Kamran Ismayilzade, Sabir Khanlarov, Nadir Shikhli CS-018

Homework Report

The overall goal of the homework is to train and prove scientific analysis abilities and skills using conventional Python methods and several associated modules. The study and work subject is particle diffusion in physics and chemistry based on the random factor. The objective is to trace the behavior of the system, model it and describe it.

First Part. Diffusion of a particle in one dimension.

The process of particle diffusion in practical sciences is represented by a series of the uncontrolled and random movements of the matter molecules in space. In order to understand the process and be able to use computer sciences to trace these random movements, a simple case with a "walker" particle going forward and back in one direction must be simulated.

1. As the position change will occur on only 1 axis(let's say x), the point x=0 can be taken as a starting point of an experiment. The moving forward of a particle will be imitated by adding one to the initial x-axis position and vice versa. The decision factor in choosing the movement direction at each step will be defined by a generated random number and the value of probability of an event. For this experiment both moving forward and backward are **equiprobable**(p = 0.5 for both events). The random number to decide which of these events must occur for this purpose is supposed to be generated between 0 and 1(`random.uniform(0,1)`) - the same boundaries as the value of probability itself has. This way, it is ensured that the probability of this randomly generated number to be below the probability value is indeed p - the coin toss situation is correctly imitated.

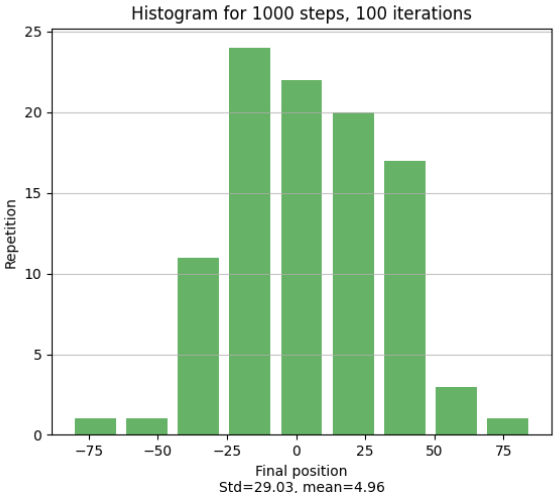
   Now, this operation of generating a random number and making an update to the position value must be performed (the number of steps) times. And there we have the final position of a particle.

2. Since the starting position of the particle is equal to 0 and the probability for the particle to go in a positive direction is equal to the one in the other direction, the value of the final position during several subsequent runs must wrap in a mean value about 0 - the same start position. And this result can be clearly observed after several runs of the program.

   For this experiment the final positions after 1000 steps of the program are recorded and the program is run 100 times. Afterwards, the mean of the program can be observed:

```
Enter number of steps for a drunk walker: 1000
Enter number of iterations: 100
Enter probability value: 0.5
[-8, 0, 64, 18, -74, -30, -54, 36, -20, 36, 0, -24, 20, 0, -12, 58, -50, -22, 10, -6, -54, -6, -22, -8, 20, -6,
-30, 12, 20, 16, -40, -10, 48, 12, -52, -88, -46, 0, 36, -30, 38, -50, 40, 12, -30, -8, -88, -6, -12, 30, 0, -44
, -30, -6, -54, -26, 26, 20, 46, 2, 8, -32, 56, -36, 38, -48, -30, -2, 44, 2, 24, 2, 20, -36, -8, 22, -8, 8, -36
, -36, -20, 22, 40, 54, 28, -4, 50, 50, 38, 16, 60, 10, 32, 58, -24, 4, -8, 38, 22, 30]
Mean 0.22
```

