Computational Physics Homework Report

Fateme RamezanZade-99100693 Problem Set 2

1. Sierpinski triangle(rand)

1.1 Functions Description

Homogeneity_func:

It takes an array which contains the x coordinate and the y coordinate of a point. It multiples the distance of the point from the origin array by ½ and returns the resulted array.

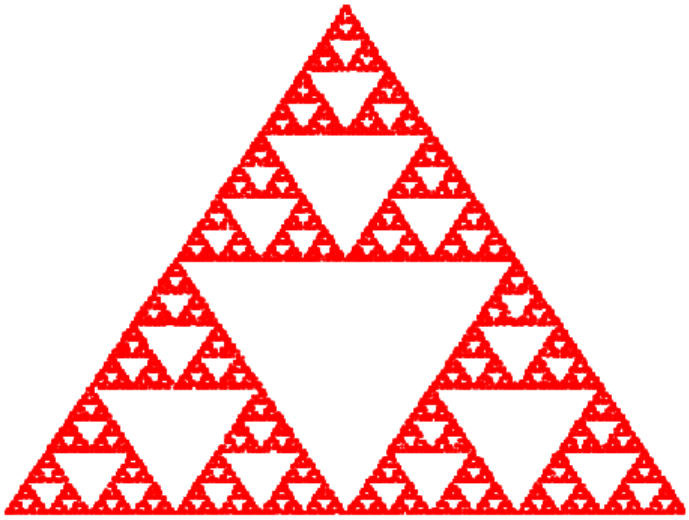
Shift_func:

It takes two arrays, one contains the x coordinate and the y coordinate of point and the other contains the shift horizontal and vertical lengths. Then adds the horizontal length to x and the vertical length to y coordinate and returns the new array of shifted point.

1.2 Main Code

Firstly, a loop is started to select a point. This loop would repeat for 2000 times in order to choose 20000 random points. Inside each turn, two integers from the interval [0,100] are randomly selected. These two are then included in an array where they would correspond to x and y coordinates of a point. Now another loop starts which is going to repeat for 20 times. The number of times we need to apply our functions until we get the self-similar fractal shape. In this loop we are going to choose a random number between 1 and 3, then apply functions to our point according to this number. If the number is 1, we will apply Homogeneity_func to our point. If it is 2 or 3, we will apply Shift_func with different amount of shift for each. We note that the exact amount of shift lengths won't matter, but we need to keep the right ratios to achieve the final result. After applying each function, the values are updated to the coordinates of the new point. After this loop ends, we plot the final point using plt.plot.

1.3 Results



Sierpinski triangle plotted by applying the functions 20 times for 20000 random points

2. The Barnsley Fern

2.1 Functions Description

Homogeneity_func:

It takes an array which contains the x coordinate and the y coordinate of a point and an array which contains the ratios for x and y axis. It multiples the horizontal distance of the point from the origin by x ratio and the vertical distance of the point from the origin by y ratio and returns the resulted array.

Shift_func:

It takes two arrays, one contains the x coordinate and the y coordinate of point and the other contains the shift horizontal and vertical lengths. Then adds the horizontal length to x and the vertical length to y coordinate and returns the new array of shifted point.

Rotation_func:

It takes an array which consists of the coordinates of a point. Next uses the result of applying the rotation matrix in order to rotate with amount (angle) radians about the origin and returns the new array of shifted points.

Reflection_func:

It takes an array which consists of the coordinates of a point. Then multiplies the x coordinate by -1 in order to reflect the point about y axis and returns the new array of reflected point.

