Monte Carlo Simulation of DLCA of Spherical Particles

Sayed Ahmad Almohri
aalmohri@umich.edu

Department of Chemical Engineering, University of Michigan, Ann Arbor 48109, USA

December 2020

Introduction

We investigated aggregates' formation due to Brownian motion of spherical particles, diffusion-limited cluster aggregation (DLCA).[1] We simulated DLCA gels using an off-lattice model and calculated the fractal dimension of the formed fractal system. This system has been studied well, and the fractal dimension reported in the literature is about 1.7-1.8[2], [3], which is similar to our results. We developed this system as a preparation to expand the simulation to other particle shapes such as discoids and ellipsoids. The team studied different particle shapes experimentally before I joined and after I got involved with the team I expressed interest in simulating the systems.

Methods

This program implements a Monte Carlo method that simulates a diffusion-limited cluster aggregation (DLCA). The work is based on a PRL paper by Ali Mohraz et al. [4] the basic idea is to move particles randomly to make clusters as they collide. First, we make N number of non-overlapping spherical particles in an L-by-L-by-L cube. Then, the particles start to move randomly between 0 and $\frac{r}{3}$ in spherical coordinates, and if a collision occurs, particles form a cluster. After that, the clusters moved in the same way until the number of iterations reaches the chosen value.

Algorithm

- We make a lattice X by Y by Z.
- We choose the number of Iterations we desire.
  - Be attentive with your choice, you don't want the system to yield a single cluster.[4]
- We choose a step size for movement.
  - We want to discretize the distance the particles move in order to avoid any missing collision or particle encroaching as the particles move.
    - ⊝ This can be also achieved by just changing the displacement but I chose to discretize instead.
  - In order to avoid particle encroachment and collisions as the particles and clusters move.
- Periodic boundary conditions are being used.
- We make N number of non-overlapping Particles with Radius r.
  - Initially each particle is a cell array in a 1-by-N cell array and each cell array contains a 1-by-3 matrix with the position of the particle centroids.
- At each iteration:
  - We make a random displacement between 0 and $\frac{r}{3}$ in spherical co-ordinates.

  - Move the particle with probability equal to μ.
    - ⊖ The displacement is multiplied by μ which is $\frac{N_{min}}{N_i}$ with N being the number of particles and i being the chosen cluster.
  - We calculate the radius of gyration of each cluster.
    - ⊖ Further discussion is provided later.

- When a collision occurs:
  - The centroids of particles from one cell gets appended to the other and the cell that was appended gets emptied.

- The program runs until the chosen iteration is reached.
  - This process could be self-operating.

List of Crucial Variables

- **mobility:**
  contains the mobility of each cluster at each iteration, what we have here is the mobility at the final iteration.

- **PAR:**
  PAR is the cell array that changes with every iteration so at the end of the iteration it contains the final position of the particles. (you can use this to access the data of the final iteration (end of the simulation)).

- **PAR_Rg:**
  PAR_Rg is the mapped version of PAR which is discussed in the radius of gyration section of the report.

- **PAR3:**
  PAR3 is the cell array that saves the centroid data at the end of each iteration, so PAR3{542} contains the centroid points in each cluster at the end of iteration number 542.

- **PAR_C**
  PAR_C is a matrix that contains the number of particles in each cluster at the end of the simulation. (If the value of PAR_C(i) = 0 this means that the correspond cell in PAR should be empty (PAR{i} = [ ])).

- **POS**
  POS is a matrix that contains the position of the particles at time = 0.

Rg3, mobility3, and PAR_Rg3 do what PAR3 does but for their respective variables.

Radius of Gyration

We calculate the radius of gyration (R_g) of a cluster by evaluating the distance between each particle and the center of mass of the cluster then divide that by the number of particles in the cluster. Due to the periodic boundary conditions that the system is using, we face problems when we evaluate R_g as it results in unrealistically high R_g values at the boundary as shown in Figure 1. To avoid these edge cases, we keep track of the clusters using two cell arrays (PAR and PAR_Rg). PAR is the original cluster data used to make the video with the boundary conditions being applied to them, and PAR_Rg is a mapped version in which the system does not have any boundaries, and the particles can move freely until all of the cluster leaves the boundary then which we make PAR_Rg = PAR since it can be used to evaluate R_g. The problem with the method that we are using is that we might lose some collisions, and to take into account those collisions every time a collision occurs with the parts that the boundary conditions are applied to, we map the cluster that collides with them to take into account the particle collision out of the box. Using the data from PAR_Rg we make a box (I recommend L-1) and use the clusters that are entirely in that box to calculate R_g in order to avoid the edge cases.

