Random Walk

Mahyar Albalan

403201223

1. Prove equation 7

I should Prove:

$$\sigma^2 =< x^2 > - < x >^2 = 4pqlt/\tau$$

for each step we know:

$$< a >= p(l) + q(-l) = l(p-q) \Rightarrow < x >= l/\tau(p-q)t$$

Now lets calculate $< a^2 >$:

$$< a^2 >= p(l)^2 + q(-l)^2 = (p+q)l^2 = l^2$$

from text we had:

$$x_{(t)} = x_{(t-\tau)} + a \rightarrow x^2_{(t)} = x^2_{(t-\tau)} + 2x_{(t-\tau)}a + a^2$$

$$\Rightarrow < x^2_{(t)} >=< x^2_{(t-\tau)} > +2 < ax_{(t-\tau)} > + < a >^2$$

$\Rightarrow < x^2_{(t)} >=< x^2_{(0)} > +2l^2/\tau((t-\tau) + (t-2\tau) + ... + (t-(N-1)\tau)) + t/\tau l^2$

which $N = t/\tau$ and $< x^2_{(0)} >= 0$ so after a bit of simpilification we get:

$$< x^2_{(t)} >= l^2(p-q)^2t^2/\tau^2 - l^2t/\tau(p^2+q^2-2pq-1)$$

$$1 = (p+q)^2 \Rightarrow < x^2_{(t)} >= l^2(p-q)^2t^2/\tau^2 - l^2t/\tau(p^2+q^2-2pq-(p+q)^2)$$

$$\Rightarrow < x^2_{(t)} >=< x_{(t)} >^2 - l^2t/\tau(p^2+q^2-2pq-(p+q)^2)$$

$$\Rightarrow < x^2_{(t)} > - < x_{(t)} >^2 = -l^2t/\tau(-4pq) = 4pql^2t/\tau$$

Proved.

2.1D Random Walk

To solve this problem, I defined two functions:

- `random_walk`: Generates a single random walk and computes its position at each step.
- `calculations`: Simulates multiple random walks and calculates the mean position ($\langle x \rangle$) and standard deviation (σ) at each timestep.

For $P = 0.75$, the results are as follows:

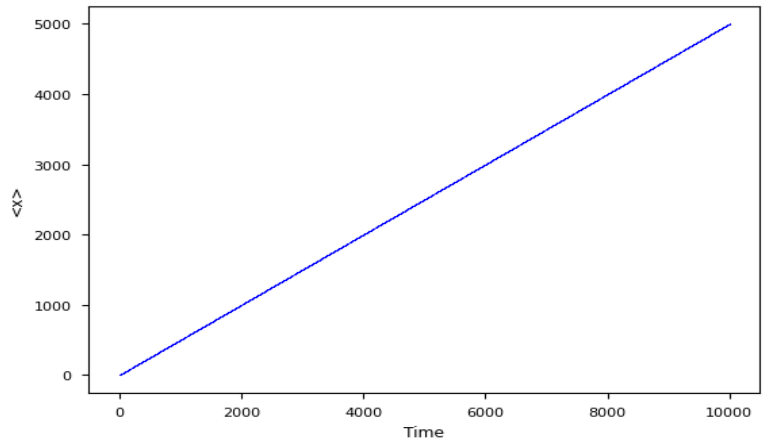

Figure 1: <x> versus time for $P = 0.75$

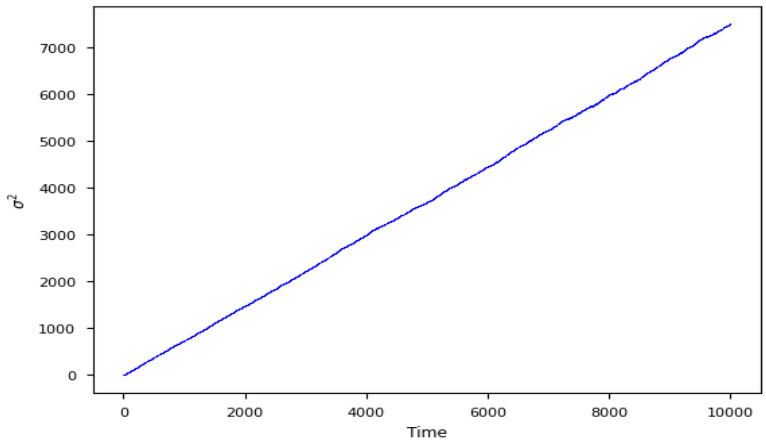

Figure 2: σ^2 versus time for $P = 0.75$

After fitting a **LinearRegression** model to each dataset:

- For $\langle x \rangle$, the slope is **0.4999**, which closely matches the theoretical expectation of **0.5**.
- For σ, the slope is **0.7516**, again aligning well with the theoretical value of **0.75**.

