Computational Physics: Problem Set 1

Mohammad Ghoraishi
99100788

February 17 2023

1 Koch Fractal

We use one recursive function to generate the fractal. The program takes the coordinates of the first line and the number of steps as initial values.

1.1 lineSplit() function

this function takes the coordinates of the two ends of a line and returns the coordinate of three new dots that make up the koch fractal shape. in the transformations we use the rotation matrix with $\theta = 60$. Within the function there is an if statement that calls the function recursively and creates the neccessary dots for plotting the entire fractal.

1.2 Plotting

We use plot.jl for all of the plots. We only need the coordinates of the dots that need to be attached but we have to make sure the order of the coordinates are correct. The plot function will take care of the rest and the dots will be connected.

1.3 Results

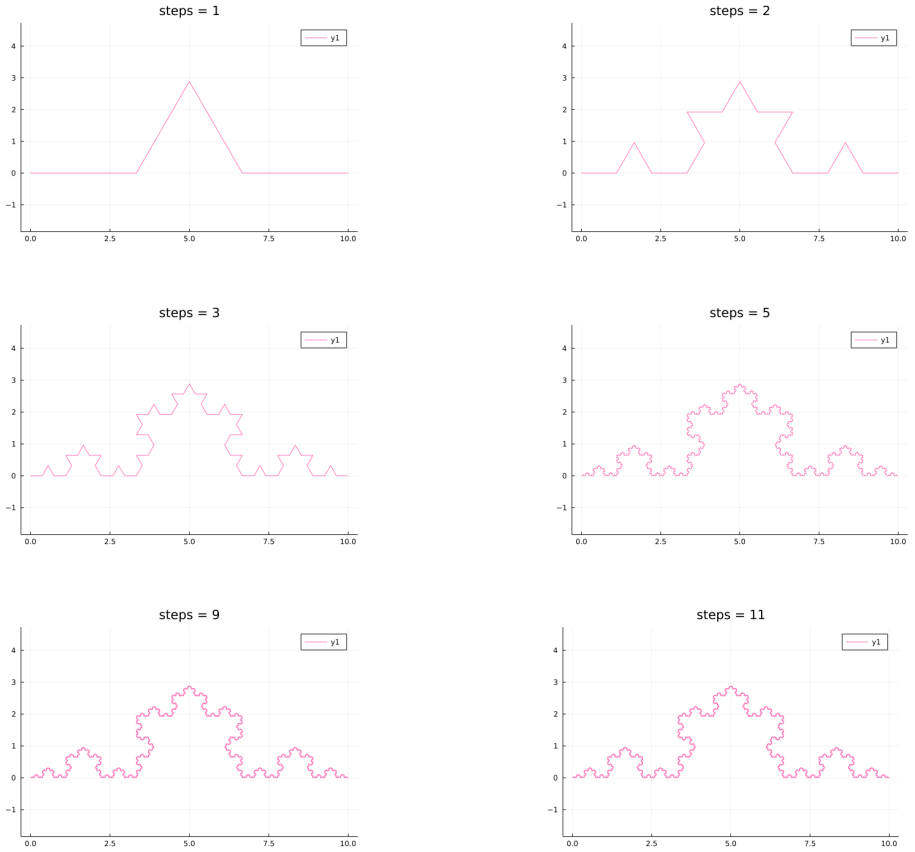

Figure 1: Koch fractal results for different step counts.

2 Heighways Dragon Fractal

For this fractal we use three functions, one for creating a point with a clockwise rotation, one for creating a point with an anticlockwise rotation and a recursive function that implements the two former functions to create the fractal.

2.1 LineSplitClockwise() and LineSplitAnticlockwise() functions

Both these functions take two points as an input and return the coordinates of a new point between these dots with either a clockwise or anticlockwise rotation. In both these functions we use the rotation matrix with $\theta = \pm 45$.

