SHARIF UNIVERSITY OF TECHNOLOGY
PHYSICS DEPARTMENT

Problem Set 2

Student name: *Ali Fayyaz - 98100967*

Course: *Computational Physcis - (Spring 2023)*
Due date: *February 24, 2023*

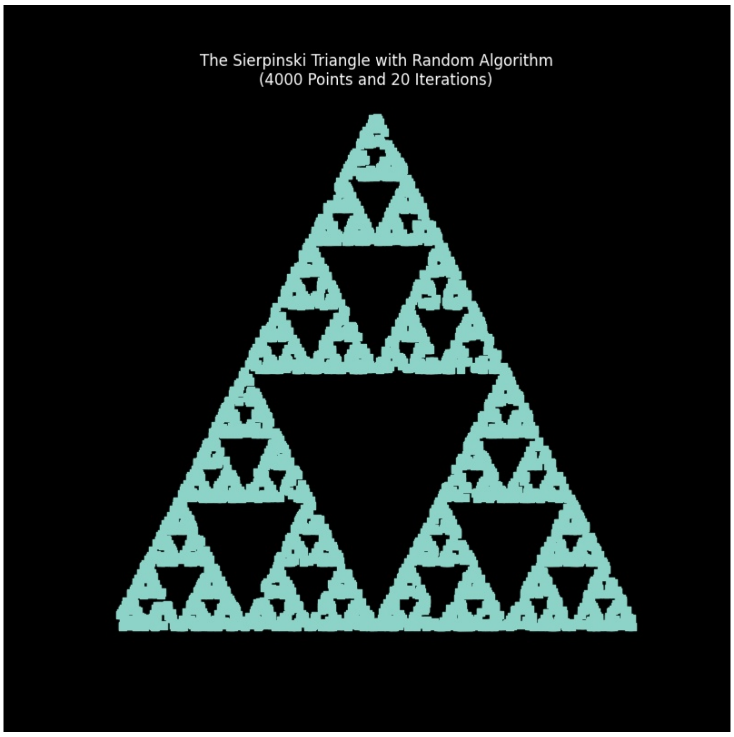
Exercise 2.5

Create the Sierpiński triangle with the random algorithm described in section 2.3 of the text book.

Answer. The approach taken to solve the problem was to utilize linear transformations on points on a plot. A first point is declared, 3 functions were defined to scale the point in 3 different ways, and the said functions were randomly chosen 20 times to be applied on the point, using a for-loop. The generated points were all gathered in lists and plotted.

- **f1()**, **f2()** and **f3()**:

  These functions take a point tuple. Then push the point to the lower left corner, lower right corner and the upper corner, respectively.



Exercise 2.6

Generate the Barnsley Fern Leaves fractal.

Answer. Again the key to generating this fractal lies in the correct form of linear transformations of an arbitrary point. Searching the web would yield the following transformation matrices as the scaling factors:

