Computer Science for Physics and Chemistry

Homework Project

Elchin Gurbanli
Laman Mammadova

22/12/2021

Part 1. Diffusion in one dimension: the drunk walker.

Libraries in use:

```
import numpy as np
import scipy.stats as sc
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.optimize import curve_fit
```

1.1

We wrote a function that takes the probability *p*, and the number of steps *N* as parameters and calculates the final position of the drunk walker with the probability of him/her taking a step forward equaling *p*, and backward — *(p-1)*.
p ranges between 0 and 1 inclusively. We used `numpy.random.choice` with non-uniform probability to get the result.

```
def calc_final_position(N, p):
  final_position = 0
  for step in range(N):
    final_position += np.random.choice([-1, 1], p = [1 - p, p])
  return final_position
```

1.2

The expected final position for *p = 0.5* is 0, with the walker taking the same number of steps backward and forward. We ran our function 100 times with the number of steps equaling 1000 steps to check it:

```
n_experiments = 100
final_positions = np.array([])

for n in range(n_experiments):
    final_positions = np.append(final_positions, calc_final_position(1000, 0.5))
print(final_positions)
```

1.3

Then we represented the results in a histogram, calculated the standard deviation (σ_{1000}), and Shapiro p-value to check the normality (Shapiro rejects the normality if it is less than or equal to 0.05)

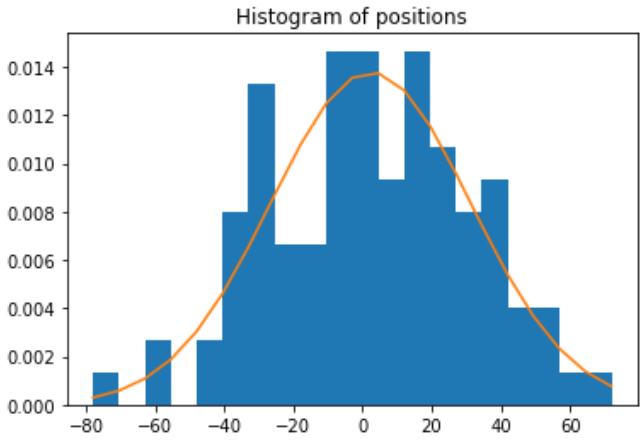
```
bins = plt.hist(final_positions, density=True, bins=20)[1]
mu, std = sc.norm.fit(final_positions)

plt.plot(bins, sc.norm.pdf(bins, mu, std))
plt.title('Histogram of positions')
print("Standard deviation sigma_1000 =", final_positions.std())


print("Shapiro: p_value ", sc.shapiro(final_positions)[1])
```

```
plt.savefig(f"plots/part1_ex3_gaussian.jpg")
```

Output:

Standard deviation sigma_1000 = 28.942315042166204 - Standard Deviation is greater than mean => the values are closer to mean.

Shapiro: p_value 0.8869155645370483 => As the p-value is greater than 0.05 the, we can deduce that the deviation is normal => it's ***Gaussian Distribution***



1.4

The same was done for 20, 50, 100, and 200 steps.

final-positions - a list for storing the final positions for *n_experiments* number of iterations

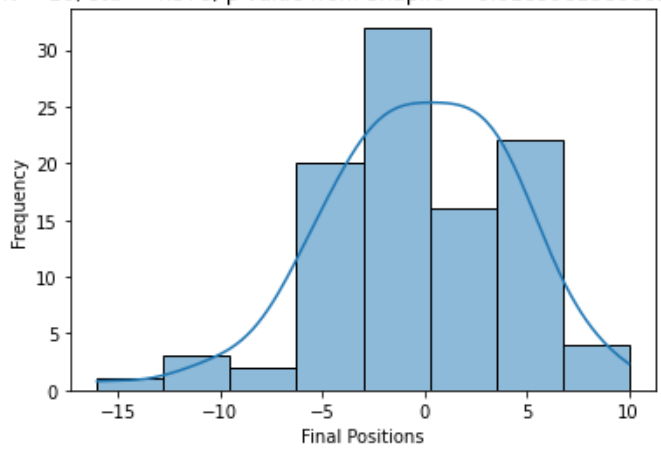
n_steps - a list containing all the number of steps for which the code will be tested

stds - a list containing values of standard deviation

```
final_positions = np.array([])
n_steps = [20, 50, 100, 200]

stds = np.array([])
for steps in n_steps:
  for i in range(n_experiments):
    final_positions = np.append(final_positions, calc_final_position(p = 0.5, N = steps))

  fig, ax = plt.subplots()
  ax = sns.histplot(final_positions, kde=True)
  ax.set_title("N = {}, std = {}, p-value from Shapiro = {}".format(steps,
round(final_positions.std(), 3), sc.shapiro(final_positions)[1]))
  ax.set_xlabel("Final Positions")
  ax.set_ylabel("Frequency")
  plt.savefig(f"plots/part1_ex4_{steps}_steps.jpg")
  plt.show()
  stds = np.append(stds, final_positions.std())
```

