EK 424 Thermodynamics
Final Project # 3:
Monte Carlo and Random Walk

Kyubi Yang (U54142590)
Maurie Zhang(U48656902)

I. Background

Monte Carlo methods became popular after World War II, and it are now commonly used in the fields of engineering, finance, and mathematics. The majority of real world processes cannot be predicted using linear systems; probabilities for real processes are not well defined, and thus there is a need for non-deterministic models to describe these situations.

A Monte Carlo simulation is a statistical technique to evaluate for various general outcomes, through a process of random sampling. The power of Monte Carlo and related statistical techniques lies in their ability to provide meaningful predictions for stochastic processes, which would otherwise be unsolvable using traditional deterministic models. With the Monte Carlo method, we can choose random cases within a given range, and be able to examine how the variables combine or interact. If we repeat the same simulation over multiple times to make the result solid, the error will be minimized.

The concept of random walk was first discussed in 1905 by Karl Pearson in a letter to Nature, where he asked the probability of man reaching distance of r with x steps when he does random walk. This was answered by Lord Rayleigh and eventually research on random walk started.

A Random Walk process is a process in which the behavior or kinetics of system components is unpredictable; it is a description of the sequential path of randomly-moving objects. Each random walk starts at the same place, the relational origin, or beginning point. Randomness exists because there is maximal entropy or disorder; the probability of taking a forward step is equivalent to the probability of taking a backward step, and thus the direction of the next step is not conventionally predictable, . Predicting the trajectory of a Random Walk Process is not definitively possible, all that can be provided are generic descriptions. To explain this concept more mathematically clearly, consider a one-dimensional randomly-moving object. This random object will take a step backward or forward from its location with equal probability. Then, after all the steps done, we will calculate how much distance it has moved from its original location.