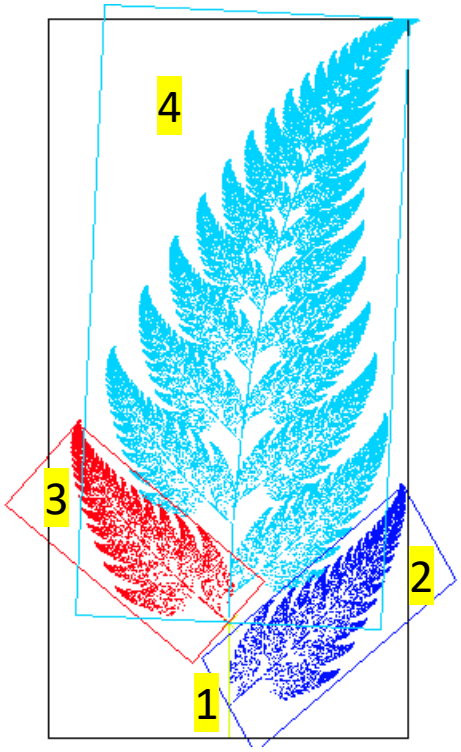
2.2 Main Code

Firstly, a loop is started to select a point. This loop would repeat for 50000 times in order to choose 50000 random points. Inside each turn, two integers from the interval [0,100] are randomly selected. These two are then included in an array where they would correspond to x and y coordinates of a point. Now another loop starts which is going to repeat for 20 times. The number of times we need to apply our functions until we get the self-similar fractal shape. In this loop we are going to choose a random number between 1 and 10, then apply functions to our point according to this number. The functions are arranged in a way that correspond to each region of figure 1. If the number is 1, we use set of transformations that correspond to region 1. The same happens for 2 and 3. If the number is larger than 3, we use set of transformations that correspond to region 4. This way we are increasing the number of times that region 4 points are created.

We note that the exact amount of shift lengths and ratios won't matter, but we need to keep the right ratios between them to achieve the final result. After applying each function, the values are updated to the coordinates of the new point. After this loop ends, we plot the final point using plt.plot.

2.3 Results

The Barnsley fern plotted by applying the functions20 times for 50000 random points

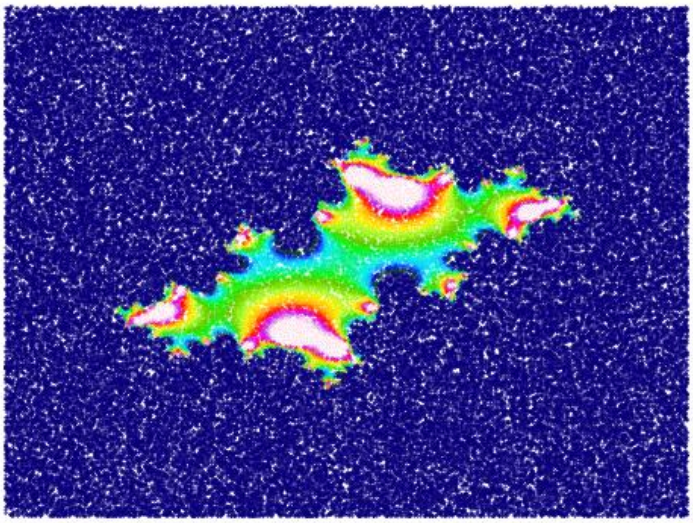


3. Julia Set

3.1 Main Code

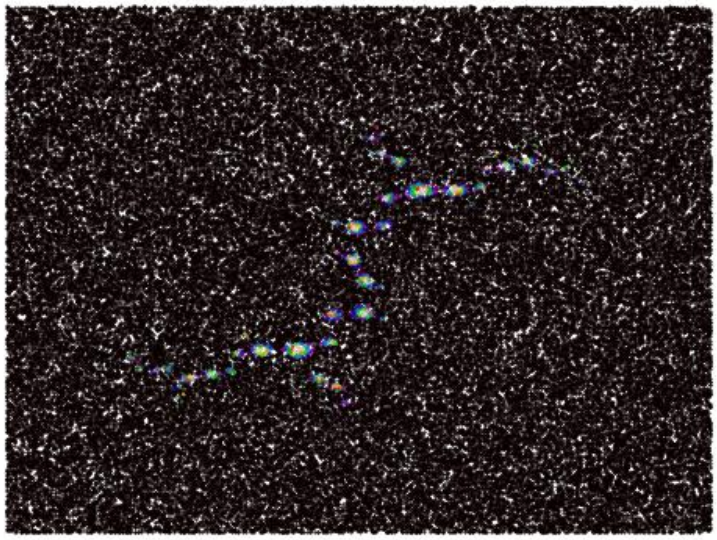
Two points are selected randomly, and a complex number is made with them. Then we find the distance of the point from the origin, after the function f(z)=z^2+c is applied to it for 7 times. Now we assign a color to this length using a colormap, then plot the original points with the assigned color. This process is repeated 100000 times for random points and results in a colormap where Julia set (the border separating numbers which their distance from the origin approaches infinity as we keep applying the function to them and numbers which their distance approaches zero.) can be seen.