Now, let's repeat the process for $P = 0.5$. The results are as follows:

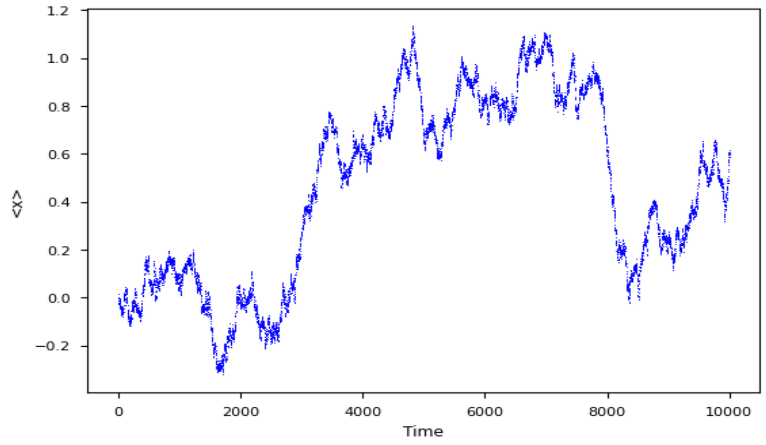

Figure 3: <x> versus time for $P = 0.5$

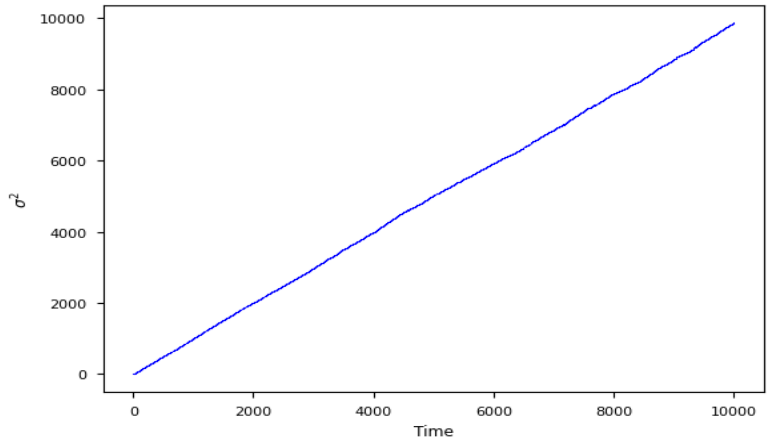

Figure 4: σ^2 versus time for $P = 0.5$

After fitting a **LinearRegression** model to each dataset:

- For $\langle x \rangle$, the slope is $6.67 * 10^{-05}$, which closely matches the theoretical expectation of **0**.
- For σ, the slope is **0.9784**, again aligning well with the theoretical value of **1**.

3.Random Walk with Boundry Conditions

For solving this question, I defined another function:

- `random_walk_lifespan` function: in this function, you can set the boundary condition, the total number of walkers, and the total steps for each walker. This function tracks which boundary a walker hits and measures its lifespan.

The results are as follows:

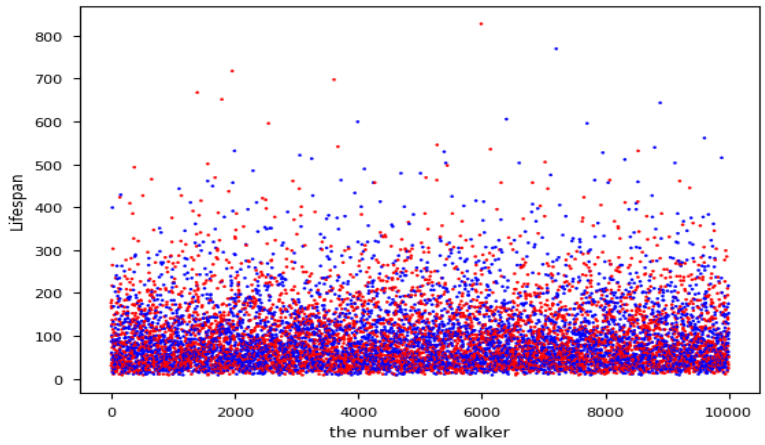

Figure 5: lifespan versus the number of Walkers,(blue for left hits and red for right hits)

average lifespan of each walker is around 100,which confirms the 1/100 relation for number of steps which was mentioned in the text.(Check the Code For more details.)

