Problem Set 1

Mauricio Sevilla

2020

We have been working a lot on the understanding of some algorithms and data types of `python`, so this problems are designed to reinforce the concepts we have been working on.

1 Python Basics

All the things we have been using until now, are programed in such a way it does not require going deeply on the language, so we are still working at a basic - intermediate level, but as we are planning to go forward soon and start developing more complex codes and models, we need to be sure we have covered all the basics first.

All the problems are going to be described step by step, so that you will be able to see some important things while you solve them.

I personally do not like when learning to program the tasks are just small codes with no meaning, so I tried to select the problems so that we can get some useful things from each one of them.

`for`

The `for` statement in python needs an *iterable* structure, it means that it needs *something* with positions such as lists, arrays or even strings, and we use a variable that takes all the values included on the *iterable* one at a time, for instance

[1]:
```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

[2]:
```
for i in ['a','b',[1,2],{1:'Test'}]:
    print(i)
```

```
a
b
[1, 2]
{1: 'Test'}
```

[3]:
```
for i in '10':
    print(i)
```

```
1
0
```

and we may calculate something on each one of the iterations we do. Let us use this to calculate π.

1.1 Pi Calculation The Leibniz

formula to calculate π goes as follows,

$$\pi = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)}$$

Create a function that calculates π as follows: - Initialize a variable named as you want where you are going to store your result. - You have to truncate the summation, so create another variable N with the largest value of k on the summation you want - Use a `for` loop to do the operation for each k saving the result on your variable. - print out the final result. - Plot how π vs k to see how it goes to the real value using as a reference the value saved on `numpy` as `numpy.pi`.

hint: To see the reference, you can use the function `axhline` of `matplotlib.pylab`.

Test Here you have the results for N iterations

N	π
10	3.0418396189294032
100	3.1315929035585537
1000	3.140592653839794
10000	3.1414926535900345
100000	3.1415826535897198
1000000	3.1415916535897743

`if`

`python` has these `boolean` variables, which basically are `True` or `False`, so let us use them to do a comparison of values inside a `for` loop.

[16]:
```
a=1
b=1.0
print(type(a),type(b))
if (a==b):
    print("Comparison between ints and floats")
if (type(a)==type(b)):
    print("Comparison between types")
```

```
<class 'int'> <class 'float'>
Comparison between ints and floats
```

[17]:
```
a=1
b=2
if (a==b):
    print("Numbers are the same")
else:
    print("Numbers are different")

a=1
b=2
if (a!=b):
    print("1st condition evaluated")
if (a<b):
    print("2nd condition evaluated")

a=1
b=2
if (a!=b):
    print("1st condition evaluated")
elif (a<b):
    print("2nd condition evaluated if 1st is not true")
```