II. Results

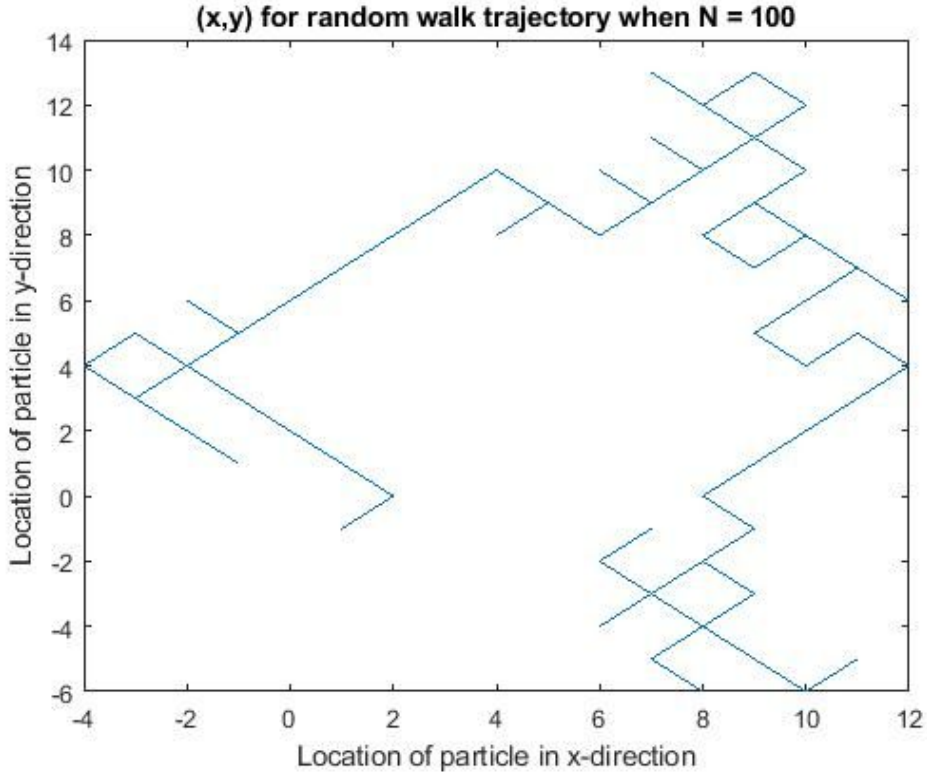

Figure 1. Random walk trajectory for 100 steps

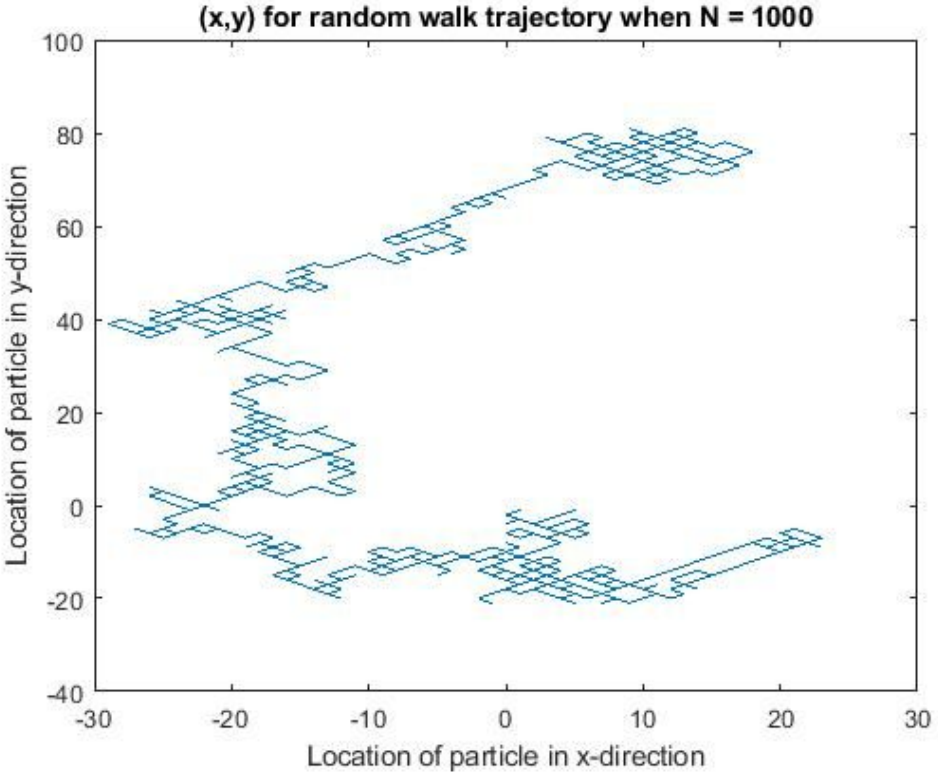

Figure 2. Random walk trajectory for 1000 steps

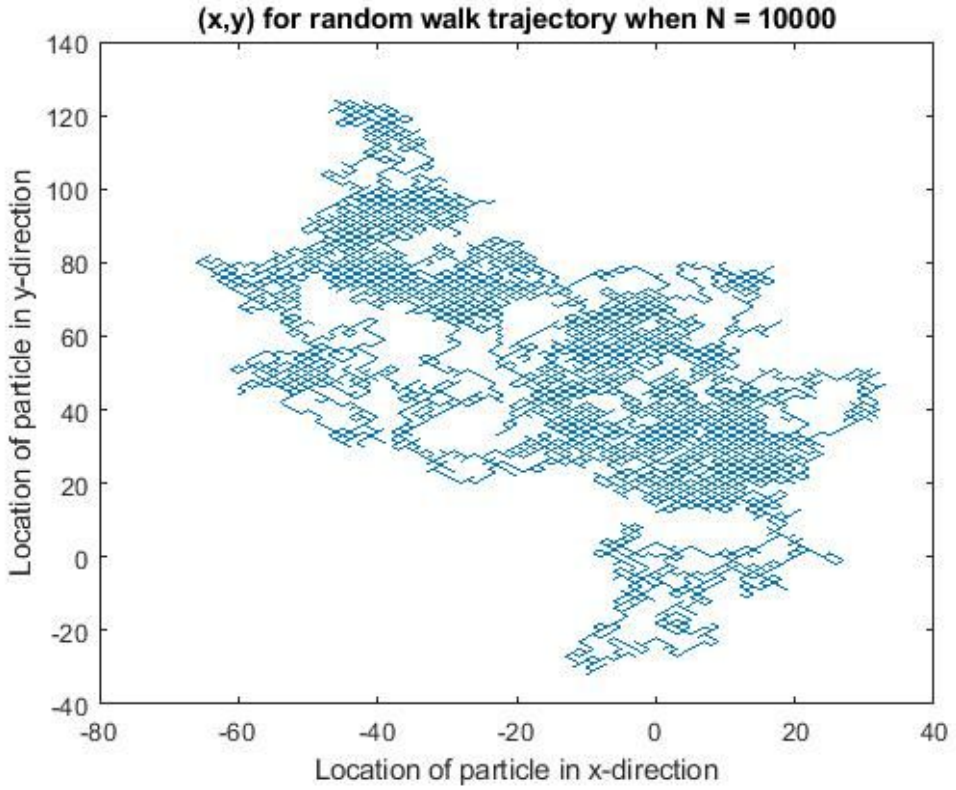

Figure 3. Random walk trajectory for 10000 steps

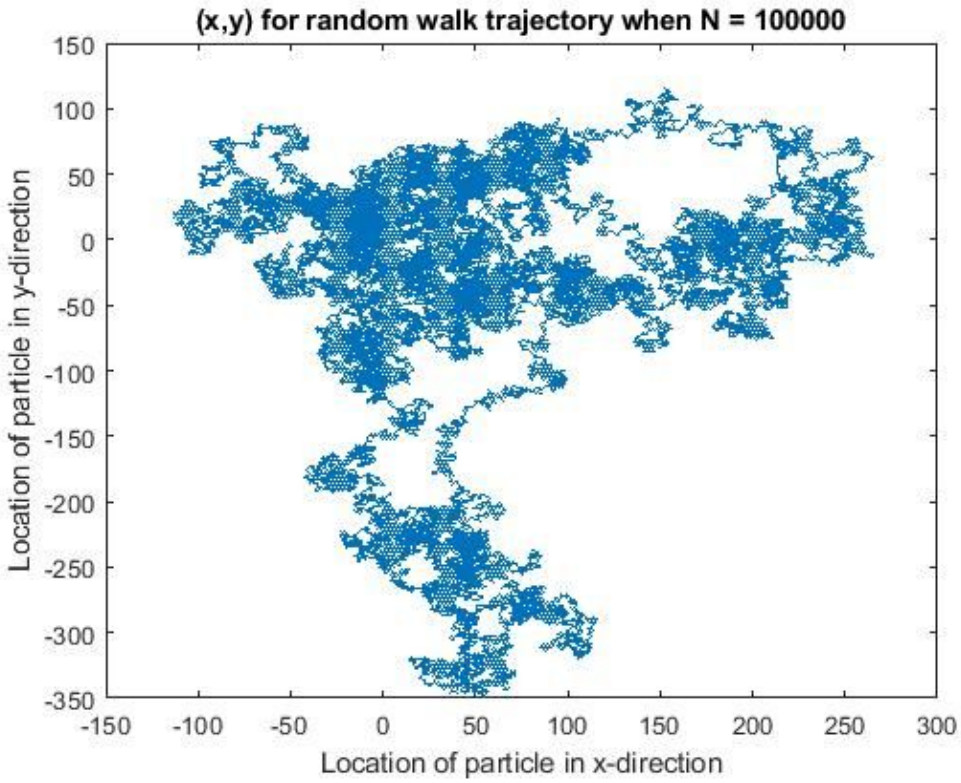

Figure 4. Random walk trajectory for 100000 steps

