

↓

Mathematical model

↓

Algorithm

↓

Software program

↓

Simulation results

Real Numbers $\mathbb{R}$

- Most scientific software deal with Real numbers. Our toy code dealt with Reals
  - Numerical software is scientific software dealing with Real numbers
- Real numbers include rational numbers (integers and fractions), irrational numbers (pi etc.)
- Used to represent values of continuous quantity such as time, mass, velocity, height, density etc.
  - Infinitely many values possible
  - But computers have limited memory. So, have to use approximations.

Representing Real Numbers

- Real numbers are stored as *floating point numbers* (floating point system is a scheme to represent real numbers)
- E.g. floating point numbers:
  - $\pi = 3.14159$,
  - $6.03*10^{23}$
  - $1.60217733*10^{-19}$

exponent

General format: $\pm x \times b^e$

mantissa
(number ranges from:
1 to b OR 1/b to 1)

base
(e.g. base 10, 8, 2, 16)

3-digit Calculator

- Suppose base, b=10 and
- $x = \pm d_0.d_1d_2 \times 10^e$ where $\begin{cases} 1 \leq d_0 \leq 9, \\ 0 \leq d_1 \leq 9, \\ 0 \leq d_2 \leq 9 \\ -9 \leq e \leq 9 \end{cases}$
- precision = length of mantissa
  - What is the precision here?
- Exercise: What is the smallest positive number?
- Exercise: What is the largest positive number?
- Exercise: How many numbers can be represented in this format?
- Exercise: When is this representation not enough?

Floating Point System - Fundamentals

- **Precision (p)** - Length of mantissa
  - E.g. p=3 in 1.00 x 10^{-1}
- **Unit roundoff (u)** – smallest positive number where the *computed* value of 1+u is different from 1
  - E.g.   suppose p=4 and we wish to compute 1.0000+ 0.0001=1.0001
  - But we can't store the exact result (since p=4). We end up storing 1.000.
  - So, computed result of 1+u is same as 1
  - Suppose we tried adding 0.0005 instead. 1.0000+0.0005=1.0005 Now, round this: 1.001

  $\Rightarrow$ **u =0.0005**
- **Machine epsilon (ϵ_{mach})** – smallest a-1, where a is the smallest representable number greater than 1
  - E.g. consider 1.001 – 1.000 = 0.001.

  $\Rightarrow$ **usually ϵ_{mach} = 2 * u**

Floating Point System - Fundamentals

- **Forward error and backward error**

$\text{Comp}(f(x)) = (1+\epsilon_1)f((1+\epsilon_2)x)$,

where ϵ_i <= u (u is unit roundoff)

Comp(f(x)) is the computed value i.e. machine representable value of f(x).

Suppose ϵ_2 is zero. Then $\dfrac{\text{Comp}(f(x)) - f(x)}{f(x)} = \epsilon_1$

Floating Point System - Fundamentals

- **Forward error example**

  Let $y = \sqrt{2}$, $z = y^2$ and

  $y = \sqrt{2}$ `implemented as: y = sqrt(2);`

  $z = y^2$ `implemented as: z = y * y;`

  with double precision floating point system

  Then $\left\{\frac{Comp(f(x)) - f(x)}{f(x)}\right\}$, can be calculated (note: f(x) = z = 2, and Comp(f(x)) = y*y)

```
y:1.41421356237
z:2
res1=z-2:4.44089209850e-16
res2=res1/z:2.22044604925e-16
```

Absolute error / relative error

Forward error
(also happens to be u for double)

Floating Point System - Fundamentals

- **Backward error example**

  Let $\mathrm{z} = \sin(2\pi)$. Then forward error is infinity!

  Subtract x with a multiple of 2π to make $0 \le x < 2\pi$

  And then compute sin(x) to get the absolute error for $x \ge 2\pi$ at most u$|x|$ (u is unit roundoff)

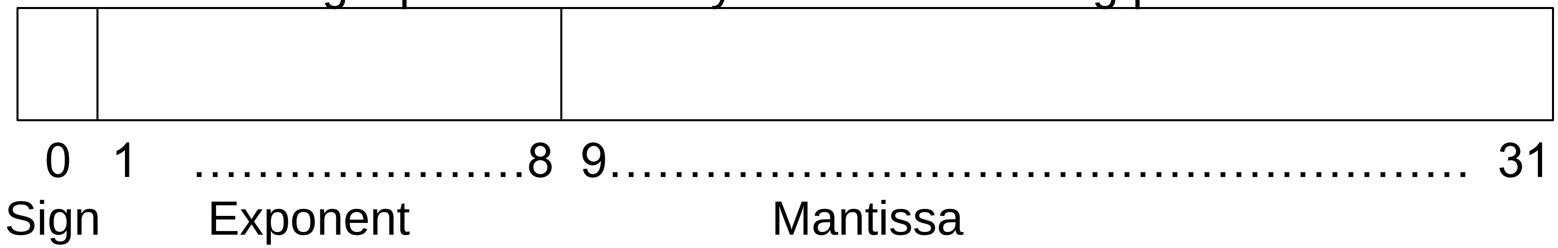
  This is perturbing the argument x (argument reduction). Instead of computing sin(x) we are computing sin($(1+\epsilon_2)x$). This is example of backward error.

IEEE 754 Floating Point System

- Prescribes single, double, and extended precision formats

Precision	**u**	**Total bits used (sign, exponent, mantissa)**
Single	6×10^{-8}	32 (1, 8, 23)
Double	2×10^{-16}	64 (1, 11, 52)
Extended	5×10^{-20}	80 (1, 15, 64)

single precision binary IEEE 754 floating point format



IEEE 754 Floating Point Arithmetic

double precision binary IEEE 754 floating point format

0	1…………………..11	1……………………………………….52
Sign	Exponent	Mantissa

- if exponent bits e_1-e_{11} are not all 1s or 0s, then the *normalized* number

$$n = \pm(1.m_1m_2..m_{52})_2 \times 2^{(e_1e_2..e_{11})_2 - 1023}$$

- **Machine epsilon** is the gap between 1 and the next largest floating point number. $2^{-52} \approx 10^{-16}$ for double.
- Exercise: What is minimum positive *normalized* double number?
- Exercise: What is maximum positive *normalized* double number?