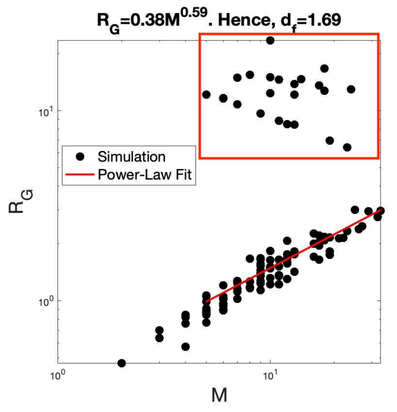


Figure 1: The figure shows the unrealistic high R_g value corresponding to the clusters that the periodic boundary is acting on

Calculating R_g:[5]

Consider a material described by matrix of center coordinates. Such a material can be a bead-spring polymer chain, colloidal aggregate etc.

Let c^i be the center coordinates of the i^{th} material (e.g. sphere, discoid, polymer bead).

$$c^i = (c^i_x, c^i_y, c^i_z)$$

Then, the co-ordinates of center of mass, c^{CoM}, of the material is defined as:

$$c^{CoM} = (c^{CoM}_x, c^{CoM}_y, c^{CoM}_z) = \frac{1}{N}\sum_{i=1}^{N} c^i = \frac{1}{N}(\sum_{i=1}^{N} c^i_x, \sum_{i=1}^{N} c^i_y, \sum_{i=1}^{N} c^i_z)$$

Then, the radius-of-gyration of the material, R_g, is:

$$R_g^2 = \frac{1}{N}\sum_{i=1}^{N}(c^i - c^{CoM})^2$$

If the cluster is a single particle:

$$R_g^2 = \frac{3}{5}r$$

Fractal Dimension

We calculate the fractal dimension of the system in order to validate our work and we do that as explained below:

- Make a smaller box in our cube and calculate the radius of gyration of the clusters inside the box.
- Fit a power law following the equation below:
  - $R_g = \alpha M^\beta$ and fractal dimension $D_f = \frac{1}{\beta}$.
  - M being mass and we set M = N, the number of particles in each cluster.
    - ⊖ We only use the clusters with M > 4, as the Mohraz et al. paper did.[4]

Running The Program

- Choose the number of particles. (N)
  - Volume fraction will be the number of particle times the volume of each particle divided by the lattice size.
  - Pick a radius r.
  - Pick a volume fraction V_f.
  - The number of particles that you should pick is $N = \frac{V_f L^3}{\frac{4}{3}\pi r^3}$.
- Choose the desired number of iterations. (ITR)
- Choose the particle radius. (r)
- Choose the lattice size which will be a cube L by L by L. (L)
- filename will save a .gif file to the current MATLAB folder, pick a relevant name.
- Now you can run the code, follow the comments provided in the script and remember to **not run large systems** on your computer the program is heavy so pick your system shrewdly and preferably run them on a cluster.

Results

Testing different different volume fractions yielded promising results, the fractal dimension was around 1.7-1.8 which is similar to the acceptable value in literature. Due to the lack of time I was not able to perform a thorough statistical analysis of the results. The number of runs with significant results were low since I was not able to test large systems until recently as I got access to the great lakes cluster. I was hoping to wrap up the sphere code much earlier and start working on a code for discoids but unfortunately getting the sphere code to an acceptable form took more than I expected as there were a lot of different possible collision cases that I had to investigate. I have started on an algorithm to estimate discoid collisions but have not implemented a code yet. I am attaching a 2 page manual draft of the discoid collision detection method and I hope to work on a code for discoids in the future.

Note that the number of particles that were tested in these results were relatively small and chosen in the interest of time since we did not get access to great lakes until a short time before this report was written.

