Homework1

Maedeh Karkhane Yousefi-98100991

September 27, 2021

1. Koch Curve

For this coding of this exercise, two main functions on each vector is operated. At the beginning we have a list containing two points with arbitrary components. In every step the variable i is set to *1* and the operation happens while i is not equal to the total number of points we have in the list. the first point that is inserted is the point that is transferred in the amount of 1/3 vector. the next point is first transferred in the amount of 2/3 vector and then rotates 60 degrees around the previous point. It's obvious that we need to have some kind of coordinated transition,too. Overall three points are inserted between the head and tail of the previous vector, all of which are shown in the figure below.

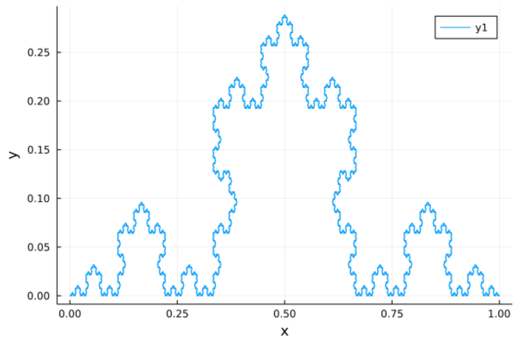

Figure 1: Koch Curve is plotted for 10 steps. The initial points are:((0,0), (1,0))

2. Dragon Fractal

our initial condition is two vectors that make a 90 degree angle with each other. The algorithm is just like the algorithm, used for Koch Curve. the main difference is that the rotation function one makes a 45 degree and once a -45 degree rotation.

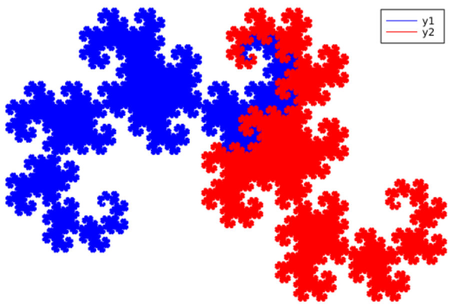

Figure 2: Dragon Fractal is plotted for 20 steps. The initial points were: ((0.00,0.00),(0.50, 0.49),(1.00,0.00))

3. Sierpinski Triangle

In this code I stored the final triangles that are produced at the end of the process in separated lists, all in one $resutl_triangles$ list. For each triangle in the $result_triangles$, we make three empty lists representing the three triangles that produce per triangle. the first innermost *for loop* goes through each point of the particular triangle to scale the points, creating the first new triangle and pushes the components of the vertices into the $first_triangle$. the second innermost *for loop* pushes the components of the second and third new triangles that are in fact the transferred vertices of the first new triangle. Eventually the $result_triangles$ becomes the $newOnes$, which consists of these three new triangles. what I thought at first was that I need **all** the triangles made along the process, which was not the case at all!
At last I built a loop over each triangle to plot this figure below.

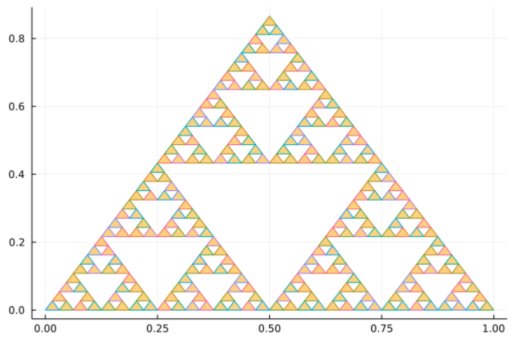

Figure 3: Sierpinski triangle is plotted for 5 steps.The main triangle's vercies are: ((0.0,0.0), (0.5,sqrt(3)/2),(1.0,0.0],[0.0,0.0))

