March 2018

COMPUTER TECHNIQUES IN PHYSICS

PHYS6009 Project B3

Investigating the Growth of Diffusion Limited Aggregation Clusters using a Computer Simulation

Daniel J Rose
27568326

School of Physics and Astronomy
University of Southampton
Southampton
S016 1BJ
UK

Abstract

This paper uses a computer simulation to investigate diffusion limited aggregation cluster formation in 2D, and looks at the additional difficulties when extending to 3D. It describes the way such fractals build up, and successfully calculates the value of the fractal dimension for 2D clusters as: d = 1.687 ± 0.002. Extension discussion lays out the groundwork for future development, and in conclusion the project was ultimately a success.

1. Introduction

Diffusion limited aggregation (DLA) refers to the process where large scale structures are built up at random by individually adding on new particles which diffuse in from infinity. Each new particle is undergoes a random walk (Brownian motion) until it makes contact with the larger structure, at which point it attaches and becomes part of the aggregate. The resulting structure that builds is known as a fractal (see Figure 1). DLA theory was first proposed by T.A. Witten Jr. and L.M. Sander in 1981, and can be used to model aggregation in any system where diffusion is the primary means of transport. Notable physical examples in which DLA is observable are the deposits that form on electrodes during electrolysis, smoke, Hele-Shaw flow and dielectric breakdown [Witten, T. A. 1981].

Due to the number of particles involved and the random nature of their motion, a computer simulation is the primary method of studying DLA. In this paper the chosen programming language is python, and the main investigation will focus on a 2-dimensional lattice, followed by an additional small investigation in 3 dimensions. Interestingly, the maths involved can accommodate any number of dimensions (the particles simply need more degrees of freedom) and while this has no current physical counterpart, research has in fact been done in up to 8 dimensions [Ball, R. 1984].