Here are the frequency plots of hits to each boundary for better understanding:

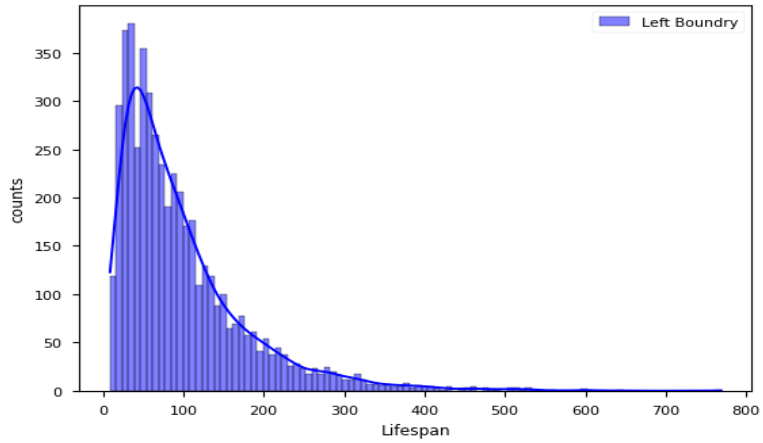

Figure 6: frequency versus lifespan(left boundry hits)

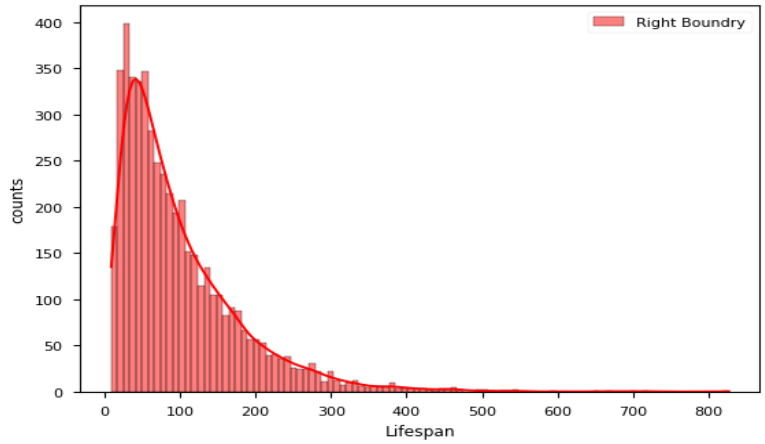

Figure 7: frequency versus lifespan(right boundry hits)

The distribution is somewhat as expected (skewed data with a right tail because there are some walkers with lifespans far beyond the average lifespan).

With a slight modification of the previous process, for each starting position, I use 10,000 walkers, each taking 10,000 steps. I evaluate the probability of hitting the right boundary for each starting position and the average lifespan of the walkers. The results are as follows:

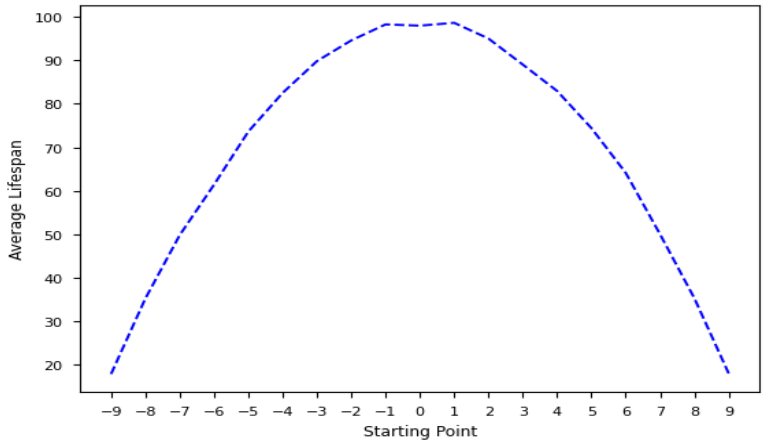

Figure 8: average lifespan versus Starting Position

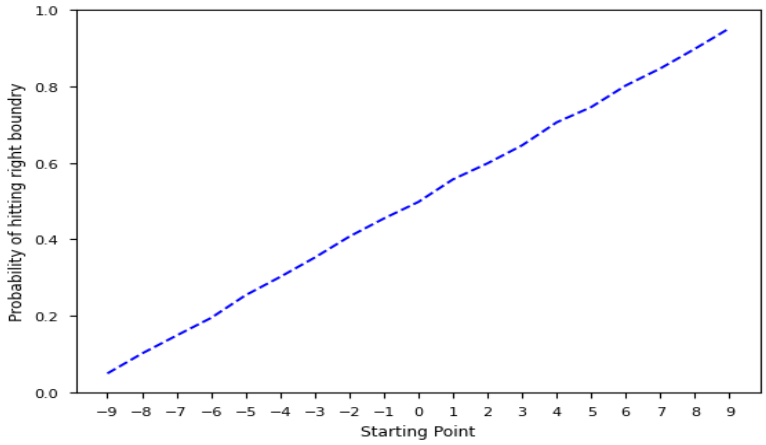

Figure 9: Probability of hitting the right boundry versus Starting Position

As expected, the probability plot is linear, and the average lifespan is quadratic with its peak at 0 and lifespan around 100, which was calculated before.

4.Enumeration

First, I create a table for $P = 0.5$, similar to the one in the text. For the first 30 steps, starting from 0, the probability propagation is as follows:

Step 0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.25	0.0	0.5	0.0	0.25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.125	0.0	0.375	0.0	0.375	0.0	0.125	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 4	0.0	0.0	0.0	0.0	0.0	0.0	0.062	0.0	0.25	0.0	0.375	0.0	0.25	0.0	0.062	0.0	0.0	0.0	0.0	0.0	0.0
Step 5	0.0	0.0	0.0	0.0	0.0	0.031	0.0	0.156	0.0	0.312	0.0	0.312	0.0	0.156	0.0	0.031	0.0	0.0	0.0	0.0	0.0
Step 6	0.0	0.0	0.0	0.0	0.016	0.0	0.094	0.0	0.234	0.0	0.312	0.0	0.234	0.0	0.094	0.0	0.016	0.0	0.0	0.0	0.0
Step 7	0.0	0.0	0.0	0.008	0.0	0.055	0.0	0.164	0.0	0.274	0.0	0.274	0.0	0.164	0.0	0.055	0.0	0.008	0.0	0.0	0.0
Step 8	0.0	0.0	0.004	0.0	0.031	0.0	0.109	0.0	0.219	0.0	0.274	0.0	0.219	0.0	0.109	0.0	0.031	0.0	0.004	0.0	0.0
Step 9	0.0	0.002	0.0	0.018	0.0	0.07	0.0	0.164	0.0	0.246	0.0	0.246	0.0	0.164	0.0	0.07	0.0	0.018	0.0	0.002	0.0
Step 10	0.001	0.0	0.01	0.0	0.044	0.0	0.117	0.0	0.205	0.0	0.246	0.0	0.205	0.0	0.117	0.0	0.044	0.0	0.01	0.0	0.001
Step 11	0.001	0.005	0.0	0.027	0.0	0.081	0.0	0.161	0.0	0.226	0.0	0.226	0.0	0.161	0.0	0.081	0.0	0.027	0.0	0.005	0.001
Step 12	0.003	0.0	0.016	0.0	0.054	0.0	0.121	0.0	0.193	0.0	0.226	0.0	0.193	0.0	0.121	0.0	0.054	0.0	0.016	0.0	0.003
Step 13	0.003	0.008	0.0	0.035	0.0	0.087	0.0	0.157	0.0	0.21	0.0	0.21	0.0	0.157	0.0	0.087	0.0	0.035	0.0	0.008	0.003
Step 14	0.007	0.0	0.021	0.0	0.061	0.0	0.122	0.0	0.183	0.0	0.21	0.0	0.183	0.0	0.122	0.0	0.061	0.0	0.021	0.0	0.007
Step 15	0.007	0.011	0.0	0.041	0.0	0.092	0.0	0.153	0.0	0.196	0.0	0.196	0.0	0.153	0.0	0.092	0.0	0.041	0.0	0.011	0.007
Step 16	0.013	0.0	0.026	0.0	0.066	0.0	0.122	0.0	0.174	0.0	0.196	0.0	0.174	0.0	0.122	0.0	0.066	0.0	0.026	0.0	0.013
Step 17	0.013	0.013	0.0	0.046	0.0	0.094	0.0	0.148	0.0	0.186	0.0	0.186	0.0	0.148	0.0	0.094	0.0	0.046	0.0	0.013	0.013
Step 18	0.019	0.0	0.03	0.0	0.07	0.0	0.121	0.0	0.167	0.0	0.186	0.0	0.167	0.0	0.121	0.0	0.07	0.0	0.03	0.0	0.019
Step 19	0.019	0.015	0.0	0.05	0.0	0.096	0.0	0.144	0.0	0.176	0.0	0.176	0.0	0.144	0.0	0.096	0.0	0.05	0.0	0.015	0.019
Step 20	0.027	0.0	0.032	0.0	0.073	0.0	0.12	0.0	0.16	0.0	0.176	0.0	0.16	0.0	0.12	0.0	0.073	0.0	0.032	0.0	0.027
Step 21	0.027	0.016	0.0	0.053	0.0	0.096	0.0	0.14	0.0	0.168	0.0	0.168	0.0	0.14	0.0	0.096	0.0	0.053	0.0	0.016	0.027
Step 22	0.035	0.0	0.034	0.0	0.074	0.0	0.118	0.0	0.154	0.0	0.168	0.0	0.154	0.0	0.118	0.0	0.074	0.0	0.034	0.0	0.035
Step 23	0.035	0.017	0.0	0.054	0.0	0.096	0.0	0.136	0.0	0.161	0.0	0.161	0.0	0.136	0.0	0.096	0.0	0.054	0.0	0.017	0.035
Step 24	0.043	0.0	0.036	0.0	0.075	0.0	0.116	0.0	0.149	0.0	0.161	0.0	0.149	0.0	0.116	0.0	0.075	0.0	0.036	0.0	0.043
Step 25	0.043	0.018	0.0	0.056	0.0	0.096	0.0	0.133	0.0	0.155	0.0	0.155	0.0	0.133	0.0	0.096	0.0	0.056	0.0	0.018	0.043
Step 26	0.052	0.0	0.037	0.0	0.076	0.0	0.114	0.0	0.144	0.0	0.155	0.0	0.144	0.0	0.114	0.0	0.076	0.0	0.037	0.0	0.052
Step 27	0.052	0.018	0.0	0.056	0.0	0.095	0.0	0.129	0.0	0.149	0.0	0.149	0.0	0.129	0.0	0.095	0.0	0.056	0.0	0.018	0.052
Step 28	0.061	0.0	0.037	0.0	0.076	0.0	0.112	0.0	0.139	0.0	0.149	0.0	0.139	0.0	0.112	0.0	0.076	0.0	0.037	0.0	0.061
Step 29	0.061	0.019	0.0	0.056	0.0	0.094	0.0	0.126	0.0	0.144	0.0	0.144	0.0	0.126	0.0	0.094	0.0	0.056	0.0	0.019	0.061