V_f	N	r	L	Box Size	Step Size	ITR
0.6%	500	0.5	35	L-1	$\frac{1}{6}$	100000
0.9%	750	0.5	35	L-1	$\frac{1}{6}$	100000
2%	1638	0.5	35	L-1	$\frac{1}{6}$	24000
3%	2500	0.5	35	L-1	$\frac{1}{6}$	12000

Table 1: Parameters used to run the simulations.

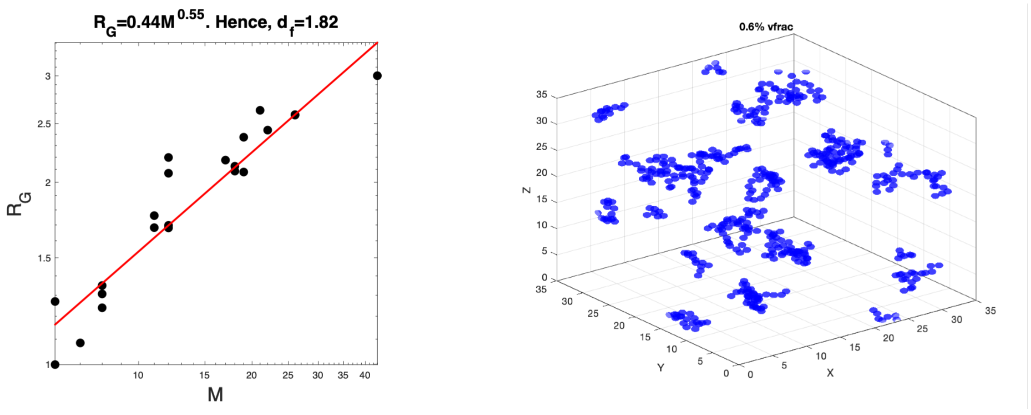


Figure 2: $V_f = 0.6\%$ number of iterations = 100000

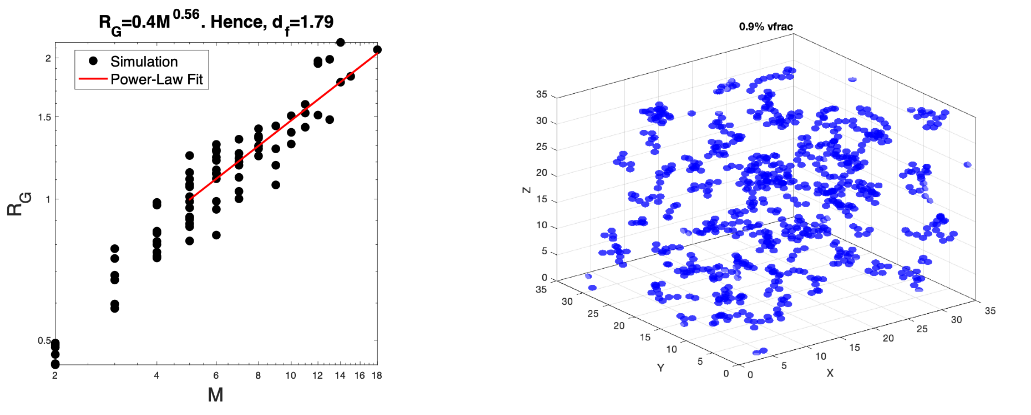


Figure 3: $V_f = 0.9\%$ number of iterations = 100000

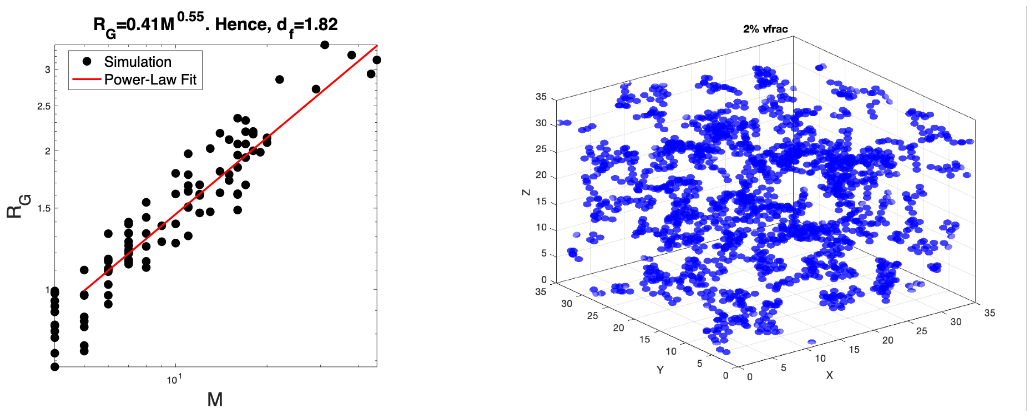


Figure 4: $V_f = 2\%$ number of iterations = 24000

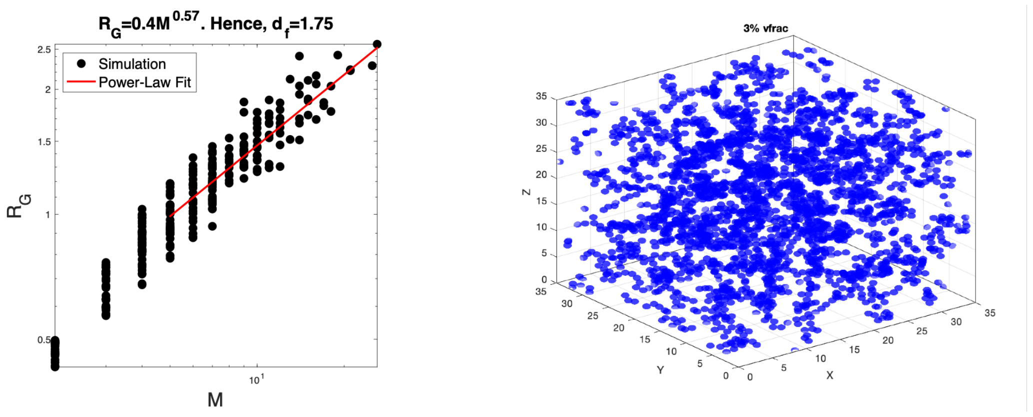


Figure 5: $V_f = 3\%$ number of iterations = 12000

What Are We Missing?

The program does not consider the rotation of clusters and omits them as earlier paper from Paul Meakin et al. works did [2], [3], [6], we agree that the rotation of clusters could matter, and it is worthy to investigate more later and see if the rotation of clusters would have a drastic impact on the results. The termination condition could be improved with making it dependant on R_g ($R_g = \frac{L}{4}$)[4] or the number of empty clusters in the cell instead of predicting how many iterations your system requires which could vary based on the size of the system.

Acknowledgements

I would like to express my very great appreciation to Dean Michael Solomon and his team for trusting in me, guiding me and encouraging me during these unprecedented times to pursue research. I also wish to thank Professor Ziff for his advice and assistance in this project. Lastly I should mention that this research was supported in part through computational resources and services provided by Advanced Research Computing – Technology Services (ARC-TS), a division of Information and Technology Services (ITS) at the University of Michigan, Ann Arbor.

References

[1] T. Witten and L. Sander, “Diffusion-limited aggregation, a kinetic critical phenomenon,” *prl*, vol. 47, pp. 1400–1403, Nov. 1981. DOI: 10.1103/PhysRevLett.47.1400.

[2] P. Meakin, “Diffusion-limited aggregation in three dimensions: Results from a new cluster-cluster aggregation model,” *Journal of Colloid and Interface Science*, vol. 102, no. 2, pp. 491–504, 1984. DOI: 10.1016/0021-9797(84)90252-2.

[3] L. M. Sander, Z. M. Cheng, and R. Richter, “Diffusion-limited aggregation in three dimensions,” *Physical Review B*, vol. 28, no. 11, pp. 6394–6396, 1983. DOI: 10.1103/physrevb.28.6394.

