1 Random Sierpinski Triangle

In order to create the Sierpinski triangle, we randomly generate points onto the screen, then we choose one of the three transformations below randomly and apply that to the coordinates of the dot to create the shape.

$$V = \frac{V}{2}$$
$$V = \frac{V}{2} + (1, 0)$$
$$V = \frac{V}{2}$$

In which V represents a vector that shows the position of a point.

1.1 Starting functions

We define the three functions, f1, f2 and f3, to be able to easily apply the transformations that have been brought up above.

1.2 Plotting

We use the standard plotting library and a scatter plot to illustrate the results.

1.3 Results

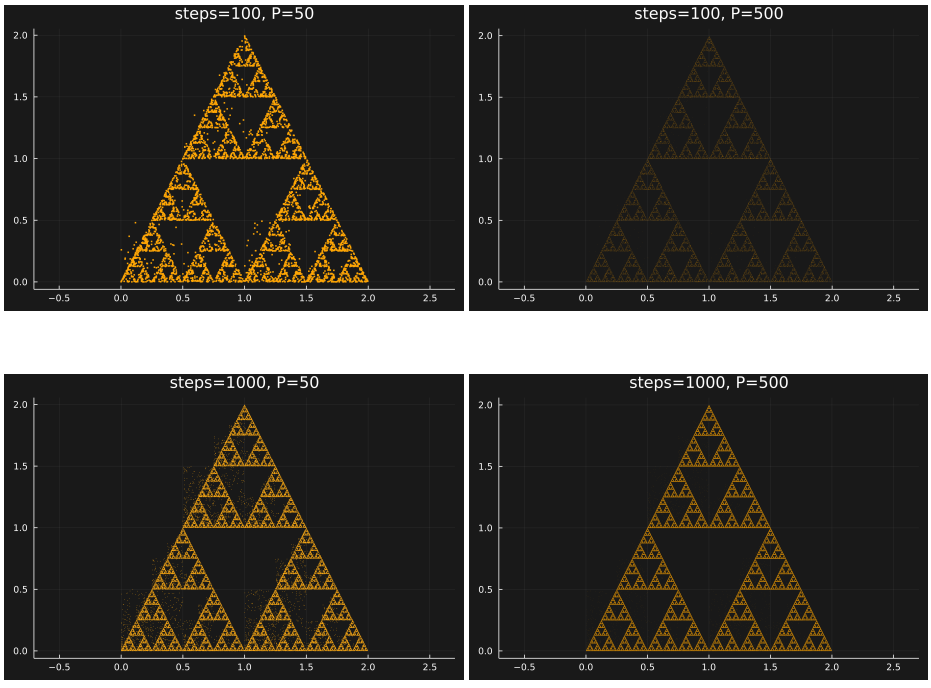

Figure 1: Results for the random Sierpinski trangle.

2 Barnsley's Fern

We use an algorithm similar to the sierpinski problem but this time we choose from four transformation functions that represent the translations and rotations needed to create the stem, right leaf, left leaf and the layer of leaves above. The details are similar to the previous problem so we won't go into detail.

2.1 Variations

By changing the starting parameters, we could create different ferns. For example by adding a scale in the x axis to the first function, we could eliminate the stems and turn them into more of a leaf shape. Also by changing the scales and translations of the other threee functions, the distance between the leaves and the angles will differ.

2.2 Results

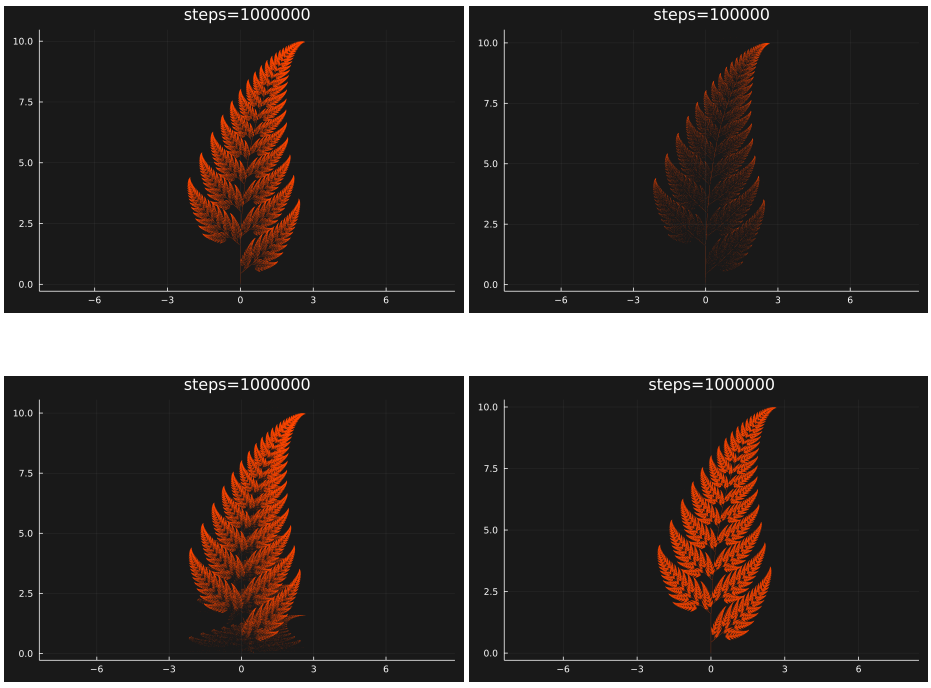

Figure 2: Results for the Barnsley's Fern fractal.

