

Numerical Methods for Astrophysics:

ACCURACY AND SPEED

Michela Mapelli

Python. Scientific notation

Scientific notation consists in using an “e” to indicate the exponent of a number expressed in powers of 10

`1e19 = 10**19`

`1.6e10=1.6 * 10**10`

`2.1e-3 = 2.1 * 10**(-3)`

Numbers in scientific notation are always FLOAT even if intrinsically integers (e.g. `1e19`)

Scientific notation is highly recommended, especially if you deal with very large or very small numbers!!!

`1e19`



`10**19`



Python. Maximum size of a variable

Python variables (as well as other programming languages) cannot hold numbers that are arbitrarily large

Maximum size for floating point: $2^{1024} \sim 1.79769 \times 10^{308}$

OVERFLOW: a variable exceeding the maximum size overflows

Python does not always give an overflow warning

```
>>> x=1e308
>>> y=10.*x
>>> y
inf
```

UNDERFLOW: a variable is too small to be represented

For python $< 2^{-1022} \sim 2.22507 \times 10^{-308}$

Python sets the value to zero

INTEGERS in python: no limit, python can represent integers of arbitrary size (=arbitrary number of digits), because it decides memory allocation based on the size of the integer. The limit is the memory of the computer.

SLOW for large numbers because of memory access time

example: `print(2**10000000)`

Python. Rounding errors

Floating points are represented on a computer to only a certain precision.

In python: standard is **16 digits**

Example π

```
>>> import numpy as np
>>> np.pi
3.141592653589793
```

ROUNDING ERROR: difference between the true value of a number and its value on the computer

For example, we know that $3.3 - 1.1 = 2.2$,
but python might print 2.1999999999999997

A practical implication is that you should never use an if statement to check equality between two floats

```
>>> x=3.3-1.1
>>> if(x==2.2):
...     print(x)
does not print anything
```



```
>>> x=3.3-1.1
>>> epsilon=1e-2
>>> if(abs(x-2.2)<epsilon):
...     print(x)
...
2.1999999999999997
```



Python. Rounding errors

Rounding error equivalent to **error of measurement** in a lab experiment

```
from math import sqrt  
x=sqrt(2)
```

Not $x = \sqrt{2}$ but rather $x \pm \varepsilon = \sqrt{2}$ with $\varepsilon \sim x/1e16$

Good assumption: error distributed according to a **Gaussian distribution**

$$\sigma = C x$$

ERROR CONSTANT C ~ 1e-16

$$Y = x_1 + x_2 \quad \text{Error on } Y? \quad \sigma = \sqrt{\sigma_1^2 + \sigma_2^2} = C \sqrt{x_1^2 + x_2^2}$$

$$Y = \sum_{i=1}^N x_i \quad \text{Error on } Y? \quad \sigma^2 = \sum_{i=1}^N C^2 x_i^2 = C^2 N \langle x^2 \rangle$$

$$\text{Percentage error: } \frac{\sigma}{\sum_{i=1}^N x_i} = \frac{C \sqrt{N} \sqrt{\langle x^2 \rangle}}{N \langle x \rangle} = \frac{C}{\sqrt{N}} \frac{\sqrt{\langle x^2 \rangle}}{\langle x \rangle}$$

Python. Rounding errors

Subtraction between two very similar and large (or small) numbers

```
x=int(1e15)  
y=10000000000000001.23456789  
print(y-x)
```

the result is 1.2 or 1.25 (depending on your python version),
because you hit the 16 digit limit

EXERCISE:

Consider the two numbers $x = 1$, $y = 1 + 10^{-14} \sqrt{2}$. We see that

$$10^{14} (y - x) = \sqrt{2}$$

Now, write a script that defines x and y as above and then prints $10^{14} (y - x)$. The result of the print will be 1.42108547152, while $\sqrt{2} = 1.41421356237$. Hence, the result is only accurate to the first decimal place.

Python. Rounding errors

EXERCISE:

Consider a quadratic equation $a x^2 + b x + c = 0$ that admits 2 real solutions.

