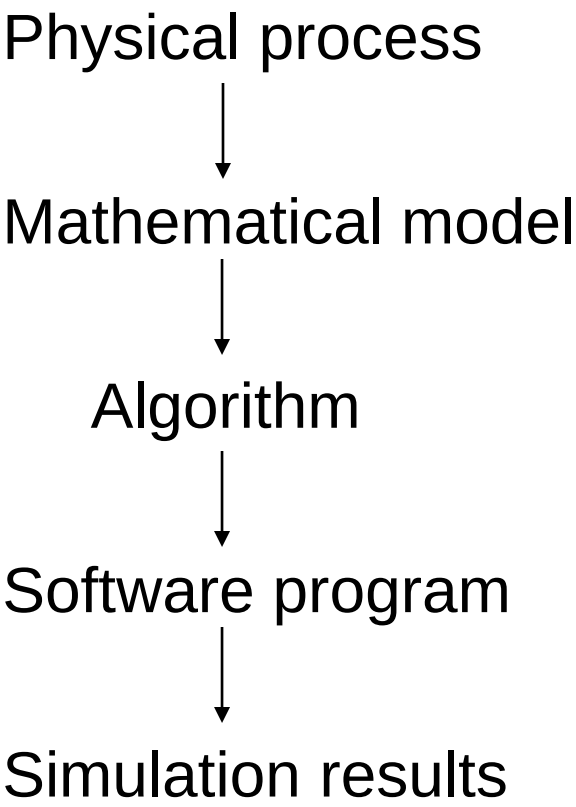
CS601: Software Development for Scientific Computing

Autumn 2024

Week2: Real numbers and their program representation

Recap: Scientific Computing



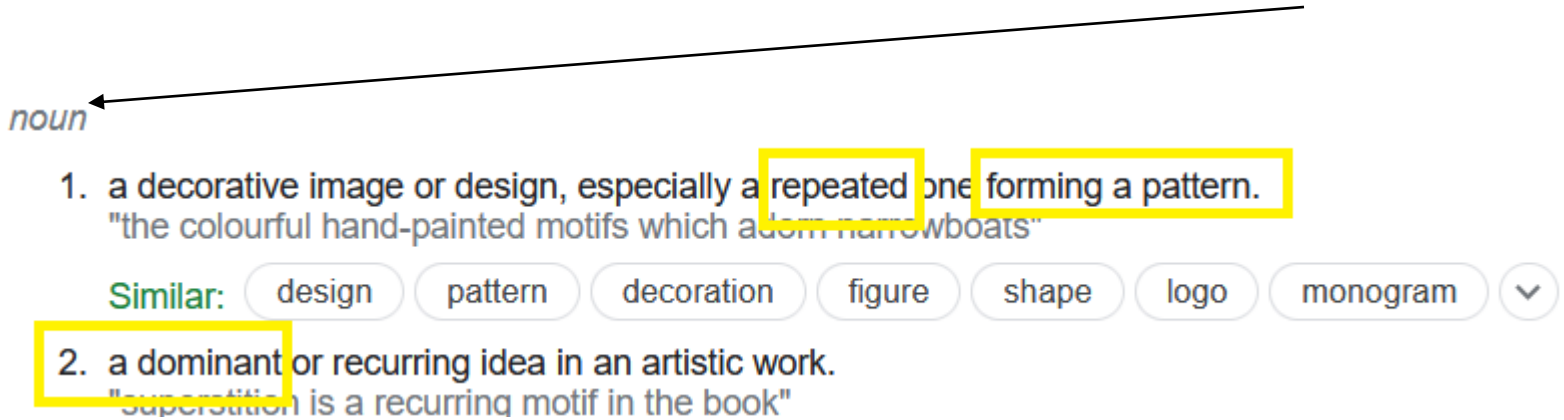
Recap: Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Recap: Computational Thinking

