**Final Project: Diffusion-limited aggregation**

**Abstract**

The goal of our project is to create a simulation of the random process (Brownian motion) and analyze the process of forming the aggregates under it. Based on the data obtained in the simulation we are going to show the relations between the distance from a sample particle and the density of the particles in a small ring at that distance. We will also show the patterns of aggregations and the relation between fractal dimension and the density exponent factor.

**Algorithm and analysis**

We firstly created a square grid with side (N= 1501), then we set the center point to be a seed particle that is fixed on that position. Then we add particles one at a time at a distance + 5 from the seed. is the maximal distance from a seed to the outer shell particle. The particle undergoes a random walk, with a uniform probability distribution to go up, down, right, or left. If the particle meets the cluster formed around the seed, the particle has the probability to stick to the cluster and if it does, it is added to the cluster and a new particle is spawned, the process then continue till the desired number of particles are added to the cluster. If the particle is moving further than 3 from the seed, the particle is getting deleted and created again at a closer distance to the seed.

To make the simulation I needed a global 2-d list: grid which is the grid where we did the simulation, global variables: N – number of cells in a grid side; - maximal distance from cell to the outer particle; - probability of sticking. I have come up with the idea to create a class named Particle (), that has the instances of the position of the particle on the grid and will have a member functions: constructor, destructor, jump(), check\_in(), check\_out(). Each of which performs following:

Constructor: Spawns a particle randomly on a circle of Rad = +5 – offset, radius around the seed. Offset is added when the particle needs to be spawned closer. Using the randint() function we set x coordinate to be random from –Rad to Rad, after we know the value for x, we have two possible values for y, thus we do the coinflip to choose between np.sqrt(Rad\*\*2 – x\*\*2) and - np.sqrt(Rad\*\*2 – x\*\*2). This is actually questionable method to generate the random point on a circle, however it is still fine. In such form the particles are spawned on a circle. Then we add to x and y the coordinates of the seed, so the circle will be around the seed.

The according coordinate on the grid is then set to be 1, which means occupied.

Destructor: Sets the grid cell, that has the coordinates of a particle to 0, deletes the particle.

Jump(): firstly, sets the according grid cell to 0, and then is choosing one of the 4 possible directions of movement, the direction is chosen by random() function, if the output of it is in the one of intervals 0, 0.25, 0.5, 0.75, 1. Than the particle moves right, left, up, down. But before moving there, we check if this place is occupied, and if it is, we don’t move the particle.

check\_in(): checks if the particle has a cluster particle in the neighborhood, if so, it sticks to the cluster with probability .

Check\_out(): check if the position of particle exceeded the 3 . if so, the destructor is called and particle allocated closer to the seed at offset += 1.

Then I implemented the main process of the simulation, the function that simulates this process, it takes number of particles spawned in total and the probability of sticking as an inputs. It alters the grid and attaches the particles that are sticked to the cluster, to the list called “Cluster”. Both, grid and cluster list are global. During the process we undergo the method explained above. We use jump() function to perform jumps and after each jump we check if the particle exceeded 3 or if it is closer then + 2 to the seed, and if it is we check if the particle sticks to the cluster by check\_in(). When it finally does, we append the new particle to the cluster and spawn another one.

If we want our cluster to have around 10000 particles, then the maximal radius should be at least 200 since the shape of the patten is widened with every particle spawned at uniform distribution at a circle, however the shape is much wider than a circle of the same amount of particles. So, if we set to 200 from the beginning it is going to take a tremendous time to get to the seed. In order to decrease the computational time I have come-up with an idea to alter , so I start with = 20 and every time there will be another hundred points added to the cluster I change round(np.sqrt(i)) where i is the number of particles added to the grid. Then, when cluster passes 1000 particles, I set , The reason for that is I was making a simulation on a smaller amounts of particles, and observed that the pattern size is growing faster than when i is above 1000 but significantly lower than , moreover, I have seen that the size of the pattern, doesn’t approach at i=10000, since the there is plenty of place between and the outer particle in the grid, also the next parts show that the density is a negative power of r, which agrees with my assumptions, which meant I could use this as an optimization. Nonetheless the simulation still takes a decent amount of time.

Then, we obtained a cluster of particles of size that we wanted, we now can plot is as a contour plot and a scatter, I did both, example figures can be found at the appendix of this file.