3 Julia set

In order to illustrate the julia sets, we first define a function to test if the transformations done on the points go to converge, diverge or somewhere in between then associate a color to the point based on this. We then plot the results using a heatmap.

3.1 JuliaColor() function

This function takes the coordinates of a point, the width and height of the plot and c as input, then outputs a number between -255 and 255 based on the result of the transformations, which represent the color of the point. We have also applied some translations and scalings on the coordinates to center the fractal.

3.2 JuliaSet() function

This function takes the height and width of the plot and c as input, then calls the JuliaColor() function multiple times to create an array of the color values of each point. Then it returns this array which is later used to plot the set of data.

3.3 Plotting

To plot our data we use the standard Plots.jl library but this time we use a heatmap which uses the the array we created to assign a color to each point of the screen.

3.4 Results

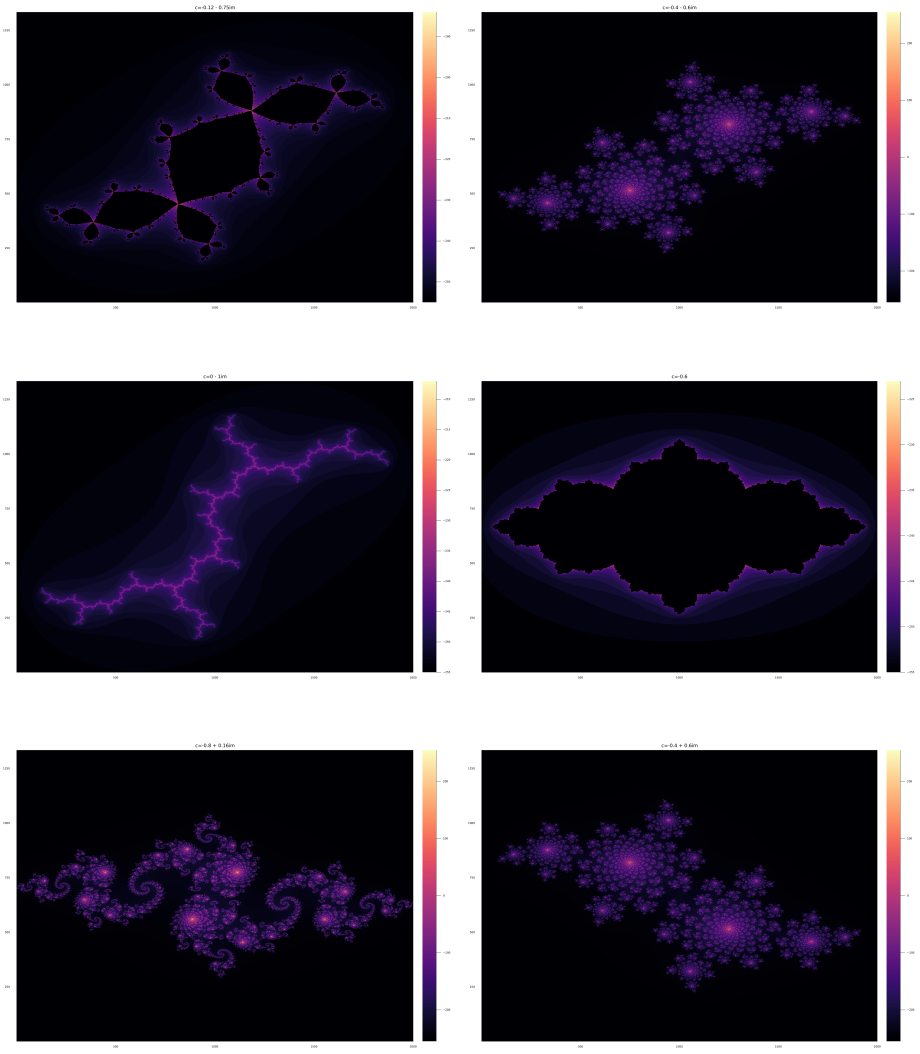

Figure 3: Juliaset results based on different values of c.

4 Random Ballistic Deposition

For this problem we create two files. rbdgraph.jl which is used to plot the overall distribution of the deposited dots using a bar graph, and rbddata.jl which is used to compute $\bar{H}_{(t)}$ and $W_{(t)}$ then it plots the results and tries to fit a curve to the data.