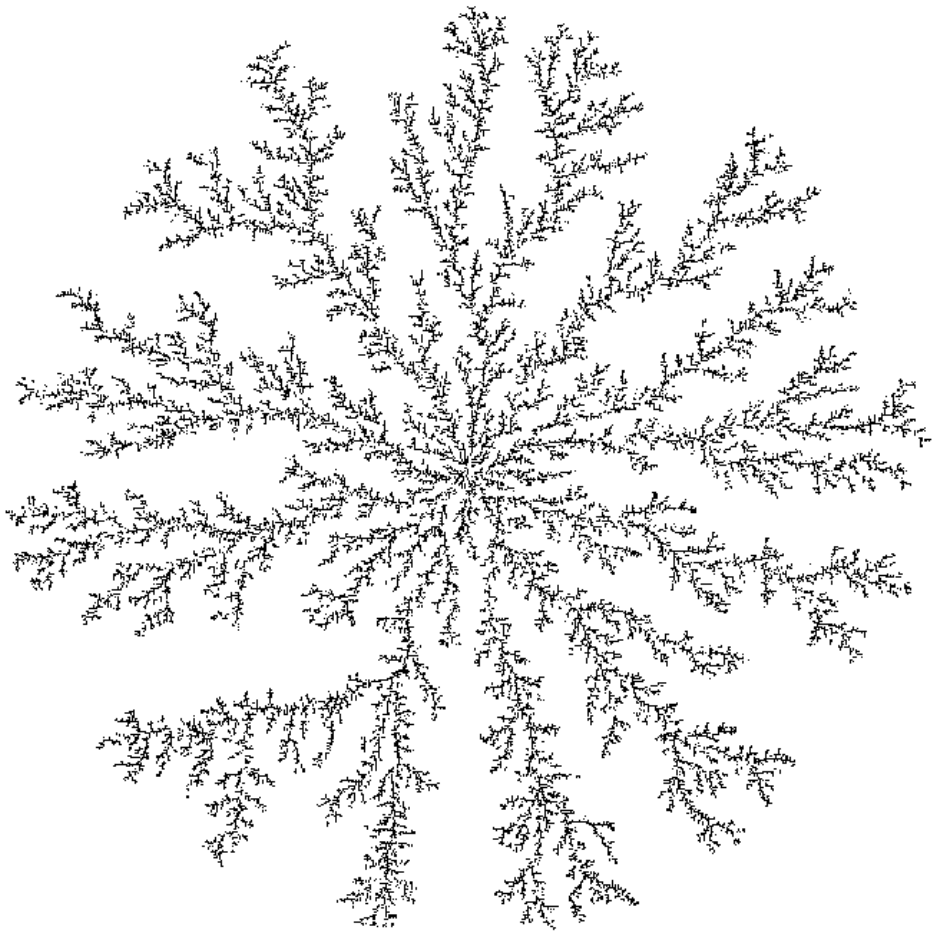

Figure 1: A constructed 2-dimensional fractal (1000x1000) displayed using Matplotlib's *pyplot.imshow* function.

As can be seen clearly from Figure 1, the fractals produced by DLA build up with definitive branching arms with empty valleys in-between. While this image is just the product of a controlled simulation, the same type of pattern also occurs throughout nature; for example the leaves of a plant grow to maximise total surface area and minimise distance from their branch [Stanley, H. E. 1986]. While this may be somewhat surprising, mathematically it makes sense. Consider a circle with a particle at its centre, if the particle is undergoing a random walk then there is equal probability that it will reach any specific point on the circumference (for a circle of any radius). If the particle is at the edge of the building fractal, and you increase the radius of the surrounding circle, the circle will always reach one of the fractal arms before the point where the arms join. As a result the fractal pattern that emerges is effectively the lowest energy configuration of such a natural system, and therefore the most likely to occur.

2. Method (for 2-dimensional DLA clusters)

The simulation will function by creating a very large 2D array of zeroes (representing the lattice), and then changing a single value at the very centre to 1, representing an occupied site. Particles (represented by ones) are then released one by one on the lattice at a great distance from the centre, and allowed to randomly move to any of their four neighbouring sites with equal probability. The random walk will continue until one of the four adjacent sites is occupied (by another 1) at which point the particle stops walking and occupies that site, and another particle is released.

For the diffusive nature of the system to dominate, it is important that there is only one particle diffusing at any one time. It is also essential that the motion is truly random (within computational limits), as this is fundamental to how the fractal forms. To this end, initial research included writing a short script to test NumPy's *random.randint* function. This script randomly produced integers 1-4, and binned them accordingly. Looping this script over a million times and comparing the final bins showed that the probability of each integer being produced was the correct 25%, with a less than 0.01% error on any individual bin, confirming that the selected NumPy function appropriate to use.

However a particle released at a great distance and allowed to diffusion randomly will likely take a very long time to reach the centre. Therefore taking available computational power into account, some improvements to the basic program were needed to efficiently grow a large scale structure. A particle diffusing randomly from a large distance should arrive at any point around the building cluster with the same probability. Therefore a good approximation is made by releasing particles from a randomly selected point on a circle of radius just larger than the fractal. The location was selected by choosing a random angle using *random.uniform*, and then using the standard conversions from polar to Cartesian coordinates.

Additionally if a particle ends up a long way from the cluster due to its random motion, it will take a very long time to diffuse back. The particle would essentially be lost in reality, and to continue simulating even a single particle until it returns could take a large amount of processing time for realistically no effect on the resulting fractal. Therefore the simulation will abandon any particle that gets to twice the radius of the current cluster. While computational time could further be improved by reducing this value (for example to 1.5 times the cluster radius), it may begin to affect the final structure as particles struggle to diffuse properly around the outside of the fractal. A similar effect was also an issue in early versions of the program, which had the fractal building right to the outer edges of the lattice. This was compensated for by doubling the size of the lattice and having the fractal only build halfway out from the centre, ultimately resulting in the same end size.

Fractals are characterised by the fractal dimension d, defined as:

$$M(R) \propto R^d \qquad \text{Equation 1}$$

Where $M(R)$ is the 'mass' contained within a circle of radius R inside the fractal [Family, F. 1990]. For our simulation $M(R)$ is equal to the number of occupied lattice points inside a circle centred on the fractal's origin, and the circle's radius R is not greater than largest radius of the fractal.