3.2 Results



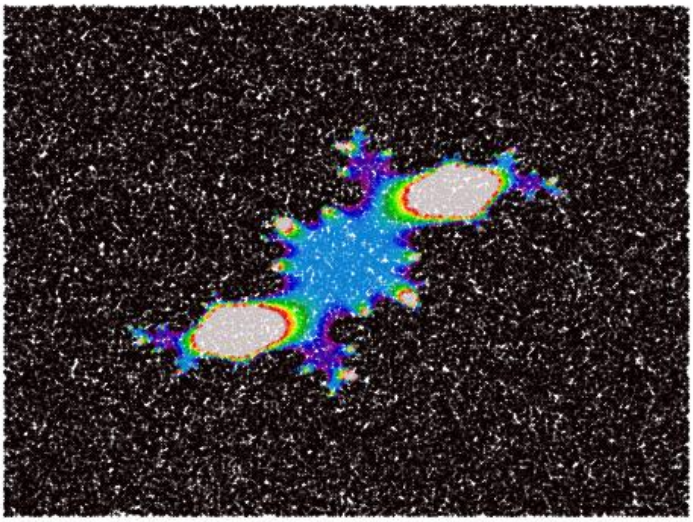
$c = -0.4 - 0.6i$

Colormap: gist_ncar



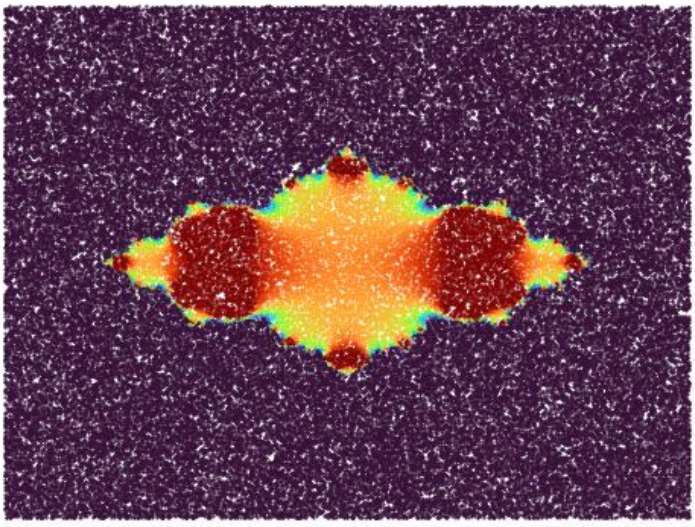
$c = -i$

Colormap: nipy_spectral



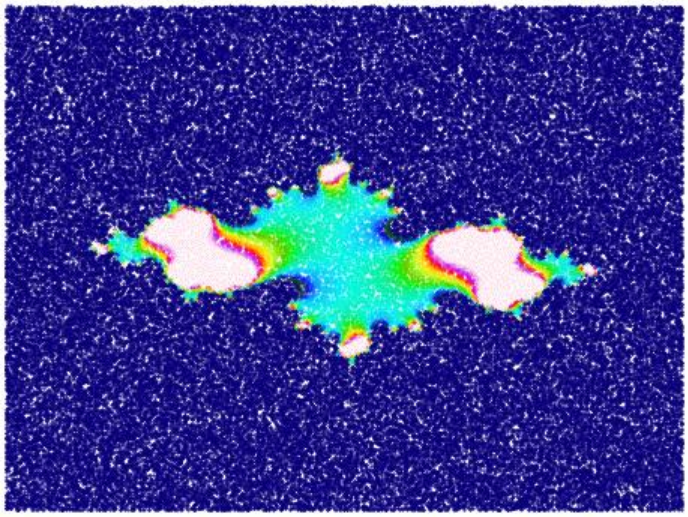
$c = -0.12 - 0.75i$

Colormap: nipy_spectral



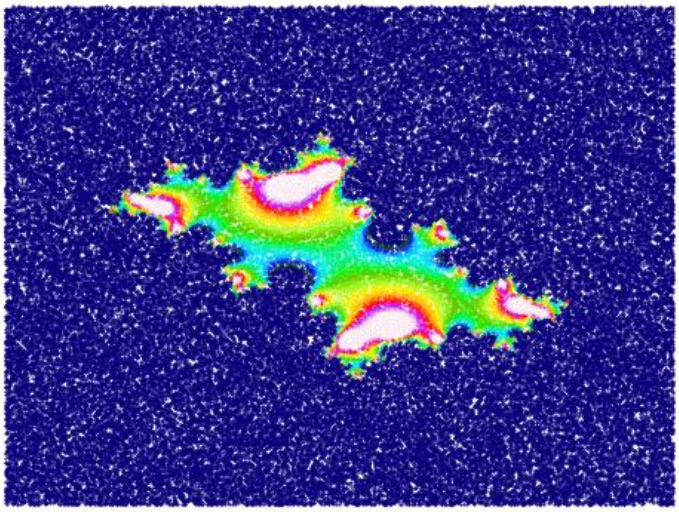
$c = -0.6$

Colormap: turbo



$c = -0.8 + 0.16i$

Colormap: gist_ncar



$c = -0.4 + 0.6i$

Colormap: gist_ncar

It is worth mentioning that in order to be able to use from a wider range of the colors in the colormaps, I used the amount (ln(1/distance)) instead if distance itself.

4. Random Deposition

4.1 Functions Description

Mean:

This function takes an array, calculates sum of all its elements and divides the sum by the number of elements. Then returns the amount, which is the mean.

Dev:

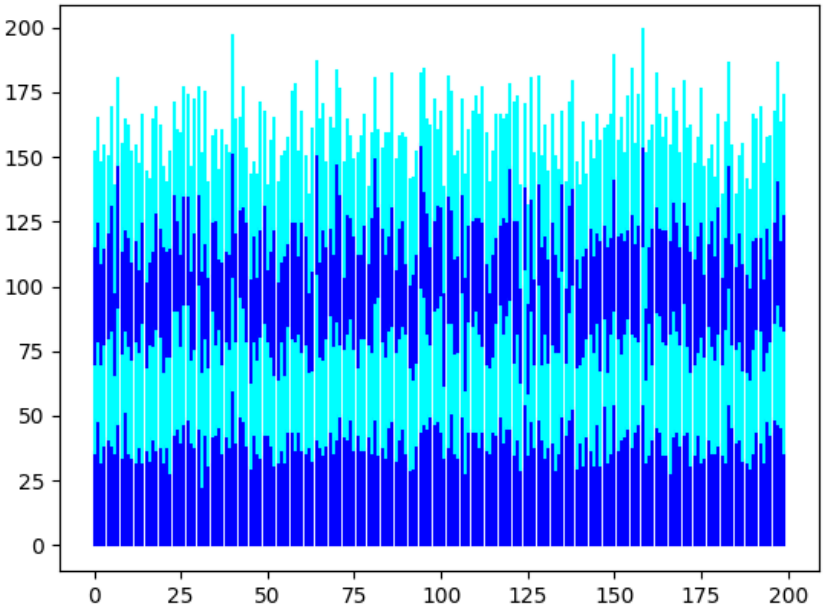
This function takes an array, first calculates the mean of its elements and then calculates the mean if the squared amount of its elements. Finally, uses standard deviation formula for them and returns the result which is the standard deviation of the elements of array.

4.2 Main Code

Firstly, an array of 200 elements is defined and each element is set equal to zero. Then in a loop which repeats 8000 times, a random number is chosen, and the element with that number from the array is increased by 1. After the loop ended, a copy of the resulted array is made. This process is repeated three times. Now for each element of the array, we plot some vertical lines. The first is plotted from the x axis up to the first copy of h, with blue color. Then we plot a line from the first copy to the second one, with cyan color. We repeat this two other last time with blue and cyan. This process of changing lines helps us see how the upper edge would look at some definite times.

4.3 Results

Each time before making a copy, the mean amount of h and its standard deviation is calculated.



