Computational Physics Homework Report

Fateme RamezanZade-99100693 Problem Set 1

(This was my first time coding with Python, I had only coded with C before. So I searched every syntax I needed and learned step by step. I did problem 1 first, then moved to problems 3 and 4, and did number 2 at last. This is the reason that the codes' styles and the functions I used are different for each problem.)

1. Koch Snowflake

1.1 Functions Description

Homogenity_func:

It takes two arrays which are the x coordinates and the y coordinates of points. It multiples the distance of every point from the first point of the array by 1/3. Then returns two arrays which are the x coordinates and the y coordinates of the new points.

Reflection_func:

It takes two arrays which are the x coordinates and the y coordinates of points. Then adds the horizontal distance of every point from the last point to the x coordinate of the last point, this way creates a mirrored image. Lastly, flips the array so that the points are back in right-to-left order, and returns their coordinates separately in two arrays.

Rotation_func:

It takes two arrays which are the x coordinates and the y coordinates of points, and a double number which is the rotation angle in radians. Next uses the result of applying the rotation matrix in order to rotate with amount "angle" radians about the first point to find the new points, and returns their coordinates separately in two arrays.

Shift_func:

It takes two arrays which are the x coordinates and the y coordinates of points, and a double number which is the shift length. Then adds this length to every x coordinate and returns the new x coordinates and y coordinates in two separate arrays.

Merge_func:

It takes four arrays, which are x and y coordinates of two sets of points, then merges the arrays for each axis and returns x and y coordinates of the merged set two separate arrays.

Plot_func:

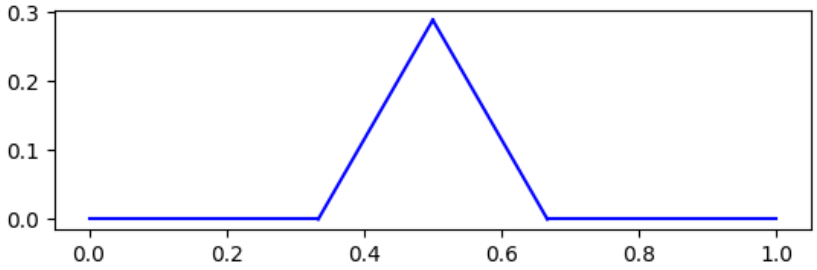
It takes two arrays which are the x coordinates and the y coordinates of points, then plots respective lines with each two using plot function in matplotlib.

1.2 Main Code

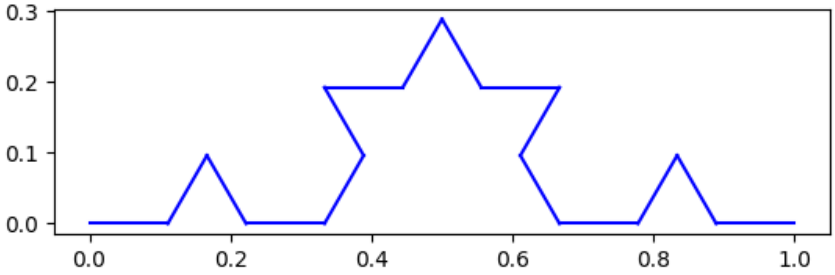
Firstly I defined two arrays which are the x and y coordinates of two points which create a line. Then I defined a double which is rotation angle in radians, which must be pi/3, and defined an integer which is the number of rounds I want to apply functions on the points. I started a loop afterwards, which repeats this process in every round: applies Homogeneity and Rotation function respectively on the set of points. Then shifts the rotated points with shift length equal to horizontal length of the shape, and merges this shifted points with the points before rotation. At last, reflects the merged points and merges them again with the unreflected ones to get a final set of points. Then updates the arrays of initial points with these new ones to repeat the process all over again. After the loop ends, the final arrays are plotted.

