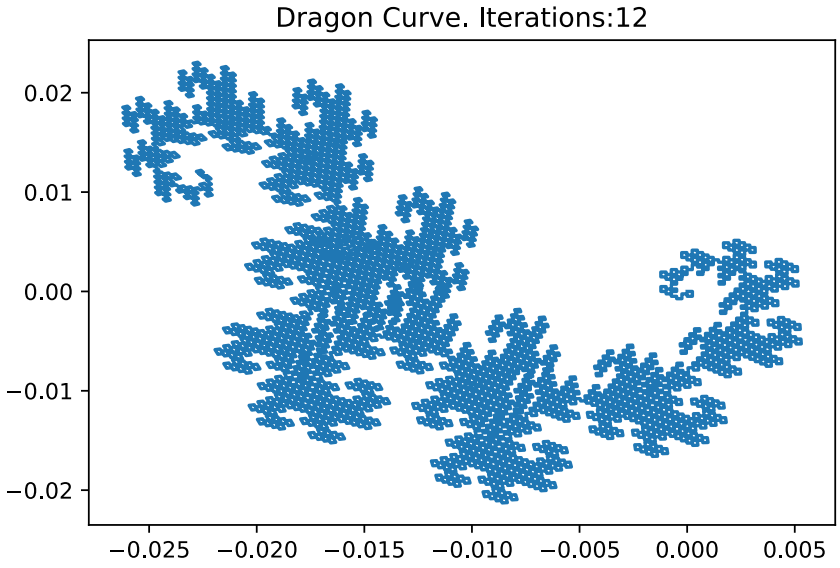
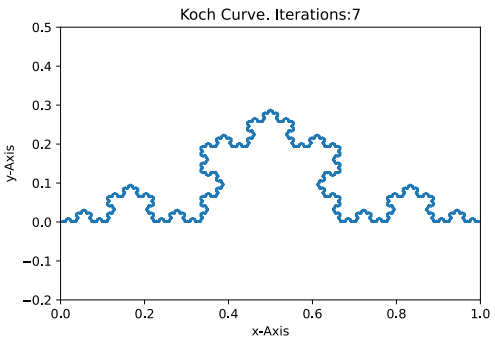
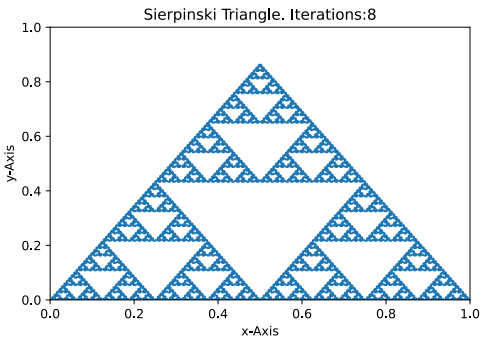
# Programming Project 2: Lindenmayer Systems

Roneet Vijay Nagale: s204091

6 December 2020



# Contents



# 1 Introduction

This programming project is a python program that generates diagrams using Lindenmayer systems. Lindenmayer systems, or L-systems are a set of formal languages used to model the growth of organisms and to generate fractal patterns. The system was originally conceived by Arisitd Lindenmayer to model the growth of plant cells. According the project description, an L-system is " *defined iteratively, and it consists of: a) an alphabet of symbols which can be used to create strings, b) an initial string used to begin the iterative construction and c) replacement rules that specify how to replace selected symbols of the string by strings of symbols (from the same alphabet)*"

This program can generate diagrams for 3 types of L-systems, Koch Curves, Sierpinski Triangles, and Dragon Curves. The diagrams are generated based on the number of iterations the user chooses. The functions implemented in this program reflect the description of L-systems provided.

The program makes use of the libraries matplotlib.pyplot for plotting the results, and numpy for processing arrays.

# 2 Functions

The program consists of 3 functions, called from the main script as necessary.

## 2.1 Main Script

The main script contains variables declared and initialized, as well as settings for the x and y axis names for the plots are generated. The script also contains a system of two menus which prompt the user for input in order to generate the desired plot.