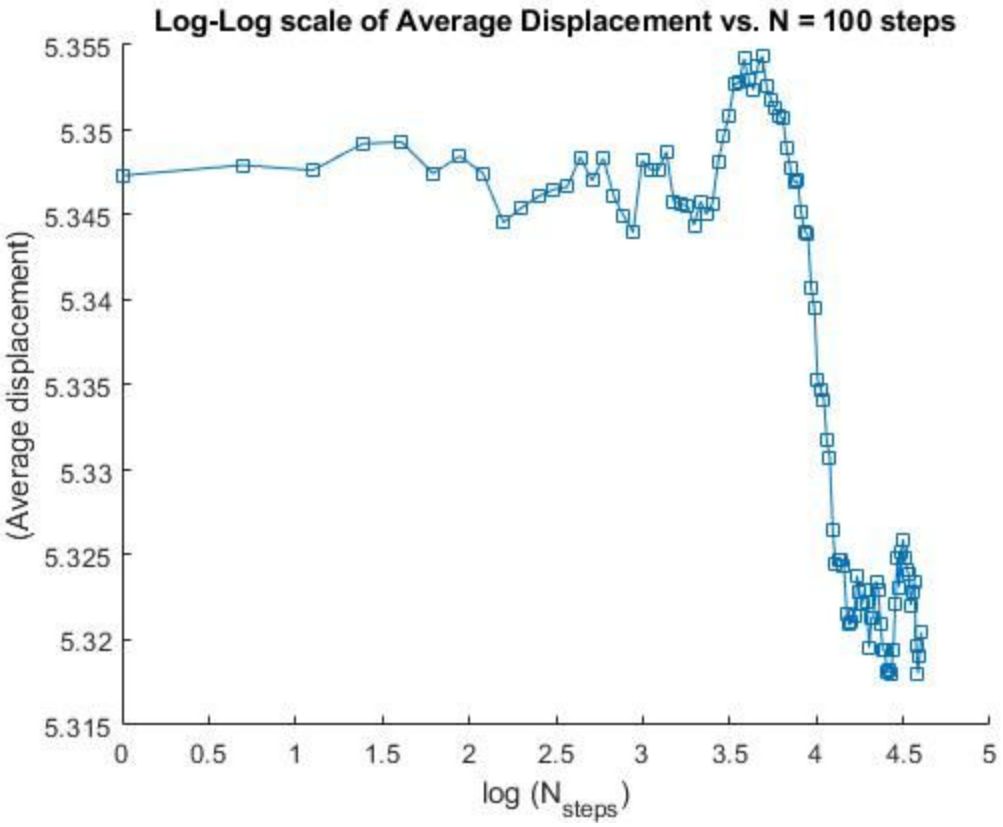

Figure 5. Log-scale distance vs. 100 steps

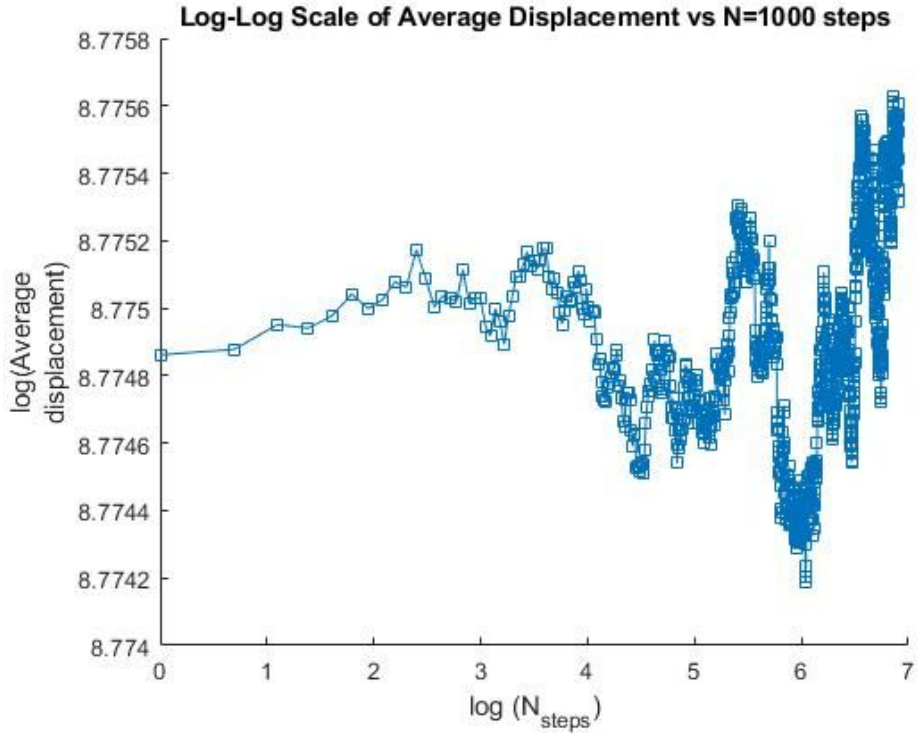

Figure 6. Log-scale distance vs. 1000 steps

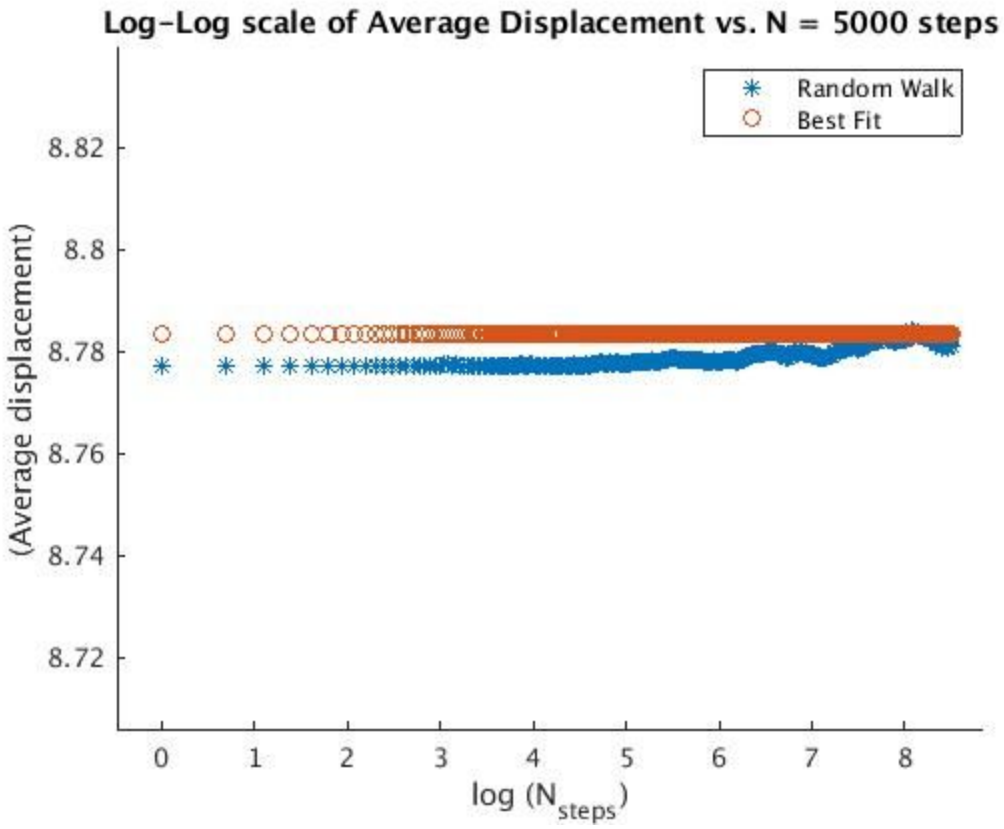

Figure 7. Log-scale distance vs. 5000 steps

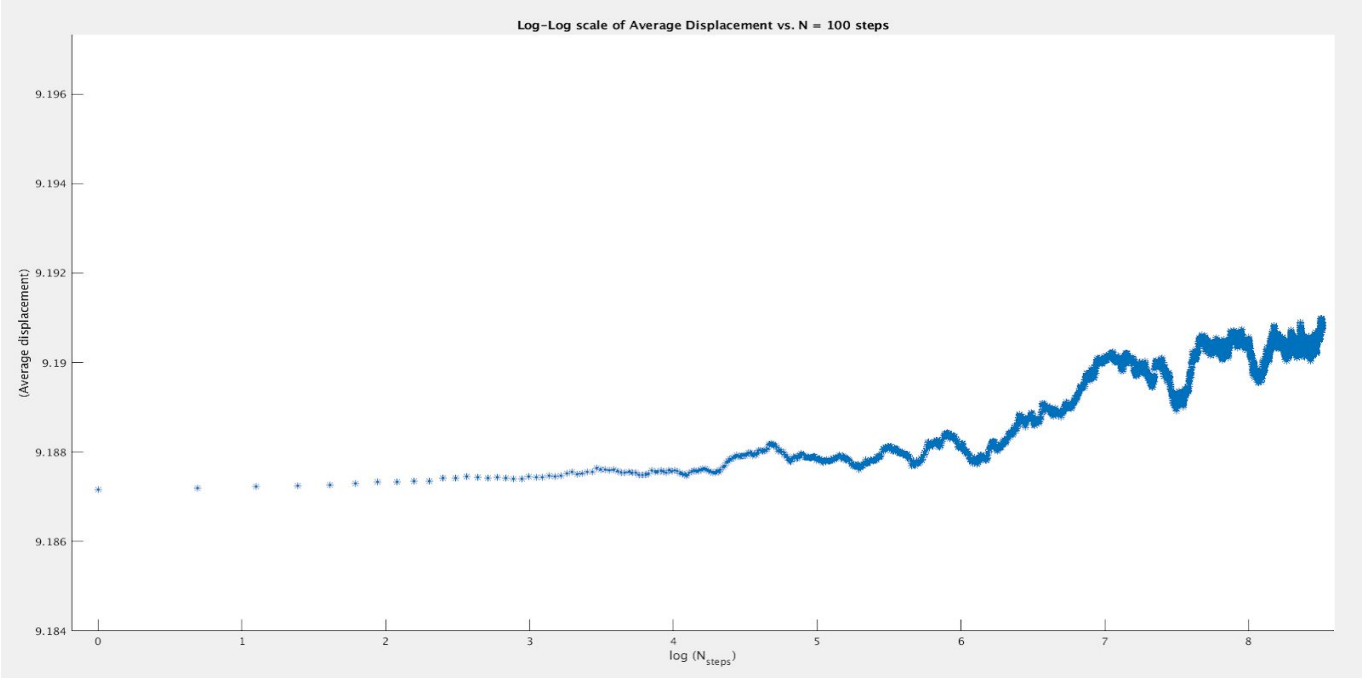

Figure 8. Log-scale distance vs. 10000 steps

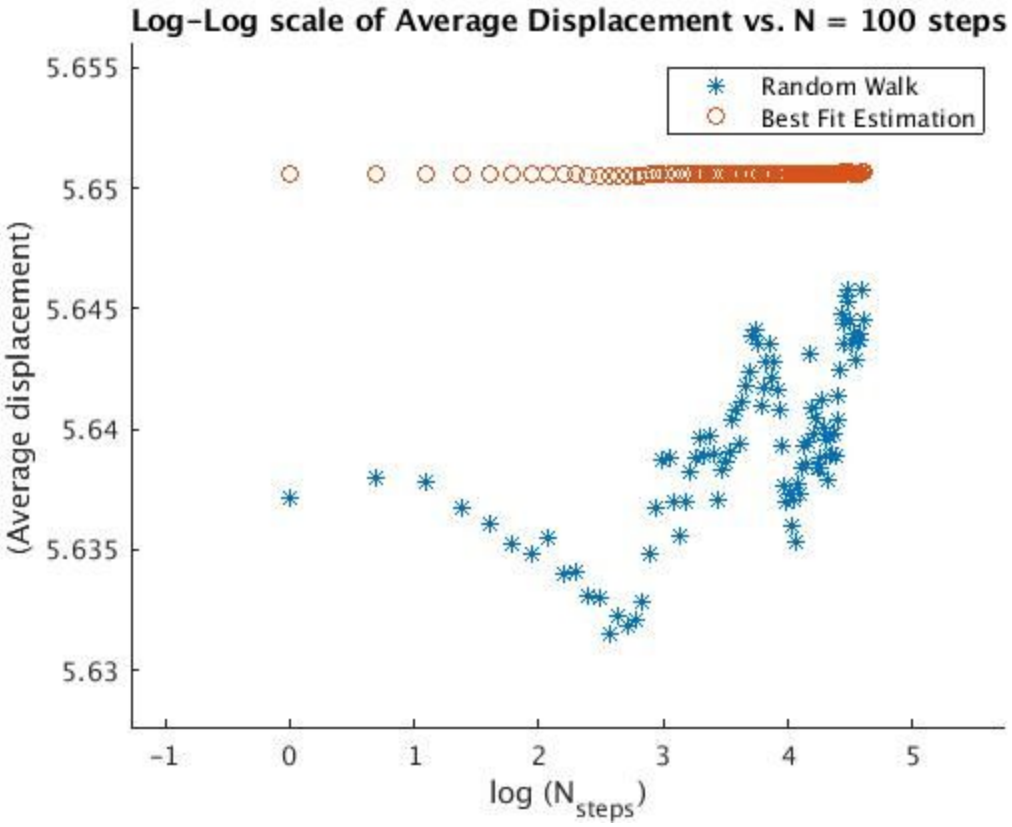