1.3 Results

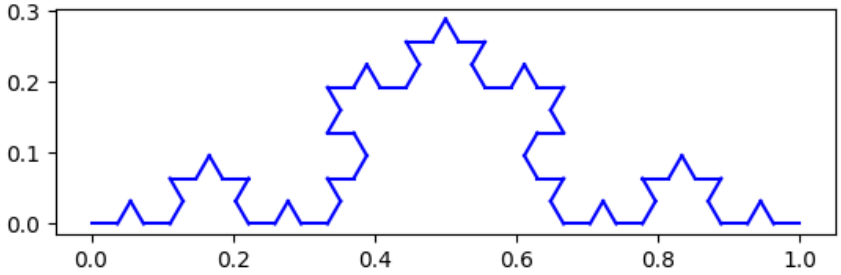
For num=1:



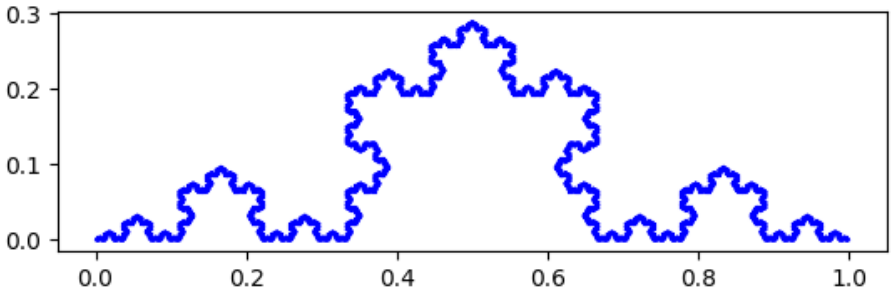
For num=2:



For num=3:



For num=6:



2. Heighway Dragon

2.1 Functions Description

Rotation_func:

It takes an array which consists of the coordinates of a set of points. Next uses the result of applying the rotation matrix in order to rotate with amount 90 degrees about the last point to find the new points. Then flips the resulted array to keep the right-to-left order, and returns it.

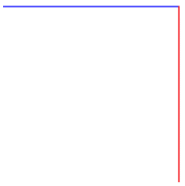
Plot_func:

It takes an array which consists of the coordinates of a set of points, then plots respective lines with each two using plot function in matplotlib. The first half of the lines is blue and the second half is red.

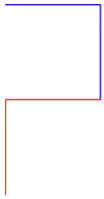
2.2 Main Code

Firstly I defined a 2-d array consisting the coordinates of two points, and an integer which is the number of rounds I want to apply functions on the points. Then I started a loop which rotates the initial points and then merges the result with them using concatenate function from numpy. As the final step, it updates the initial points with the new ones to start the loop over. After the loop ended, the resulted array is plotted.