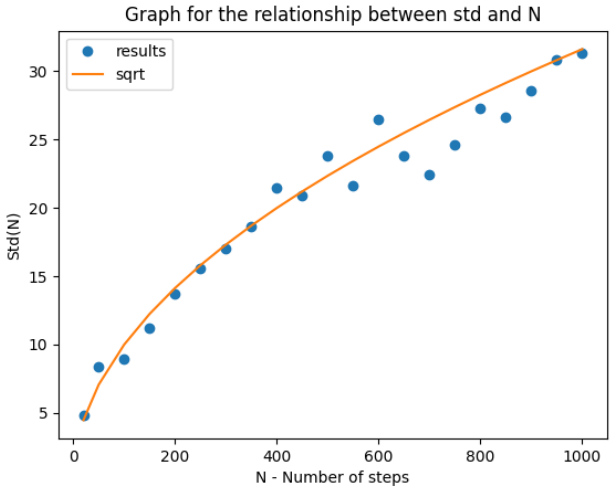
3. Using the Python module **`statistics`**, it is easy to find the statistical parameters for a set of data in the form of a list. The **`matplotlib.pyplot`** allows us to plot the dataset.




```
Enter number of steps for a drunk walker: 1000
Enter number of iterations: 100
Enter probability value: 0.5
Standard deviaton 29.03
Mean 4.96
Percentage of values in first partition: 66
Percentage of values in second partition: 96
Percentage of values in third partition: 100
```

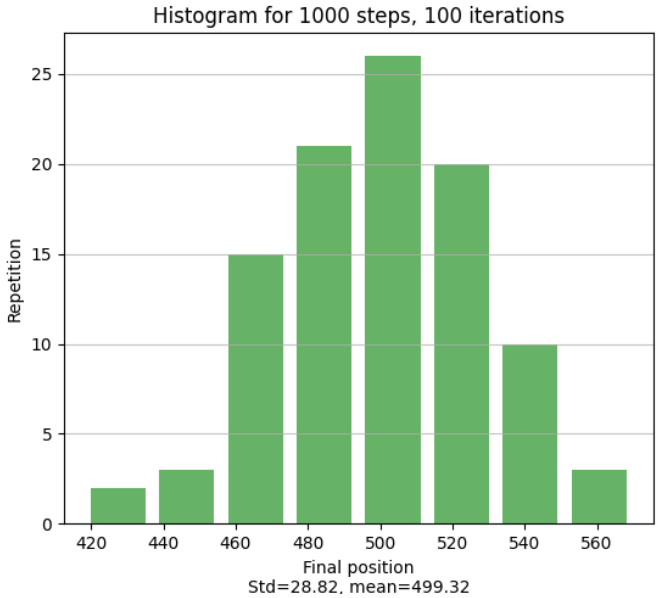
The distribution in this case is called **Gaussian** or **Normal**. The bell shape of the histogram points to it. In addition, in Normal Distribution there is a particular set of percentages for the number of records that fall into so-called 1st, 2nd and 3rd partitions(mean +- std, +-2*std and +-3*std, respectively) range. These are approximately 68%, 95% and 99.7%, which is almost the case here as well.

4. A set of other input data for the number of steps was taken for the analysis. The values for their standard deviations were observed to be similar to the results of sqrt function applied to the same input data. The similarity was then observed on the plot graph.



5. When the probability is inclined more towards step forward, obviously the mean of the final positions with all the datasets will, as expected, move to the right - positive direction as well. This was observed clearly with the p value equal to **0.75**. The std values declined as well, even though with an unclear pattern.

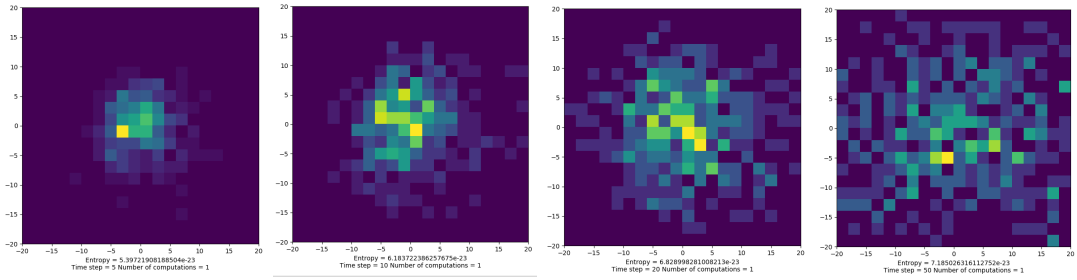
```
Enter number of steps for a drunk walker: 1000
Enter number of iterations: 100
Enter probability value: 0.75
Standard deviaton 28.82
Mean 499.32
Percentage of values in first partition: 68
Percentage of values in second partition: 94
Percentage of values in third partition: 100
```



Second Part. Diffusion of a particle on a plane.