Figure 9. Log-scale distance vs. 100 steps with Fit function

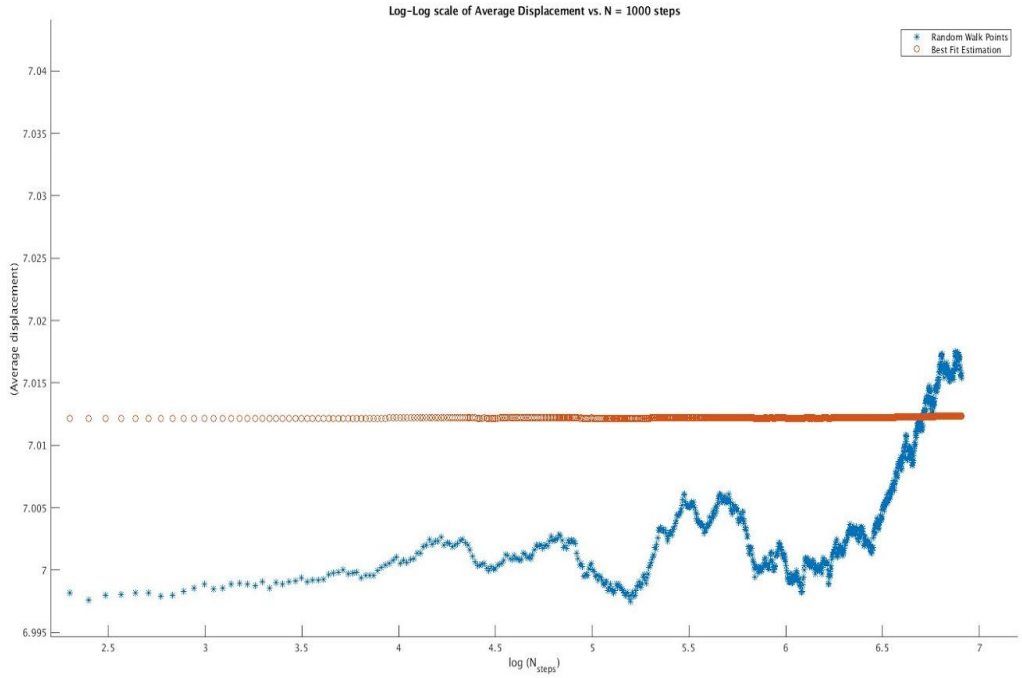

Figure 10. Log-scale distance vs. 1000 steps with Fit function

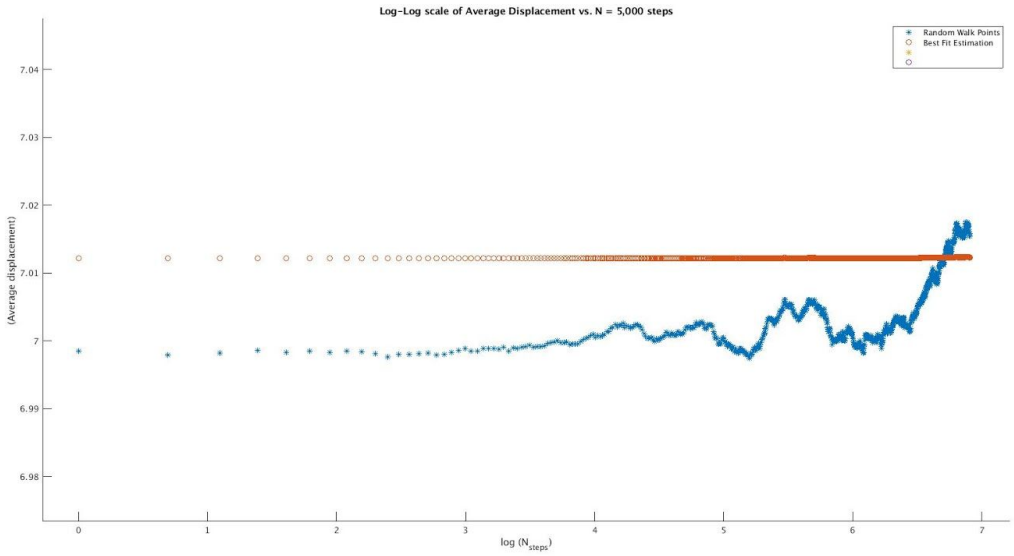

Figure 11. Log-scale distance vs. 5000 steps with Fit function

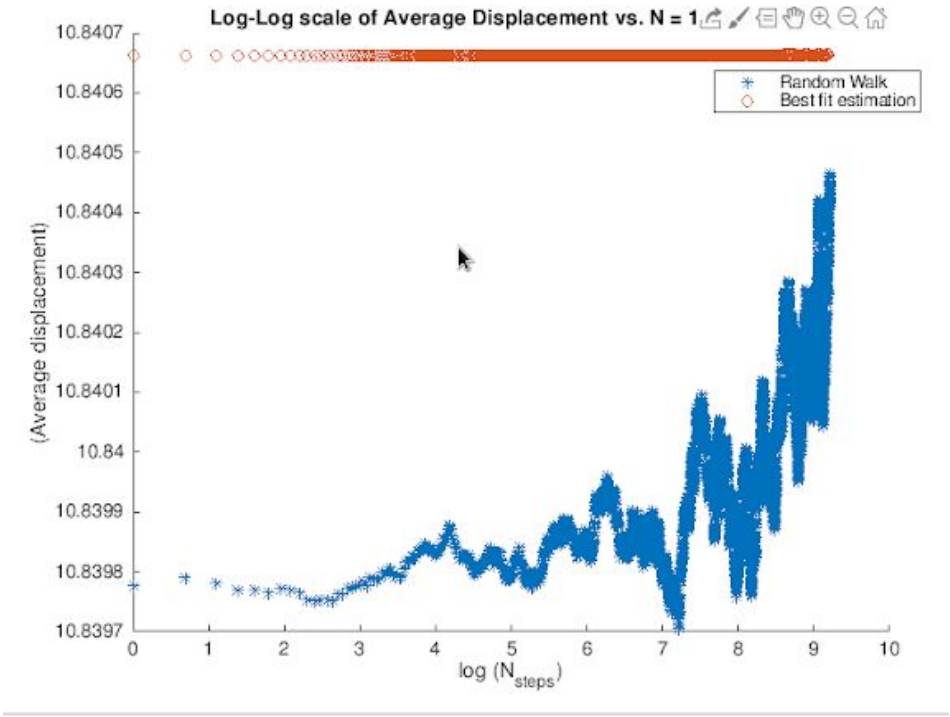

Figure 12. Log-scale distance vs. 10000 steps with Fit function

III. Discussion

For Figure 1 to 4, we could tell that with greater step numbers, the particle's magnitude of maximum and minimum displacement got greater. Moreover, with greater step size, particles

tend to be concentrated in one places. With lower step sizes, particles move around to further edges.

For our graphs for part 3 and 4, we used Ntotal = {100, 1000, 5000, 10000} instead of {100, 1000, 10000,100000} because the last value 100000 was too large and the code wouldn't finish running for an hour. Therefore we used slightly lower steps value. Although the p value would be much closer to 0.5 if we used 100000 for our f = c*N^p, we decided that we could see the pattern even with smaller values.

We figured out that the distance will approximately get closer to sqrt(N) as we processed more graphs with larger step size in log-log scales. Theoretically, total displacement should be equal to sqrt(N), and we proved it. Proof is shown below as Figure 13.

Our p values for N total = 100 is 1.489, for N_total = 1000 p value is 0.795, for N_total = 5000, p value is 0.688, and when N_total = 10000 p value is 0.592. From this, we can read the trend

that p value gets closer to 0.5 as our sample size increases.