- a) Write a script that takes in input the three numbers a , b , c and returns the solution x using the standard formula

$$x = \frac{-b \pm \sqrt{b^2 - 4 a c}}{2 a} \quad (7)$$

Apply this to the solution of the equation where $a = c = 0.001$, $b = 1000$.

- b) Write the solution x in another way, by multiplying numerator and denominator by $-b \mp \sqrt{b^2 - 4 a c}$:

$$x = \frac{2 c}{-b \mp \sqrt{b^2 - 4 a c}} \quad (8)$$

Apply this to the solution of the equation where $a = c = 0.001$, $b = 1000$. What do you see? How do you explain it?

- c) Using what you learned, write a script that calculates both solutions accurately.

Python. Rounding errors

With method a) I get $x1a=-9.999894245993346e-07$, $x2a=-999999.999999$

if I fold them back to the original equation, I get result
 $1.0575401665491313e-08$, $7.247924804689582e-08$

With method b) I get $x1b=-1.0000000000001e-06$, $x2b=-1000010.5755125057$

if I fold them back to the original equation, I get result
0.0, **10575.62534720993**

Error on $x2b$ much worse than on $x2a$

- $x2a$ much better than $x2b$ because I avoid the subtraction
of two similar numbers

$b=1000.0$

$\text{np.sqrt}(b * b - 4 * a * c)=999.999999998$

Python. Speed

The computer cannot be infinitely fast


10^6 operations \rightarrow \sim seconds on laptop

10^7 operations \rightarrow minutes to hours

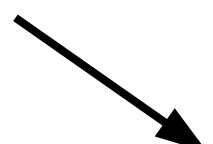
Try to estimate the computational cost before you start coding

1. if $\leq 10^7$ operations,
you can make it

2. if $> 10^7$ operations,
things get harder



LOOK FOR A
SMART TRICK
(e.g. histogram
with indexes)



Go for a
super-computer
(e.g. CINECA)

Some simple smart
tricks can often
save your life

Python. Speed

EXERCISE :

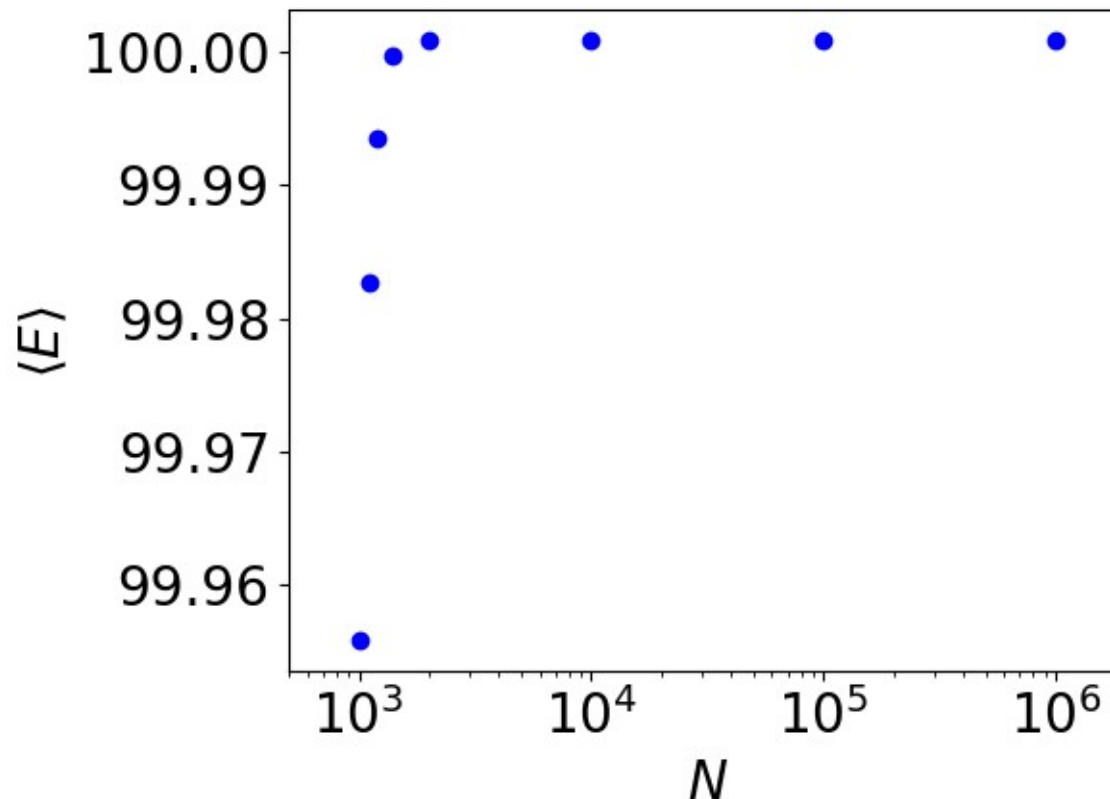
Consider for example the quantum simple harmonic oscillator with energy levels $E_n = \hbar \omega (n + \frac{1}{2})$, where $n = 0, 1, 2, \dots, \infty$. According to Boltzmann and Gibbs, the average energy of a simple harmonic oscillator at temperature T is