4.1 rbdgraph.jl

To draw the graphic we first define a function that deposits $10 \times L$ dots at random. we then run this function four times and draw the deposited dots of each loop in a different color.

4.1.1 deposit() function

This function inputs L(The number of available bins) and the number of dots that are supposed to be deposited. It then outputs an array with the number of dots deposited in each bin.

4.1.2 Plotting

We use StatsPlots.jl for this plot because it has support for stacked bar plots. We then plot the arrays we have calculated in different colors.

4.1.3 Results

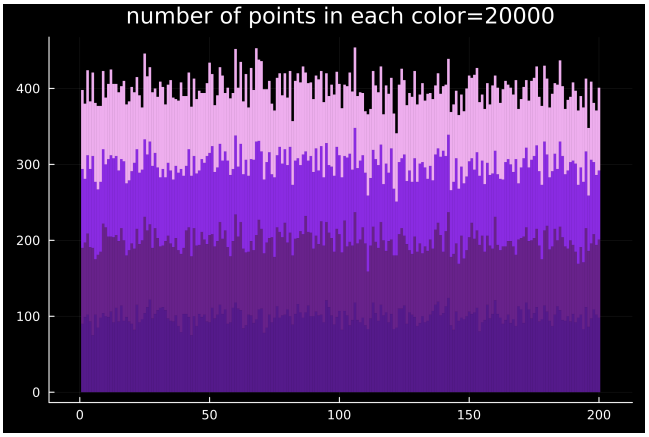

Figure 4: Random Ballistic Deposition Result Graphic.

4.2 rbddata.jl

4.2.1 deposit() function

This function is the same as before but this time the function also takes time as an input and deposits dots up until that time with the rate of 1 dot per second.

4.2.2 Hbar() and Hbar2() functions

The Hbar function takes the array H representing the heights of the bins as an input and calculates the average of the heights. The Hbar2() function does the same but calculates the average of height to the power of two.

4.2.3 WGen() function

This function takes the heights array, H as input and outputs W calculated by the formula below:

$$W = \sqrt{\bar{h^2} - \bar{h}^2}$$

4.2.4 Fitting Curves

To fit lines and curves to our data, we use the Polynomials.jl package. We use the fit(x, y, deg) function that returns a polynomial of the degree "deg".

4.2.5 Plotting

We use a scatter plot that we have used before to represent the data. In one plot we plot $\bar{h}$ based on t and in the other we plot $Log(W)$ based on $Log(t)$. We then add the fitted polynomials on top of the plot.

4.2.6 Results

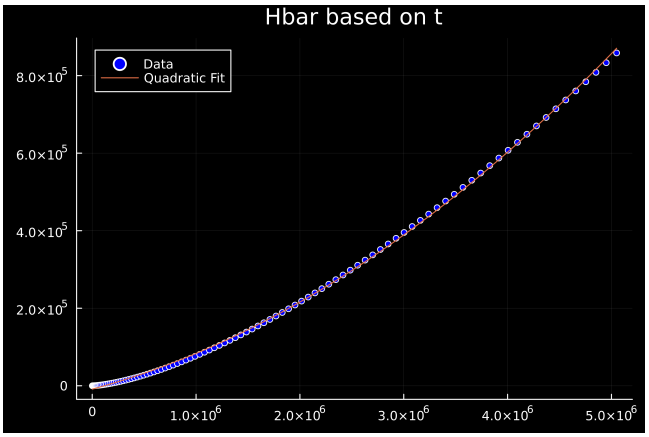

Figure 5: Hbar based on t with a fitted 2nd degree polynomial.

Fitted polynomial:

$$Hbar \simeq -8677 + 0.0718t + 2.026t^2$$

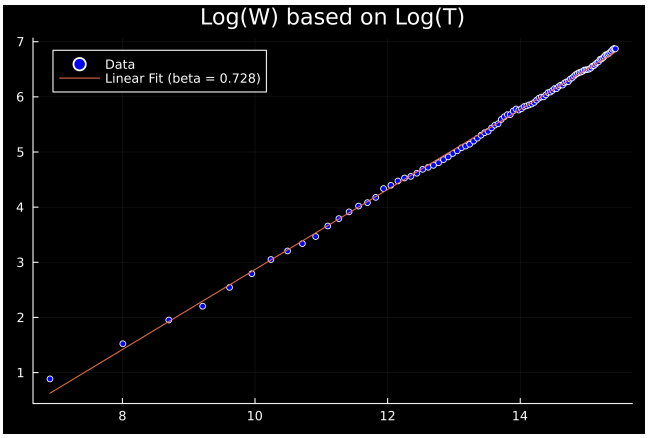

Figure 6: $Log(W)$ based on $Log(t)$ with a fitted line.

Fitted line:

$$Log(W) \simeq -4.534 + 0.735 Log(t)$$

Therefore for the value of β:

$$\beta \simeq 0.735$$