Figure 10: Probability of being at each position at each moment for P=0.5 .

For making this table, I defined a function, `probability_table`, which calculates the probability of being at each position at each step based on the walker's starting position. The probability of hitting one of the boundaries versus time is as follows:

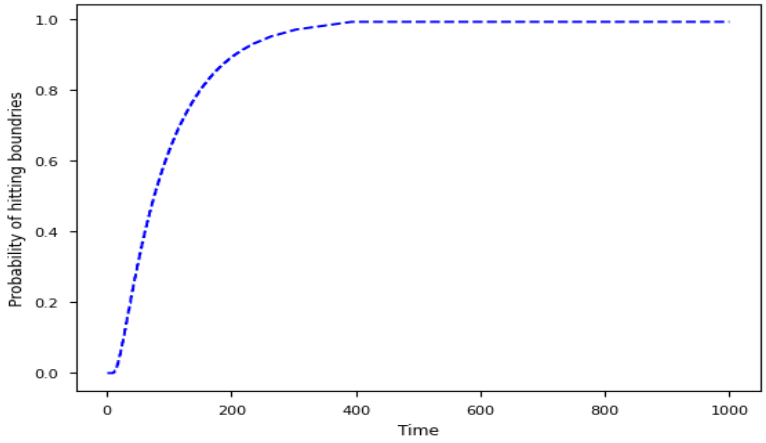

Figure 11: Probability of hitting boundries at each time.

As expected, for large enough times the probability is approximately 1. The average lifespan can be calculated if we find how many walkers hit the boundaries at each time step. This can be easily computed because we have the total hits at each step. By subtracting the probability of hitting at each step from the probability of hitting at the previous step, we obtain the probability of hitting the boundaries only at that given step (if this is unclear, check the code: I simply calculate how many walkers "died" at each step. We have the total deaths at each step, and by subtracting the total deaths at each step from the previous step, we get the number of walkers that died at that step and not before).