Taking the logarithm of both sides gives: $\log M \propto \log R^d = d \log R$, and then rearranging:

$$d = \frac{\log M}{\log R} \qquad \text{Equation 2}$$

Hence the value of the fractal dimension d is given by the gradient from a plot of $\log M$ against $\log R$.

To plot the required graph mass values for many different radii are needed. For this paper it was opted to construct an entire fractal of a predetermined final size and then calculate its total mass, for multiple fractals of varying radius. The initial program started by building a very small fractal ($R = 25$) and then working its way up, each time building a new fractal with a radius one greater than previous. The aim was to go up to $R = 500$ (a 1000x1000 fractal), so that the final cluster was sufficiently greater than the lattice spacing. However it was quickly determined that building a new fractal for each new measurement would be far too slow, as working up to just $R = 150$ took around 90 minutes, and making larger and larger fractals would just take exponentially longer.

Instead the program was altered so that values of the cluster mass could be taken as it built up, with a snapshot being taken each time a new aggregating particle increased the cluster radius by 1. Whilst this method initially seems to have the issue that particles can join inside the current radius after a snapshot is taken, effectively there is no difference to the previous method; a fractal of any 'final' size can have more particles attach inside without increasing its total radius if random diffusion is allowed to continue. Then to make sure the results were not biased by one single fractal randomly building in

some unusual way, the whole process would be repeated ten times and all the results used to calculate the fractal dimension of a two-dimensional DLA cluster.

Firstly, processing speed was tested by just building up one cluster. A single 1000x1000 fractal ($R = 500$) took around two and a half hours on the system in question, which was deemed somewhat too long for a single run. 900x900 was around an hour quicker, but still not quite as fast as required for multiple runs, so instead it was opted to use an 800x800 fractal ($R = 400$) to produce the final graph.

3. Results and Analysis

The full simulation took roughly 9 hours to complete, and constructed ten 1600x1600 fractals. The mass values taken for different radii while the clusters built up are plotted in Figure 2 below.