$$\langle E \rangle = \frac{1}{Z} \sum_{n=0}^{\infty} E_n \exp(-\beta E_n), \quad (9)$$

where $\beta = 1/(k_B T)$, with k_B being the Boltzmann constant and $Z = \sum_{n=0}^{\infty} \exp(-\beta E_n)$. Suppose we want to calculate the value of $\langle E \rangle$ for $k_B T = 100$. The worst term from the computational point of view is the sum. Let's consider first $n = 0, 1, \dots, 10^6$. Let's work in units where $\hbar = \omega = 1$.

Python. Speed

- Initialize all the constants at the beginning of the script. It will be easier to change them.
- Minimize the number of loops.
- The exponential term is the same for both Z and E. Calculate it only once per each term of the loop.



Python. Speed

Calculate the product of two matrices

EXERCISE:

Let's calculate the product between two $N \times N$ matrices. How long it takes to do the product if $N = 100$? And if $N = 1000$?

Python. Speed

Package time to calculate
time spent
in a given part of the code:
good to **profile** your code

Three for loops nested
2 operations: + and *
The cost is $2 N \times N \times N$

Function to calculate
matrix product in python

If $N=1000$

My function: ~600 s

Python function: ~0.7 s

```
import time
import numpy as np

N=1000
A=np.zeros([N,N],float)
B=np.zeros([N,N],float)
C=np.zeros([N,N],float)

A[:,:]=1.0
B[:,:]=2.0

start = time.time()
for i in range(N):
    for j in range(N):
        for k in range(N):
            C[i,j]+=A[i,k]*B[k,j]
end = time.time()
print(end-start)

start = time.time()
D=np.dot(A,B)
end = time.time()
print(end-start)

print(C[17,13],D[17,13])
```

Python. Additional remarks

- If you want a variable to be a float, please define it as a float
e.g. try with **python2**

```
x=10.0
a1=x**(3/2)
a2=x**(3./2.)
a3=x**1.5
print(a1,a2,a3)
(10.0, 31.622776601683793, 31.622776601683793)
```

Note: $a3=x^{1.5}$ is better because you avoid calculating the division

- Use scientific exponential notation instead of powers.
- Try to minimize the usage of arrays.
It will save RAM memory and it will make easier to code.
- Use stack overflow <https://stackoverflow.com/> or similar internet resources (start with a google search) to understand the errors you get from your code and to find better functions for the problem you want to tackle. **You cannot learn all the python by heart.**
- Try to minimize nested loops, because they really slow down your code.
Sometimes, a single function of python can do the same without any loop.
Please, look on the online python manuals and on slack overflow to find the best function for your case.

Python. Additional remarks

- **Choose your units of measure** to make it easier for the computer to do the calculations.
If you want to calculate the mass of a galaxy cluster, it might be not-so-smart to use grams: you end up with $\sim 10^{46}$ g. Rounding errors might dominate your result.
- **Check units of measure** 100 of times in your code. Bugs hide very well in there.
- If some constants are used many times to a certain power or in a certain combination, **define a new constant that calculates this combination** just once.
It will save time and accuracy.

Gravitational-wave related example

$G3c5 = G^{*3} / c^{*5}$