4. Khayyam Triangle

For this code I intended to store all the khayyam numbers in a list named **numberslist**. So, basically I added the first two elements of this series myself, and started the loop for creating the numbers from step 3. I set a condition that the first and the last element of each series of numbers in each step, is *1* and new numbers are created as shown bellow:

$$number_{R,n} = number_{R-1,n-1} + number R - 1, n$$

consider R as each row of elements in the Khayyam triangle, and n as the number of each element in one row.
At the end all of these numbers are pushed into a list named *R* containing the fist and last number *1*. All of these *R*s are pushed into the main list.
for plotting, if we consider the starting point at (x,y)=(0.0, 0.0), we can observe a pattern in the sequence of the elements' positions, such that each row decrease by one *y-=1*) and each element in that row increase by one(*x+=1/2*), taking note of the fact that the x component of the first element of each row is the row's number multiplied by 1/2.
In the loop it will be check whether the number is even or not, and then it's position pushed into even or odd number's poses' list,respectively to be drawn with particular colors.

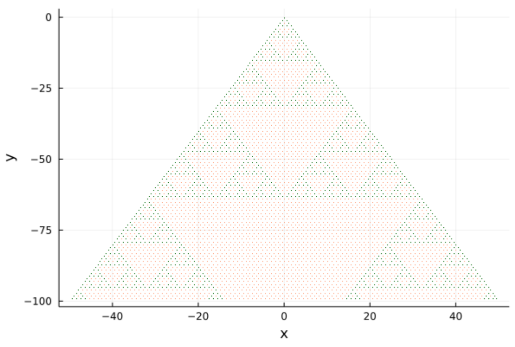

Figure 4: Khayyam triangle is plotted for 100 steps. The starting point is at (0.0, 0.0)

5. Random Sierpinski Triangle

In this random version of Sierpinski Triangle, I considered 3 functions, that scale the point and transfer it to two different directions. the point is randomly built and putted randomly into one of these functions. this process is repeated for each point several times and the final point is pushed into the *finalPoints* list. we run the code for enoght number of points till the Sierpinski Triangle is observed.

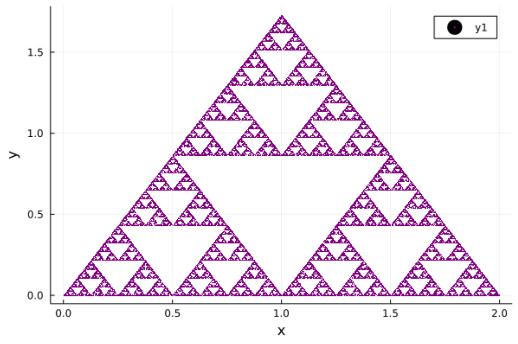

Figure 5: Random Sierpinski triangle is plotted for 50000 points and repeated 100 times for each point.

6. Random Barnsley Fern

By measuring and the information available in internet, accurate transfer and rotational matrices were built and introduced as four functions. assuming the starting point have the components (0.0,0.0) and putting other random points randomly into one of these four functions can lead us to the desired figure. Of course we must take into account that the probability of choosing these functions randomly is different from one another; Otherwise, the Barnsley Fern is going to be imperfect! Therefore, I used the *StatsBase* package in julia to include this weight characteristic. the functions are:

f1= $\begin{bmatrix} 0 & 0 \\ 0 & 16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

f2= $\begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$

f3= $\begin{bmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$

f4= $\begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.44 \end{bmatrix}$

All of these functions are chosen 1%, 85%, 7% and 7% of the time, sequentially.

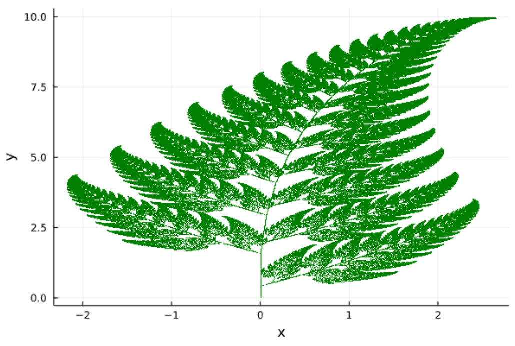

Figure 6: Random Barnsley Fern is plotted for 500000 points and repeated 100 times for each point.The first point has components (0.0, 0.0)