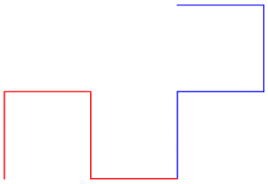
2.3 Results



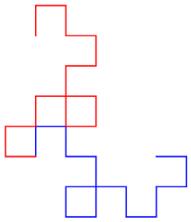
For num=1



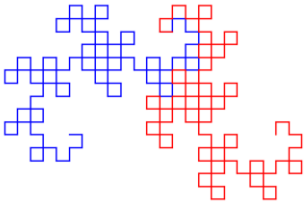
For num=2



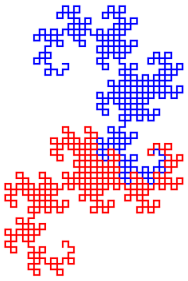
For num=3



For num=5

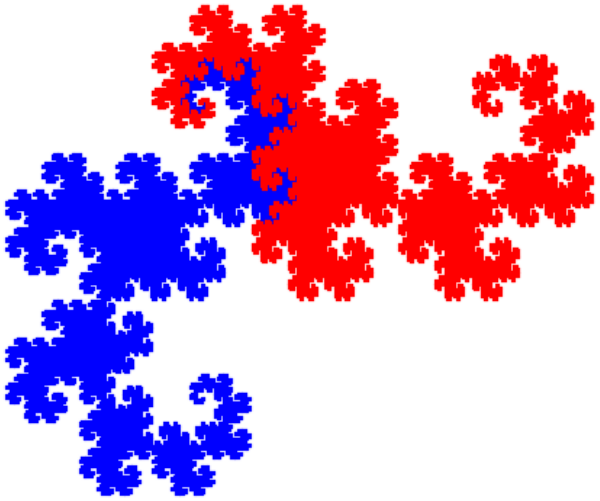


For num=8



For num=10

For num=15



3. Sierpiński Triangle

3.1 Functions Description

Homogeneity_func:

It takes an arrays which contains the x coordinates and the y coordinates of points. It multiples the distance of every point from the first point of the array by ½ and returns the resulted array.

Shift_func:

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

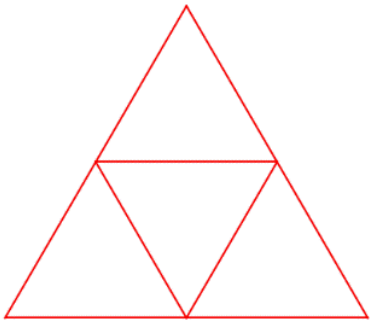
plottri:

This function is responsible for plotting triangles. It takes an array of points and separates every respective 3 points using array_split function in numpy. Then adds the first element of each separated array to the end of it, using append function in numpy. Finally, it plots respective lines with each two points in the resulted array, using plot function in matplotlib.

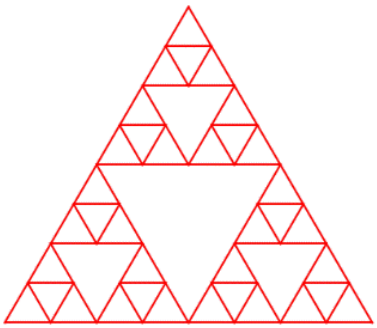
3.2 Main Code

Firstly I defined a 2-d array consisting the coordinates of three points (the first triangle), and an integer which is the number of rounds I want to apply functions on the points. Then I started a loop which does so: applies Homogeneity_func to the initial points, then applies two shifts. First a shift with horizontal length of the difference between largest x value in the points and the first point's x value, then a shift with horizontal length of half the difference between largest x value in the points and the first point's x value and vertical length of the difference between largest y value in the points and the first point's y value. Lastly, it merges three arrays, the unshifted array, first shifted one and second shifted one using concatenate function from numpy. As the final step, it updates the initial points with the new ones to start the loop over. After the loop ended, the resulted array is plotted using plottri.

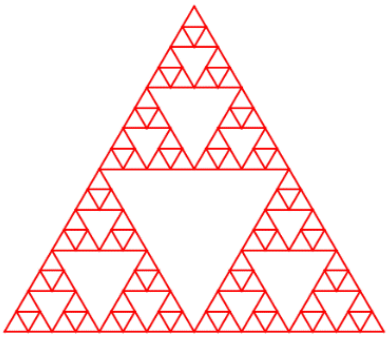
3.3 Results



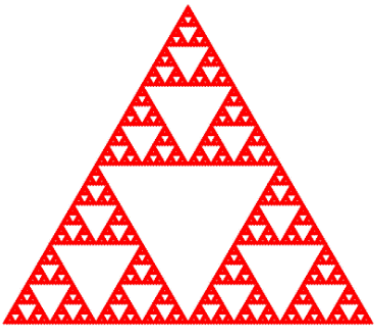
For num=1



For num=3



For num=4



For num=7

4. Kkayyam-Pascal Triangle

4.1. Functions Description

tri_construct:

This function takes an integer number and constructs a Kkayyam-Pascal triangle with that number of rows. It stores the numbers in a 2-d array where the first element of each row is one, and the following elements are calculated by summing two elements from the above row: the element in the same column and the next one. After the number of columns reaches the number of the specific row, the elements that are left would be zero. It finally returns the array.