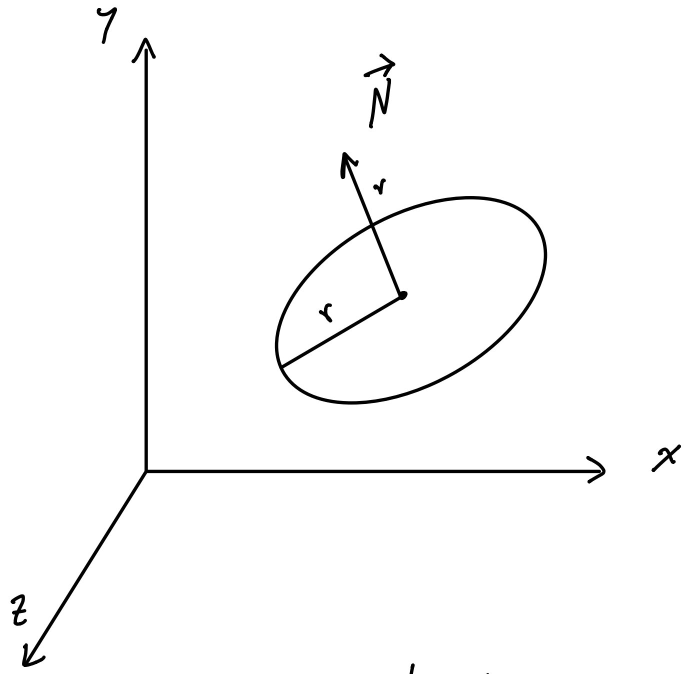
[4] A. Mohraz, D. B. Moler, R. M. Ziff, and M. J. Solomon, “Effect of monomer geometry on the fractal structure of colloidal rod aggregates,” *Physical Review Letters*, vol. 92, no. 15, 2004. DOI: 10.1103/physrevlett.92.155503.

[5] M. Ganesan, personal communication, 2020.

[6] P. Meakin, “Formation of fractal clusters and networks by irreversible diffusion-limited aggregation,” *Physical Review Letters*, vol. 51, no. 13, pp. 1119–1122, 1983. DOI: 10.1103/physrevlett.51.1119.

Appendix A Discoid Collision Approximation

- See if two discoids are closer than 2r
- Compute projection
- If projections are on each other, they're colliding



$\vec{N}_{00} = (0, 0, r)$ $\left[\text{normal vector when all angles are zero} \quad \theta = \phi = \psi = 0 \right]$ $A_0 = (\phi_0, \theta_0, \psi_0)$ initial Angles

Euler ratation :— $R = R_z(\psi) \; R_x(\theta) \; R_{z_2}(\phi)$

$$R_z = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_{z_1} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\vec{N}(0) = R_z(\psi)\ R_x(\theta)\ R_{z2}(\phi)\ \vec{N}_{00}$ * This will be a vector

$\vec{N}(i+1) = R_z(\text{Random } \Delta\psi)\ R_x(\text{Random } \Delta x)\ R_z(\text{Random } \Delta\phi)\ \vec{N}(K)$

$\vec{CC'}$: a vector connecting 2 centroids when they're closer than $2r$

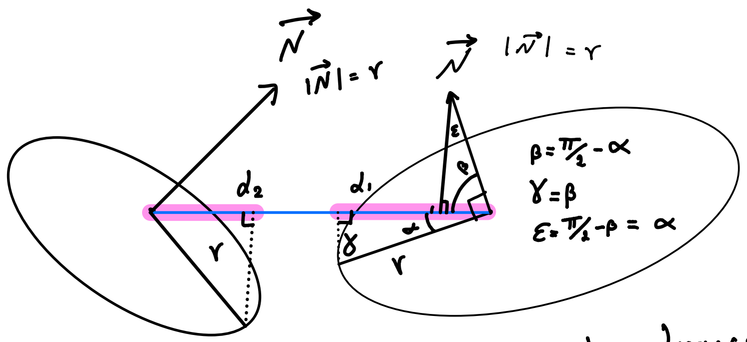
$$\text{Proj}(K+1) = \frac{\vec{CC'} \times \vec{N}(i+1)}{|\vec{CC'}|}$$

* Cross Product

$$C = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad C' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \quad \vec{CC'} = \begin{bmatrix} x - x' \\ y - y' \\ z - z' \end{bmatrix}$$

Why cross product??

