Computational Physics – PH 354

Manish Jain
Prateek Sharma

Email: mjain@iisc.ac.in
Email: prateek@iisc.ac.in

Logistics

- Homework 1 has been posted – Due date 20th Jan.
- https://iiscphy354.github.io/computational-physics/
- All homeworks have to submitted via github.

Python Tutorial

- Sanat has already given a python tutorial.
- Project will be decided by you in consultation with your Masters/PhD/Bachelors advisor – subject to our approval as well.
- Please send us a short paragraph about what you are planning to do for the project.

Machine representation, Precision and Errors

- Representation on a computer.
- Machine precision.
- Errors.

Machine Representation

Every computer has a limit how small/large a number can be.

Machine Representation

Every computer has a limit how small/large a number can be.

- A computer represents numbers in a binary form.

Machine Representation

Every computer has a limit how small/large a number can be.

- A computer represents numbers in a binary form.
- Word length: number of bytes used to store a number. The number of bits processed by a computer's CPU in one go.

Machine Representation

Every computer has a limit how small/large a number can be.

- A computer represents numbers in a binary form.
- Word length: number of bytes used to store a number. The number of bits processed by a computer's CPU in one go.
- Most common architecture:
  Word length = 4 bytes = 32 bits.
  Word length = 8 bytes = 64 bits.
  (1 byte = 1 B = 8 bits: 00000000)

Integer Representation

- Integers are represented exactly on a computer.

Integer Representation

- Integers are represented exactly on a computer.
- Range usually depends only on the machine!

Integer Representation

- Integers are represented exactly on a computer.
- Range usually depends only on the machine!
- Python is an exception – can represent arbitrarily large integers – show 2^{10000}

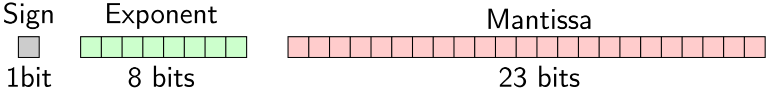
Integer Representation

- Integers are represented exactly on a computer.
- Range usually depends only on the machine!
- Python is an exception – can represent arbitrarily large integers – show 2^{10000}
- For most other languages – dependent on the size of the integers:
  integer*4 : 32 bits – highest number should be 2^{32} - 1
  But first bit is reserved for sign:
  -2^{31} – 2^{31}-1

Floating point representation – single precision

For eg. $123.45e6 = 0.12345e9$

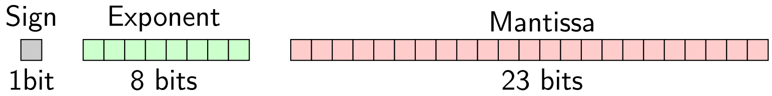
sign: $+$, exponent: $+9$, mantissa: 12345



Floating point representation – single precision

For eg. 123.45e6 = 0.12345e9

sign: +, exponent: +9, mantissa: 12345

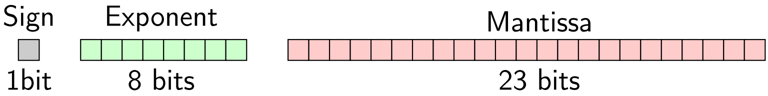


- Range of exponent: [-127, 127] ($2^{127} \sim 10^{+38}$)

Floating point representation – single precision

For eg. 123.45e6 = 0.12345e9

sign: +, exponent: +9, mantissa: 12345

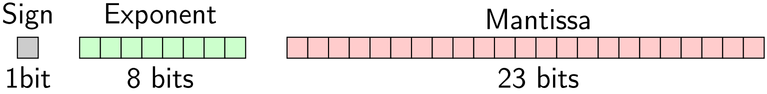


- Range of exponent: [-127, 127] ($2^{127} \sim 10^{+38}$)
- Single precision: 6-7 decimal places ($1/2^{23} \sim 10^{-7}$)

Floating point representation – single precision

For eg. 123.45e6 = 0.12345e9

sign: +, exponent: +9, mantissa: 12345



- Range of exponent: [-127, 127] ($2^{127} \sim 10^{+38}$)
- Single precision: 6-7 decimal places ($1/2^{23} \sim 10^{-7}$)
- Range max: $\pm 3.4 \times 10^{38}$.
- Range min: $\pm 1.4 \times 10^{-45}$.

Example

Getting a problem with single precision is quite easy:

Example: Bohr's radius:

$$a_0 = \frac{4\pi\epsilon_0\hbar^2}{m_e e^2}$$

where

$$\epsilon_0 = 8.85 \times 10^{-12}\mathrm{C}^2/\mathrm{N}/\mathrm{m}^2$$

$$\hbar = 6.63 \times 10^{-34}/2\pi\mathrm{J\ s}$$

$$m_e = 9.11 \times 10^{-31}\mathrm{Kg}$$

$$e = 1.60 \times 10^{-19}\mathrm{C}$$

Numerator is: 1.24×10^{-78} and Denominator is: 2.33×10^{-68}.

