French-Azerbaijani University

Computer Sciences for Physics and Chemistry

report on homework

Diffusion in 1D and 2D

Group:
CS-016

Author:
Emin Sultanov

December 17, 2019

Contents

1 Introduction

This report looks into the procedure of the working of two programs that helps to calculate the final position after diffusion in a different time and different dimensions. Purpose of this project, the working process, the result will be also discussed in this report.

2 Description

The main purpose of this project is to understand the process of diffusion while calculating the standard deviation of final positions in 1D and entropy in 2D.
The second purpose was to implement needed functions of different libraries of python such as `scipy`, `numpy` and `matplotlib`.

3 Creation Process

3.1 Part 1

3.1.1 Steps of the creation

There are five steps that have to be done in the first part of the homework:

1. Write a function that calculates the final position of the walker as a function of the parameter p, which is the probability of the drunk man to move forward, and the number of steps N.
2. Find the final position by calculating the average of final positions after 100 times of repetition for 1000 steps. Taking into account that we have the probability of moving forward 0.5, which means that the man can go forward and backward with the same probability, we can expect that the **final position will be near to 0 or equal to 0**.
3. Represent the results of the final positions after 100 iterations as a histogram and calculate standard deviation demonstrating that the distribution is Gaussian.
4. Repeat the second and third steps for N equal to 20, 50, 100 and 200 and fit the function for standard deviation depending on N steps.
5. Find the final position after 100 times of iteration for N equal to 20, 50, 100, 200 and 1000 with p=0.75. As for now, the probability of moving forward is bigger than backward we can predict the final position by calculation.
   Knowing that for now we have p=0.75, we can suppose:

$$Final\ position = p \cdot N - (1 - p) \cdot N = 0.75 \cdot N - 0.25 \cdot N = 0.5 \cdot N$$

3.1.2 Implementation

1. For the first step, it was supposed to create a function that finds the final position of the man. In the part of the code below, we can observe the function `position` that takes as input p and N and returns final position and list of steps:

```
def position(p, N):
    '''
    Get the position after N steps knowing the probability to move forward

    p - probability to move forward
    N - number of steps
    '''
    r = np.random.uniform(size=N)       # randomizing probabilities of each step

    position = 0                        #initializing the position
    steps = np.zeros(N)
    for i in range(N):
        if r[i] <= p:
            steps[i] = 1
            position += 1               # 1 meter forward
        else:
            steps[i] = -1
            position -= 1               # 1 meter backward
    return position,steps
```

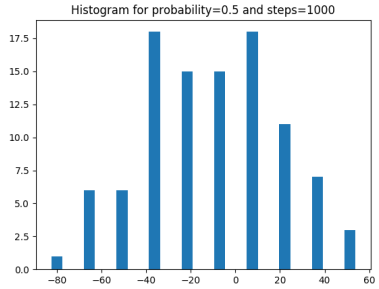
2. The second step is to implement the `position` function 100 times to calculate the average final position. In the code below we can observe this implementation:

```
guess = np.zeros(times)             # list of geussed final positions

# Calculating
for i in range(times):
    guess[i], steps = position(p,N) # getting guessed position and each step after n times of the computation
avg = np.sum(guess)/100             # calculating average of the final positions after n times of the computation
```

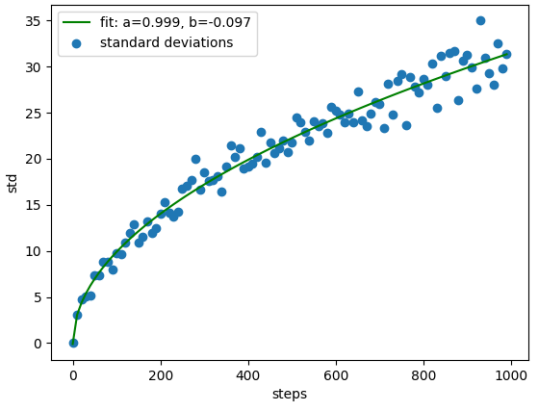
   The obtained result after compiling the code is -3.68 which is close to the expected value. Therefore it is possible to say that the function works correctly for this case.

3. For the third step, the function `scipy.histogram` was used to calculate the histogram and the function `matlplotlib.pyplot.plot` was used to plot it. The image below represents the result:



Observing this diagram it is possible to say in advance that the distribution is normal as this diagram is similar to the Gaussian distribution.
In order to calculate the standard deviation the function `scipy.nanstd` was used and the test for Gaussian distribution was done by Shapiro test by using `scipy.stats.shapiro` function. After calculations, the result of the standard deviation is 30.370670061755305 and Shapiro test gives a positive result, which was predicted previously .

4. In the fourth step, we are using function `guess` with input times, p and N, where we give values 20, 50, 100 and 200 to N to repeat the previous two steps. In order to fit standard deviation as a function of the number of steps N it was supposed to use function `scipy.optimize.curve_fit` and function `my_std` to get different values of standard deviation for different steps.
As `scipy.optimize.curve_fit` requires for one of the inputs the fit function, it was defined to use $a \cdot \sqrt{x} + b$ function, where a and b are the coefficients that fit the function and x is the number of steps, by taking into consideration that $\sigma_{mean} = \frac{\sigma}{\sqrt{N}}$. The achieved results are below:



5. In this step, we are repeating the second and the third steps to calculate the final position but with the value p equals to 0.75. The obtained results are the following:

Steps	Final Obtained Position	Final Expected Position
20	10.04	10
50	25.62	25
100	49.48	50
200	100.78	100
1000	501.46	500

Seeing that the expected and the obtained results are approximately the same, we can say that the initial assumption was correct.