- Abstractions
  - Our "mental" tools
  - Includes: choosing right abstractions, operating at multiple layers of abstractions, and defining relationships among layers
- Automation
  - Our "metal" tools that amplify the power of "mental" tools
  - Is mechanizing our abstractions, layers, and relationships
    - Need precise and exact notations / models for the "computer" below ("computer" can be human or machine)
- *Computing is the automation of our abstractions*

Scientific Software - Motifs



1. Finite State Machines
2. Combinatorial
3. Graph Traversal
4. Structured Grid
5. Dense Matrix
6. Sparse Matrix
7. FFT
8. Dynamic Programming
9. N-Body (/ particle)
10. MapReduce
11. Backtrack / B&B
12. Graphical Models
13. Unstructured Grid

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:
  - $\pi = 3.14159,$
  - $6.03 * 10^{23}$
  - $1.60217733 * 10^{-19}$

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

exponent

mantissa

base

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

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

If 1 to b then
$x_0 . x_1 x_2 x_3 \ldots x_{m-1})$

3-Digit Decimal Representation

- Suppose base, b=10 and
- $x = \pm d_0.d_1d_2 \times 10^e$ where $\begin{cases} 1 \le d_0 \le 9, \\ 0 \le d_1, d_2 \le 9, \\ -9 \le e \le 9 \end{cases}$
- 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 - Terminology

- **Precision (p)** - Length of mantissa
  - E.g. p=3 in 1.00×10^{-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** $\epsilon_{mach} = 2 * u$

Exercise: 3-Digit Binary Representation

- Suppose base, b=2 and
- $x = \pm b_0.b_1b_2 \times 2^E$,where $\begin{cases} 1 \leq b_0 \leq 1, 0 \; iff \; b_1, b_2 = 0 \\ 0 \leq b_1, b_2 \leq 1, \\ -1 \leq E \leq 1 \end{cases}$
- What is the precision?
- What is the unit roundoff?
- What is the machine epsilon?
- What are the range of numbers that can be represented?

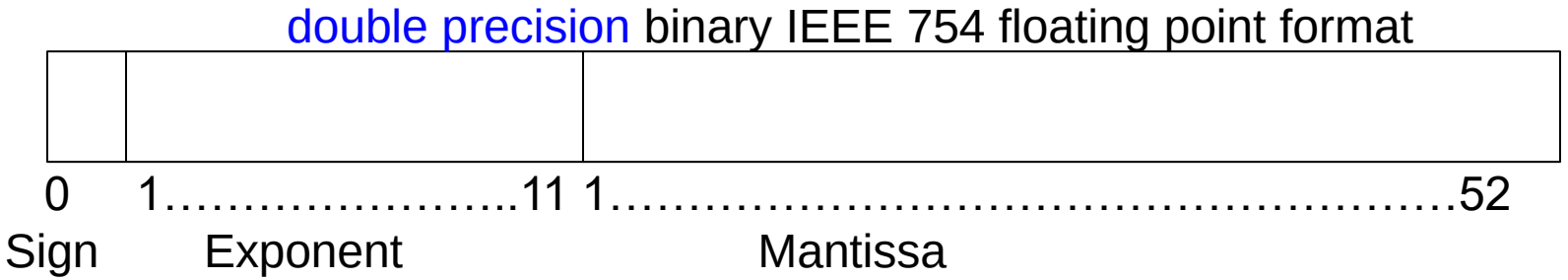
IEEE 754 Floating Point System

- Prescribes single, double, and extended precision formats

Precision	**u**	**Total bits used (sign, exponent, mantissa)**
Single	$6x10^{-8}$	32 (1, 8, 23)
Double	$2x10^{-16}$	64 (1, 11, 52)
Extended	$5x10^{-20}$	80 (1, 15, 64)