The calculated **average lifespan is 96.37**, which is very close to what we achieve by simulation: 99.03. The average lifespan and the probability of hitting the right boundary plots are as follows:

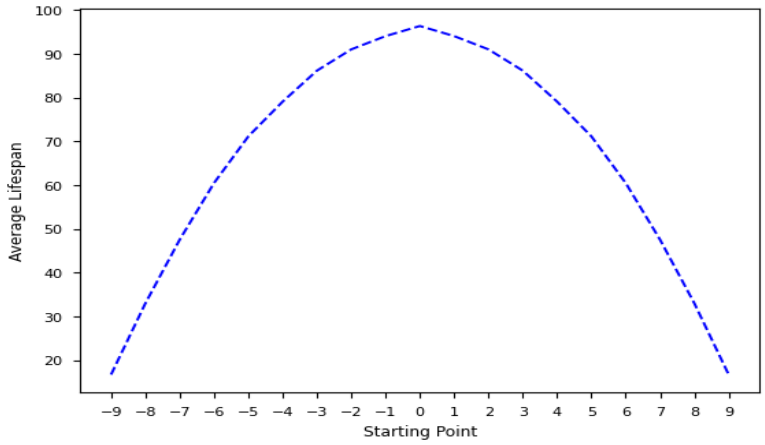

Figure 12: Average Lifespan at each starting position.

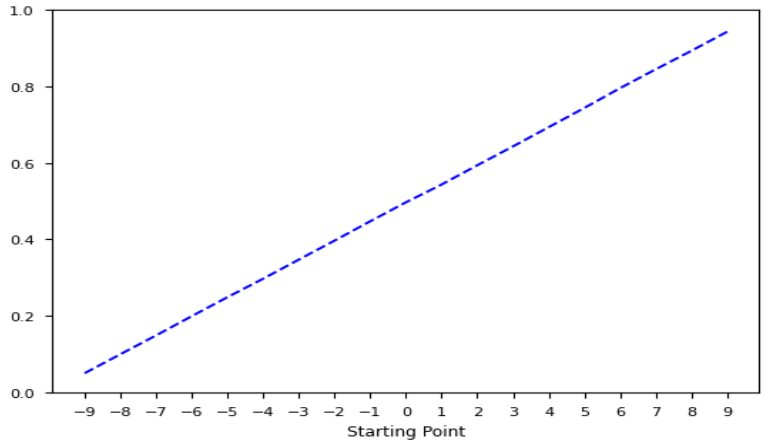

Figure 13: Probability of hitting the right boundary at different starting Positions.

As you can see, the plots match the simulation expectations.

5.2D Random Walk

This problem is very similar to the 1D random walk with a slight modification of previouse functions we can simulate this proccess, I defined the function `random_walk2D`, which saves the walker's position at each step (for more details, check the code). Here are two walkers' trajectories with different total steps:

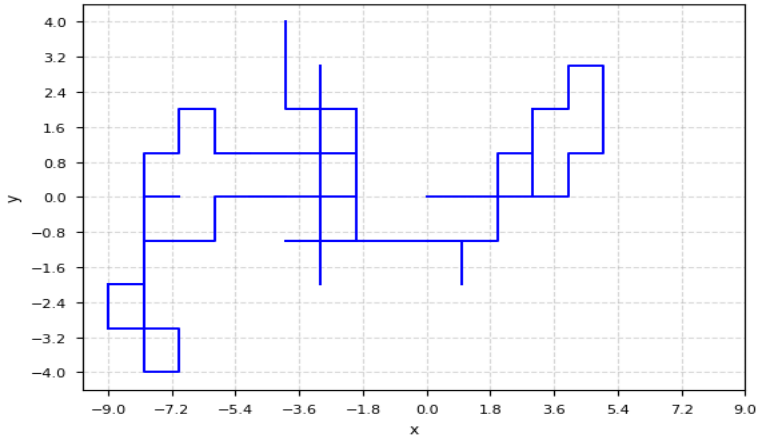

Figure 14: 2D random walk for 100 steps.