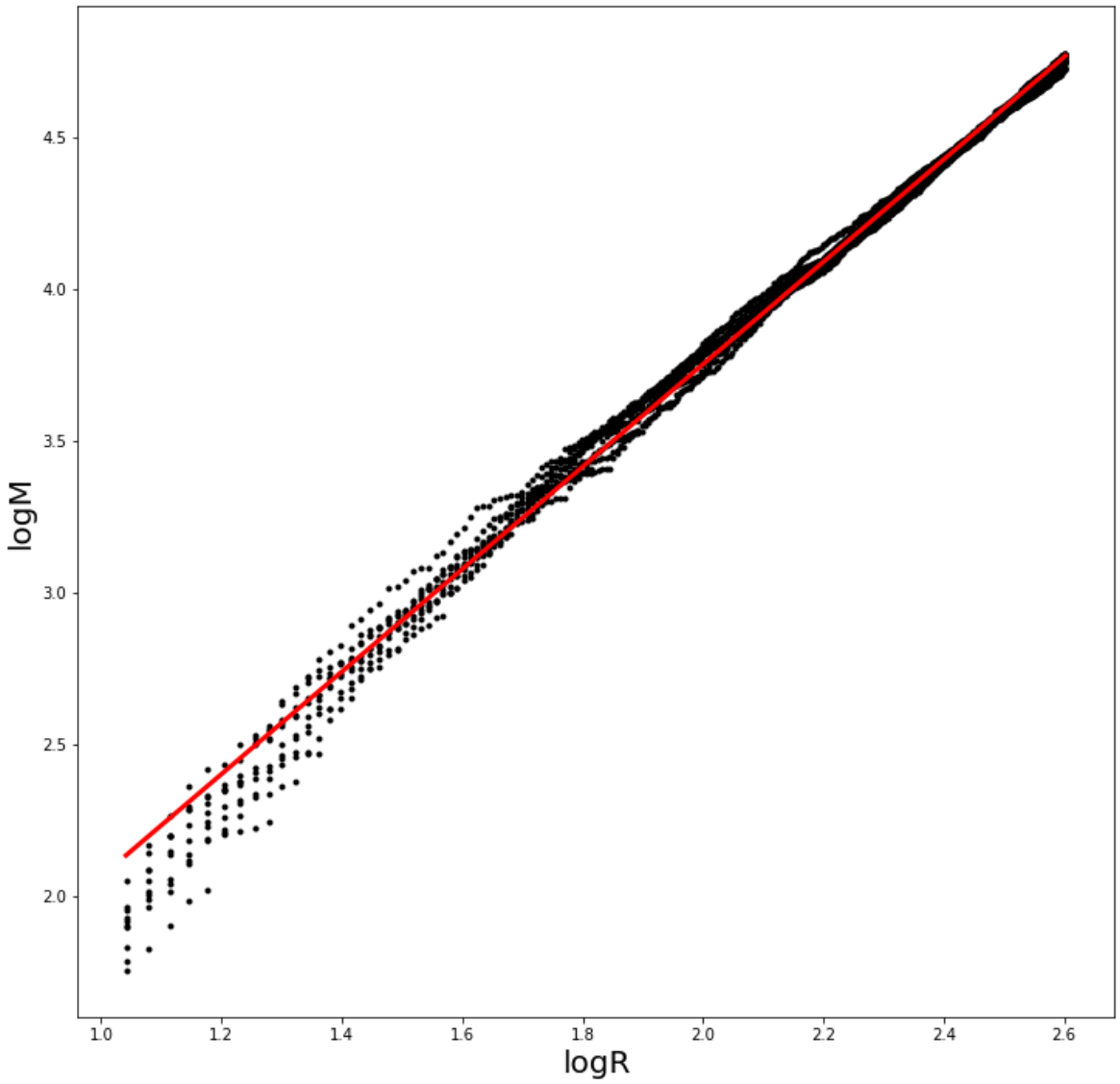

Figure 2: A log-log plot of mass against radius for ten 1600x1600 fractals, with a linear line of best fit (red).

A linear line of best fit was plotted using regression analysis via NumPy's *polyfit* function (which uses the method of least squares fit). The gradient of this line and hence the value of the fractal dimension calculated was:

$$d = 1.687 \pm 0.002$$

With the error on d calculated by returning *polyfit*'s covarience matrix and extracting the zeroth element.

As the program measured the mass at the same radius intervals for each fractal, the graph displays a spread of 10 points for each radius value. This distribution of these points represents a sample from the standard deviation on each set, and is effectively a range of error for each mass. As can be seen from the plotted points, the spread is much larger when the cluster is still small. However as the size of the cluser is not hugely greater than the lattice spacing at this point, an increased error as such is to be expected. Also as the radius values increase by one each time, but the scale is logarithmic, the values are much more bunched together at greater radii. While this is not ideal for clarity, this does at least mean the best fit line is more heavily weighted to the higher values which is ideal as to minimise the error involved with the lattice spacing at small cluster sizes.

From the book 'Universalités et fractales', the value of the fractal dimension quoted for a two-dimensional DLA cluster is around: $d = 1.70$. While this is not quite within the error boundaries on the value calculated in this paper, it is still fairly close (within 0.8%). Whilst being 'close' is not usually scientifically acceptable, in this case it is also stated in the book that the value of 1.70 is not exact [Sapoval, B. 2001]. As the cluster formation is still essentially random, it is not possible to obtain an exact value for the fractal dimension that will hold true in every case. Taking this into account and the fact that DLA simulations can be programmed in slightly different ways, it is fair to conclude that the value obtained in this paper is not unreasonable as an initial result.

A linear fit was chosen as the data points appear to exhibit a fairly strong linear correlation. However to check if there was and systematic variation with radius, a second order polynomial fit was also applied to the data points, again using *polyfit*. The resulting graph is displayed in Figure 3 below.