$$f1 = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$$

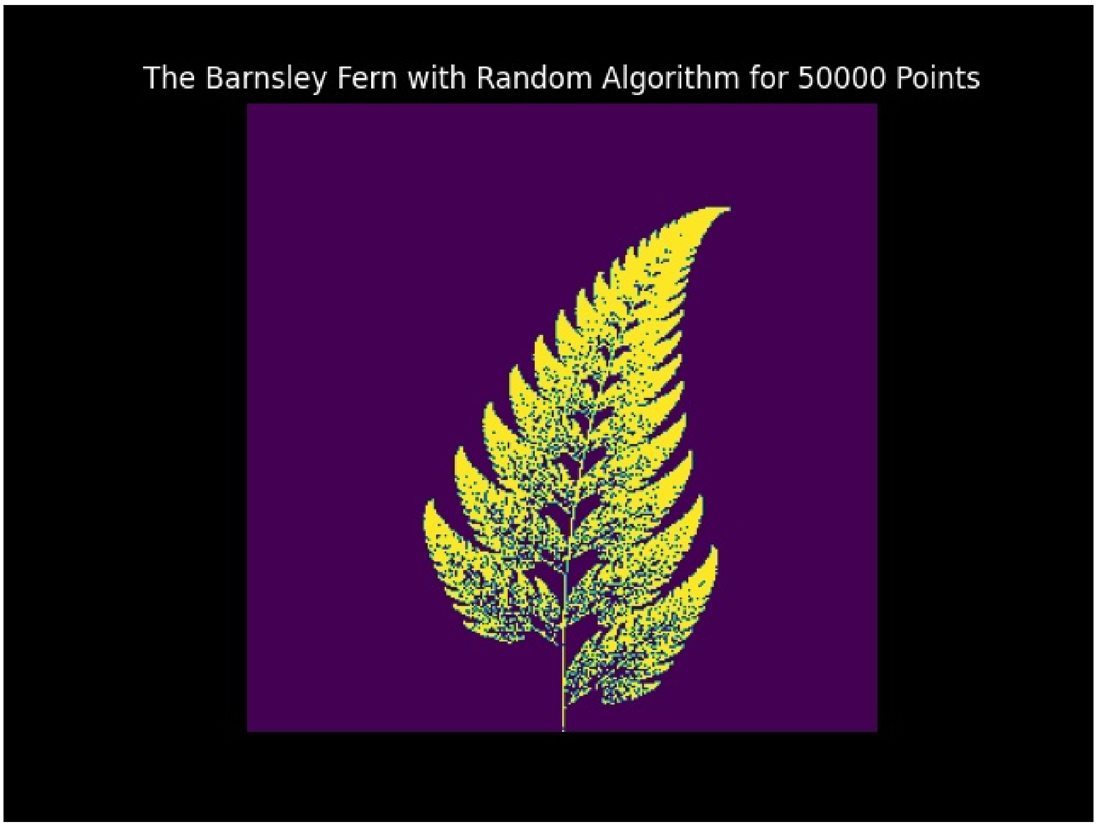
$$f2 = \begin{bmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$$

$$f3 = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.44 \end{bmatrix}$$

$$f4 = \begin{bmatrix} 0 & 0 \\ 0 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The corresponding functions **f1()**, **f2()**, **f3()** and **f4()** were declared. These functions were chosen in a weighted random fashion to act on a point tuple. It must be emphasized that the functions are not chosen with equal probabilities, rather the probability of choosing them is as follows: [85%, 7%, 7%, 1%]

The transformed points were put in a 2D *Numpy* array and finally plotted. The algorithm and the probability distribution were adapted from scipython.com.

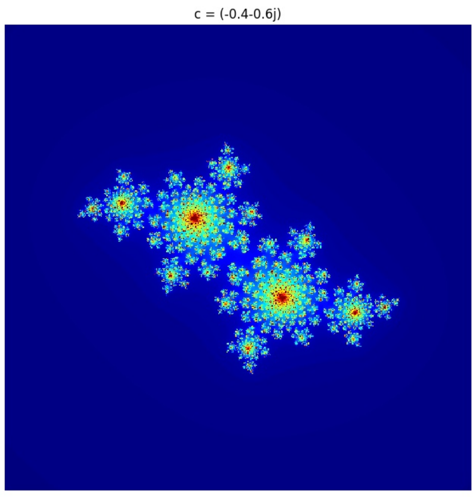


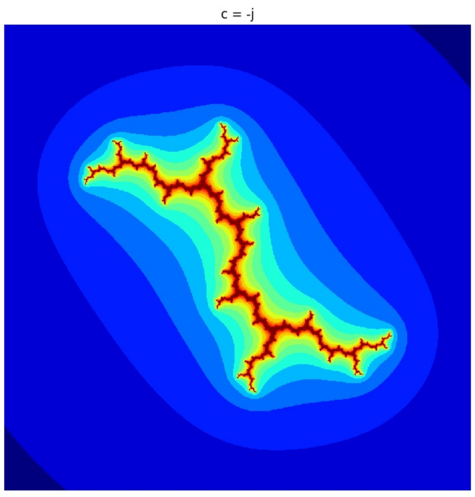
Exercise 2.7

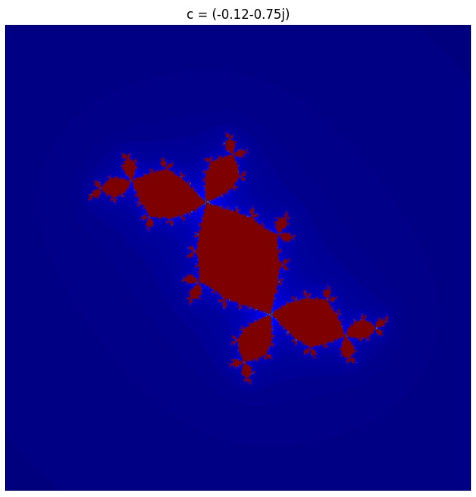
Generate the Julia Set Fractals for different values of c.