3.2 Part 2

3.2.1 Steps of the creation

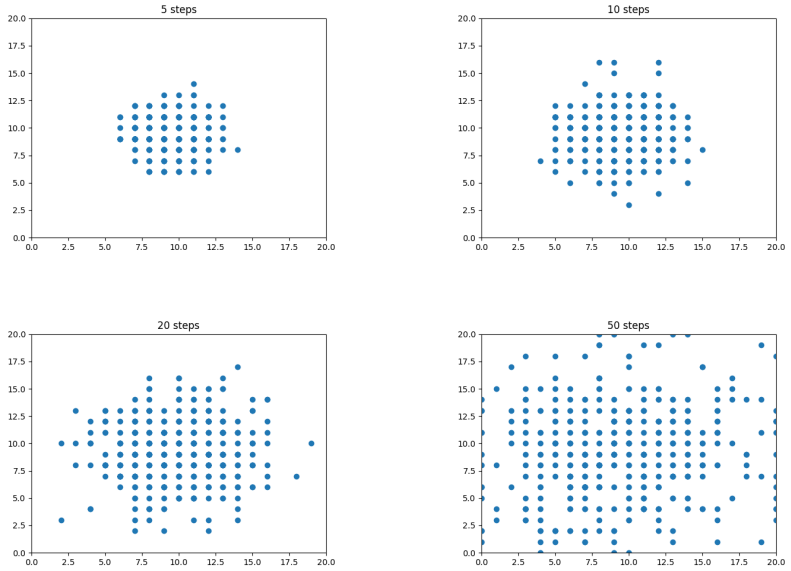
For this part of the homework there are 4 steps of the creation:

1. Write the function that computes the final position of each molecule walker as a function of the parameter p, which is the probability of the molecule to stay at the same position after one step, and the number of steps N.
2. Represent the positions of the molecules in a scattered plot after 5, 10, 20 and 50 time steps.
3. Compute the entropy after 25 computations with N equals to 50.
4. Repeat the third step for N = 5, N = 10, N = 20 and N = 50. Plot S(N) and find a mathematical function that fits the variation of entropy with respect to the number of time steps S(N)

3.2.2 Implementation

1. In this step, the function `move` was created. This function takes as input the parameter p and the number of steps N and returns the list of the final positions of each particle. The first part of this function initializes the starting positions of the particles, which are the 4 central cells with 100 particles in each. The second part of this function modifies the position of each particle N times according to the rule which is defined in the function `getDirection` and knowing that after every movement the particle goes from one cell to another without the possibility to go out of the lattice.
   The function `getDirection` takes as input current position of the particle and the parameter p and returns the direction of the particle movement, taking into account that all possible moving directions are equiprobable and that the particle can stay on the same place.

2. In the second step we implemented the function `move` with p=0.2 and N=5;10;20;50. The result of the scattered positions of the particles are below:



Observing that the particles are moving in all directions we can assume that the function works correctly, as the diffusion is the movement of the particles from an area of high concentration to an area of low concentration.

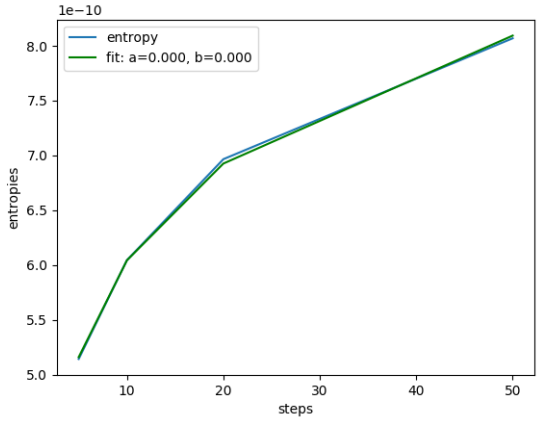
3. The third step is to calculate the entropy after 25 computations with N=50. For this purpose the function `entropyComp` was created which takes as input number of computations, the parameter p and the number of steps N. In order to calculate the entropy after 25 computations it was firstly needed to calculate the average probability of the particle being in each box. Knowing that the average of probabilities is calculating by $P_i = \frac{\frac{\sum number\ of\ particles\ in\ the\ cell}{number\ of\ all\ particles}}{number\ of\ computations}$, we can simplify this as $P_i = \frac{\sum number\ of\ particles\ in\ the\ cell}{number\ of\ all\ particles \cdot number\ of\ computations}$. And the formula of the entropy is

$$S = -k_b \cdot \sum P_i \cdot \ln(P_i)$$

4. The last step is to implement the function `entropyComp` with number of computations equal to 25, p equal to 0.2 and N equal to 5, 10, 20 and 50. The obtained results of the entropy are below:

Steps	Entropy
5	5.15690406e-10
10	6.06794927e-10
20	6.96216957e-10
50	8.08088408e-10

It was also required to find the mathematical function that fits the variation of the entropy. Knowing that one of the step of the entropy calculation is consisting by finding ln (P_i) we can assume that the fit function for the entropy depending on the steps number is $a \cdot \log(x) + b$, where a and b are the coefficients that fit the function and x is the number of steps. The function `scipy.optimize.curve_fit` in order to find the coefficients a and b. the results of this function are below:



4 Notes*

- The full description of the developed scripts is written in the comments in each of two programs.
- Some parts of the programs were tested by comparing the expected results and the obtained ones or by observations of the graphs.
- The results after each compilation of the programs are different as the values of the probabilities are random.