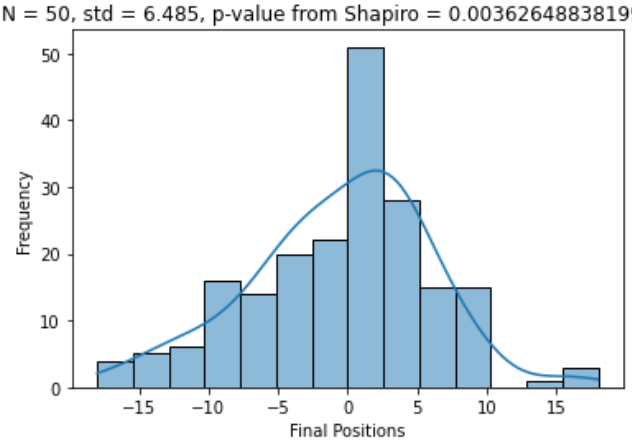
(1) Histogram for 20 steps

N = 20, std = 4.578, p-value from Shapiro = 0.01859823800623417



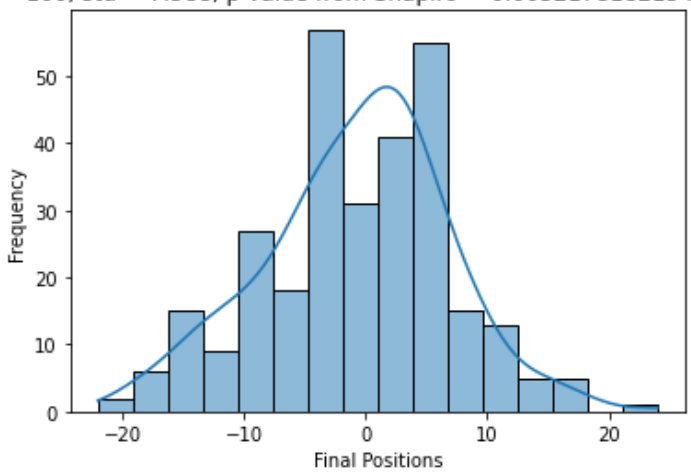
(2) Histogram for 50 steps

N = 50, std = 6.485, p-value from Shapiro = 0.003626488381996751

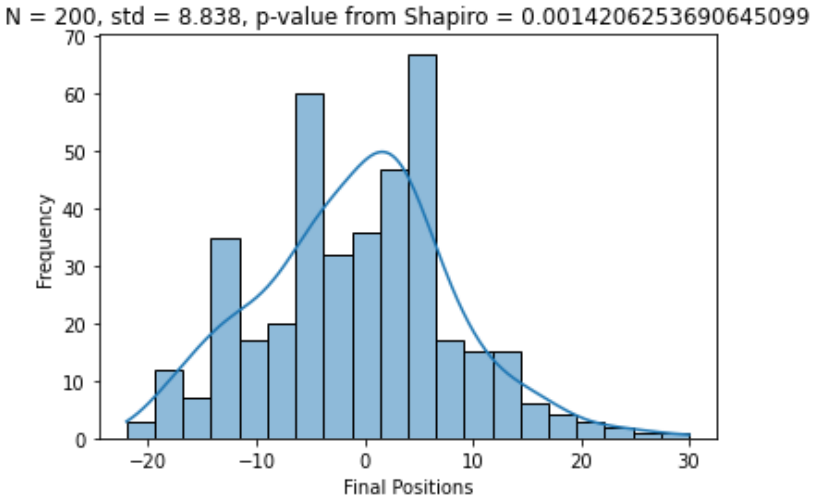


(3) Histogram for 100 steps

N = 100, std = 7.588, p-value from Shapiro = 0.0032178182154893875

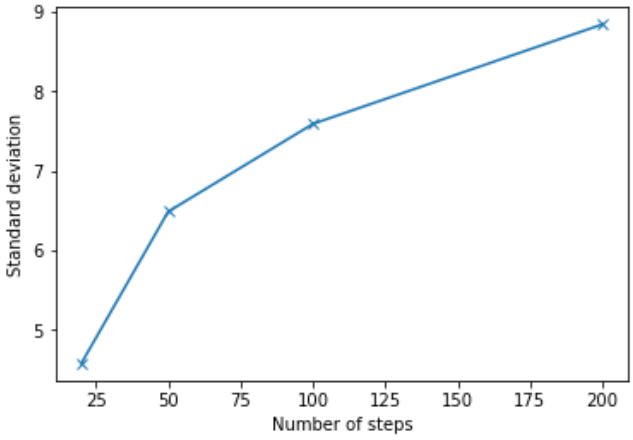


(4) Histogram for 200 steps



Then we fit the standard deviation as the function of the number of steps (σ_N)

```
for N in n_steps:
  for i in range(n_experiments):
    final_positions = np.append(final_positions,[calc_final_position(N, p = 0.5)])
plt.xlabel("Number of steps")
plt.ylabel("Standard deviation")
plt.plot(n_steps, stds, '-x')
plt.savefig(f"plots/part1_ex4_fitted_curve_std.jpg")
plt.show()
```

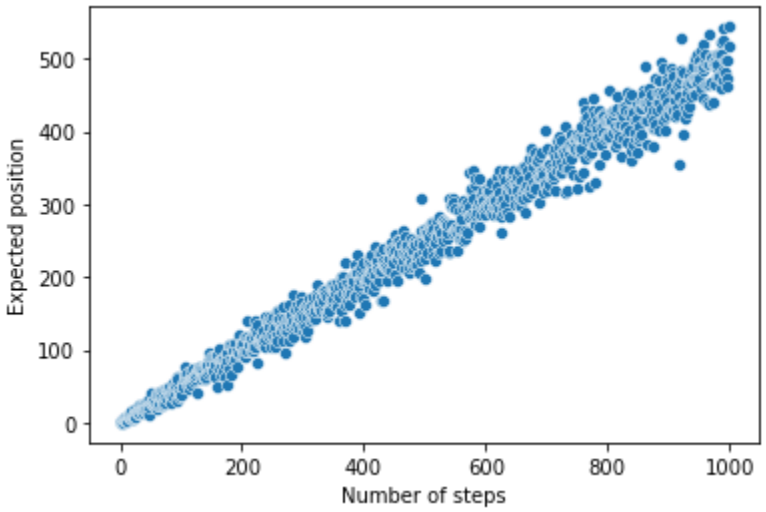


1.5

Now p = 0.75. The expected value is 250 for 1000 steps

```
p = 0.75
N = 1000
steps = np.arange(1000)

final_positions = np.array([])
```

```
for step in steps:
  final_positions = np.append(final_positions, calc_final_position(step, p))
print(final_positions.mean())

sns.scatterplot(x = steps, y = final_positions)
plt.xlabel("Number of steps")
plt.ylabel("Expected position")
```



Part 2. Diffusion in 2 dimensions: diffusion of a dye in water

2.1

According to the rules of particles movement, we wrote a function that computes the final position of each molecule walker as a function of the parameter p and the number of steps *N*

```
def calc_pos(N, p = 0.2):
    '''
    Args:
        N:          Number of steps each molecule moves
        p:          The probability of a molecule to stay in the cell
    Returns:
        mol_pos: Final position of the molecules as an array
    '''
    molecule_position = np.zeros((400, 2))
    molecule_position[0:100] = [9, 9]