2.2 fractalize() function

This function takes the coordinates of two dots, the number of steps and an i value that is used to determine which rotation function we should use. The

function oscillates between clockwise and anticlockwise rotations in each segment. There is also an if statement that makes the function recursive so the output will be the coordinates of all the dots needed to plot the fractal (with the exception of the first dot which we add later).

2.3 Plotting

The plotting code used is similar to the one used in the koch fractal so we won't expand on it further. The only difference is that we use the fractalize() function twice for two different initial lines and plot them on top of eachother with different colors.

2.4 Results

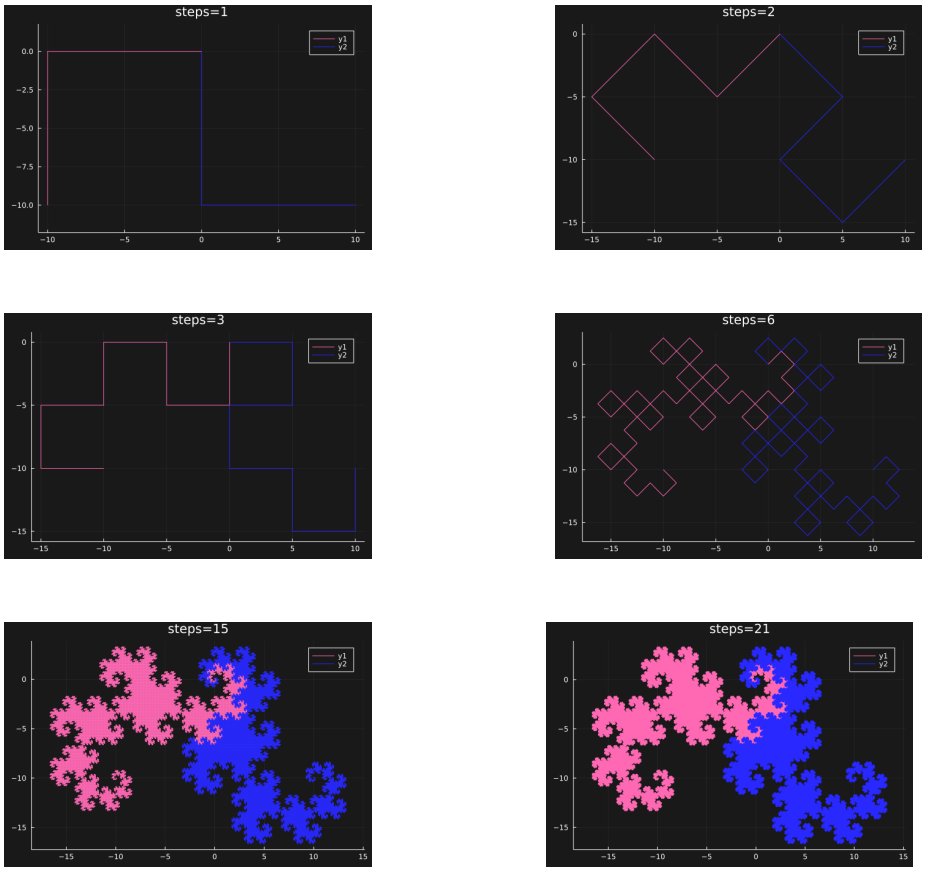

Figure 2: Heighways dragon fractal results for different step counts.

3 Sierpinki Fractal

To draw the Sierpinski triangle we only use one function which in essence is really simillar to the koch program.

3.1 newdots() function

This function takes the coordinates of the center of a triangle, the number of steps and a value i that is used to choose the correct scale for the transformations. This function returns the coordinates of three dots that make up the centers of the triangles of the next step. There is also a recursive if statement that constructs the entire fractal.

3.2 Plotting

The plotting method used in this program is different from the previous two. First we define a triangle with equal sides as a shape and use that as a marker in a scatter plot which each of the dots placed on the graph represent the center of a triangle. We also scale the size of the marker using the number of steps.