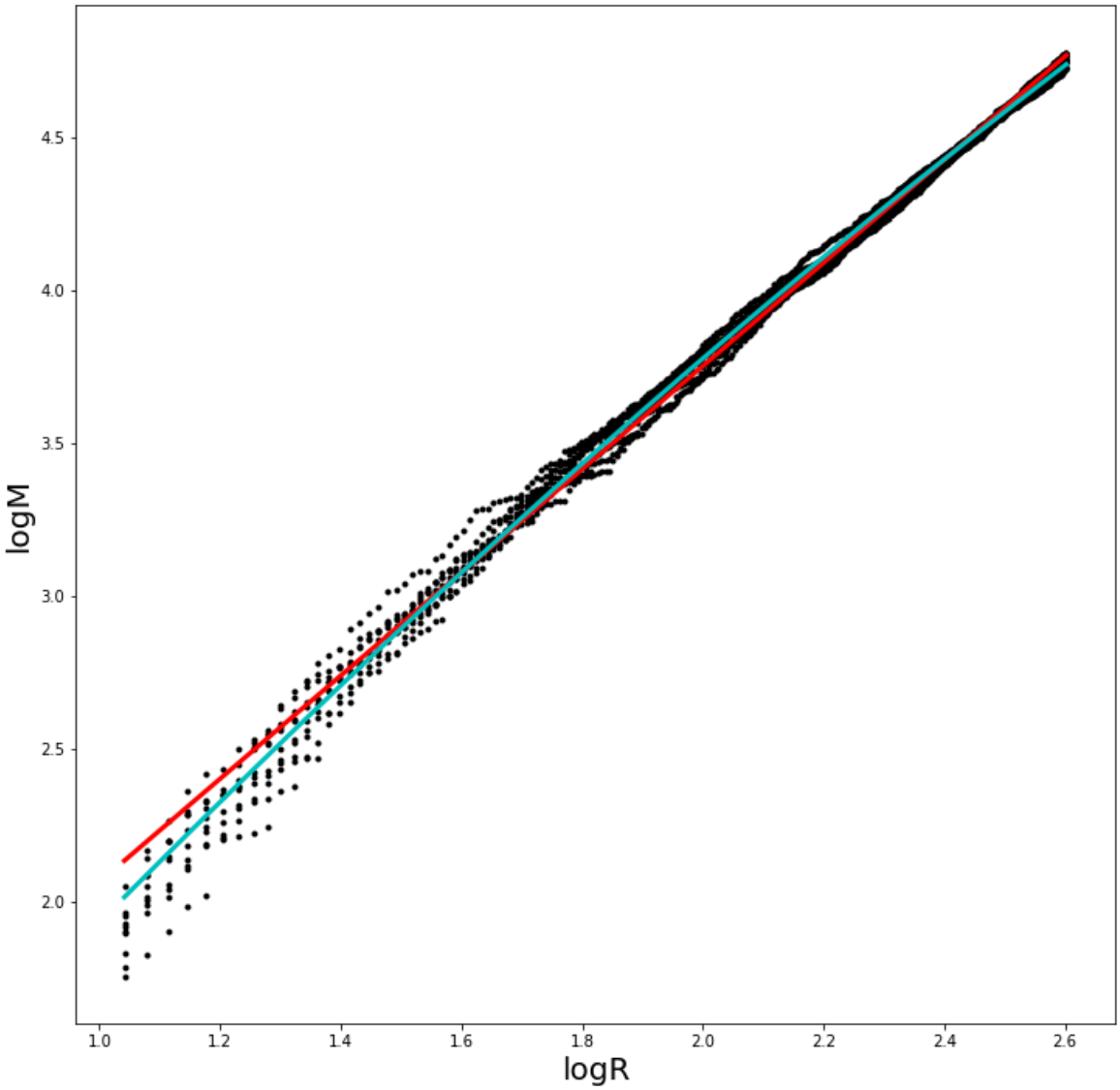

Figure 3: The same data points as figure 2, but now also applying a second order polynomial fit (blue).