What can one do?

- Restructure the equation.

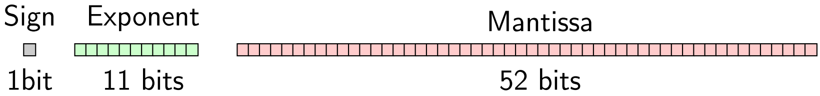
What can one do?

- Restructure the equation.
- Change units – work in atomic units where all these quantites are $\mathcal{O}(1)$.

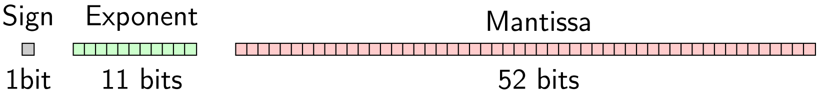
What can one do?

- Restructure the equation.
- Change units – work in atomic units where all these quantites are $\mathcal{O}(1)$.
- Increase precision!

Floating point representation – double precision

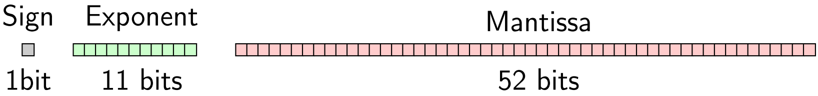


Floating point representation – double precision



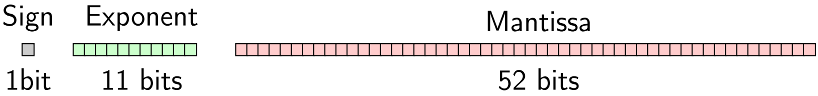
- Range of exponent: [-1023, 1023] ($2^{1023} \sim 10^{+308}$)

Floating point representation – double precision



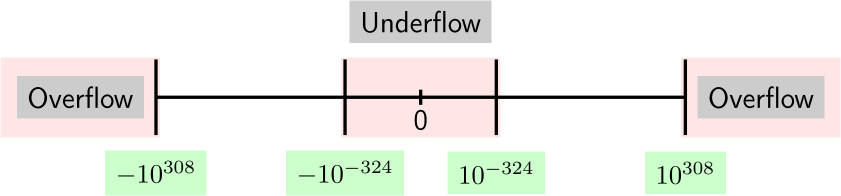
- Range of exponent: [-1023, 1023] $(2^{1023} \sim 10^{+308})$
- Single precision: 15-16 decimal places $(1/2^{52} \sim 1.2 \times 10^{-15})$

Floating point representation – double precision



- Range of exponent: [-1023, 1023] ($2^{1023} \sim 10^{+308}$)
- Single precision: 15-16 decimal places ($1/2^{52} \sim 1.2 \times 10^{-15}$)
- Range max: $\pm 1.78 \times 10^{308}$.
- Range min: $\pm 4.94 \times 10^{-324}$.

Floating point representation – double precision



Machine Precision

Machine precision is the smallest number ϵ such that the difference between 1 and $1 + \epsilon$ is nonzero, ie., it is the smallest difference between two numbers that the computer recognizes.

```
def machineEpsilon(func=float):
    machine_epsilon = func(1)
    while func(1)+func(machine_epsilon) != func(1):
        machine_epsilon_last = machine_epsilon
        machine_epsilon = func(machine_epsilon)/func(2)
    return machine_epsilon_last
```

Machine precision

```
>>> machineEpsilon(float)
2.22044604925031e-16
>>> import numpy as np
>>> machineEpsilon(np.float32)
1.1920929e-07
>>> machineEpsilon(np.float64)
2.220446049250313e-16
```

Three types of Errors

- Grammatical
  Using what is NOT in the programming language – the compiler finds these.

Three types of Errors

- Grammatical
  Using what is NOT in the programming language – the compiler finds these.
- Run time errors
  (n-1) errors; Inversion of logical tests etc. – we have to find them

Three types of Errors

- Grammatical
  Using what is NOT in the programming language – the compiler finds these.
- Run time errors
  (n-1) errors; Inversion of logical tests etc. – we have to find them
- Mirabile visu (strange to behold)
  They show up only for some input parameters. The code works for the test cases but blows up for some values of parameters!
  Reason: Loss of significant digits (round off errors), unstable algorithms etc.