    molecule_position[100:200] = [9, 10]
    molecule_position[200:300] = [10, 9]
    molecule_position[300:400] = [10, 10]
    for i in range(molecule_position.shape[0]):
        position = molecule_position[i]
        for j in range(N):
            # at center
            if 19 > position[0] > 0 and 19 > position[1] > 0:
                choices = [[0, 0], [0, -1], [-1, 0], [0, 1], [1, 0]]
                position += np.array(choices[np.random.choice(len(choices), 1,
                                                 p=[p, 0.25 * (1 - p), 0.25 * (1 - p), 0.25 *
(1 - p), 0.25 * (1 - p)])[
                    0]])

            # at upper left corner
            elif position[0] == 0 and position[1] == 00:
                choices = [[0, 0], [0, 1], [1, 0]]
                position += np.array(choices[np.random.choice(len(choices), 1, p=[p, 0.5 * (1
- p), 0.5 * (1 - p)])[0]])

            # at upper boundary
            elif position[0] == 0 and 19 > position[1] > 0:
                choices = [[0, 0], [0, -1], [0, 1], [1, 0]]
                position += np.array(
                    choices[np.random.choice(len(choices), 1, p=[p, 0.33 * (1 - p), 0.33 * (1
- p), 0.33 * (1 - p)])[0]])

            # at upper right corner
            elif position[0] == 0 and position[1] == 19:
                choices = [[0, 0], [0, -1], [1, 0]]
                position += np.array(choices[np.random.choice(len(choices), 1, p=[p, 0.5 * (1
- p), 0.5 * (1 - p)])[0]])

            # at left boundary
            elif 19 > position[0] > 0 and position[1] == 0:
                choices = [[0, 0], [-1, 0], [0, 1], [1, 0]]
                position += np.array(
                    choices[np.random.choice(len(choices), 1, p=[p, 0.33 * (1 - p), 0.33 * (1
- p), 0.33 * (1 - p)])[0]])

            # at right boundary
            elif 19 > position[0] > 0 and position[1] == 19:
```