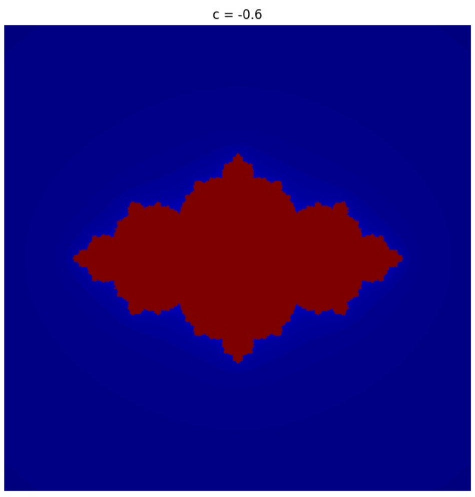
Answer. A boundary was defined to determine divergence (max distance from the origion). If after a sufficiently large number of iterations, the point were still bounded, then one can assume that the value will not diverge in the future. Another variable for this "sufficiently large number" was defined. Also, each point on a plot (equivalently each pixel on an image) was declared with a complex number.

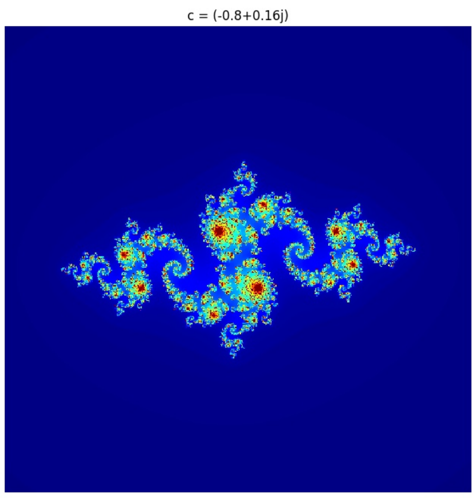
Using a nested for-loop, the Julia function $f(z) = z^2 + c$ was iteratively applied to the points. The ratio of the number of iterations needed for a point to diverge, to the maximum number of iterations, i.e. the "sufficiently large number", gives a useful scale for the image points to be mapped to a color range; implementing this would give a color to every point on the plot, and the Julia set for the corresponding c will emerge.

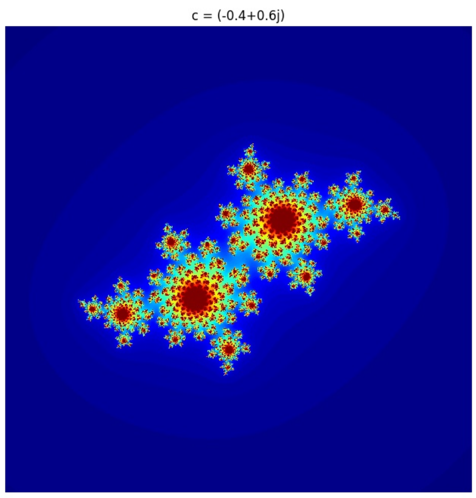












Exercise 3.1

Simulate a Random Ballistic Deposition model for a line of width 200 units.

(a) Show the dynamics of the model on the screen.

(b) Calculate the mean height and the roughness (standard deviation) of the surface for a number of time intervals.

(c) Sketch the surface roughness vs. time.

(d) Calculate β and explain what you understand from its value.

Answer. Two seperate methods were implemented for this problem:

The first one, the *Histogram Method*, was rather a trick at graphing the model. Basically, this method implements the stacked histogram of 4 sets of data with 200 bins of random heights.

The second, the more systematic *Animation Method* executes the algorithm described in section 3.2 of the text book. *Numpy* arrays were used to show the number of particles in each position (bin). Then, plotting these arrays, an animation of the deposition was created.

After graphing the model, the data generated from the latter method, containing the information of 100,000 particles in total, was used for the next parts of the question.

Numpy array methods were used to calculate the mean height and the standard deviation of the surface for every 25,000 particles that were deposited.