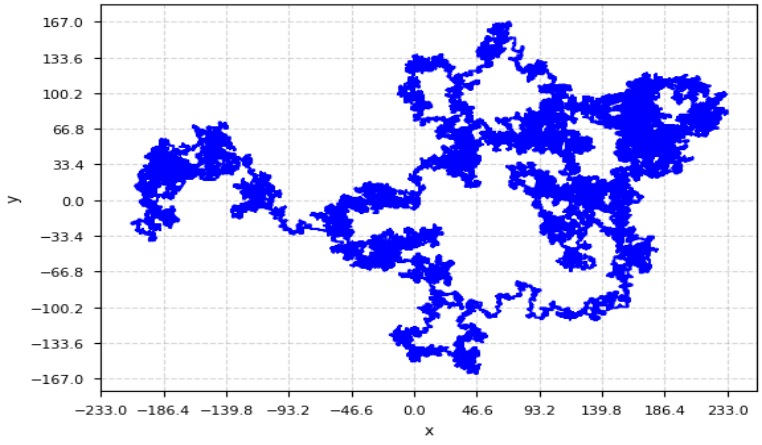

Figure 15: 2D random walk for 100,000 steps.

Now, let's plot r^2 versus time to check if the slope matches the theoretical expectation of $m = dDt = 1$. (Note: In the text, the formula is incorrect because it treats the 2D random walk as simply two independent 1D random walks added together. However, this is not accurate since, in a 2D random walk, the walker either moves up, down, left, or right at each step. This means that if it moves in one direction, the other direction remains unchanged, which is not considered in the text's proof of the equation.
The 2D random walk here consists of two 1D random walks with $p = 0.25$ to move right (up), $p = 0.25$ to move left (down), and $p = 0.5$ to remain stationary. For better understanding, you can consult mathematical textbooks on random walks.)
Here is the average squared distance at each step, calculated for 1000 walkers with 1000 steps:

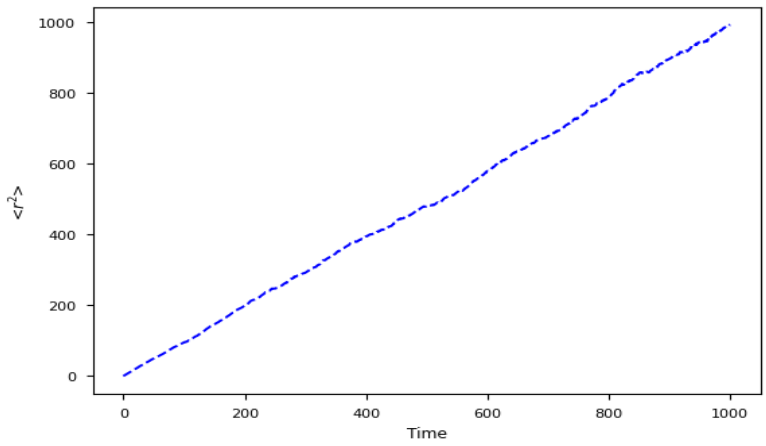

Figure 16: $< r^2 >$ at each time.

You can see the linear relation as expected, and the slope of this plot is **0.991**, which is very close to the theoretical expectation with only 1.9 % error.

6.Diffusion limited Aggregation (DLA)

For this question, I defined the function, `DLA`, which simulates the process with random walkers with initial positions based on the maximum height of the barrier (check the code for more details). Each particle performs a random walk and can stick to the side, top, or bottom of the barrier. The runtime is very high, and the growth process is very slow, even after several attempts at optimization. Here is the result for only 6000 particle depositions:

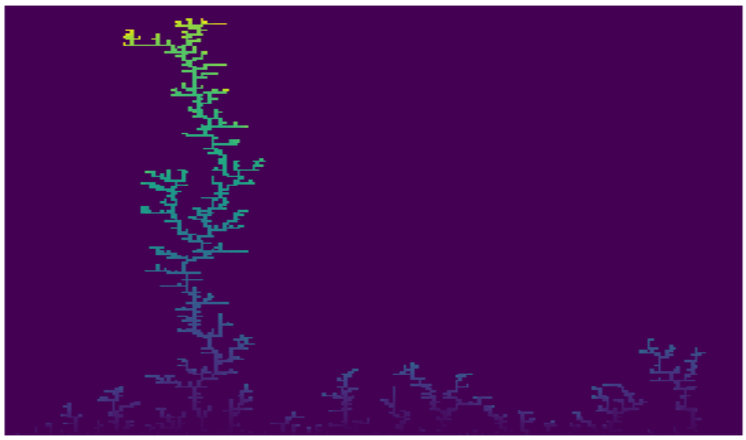

Figure 17: growth after around 6000 particle deposition.