Typical Errors

- Round off errors: Any number is represented by a finite number of bits.
  The difference between the true value of the number and its value on the computer is called round off error.
- Approximation errors/ Truncation errors: From using approximations such as replacing

$$\int_0^\infty f(x)dx \text{ with } \int_0^L f(x)dx \text{ with finite L}$$

Round off Errors

- Loss of significant digits

$$x = 1000000000000000.0$$
$$y = 1000000000000001.234567$$

Calculating $y - x = 1.234567$ but the computer calculates this as $y - x = 1.25$ – instead of 16 figures we only have 2 figures!

Round off Errors

- Loss of significant digits

$$x = 1000000000000000.0$$
$$y = 1000000000000001.234567$$

  Calculating $y - x = 1.234567$ but the computer calculates this as $y - x = 1.25$ – instead of 16 figures we only have 2 figures!

- Loss of precision
  Erosion by repeated rounding errors (least significant digits being eroded first).
  The average accumulated mulipication error after N multiplications is $\sqrt{N}\epsilon_0$.

Round off Errors

- Loss of significant digits

$$x = 1000000000000000.0$$
$$y = 1000000000000001.234567$$

  Calculating $y - x = 1.234567$ but the computer calculates this as $y - x = 1.25$ – instead of 16 figures we only have 2 figures!
- Loss of precision
  Erosion by repeated rounding errors (least significant digits being eroded first).
  The average accumulated mulipication error after N multiplications is $\sqrt{N}\epsilon_0$.
- Some times the problem is not round-off errors but numerical stability of the algorithm. Even tiny round-off errors grow rapidly if algorithm is not numerically stable.

Loss of significant digits

Loss of significant digits occurs in so many ways that it defies useful classification and lack systematic cures!

```
from math import sqrt
x = 1.0
y = 1.0 + (1e-14)*sqrt(2)
print (1e14)*(y-x)
print sqrt(2)
```

```
1.42108547152
1.41421356237
```

Calculation is accurate only to first decimal place – rest is garbage!

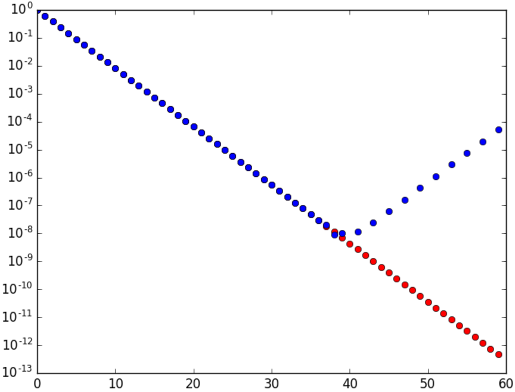
Numerical instability

Calculate the series $a_n = \phi^n \; n = 0, 1, 2 \ldots$
where ϕ is the golden ratio:

$$\phi = \frac{\sqrt{5} - 1}{2}$$

- Method 1: $a_0 = 1$ and $a_n = a_{n-1}\phi$
- Method 2: $a_0 = 1 \; a_1 = \phi$ and $a_n = a_{n-2} - a_{n-1}$

Numerical instability



Method 1 is stable – while method 2 is not!

- Dealing with infinity – sometimes change of variables can help (if it does not introduce any singularities).
Other times "tails" can be evaluated analytically:

$$\int_0^\infty \frac{\sqrt{x}}{x^2+1} = \int_0^L \frac{\sqrt{x}}{x^2+1} + \int_L^\infty \frac{\sqrt{x}}{x^2+1}$$

for $L >> 1$:

$$\int_L^\infty \frac{\sqrt{x}}{x^2+1} \approx \int_L^\infty \frac{1}{x^{\frac{3}{2}}} = \frac{2}{\sqrt{L}}$$

- When a continuous problem is discretized – Use of Taylor series expansion etc
Use of second order Taylor expansion vs first order can control this error better.

Truncation vs Round-off error

- Truncation error controlled by programmer; choose a more accurate method!

Truncation vs Round-off error

- Truncation error controlled by programmer; choose a more accurate method!
- Round off error is fixed (16 decimal places in DP); less control

Truncation vs Round-off error

- Truncation error controlled by programmer; choose a more accurate method!
- Round off error is fixed (16 decimal places in DP); less control
- Typically truncation error » Round-off error; e.g., $\Delta x = 10^{-3}$ then Truncation error for second order expansion $\sim 10^{-6}$.

Truncation vs Round-off error

- Truncation error controlled by programmer; choose a more accurate method!
- Round off error is fixed (16 decimal places in DP); less control
- Typically truncation error » Round-off error; e.g., $\Delta x = 10^{-3}$ then Truncation error for second order expansion $\sim 10^{-6}$.
- In general, order of accuracy not the sole metric for a better algorithm – Stability, Robustness, Mathematical properties are more crucial.