*Not to scale



*consider the disc with all triangle drawn on, the 2 triangles are identical (3 angles, 1 side)

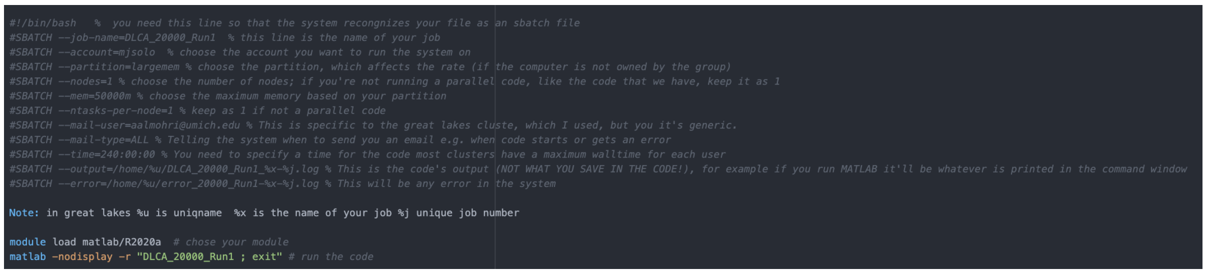
$$\Longrightarrow d_1 = \frac{|\vec{N}|\,|\vec{CC'}| \sin(\beta)}{|\vec{CC'}|} \Rightarrow d_1 = \frac{\vec{N} \times \vec{CC'}}{|\vec{CC'}|}$$

* if $\vec{N} \cdot \vec{CC'} < 0$ $\Rightarrow$ take $-\vec{N}$

* if $d_1 + d_2 > |\vec{CC'}| \Rightarrow$ discs collide

Appendix B Running a code on a Cluster

I ran the code on the Great Lakes HPC and it uses Slurm to schedule the jobs so I will describe how to submit a job to a cluster. To submit jobs you need to know some Linux command which can be found online easily so I will not list the different Linux commands you need to know. First you'll need to ssh to the cluster, in our case the great lakes, by typing "ssh uniqname@greatlakes.arc-ts.umich.edu" in our prefered terminal software (I use iTerm). I'll assume that you know how to transfer files between your computer and the cluster either by using a program like filezilla or just a command like SCP (there are many alternatives). There are many ways to run a code through Linux but specifically to run something on Great Lakes you'll need to submit a batch script; an example of a MATLAB batch script to the Great Lakes is in Figure B.1 I have added some comments on the file which should be removed if you want to copy out the same script and submit a job.



```
#!/bin/bash   %  you need this line so that the system recongnizes your file as an sbatch file
#SBATCH --job-name=DLCA_20000_Run1  % this line is the name of your job
#SBATCH --account=mjsolo  % choose the account you want to run the system on
#SBATCH --partition=largemem % choose the partition, which affects the rate (if the computer is not owned by the group)
#SBATCH --nodes=1 % choose the number of nodes; if you're not running a parallel code, like the code that we have, keep it as 1
#SBATCH --mem=50000m % choose the maximum memory based on your partition
#SBATCH --ntasks-per-node=1 % keep as 1 if not a parallel code
#SBATCH --mail-user=aalmohri@umich.edu % This is specific to the great lakes cluste, which I used, but you it's generic.
#SBATCH --mail-type=ALL % Telling the system when to send you an email e.g. when code starts or gets an error
#SBATCH --time=240:00:00 % You need to specify a time for the code most clusters have a maximum walltime for each user
#SBATCH --output=/home/%u/DLCA_20000_Run1_%x-%j.log % This is the code's output (NOT WHAT YOU SAVE IN THE CODE!), for example if you run MATLAB it'll be whatever is printed in the command window
#SBATCH --error=/home/%u/error_20000_Run1-%x-%j.log % This will be any error in the system

Note: in great lakes %u is uniqname  %x is the name of your job %j unique job number

module load matlab/R2020a  # chose your module
matlab -nodisplay -r "DLCA_20000_Run1 ; exit" # run the code
```

Figure B.1: sbatch file example for running a MATLAB script on the Great Lakes HPC