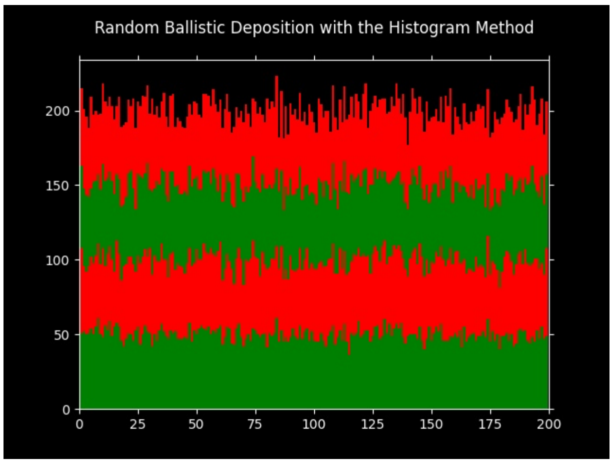
Again, utilizing *Numpy* arrays and *seaborn.regplot()*, the roughness $w(t)$ was graphed for 25 layers (every 5000 particles).

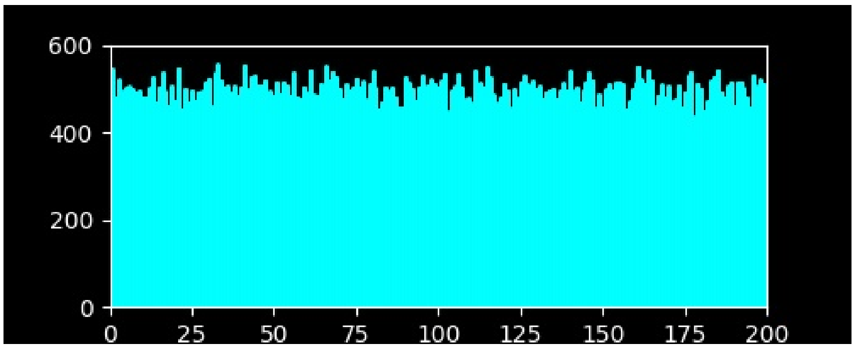
In the end, the function **w_func()** was defined for $w(t) = at^{\beta}$ and *scipy.optimize.curve_fit()* was used to estimate β. a and β were obtained to be, respectively:

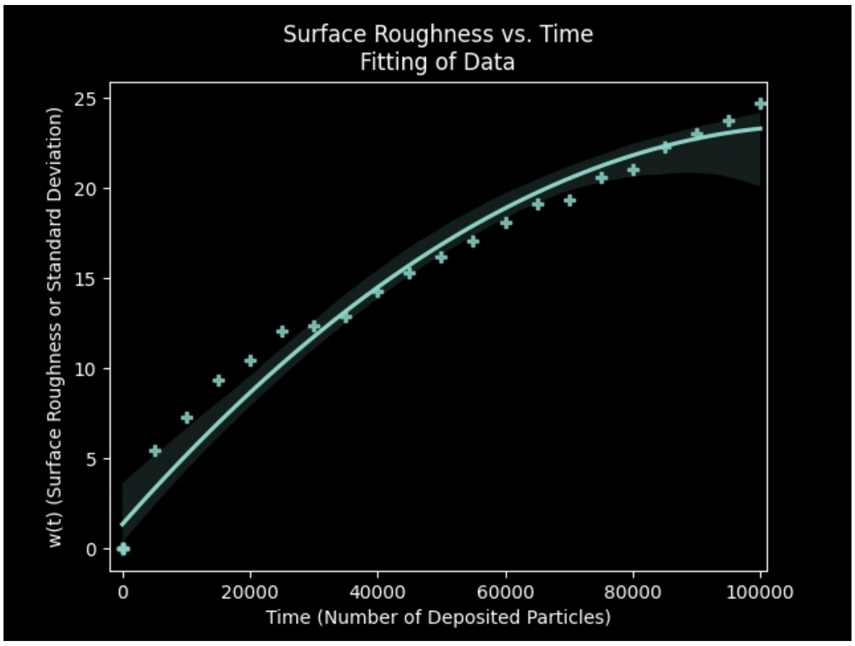
$$a = 0.0506 \qquad \beta = 0.5353$$

This means the roughness increases with approximately $\frac{1}{2}$ the power of time, or:

$$w(t) \sim \sqrt{t}$$


Random Ballistic Deposition with the Histogram Method


0
200
400
600
0
25
50
75
100
125
150
175
200


Surface Roughness vs. Time
Fitting of Data
w(t) (Surface Roughness or Standard Deviation)
Time (Number of Deposited Particles)