Again, the output matches the expectation that the larger the columns, the more particles will be trapped.

7.DLA Growth from centeral point

The process is similar to the previous question. The only challenging part was applying the boundary conditions for the circular case. Instead of randomly generating a point, I generate an angle and then calculate the initial starting point of each particle. The runtime was significantly better than the previous question. The results are as follows:

Figure 18: square Condition.

Figure 19: circle condition.

Because the particles in the circular boundary condition on average start closer to the cluster, they will stick faster than in the square condition. For the circular condition, the probability of growing in several directions is higher.

8.Self Avoiding Random Walk

With a little modification of the previous function, we can do this simulation. The process isn't hard, and it's pretty straightforward. (Check the code for more details). Here is the distribution of each walker's total steps until reaching the dead end:

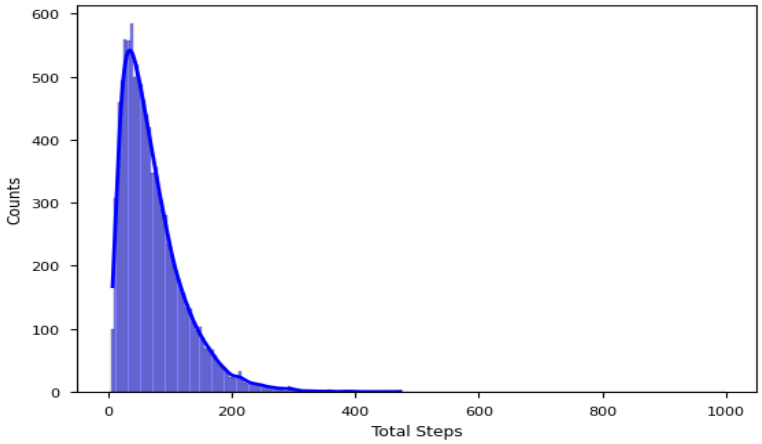

Figure 20: distribution of total steps.

We can see from the plot that it decays exponentially. In fact, after 10 to 50 steps, half of the walkers will reach the dead end.

9.Calculating Total availabel paths for Self Avoiding Random Walk

Simulating this problem was rather challenging. I count we can count every possible path starting from the initial position. At every new step, the SAW has at most 4 options, and each option creates a new path. I defined the function, `generate_saws`, for implementing this process. (Check the code for more details). Here is the plot of the ratio of paths and another for comparing the total path of a normal Random Walk versus SAW: (I just simulated the first 10 steps; the runtime for larger steps was very high).

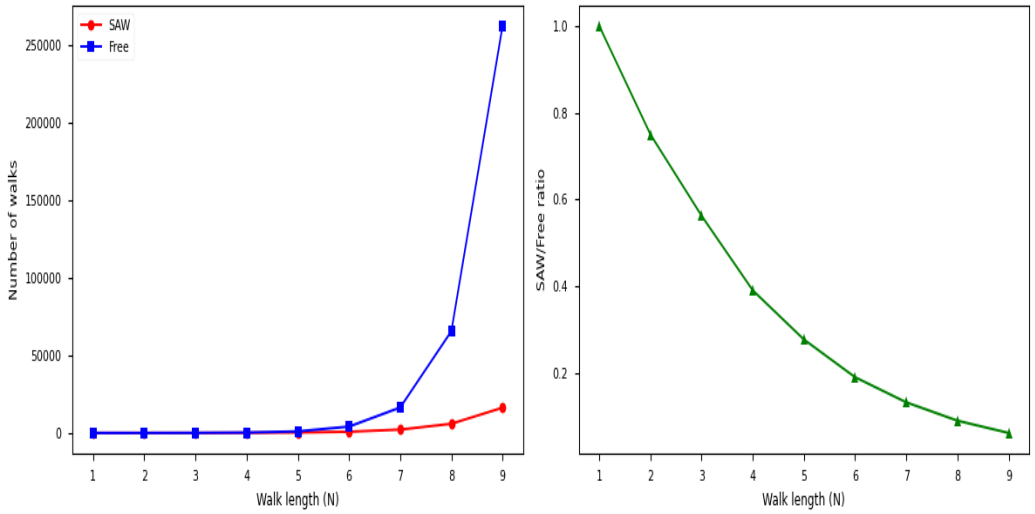

Figure 21: Normal Random Walk and SAW .

As expected, when the number of steps grows, the difference between the free walker and SAW will grow drastically.