$\bar{h}(8000\tau) = 40$

$w(8000\tau) = 6.47302$

$\bar{h}(2 \times 8000\tau) = 80$

$w(2 \times 8000\tau) = 9.93227$

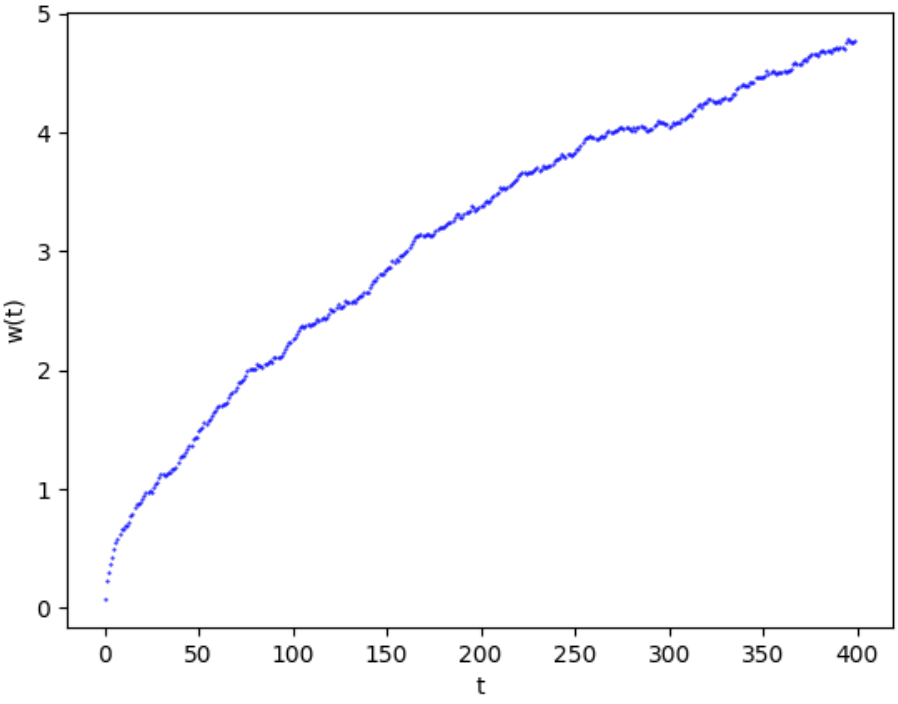
$\bar{h}(3 \times 8000\tau) = 120$

$w(3 \times 8000\tau) = 12.09628$

$\bar{h}(4 \times 8000\tau) = 160$

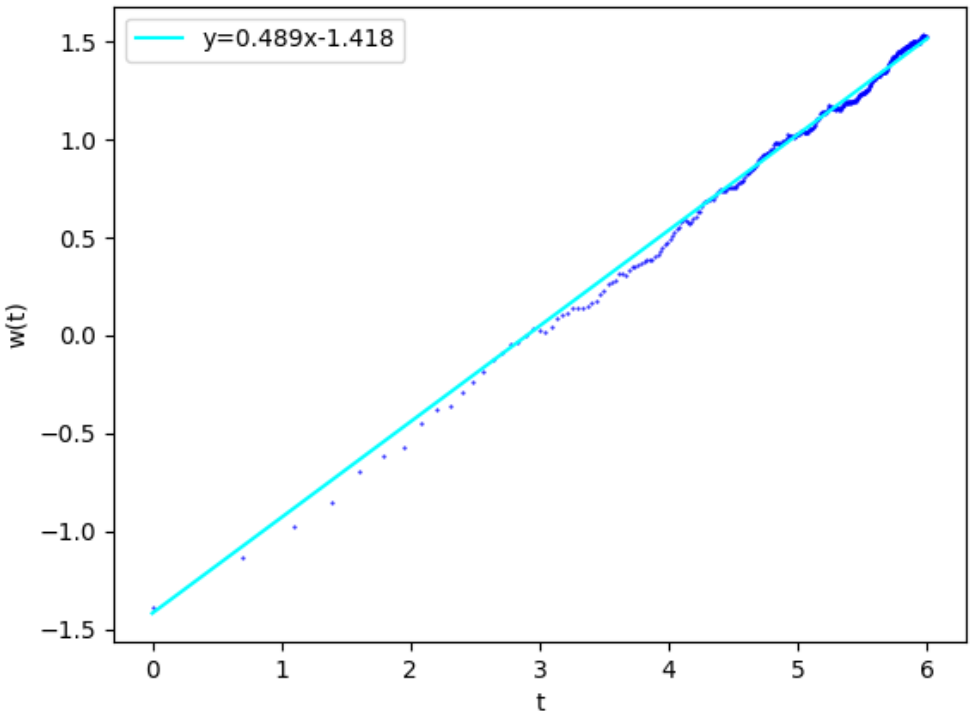
$w(4 \times 8000\tau) = 13.22989$

We can see that $\bar{h} = \frac{t\tau}{l}$ as we expected. Now we want to find out how w changes with time. Here I have plotted a graph of w(t) for each 10τ:



This plot is a t^{β} graph. So I plotted a log-log graph and fitted a line to it using least-squares method.

And we see that $\beta = 0.498$.


y=0.489x-1.418
w(t)
t
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5
6