While the second order polynomial fit appears to follow along the shape of the data points which drop away at the low end, there are still many more higher values clustered together so the upper end is much more heavily weighted. Looking at the middle and top end of the graph, it can then be seen that curved fit, while very close to the linear fit, starts off with a slightly steeper gradient and then becomes shallower. This would therefore imply that the variation of cluster mass with respect to radius cannot quite be described by as simple a relationship as Equation 1 suggests. This suggests that for the initial cluster growth the fractal dimension d is slightly greater than for the larger cluster size. This may be a consequence of the lattice size compared to the initial cluster.

4. An extension to 3-dimensions

An attempt was then made to modify the program accordingly so that it would simulate DLA in 3-dimensions. Altering the code itself was unexpectedly straightforward, the array would iterate over 3 dimensions with an extra coordinate needed to refer to a position. The diffusing particle could begin anywhere on the surface of a sphere, and had an extra two directions (orthogonal to the 2D plane) in which it could randomly move. Finally the radius vector had three components instead of two, and the 'mass' referred to the effective volume instead of an area.

Issues arose however, when the simulation began. It became immediately clear that the processing power required for this modification was much greater. Not only did the diffusion process for each particle take much longer, but simply summing the array to get the mass (which took milliseconds in 2D) took a much greater amount of time. Relatively small fractals of only 250x250x250 ($R = 125$) took over 2 hours to produce. While some time was spent attempting to improve the speed of the simulation, ultimately this persisted as the main issue

Unfortunately it was not possible to produce a fractal large enough to properly eliminate the effect of the lattice spacing, and as a result the building of such small clusters varied significantly. Multiple runs were done and plotted, following Equation 2 to get a value for the fractal dimension, however the results were not consistent, varying by ± 0.3 around $d = 2$. The value quoted for a 3D DLA cluster in 'Universalités et fractales' is around: $d = 2.50$ [Sapoval, B. 2001]. Like before this is not an exact value, however in this case our value is not close, even taking the fluctuation error boundary into account. It is at least of the right order of magnitude, suggesting the issue may be to do with not being able to produce a large enough fractal, due to performance limitations. Or, it may be due to an error in the method of calculation, however, this was not investigated further due to time constraints.

5. Discussion and Conclusion

In this paper we have investigated DLA cluster formation in 2-dimensions via computer simulation, and successfully obtained the value of the fractal dimension to be $d = 1.687 \pm 0.002$. The value and error of d found here may be influenced by the exact method used by this program. It would be informative to compare the results determined by a number of such programmes, and depending on the spread of those results, look to identify significant factors in the programming method.

The clarity of the plotted graphs suffers at the higher values due to the logarithmic scale used. It may have been beneficial (to both runtime and clarity) to calculate beforehand which radius values would have given an even spread of points on a logarithmic scale, and only calculate at these values.

An attempt was made to investigate DLA in 3 dimensions, however this was mostly unsuccessful due to processing limitations. Therefore if extension research was to be done, it would be ideal to firstly run the current simulations on a much more powerful system, as this would generate results for large arrays in a reasonable time.

Research on DLA is significant for many real world processes, examples of which are given in the introduction. Many systems are also not as simple as particles aggregating onto a single initial point, so another extension could be to simulate DLA with particles attaching to existing objects (for example a line or more interesting shape).

To conclude this paper's main goal was to obtain a value of d for two dimensions, which was achieved successfully. Preliminary extensions of the simulation have laid out the groundwork for future development. As an initial development, with the results and findings, the project was a success.

References

Ball, R.; Nauenberg, M.; Witten, T. A. (1984), *Diffusion-controlled aggregation in the continuum approximation* [Online], Physical Review A (General Physics), Volume 29, Issue 4, April 1984, pp. 2017-2020. Available at: <https://doi.org/10.1103/PhysRevA.29.2017> [Accessed March 2018].

Family, F.; Vicsek, T. (1990), *Simulating Fractal Aggregation* [Online], Computers in Physics Volume 4, January/February 1990, pp. 44. Available at: <https://doi.org/10.1103/PhysRevA.29.2017> [Accessed March 2018].

Sapoval, B. (2001) *Universalités et fractales* [Book] Flammarion, Champs-University, January 2001. ISBN-10: 2-08-081466-4. ISBN-13: 978-2080814661.