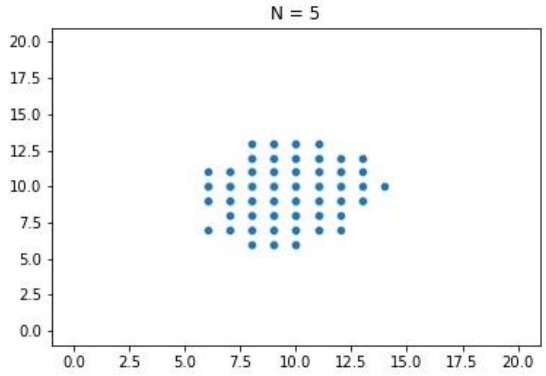
```
                choices = [[0, 0], [0, -1], [-1, 0], [1, 0]]
                position += np.array(
                    choices[np.random.choice(len(choices), 1, p=[p, 0.33 * (1 - p), 0.33 * (1
- p), 0.33 * (1 - p)])[0]])

            # at bottom boundary
            elif position[0] == 19 and 19 > position[1] > 0:
                choices = [[0, 0], [0, -1], [-1, 0], [0, 1]]
                position += np.array(
                    choices[np.random.choice(len(choices), 1, p=[p, 0.33 * (1 - p), 0.33 * (1
- p), 0.33 * (1 - p)])[0]])

            # at bottom left corner
            elif position[0] == 19 and position[1] == 0:
                choices = [[0, 0], [-1, 0], [0, 1]]
                position += np.array(choices[np.random.choice(len(choices), 1, p=[p, 0.5 * (1
- p), 0.5 * (1 - p)])[0]])

            # at bottom right corner
            elif position[0] == 19 and position[1] == 19:
                choices = [[0, 0], [0, -1], [-1, 0]]
                position += np.array(choices[np.random.choice(len(choices), 1, p=[p, 0.5 * (1
- p), 0.5 * (1 - p)])[0]])
        molecule_position[i] = position

    return molecule_position
```

2.2

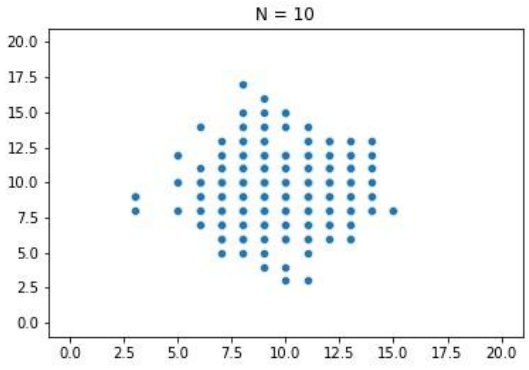
With the fixed value of p equaling to 0.2 we represented the positions of the molecules in a scattered plot after 5, 10, 20 and 50 time steps.

```
steps = [5, 10, 20, 50]
for _, step in enumerate(steps):
    output = calc_pos(N = step)
    f, ax = plt.subplots()
    ax = sns.scatterplot(x = output[:, 0], y = output[:, 1])
    ax.set_title("N = " + str(step))
    ax.set_xlim(left =- 1, right = 21)
    ax.set_ylim(bottom =- 1, top = 21)

    plt.savefig(f"part2_plots/ex2_{step}_step.jpg")
    plt.show()
```

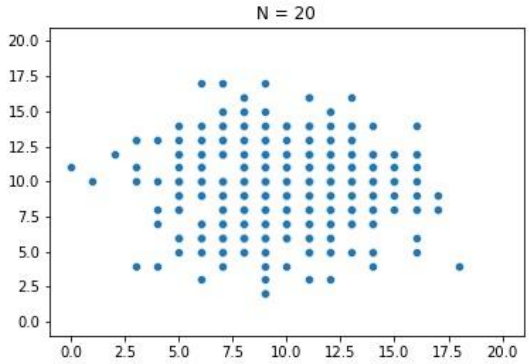
1) Scatter plots for 5 steps



2) Scatter plots for 10 steps



3) Scatter plots for 20 steps



4) Scatter plots for 50 steps