3.3 Results

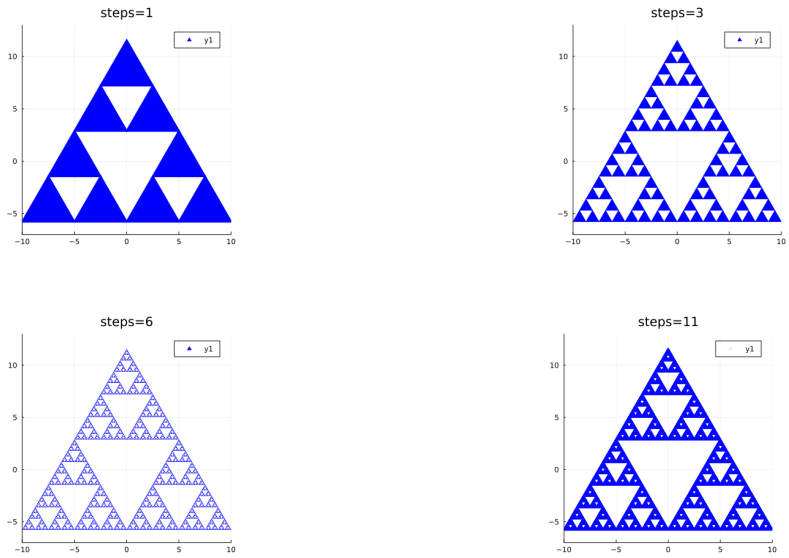

Figure 3: Sierpinski fractal results for different step counts.

4 Khayyam Triangle

We take a completely different approach to this problem and we won't use a recursive function to construct the shape and we will use only one function to get the job done.

4.1 KhayyamNumber() function

This function takes two numbers representing the index of the cell in the khayyam triangle and returns the number that should be placed in that cell using the formula below:

$$num = \frac{n!}{m!(n-m)!}$$

4.2 Nested if statements

The nested if statement is used to go through all of the cells of the Khayyam triangle, check if each cell is odd or even and put the coordinates of each cell with an odd number in an array which will later be plotted.

4.3 Plotting

We use a scatter plot and a square marker without a fixed size as our means to show the data with the coordinates of each dot being the center of a square. With the increase of the steps variable, the shape created more and more represents a Sierpinski triangle.

5 Results

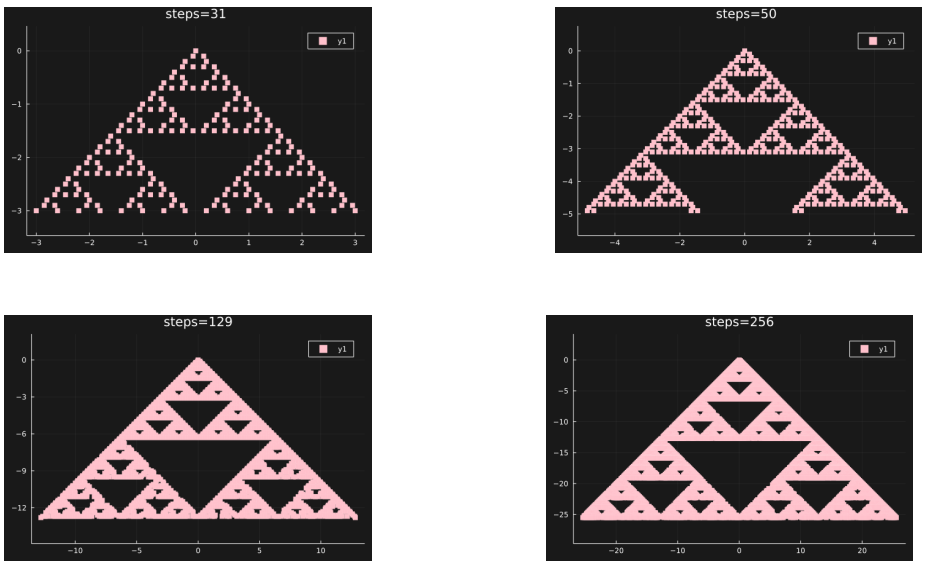

Figure 4: Khayyam Triangle results for different step counts.