```
Numbers are different
1st condition evaluated
2nd condition evaluated
1st condition evaluated
```

Inside the `for` loops you can add some conditions using the `if` statements.

1.2 Standard map

Let us consider the following map,

$$p_{n+1} = p_n + K\sin(\theta_n) \tag{1}$$
$$\theta_{n+1} = \theta_n + p_{n+1} \mod 2\pi \tag{2}$$

As the variable θ is an angle, one should expect to have it bounded. So

- Import the library `math` to have the sin function and use $K = 1.5$
- Consider the initial conditions $\theta_0 = 0$ and $p_0 = 1$.
- Construct a loop that runs N times.
- If $\theta > 2\pi$ then use the operator `%` (Module) to make the map bounded.
- Use the value of π you just calculated.

Test Here you have the results for N iterations

N	θ	p
10	1.0	1.0
100	2.2622064772118446	3.2622064772118446
1000	2.081724086075396	5.343930563287241
10000	0.8710465864908559	6.214977149778097
100000	0.7688136640385472	0.7006075066370956
1000000	1.735836986591558	2.4364444932286533

1.3 Standard map with extended angle coordinates

Consider the Standard map we have already studied a little,

$$
\begin{aligned}
p_{n+1} =& p_n + K\sin(\theta_n) && (3)\\
\theta_{n+1} =& \theta_n + p_{n+1} && (4)
\end{aligned}
$$

With the difference now θ is not periodic anymore.

A quantity that one can study from here is the *diffusion*. The diffusion is always measured from the difference of an evolved coordinate minus the initial conditions. The dependence of the difference (On average) with the step (Or time for continuous evolution) goes generally as a power law if there is any diffusion, where the constant going with the power law is called the diffusion constant D, while the exponent α determines the rate of the diffusion.

$$\left\langle (\theta_0 - \theta_n)^2 \right\rangle = Dn^{\alpha}$$

If the exponent $\alpha = 1$, it is called diffusion, but if $\alpha \neq 1$, it is called anomalous diffusion. if $\alpha > 1$ it is called superdiffusion and $\alpha < 1$ is called subdiffusion.

Show that the standard map with extended angle coordinates, exhibits superdiffusion.

Hint

- Consider a large value of K (Chaotic regime) $K \approx 10$ is good.
- Take a large set of initial conditions (randomly distributed).
- Save the initial conditions.
- Evolve, saving after certain evolutions to do the plot.
- Average the square differences.
- Plot the results.

1.4 Wigner Semicircle Law

There is a very important distribution on random matrix theory, which is called Wigner Semicircle Distribution named after the physicist Eugene Wigner (Nobel prize awarded), also known for his phase space representation of quantum mechanics (I hope we can discuss this later).

This distribution can be found by calculating the distribution of the eigenvalues of symmetric random matrices with Gaussian distributed entries, and that is exactly how we will calculate it.

We will generate 1000 random matrices of size 100×100, so it is a good idea for you to define those values before starting.

- Do a for loop running on the number of matrices (1000 in our case), and then inside that for, generate a Gaussian distributed matrix by using the function `np.random.normal` using $\mu = 0$ and $\sigma = 1$.
- Make the matrix symmetric. (Add it by the transpose.).
  - **Hint:** The transpose of a matrix `M` on python can be calculated as `M.T`
- For each matrix, calculate its eigenvalues, it is easily done with the function `eigen=np.linalg.eigvals(matrix)`
- Save on an array the values of all the eigenvalues of all the matrices, you can use the function append, such as `eigen_vals=np.append(eigen_vals,eigen)`
- Do a histogram of them. (Use 100 bins)

1.5 Maxwell Boltzmann Distribution

This is one of the most important distributions on molecular dynamics, proposed by Maxwell and then by Boltzmann back in 1860 and 1872 (1877) respectively.

Based on the kinetic theory to describe ideal gases, the velocities on each direction distribute as a Gaussian but when combining them we get a different distribution,

$$v = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

One has to take a look to the derivation (Which with the statistical mechanics theory becomes very easy), but on the literature one can find that the velocity distribution is

$$f(v) = 4\pi v^2 \left(\frac{m}{2\pi k_B T}\right)^{3/2} e^{\frac{-mv^2}{2k_B T}}$$

but this result only applies to a 3 dimensional gas and here we will use results from a 2 Dimensional simulation!!!.

If you want to see a small demonstration, the following link has a small gif of the simulation we are studying link

The relationship we may use is

$$f(v) = 2\pi v \left(\frac{m}{2\pi k_B T}\right) e^{\frac{-mv^2}{2k_B T}}$$

On the following link, you may find the data result from the simulation link. There are 3 columns, v_x, v_y and $v = \sqrt{v_x^2 + v_y^2}$

- Collect the data.
- Convince yourself that the distributions of v_x and v_y are Gaussians by doing the histogram.
- To prove that the speeds distribute as we said before, we are going to do a fit of the histogram to the model, so first, define a function you want to use as your model.
  - **Hint:** You do not have to use all those values of $k_B T$ and m, use a parameter for the amplitude and another for the exponent.

- Do the histogram (with 20 bins) of the velocities (third column of the file) but!, save the values. (`histogram = plt. hist ( velocities , bins =20 )`). Then you will have save on histogram the amplitudes and limits of the boxes. To have one value for each box, we take the average

```
x_vals =( histogram [1][1:]+ histogram [1][:-1])/2
frequencies = histogram [0]
```

- Use the function `sciypy.optimize.curve_fit` with the variables `x_vals` and `frequencies`. **Hint:** you can use p0=[1000,1000**2.] as your initial parameters.
- Plot again the histogram of velocities, but also the result from the fit on the same plot.