Stanley, H. E.; Ostrowsky, N. (1986), *On Growth and Form: Fractal and Non-Fractal Patterns in Physics* [Book], NATO Science Series E, Volume 100, pp. 3-4. Springer Book Archive ISBN: 978-94-009-5165-5. Available at: <https://doi.org/10.1007/978-94-009-5165-5> [Accessed March 2018].

Witten, T. A.; Sander, L. M. (1981), *Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon* [Online], Physical Review Letters, Volume 47, Issue 19, November 1981, pp. 1400-1403. Available at: <https://doi.org/10.1103/PhysRevLett.47.1400> [Accessed March 2018].

```
from numpy import random, pi, sin, cos, log10, unique, poly1d, polyfit, sum
from matplotlib import pyplot
from datetime import datetime
start_time = datetime.now()

radii = []; masses = []

for repeats in range(10): #loops whole fractal build for end graph
    size = 1601 #matrix size
    empty_space = 0
    matrix = [[empty_space]*size for i in range(size)] #creates empty matrix
    mid = int(size/2) #max fractal size
    maxR = int(mid/2) #max fractal radius
    matrix[mid][mid] = 1 #makes fractal origin occupied
    radius = 10 #initial diffusion start radius

    while True: #loops for multiple diffusion particles
        theta = random.uniform(0, 2*pi) #chooses random angle for diffusion start
        x = int(round(radius*cos(theta))) #transfroms polar to cartesian
        y = int(round(radius*sin(theta)))
        n = mid + x; m = mid + y #diffusion start in cartesian
        matrix[n][m] = 1 #diffusion particle start

        while True: #loops random particle movement
            matrix[n][m] = empty_space #sets previous particle location to empty
            direction = random.randint(4) #random diffusion direction
            if direction == 0:
                n = n-1
            if direction == 1:
                m = m+1
            if direction == 2:
                n = n+1
            if direction == 3:
                m = m-1
            matrix[n][m] = 1 #new particle position

            xlength = n-mid; ylength = m-mid #x and y distances from origin
            rlength = int(round((xlength**2 + ylength**2)**0.5)) #particle radius vector

            if rlength == 2*radius: #abandons particle if it diffuses too far away
                matrix[n][m] = empty_space
                break

            if matrix[n-1][m] == 1 or matrix[n][m+1] == 1\
            or matrix[n+1][m] == 1 or matrix[n][m-1] == 1: #particle joins cluster
                if rlength > radius:
                    radius = rlength #diffusion start radius increases as cluster grows
                    radii.append(radius) #radius and mass values taken
                    masses.append(sum(matrix))
                break
        if radius == maxR: #cluster stops growing when it reaches predetermined size
            break
    print('Fractal', repeats+1, 'complete. Runtime =', datetime.now() - start_time)

logR = log10(radii); logM = log10(masses) #logs radii and masses for graph

pyplot.figure(figsize = (12,12))
pyplot.scatter(logR, logM, c='k', marker='.')
pyplot.plot(unique(logR), poly1d(polyfit(logR, logM, 1))(unique(logR)), linewidth=3, c='r')
pyplot.xlabel('logR', fontsize=20)
pyplot.ylabel('logM', fontsize=20)
pyplot.show() #plots data points with first order polynomial fit

pyplot.figure(figsize = (12,12))
pyplot.scatter(logR, logM, c='k', marker='.')
pyplot.plot(unique(logR), poly1d(polyfit(logR, logM, 1))(unique(logR)), linewidth=3, c='r')
pyplot.plot(unique(logR), poly1d(polyfit(logR, logM, 2))(unique(logR)), linewidth=3, c='c')
pyplot.xlabel('logR', fontsize=20)
pyplot.ylabel('logM', fontsize=20)
pyplot.show() #plots data points again with second order polynomial fit on top

gradient, grad_err_squared = polyfit(logR, logM, 1, cov=True) #calculates d and error
print('d = ', gradient[0], '+/-', (grad_err_squared[0][0])**0.5) #extracts matrix elements
print('Total runtime = ', datetime.now() - start_time) #program runtime
```