single precision binary IEEE 754 floating point format

IEEE 754 Floating Point Arithmetic

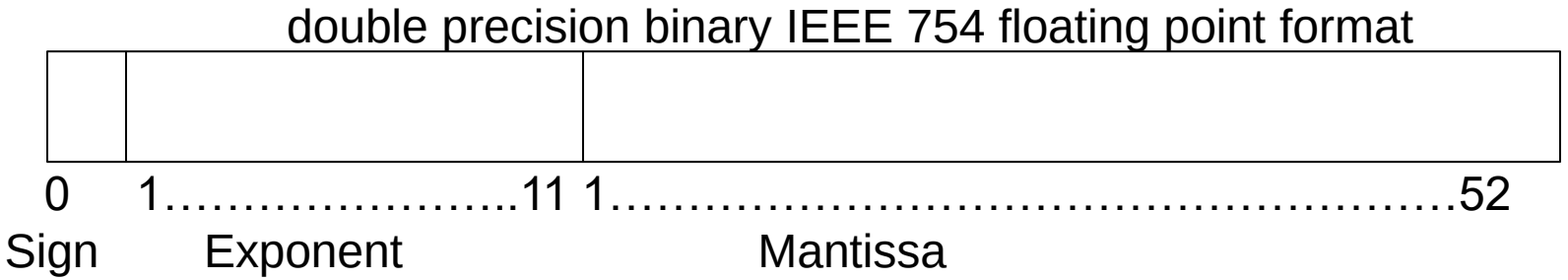


- if exponent bits e_1-e_{11} are not all 1s or 0s, then the *normalized* number

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

- **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?

IEEE 754 Floating Point Arithmetic



- if exponent bits e_1-e_{11} are all 0s, then:
  the *subnormal* number

$$n = \pm(\mathbf{0}.m_1 m_2 .. m_{52})_2 \times 2^{(e_1 e_2 .. e_{11})_2 - 102\mathbf{2}}$$

- if exponent bits e_1-e_{11} are all 1s, then:
  we can get –inf, NaN, or +inf based on value of $m_1 m_2 .. m_{52}$
  - If any m is non-zero, the number is NaN (not a number)

IEEE 754 Floating Point Arithmetic

- Don't test for equality
  - The special case of `x=y; if(y == x)`
- Order is important
  - Floating point arithmetic is *not associative*
    - (x+y)+z not the same as x+(y+z)
- Explicit coding of textbook formula may not be the best option to solve
  - $x^2 - 2px - q = 0$ p and q are positive: p=12345678, q=1
  - **Exercise:** find the minimum of the roots.
- Do intermediate calculations in higher precision than needed for end result.
- Subtracting approximations of two nearby numbers results in a bad approximation of the actual difference – **catastrophic cancellation**

Floating Point System - Fundamentals

- **Forward error and backward error**

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

where $\epsilon_i <= u$ (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 forward error, $\left\{\frac{Comp(f(x)) - f(x)}{f(x)}\right\}$, can be calculated

  (note: f(x) = z = 2, and Comp(f(x)) = y*y)

```
y:1.41421356237
z:2
res1=z-2:4.4408920985e-16
res2=res1/z:2.22044604925e-16
```

Absolute error / relative error

Forward error (also happens to be `u, unit roundoff, for double`)

Floating Point System - Fundamentals

- **Backward error example**

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

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

  And then compute sin(x) to get the absolute error for $x \geq 2\pi$ 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+ ϵ_2)x). This is example of backward error.

IEEE 754 Floating Point – Misc..

- **+0, -0, Inf, and NaN –**
  - Stop your program when you see a NaN (indicative of a bug)
  - How to check if a number is NaN?

    `if (x == x)` is `false`

    Exercise: Give an example when you get a NaN?

- **Rounding modes –** Round up, Round down, Round to nearest, Round towards zero
  - Default is round to nearest. Can be set using compiler options and library methods. Avoid changing rounding modes.
  - Can use this to flush out bugs! (change round modes and results shouldn't change drastically).