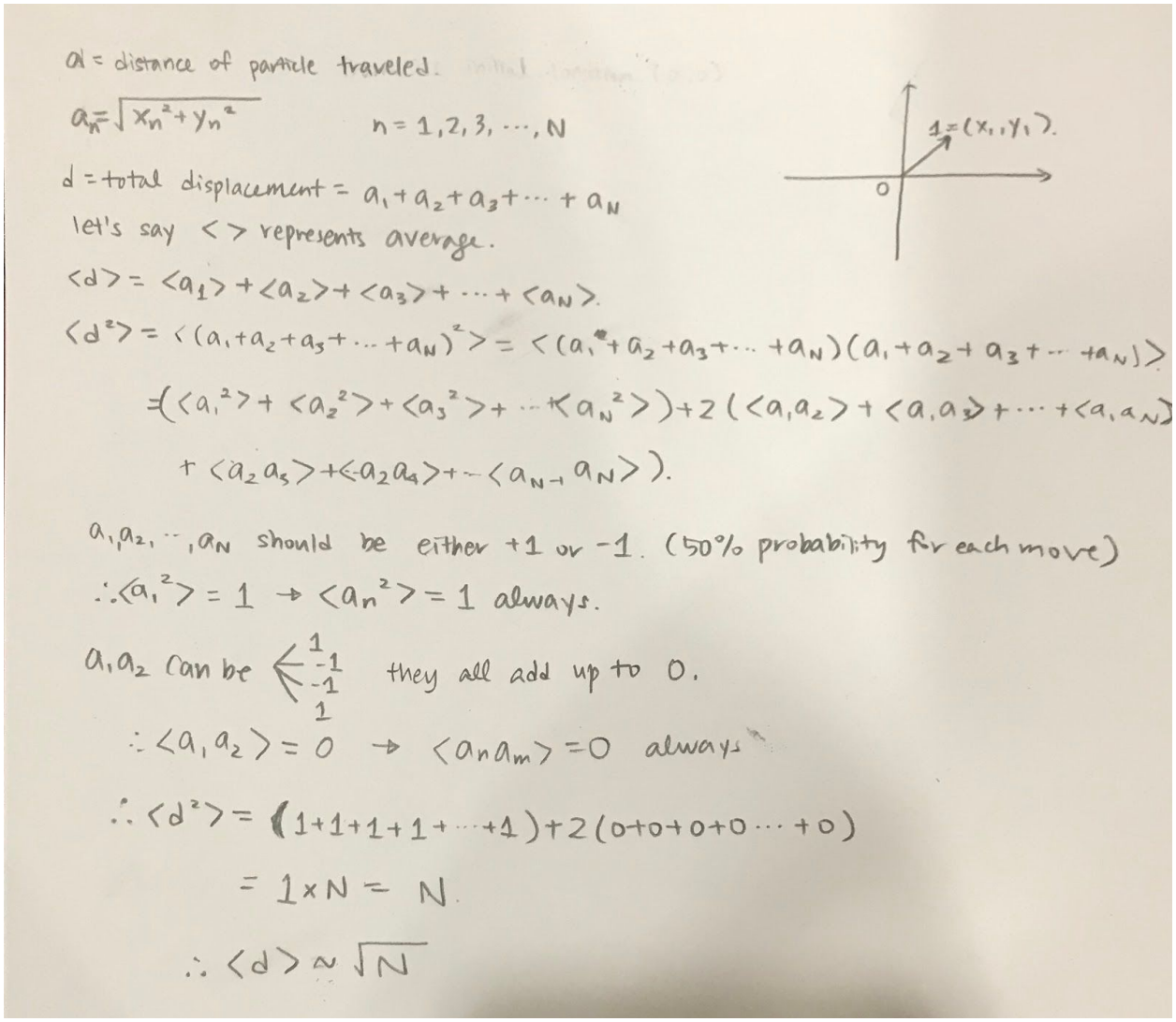

Figure 13. Proof of d = sqrt(N) in Random Walk simulation

Although we proved that displacement should be close to sqrt(N) value, we couldn't get accurate results from our graphs. For step sizes 100 and 1000, inaccurate results are due to small range of collected variables.

IV. Reference

"Random Walks". *Mit.Edu*, 2019, https://www.mit.edu/~kardar/teaching/projects/chemotaxis(AndreaSchmidt)/random.htm.

"Random Walk - Statistics How To". *Statistics How To*, 2019, https://www.statisticshowto.datasciencecentral.com/random-walk/.

"Random Walk". *En.Wikipedia.Org*, 2019, https://en.wikipedia.org/wiki/Random_walk.

"Monte Carlo Method". *En.Wikipedia.Org*, 2019, https://en.wikipedia.org/wiki/Monte_Carlo_method.

Dizikes, Peter. "Explained: Monte Carlo Simulations". *MIT News*, 2019, http://news.mit.edu/2010/exp-monte-carlo-0517.

V. Appendix

- modified code for part 4

```
% This code considers the random-walk motion of a single particle over the
% course of steps.

%if you don't see steps # of points @ the end on the final plot, then that
%means that the particle backtraced into one of the old spots

%preallocate global variables
counter = 1; %keeps track of steps taken
posnx1 = 0; %keeps track of coordinate x
posny1 = 0; %keeps track of cooordinate y

% steps vector defines different random walk trajectories
% used to create a 2-level nested for loop, which creates single plots for
% each
steps=[100,1000,5000,10000];

% HALFst(i) = 1/2 .* steps(i);
% will be used in the 2-level for loop;

HALFst= zeros(1,length(steps));
for i=1:length(steps);
HALFst(i)=steps(i)/2;
end
threshx = 0.5;
threshy = 0.5;

for i=1:length(steps)
for j = 1:steps(i)
% preallocate data_structure to store the 1- 10 trials of random walk
% for each N_step value
Distances_X = zeros (10,steps(i)); %holds the x distances
Distances_Y= zeros(10,steps(i)); % holds the y distances
d = zeros(10,steps(i)); %combines x and y distance
%plot(Distances_X, Distances_Y);
% gca = get current axes or figure
% set (gca, 'XLim', [lower_limit, upper_limit])
% the following two lines manually set the axes limits
% upper lim and lower lim = +/-(steps/2)
```

```
set(gca,'XLim',[-HALFst(i), HALFst(i)]);
set(gca,'YLim',[-HALFst(i), HALFst(i)]);
% for each step, get the corresponding frame
%F(j) = getframe;

for k=1:10
for s = 1:steps(i)
% temporary holder variable is a random number
% rand (rows, columns) [=] tmp has dimensions 2 rows, 1 column
tmp = rand(2,1);
if tmp(1,1) > threshx
posnx1 = posnx1 + 1;
else
posnx1 = posnx1 - 1;
end

if tmp(2,1) > threshy
posny1 = posny1 + 1;
else
posny1 = posny1 - 1;
end
% store tmp indices into Distances_X(i,k) and Distances_Y(i,k)
Distances_X(k,s) = posnx1;
Distances_Y(k,s) = posny1;
end
end
%Average values in the d vector which holds all the distances
%d_AVG= zeros(1,steps(i));
d = (Distances_X.^2 + Distances_Y.^2).^0.5;
d_AVG= mean(d); % mean (d) gives a row vector of the mean of each columnn
log_AVG = log(d_AVG);
log_steps = log(1:steps(i));
end
figure(i);
hold on;
%loglog(log_steps, log_AVG,'-s');
xlabel('log (N_{steps})');
ylabel('(Average displacement)');
title('Log-Log scale of Average Displacement vs. N = 10,000 steps');
legend('Random Walk points', 'Best fit estimation');

N=log_steps; % "x" value [=] log steps taken
c= log_AVG; % "y" value [=] log average distance given steps taken

p = polyfit(N,c,2);
%f(N) = cN^p
%G = log(f(N))= log(cN^p)
%G = log(c) + (p(3) .* log(N));
```

```
G = (polyval(p,c)); % newY=polyval(coeff2, log_x_value);
hold on;
loglog(N,c, '*')
hold on;
loglog(N, G, 'o');
% hold off;
hold off;
end
```