coords_construcs:

This function is used in order to assign a coordinate to each element of a 2-d array. It takes an integer which is the number of rows and columns of the array, then constructs a 3-d array which contains two values for each element of the 2-d array. These two values are treated as x and y coordinates. Then it sets the y coordinates of the first row equal to 100, and reduces this amount by 1 as goes down in rows. In order to set the x coordinate, it calculates the difference of the number of column from the number of row, and adds it to the number of column. This way the points will be assigned to an equilateral triangle shape in the x-y plane.

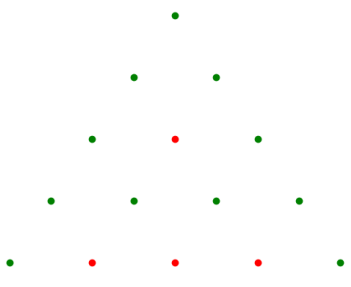
plot_dots:

This function takes two arrays, and an integer number. Then loops through the first array, checking if the value of each element is even or odd. For non-zero values, if it was odd, plots a green dot with the same element's coordinates, obtained from the second array, using scatter function from matplotlib. The dot would be red if the value was odd.

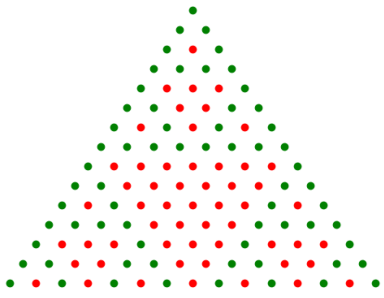
4.2. Main Code

Here I defined an integer number which is the number of rows I want my triangle to have. Then I defined two arrays, which are resulted from applying tri_construct and coords_construcs functions to the number I had. Lastly I used the plot_dots function on these.

4.3. Results

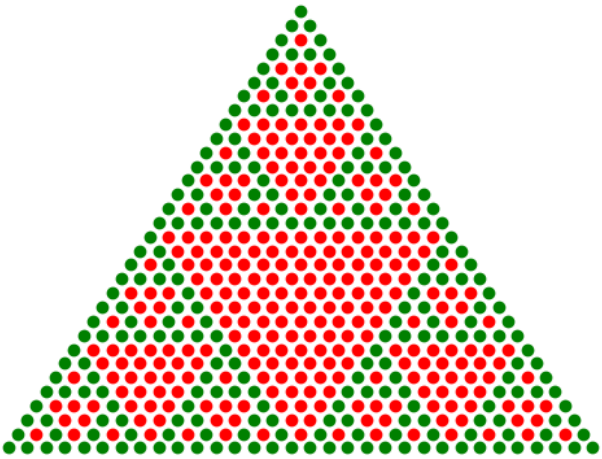


For num=5

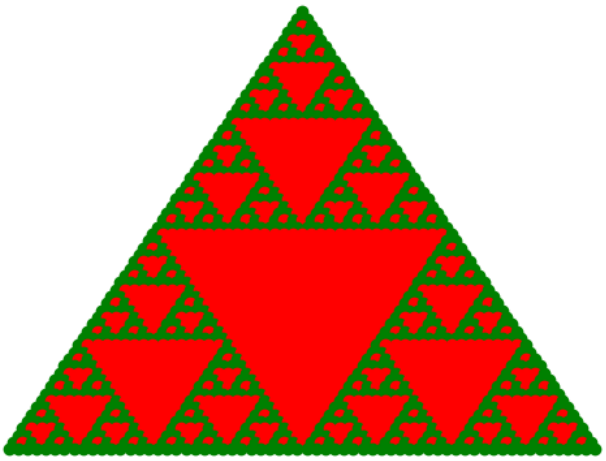


For num=15

For num=32



For num=64



We can see that the result would look ike the Sierpiński triangle for numbers equal to powers of two.