Then I started analyzing the obtained data from the simulation. Firstly, I was interested in the density-density correlation function, so I calculated it as was asked, for each particle, the number of particles at a particular distance from it. In order to count that properly, I have implemented the function which returns the distance between the two particles, in the grid spacing terms, abs(x1-x2) + abs(y1 – y2); then used the [broad first search (BFS) algorithm](https://en.wikipedia.org/wiki/Breadth-first_search) to efficiently add the particles, the essence of the algorithm is that we create a queue that stores the processed particles, we then estimate the area in which the distances are within the interested range, and for the next particles it is checked whether it is in the zone, then top particle popped and algorithm repeated for next. Then I calculated the log of the density and the log of distance and compared the two, the graph of the points was plotted in the jupyter notebook, and the same graph was plotted in the excel file, there I have used a data points obtained in the simulation. I couldn’t do the sampling for 20 clusters, because the time needed to generate a single cluster is above 20 minutes for =1 and above 2 hours for =0.3 so, to create 20 clusters we would need above 40 hours of interpreter computational time, given all the optimizations performed. In the ‘Smaller simulate.py’ file that I have sent with the project the program is capable to perform the calculations for 20 clusters for both =1 and =0.3 but it is going to take couple of days of runtime if sampling is 10000. So I used a smaller sampling number (800) to do the 20 clusters avareging I have included the picture of the averaging value for this case of 800 points in Appendix figure 7. I used ~50 data points for different distances to find the density average among all the particles in clusters. Which I believe is sufficient to trust the simulation data. In the excel plotted graph that I included in this report appendix I performed a linear regression to get the value of the and I also have to admit that the graph obtained is very close to the line, which proves the relation given in [the paper](https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.47.1400) relation: . According to the obtained results (See Figure 3) the for = 1. Which imply that the dimension of a fractal is which is quite reasonable and compares good (Similar patterns have a similar dimensions) [with the table of the fractal dimensions](https://en.wikipedia.org/wiki/List_of_fractals_by_Hausdorff_dimension).

For the = 0.3 we got the result (See Figure 4), Which imply that the dimension of a fractal is which is quite reasonable and compares good (Similar patterns have a similar dimensions) [with the table of the fractal dimensions](https://en.wikipedia.org/wiki/List_of_fractals_by_Hausdorff_dimension). From the Figure 4 and 5 we can see that the pattern of the graph with sticking probability = 0.3 is more dense, unsurprisingly. Thus the difference is caused by sticking probability creates difference in dimensions.

In conclusion I would say that this project helped me to learn about doing the simulation of a big number of sample points and through it, I was able to learn some advanced technics in optimization and the have seen validity of those methods in real life. I have also learned about the analysis of the random process and a finding the statistical errors.

Important: The code supplied in main.py file is altered a little rather than the code provided in Final project.html, the jupyter file is given to display the ran results.

The HTML jupyter file has one of the latest versions with the successful run of the code, however it does not create more than 1 cluster at different P’s but it display all the results explicitly and is very clear.

The file smaller simulate makes the 20 cluster simulation and computes the relations with an averaging of those 20 clusters, but does so in a small sample points in order to do it in adequate time

**References:**

[1] BFS: https://en.wikipedia.org/wiki/Breadth-first\_search

[2] Witten Jr T A, Sander L M. Diffusion-limited aggregation, a kinetic critical phenomenon[J] : <https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.47.1400>

[3] Fractal dimensions: https://en.wikipedia.org/wiki/List\_of\_fractals\_by\_Hausdorff\_dimension

**Appendix**

![Chart, scatter chart

Description automatically generated]()

Figure 1. Scatter plot of a cluster of 10000 particles for =1

Chart, histogram

Description automatically generated

Figure 2. Contour plot of a cluster of 10000 particles for =1

Chart, line chart

Description automatically generated

Figure 3. Relation of the particle’s density-density correlation and radius logarithms for

=1

Chart

Description automatically generated

Figure 4. Contour image of the fractal cluster =0.3

A picture containing text

Description automatically generated

Figure 5. Scatter image for the fractal cluster =0.3

Figure 6. Relation of the particle’s density-density correlation and radius logarithms for =0.3

![Chart, line chart

Description automatically generated]()

Figure 7. 20 cluster averaging.