In this case, the goal is to trace the behavior of more than one particle in a space with more than 1 dimension. The lattice of water is represented by a 20x20 sized matrix, where the integer value at each of the cells represents the number of dye particles inside it at a time. Same as in the previous case, the particles' movement is defined by the random event. But in this case, the amount of possible outcomes is greater - the particle has a probability of p to stay inside a cell. While for the rest of the cases, when the dye particle moves, the movements to any available direction are again equiprobable. Since in case of movement, the probability among movements in each available direction must be one and the same, the lattice can be divided into 3 logical areas. One is the corners of the lattice, where the movement can occur in only 2 directions(hence probability is 0.5*(1-p) for each). The same logic for other border areas with 3 open directions and all other cells in between borders where the movement is free in all 4 directions.

1. In order to initiate the time step of the diffusion process, the occurrence of the event may be needed to check for more than one time - first to define whether the particle must remain in the box, and if not, then to define the direction of position change. This check must be done for all of the individual particles simultaneously in each individual cell. The lattice is initiated inside the main body of the program, and **the time_step** function calls for every cell to simulate a one-time step. For this purpose a separate function **`move_rand`** is called for every cell. This function is responsible for identifying in which logical area the cell is situated and providing a separate set of actions for each case. For this purpose, the variable storing the respective value is created. For each present particle in a spell, the dice is rolled. Once a particle has to move, the dice is rolled once more in order to be able to identify the direction of the particle's subsequent movement. This second random value and the "area" variable is passed now to another function - **dir_sel**, which in its term is responsible for control over the direction selection in order to prevent situations in which dice roll combination and box location are in conflict. For this purpose , a set of **If_Else** conditions was implemented.
2. The mechanism of diffusion imitation is explained. Now the practical observations may be applied. The p value is 0.2, which means that with probability 0.8, the particle will move. Here is the observation on the lattice after doing 5, 10, 20 and 50 steps.

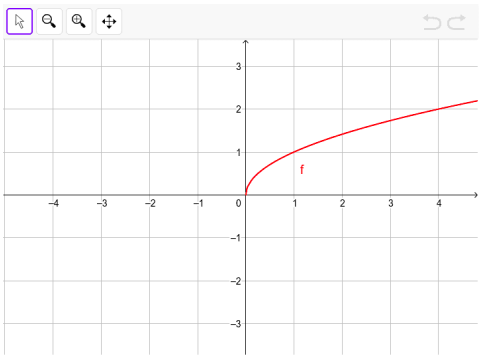


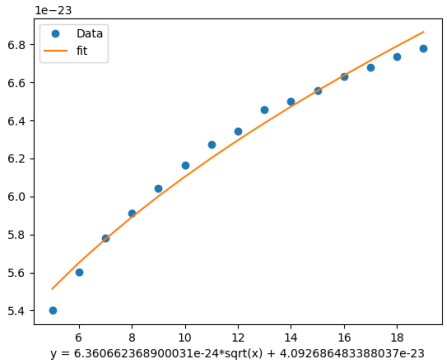
$$S = -k_B \cdot \sum_i P_i \cdot ln(P_i)$$

3. Entropy is one of the key parameters of a system that usually needs to be assessed in physics and chemistry. It is sometimes referred to as “the level of chaos”. After some chaotic movements have been done by all particles(50 time steps, for instance) stay in their last visited box. After that graph is plotted and all necessary data is being collected. In case of multiple Entropy calculations, the lattice should be returned to its initial state. In order to calculate the average value Pi of each box and then gather it for the whole lattice, **EntropyCalc** function was created. It gradually computes Pi for each cell and summarizes received data, after that Boltzman coefficient is applied to obtain the end result.
4. Partially same method applied to investigate N=5,10,20,50. Since we now have to proceed several time step values one by one, **MeanEntropy** function was implemented. As N values suggest we need to hold 4 separate Entropy calculation processes, each of them has to be computed 25 times. Hence we need to compute average entropy from 25 results for 4 procedures.

   After that we can plot a graph which represents dependency between entropy and time step numbers for 4 different cases.

   According to the visual similarity we can guess that the S(N) formula is based on the square root of time_step .





Furthermore we can assume that **S(N) = A * sqrt(timeStepVar) + B**

With help of **PotentialFormula** function and **Scipy.ompimize curve fit** we can calculate A and B coefficients and prove that this formula satisfies provided data.