### 2.1.1 Main menu

The main script displays a menu in the console that asks for the user's input. It is implemented with a while loop which breaks when the user decides to exit the program, thereby ending the program. The user is first provided three options to pick between, 1. The type of L-system to generate the diagram for, 2. generate the plot for a chosen L-system, and 3. quit the program.

The second option, generating a plot, requires the user to have picked an L-system beforehand. If this option is chosen first, the program does not continue and asks the user to choose a system first. If the user picks option 3, the while loop breaks, and the program exits after displaying a message that it is exiting.

### 2.1.2 Sub-menu

Picking option 1 displays another menu, this time with 4 options. The first three options are each types of the available L-systems the user can choose between. If one of these is picked, the program then asks the user for the number of iterations they wish to calculate for the chosen system. After entering the number of iterations, the program displays a message saying that the input was accepted, and instructs the user to pick option 2 in the main menu before sending the user back to the main menu.

Option 4 simply lets the user return to the main menu without either picking a system, or picking a new system if a system has already been picked.

When the user picks an L-system, the name of the L-system is saved as a string to a variable called "System", and the number of iterations are saved to another variable "N" as an integer.

### 2.1.3 Generating plots

Picking option 2 in the main menu generates the plot of the chosen L-system with the chosen number of iterations and displays it to the user.

When the user picks option 2, the three functions are called one after the other in sequence, with the first one being LindIter. The LindIter function is given the arguments "System" and "N" (the name of the L-system and the number of arguments). The LindIter function's output is then passed to the turtleGraph function, along with the number of iterations. The output from the turtleGraph function is finally passed to the turtlePlot function, which outputs the generated plot to the screen.

The script also warns the user if they enter an invalid input. This is done by using if-statements for picking menu options, and a try-catch error handling for the number of iterations, to check if the input can be actually parsed as an integer.

The script sets different settings for the graphs to be generated, depending on the type of L-system chosen. The Koch Curve and Sierpinski triangle systems look best with a limited size for the x and y axes, and the Sierpinski triangle also needs the axes flipped and the y axis to go from 0 to -1 to look correct for odd numbers of iterations.

## 2.2 LindIter

The LindIter function takes a string containing the L-system name and an integer containing the desired number of iterations as arguments. Within the function are defined three nested functions, one for each type of L-system the program includes.

Using if-statements, the function recognizes what the chosen L-system is, and calls the relevant nested function while passing the initial string as an argument. The nested function is called within a for-loop, which runs for the number of iterations of the L-system desired.

The nested functions themselves take a string containing the initial string for the particular L-system they process, and output another string after making the necessary replacements in the initial string. Their output is passed to the nested function itself again, so the replacement is carried out on the output. This repeats until a final output string processed for N iterations is created, and the LindIter function returns this string.

```
# Convert the intial string for Koch curves (S) to an L-string.
    def KochConvert(String):
        #Split the string into an array.
        arrayString = list(String)
        outputString = ''

        #Replace characters according to L-system rules.
        for character in arrayString:
            if character == 'S':
                indexS = arrayString.index('S')
                arrayString[indexS] = 'SLSRSLS'
            elif character == 'L':
                  indexL = arrayString.index('L')
                  arrayString[indexL] = 'L'
            elif character == 'R':
                  indexR = arrayString.index('R')
                  arrayString[indexR] = 'R'

        # Join the array after the loop finishes, and return the string.
        outputString = outputString.join(arrayString)
        KochString = outputString
        return KochString
```

Listing 1: Example of a nested function 'KochConvert' which makes replacements based on the Koch Curve rules.

## 2.3 turtleGraph

The turtleGraph function takes the string generated by the LindIter function as an argument, translates the string of characters to an array of numbers; alternating length and angle specifications which are then used by the turtlePlot function as instructions. The function also requires the number of iterations as an argument; this is used to scale the subsequent lengths of line segments based on the number of iterations.

The turtleGraph function does not take the name of the L-system as an argument. It instead uses the first character in the Lindenmayer string to decide which system it is going to be working with. This is done using if-statements. Once the system is identified, a for loop reads each character of the L-string, and processes it based on the defined rules for the L-system. Based on how the L-strings are generated, it is assumed that the characters alternately indicate length and a rotation in the case of Sierpinski Triangle and Koch Curve. Characters indicating 'left' or 'right' turns imply a positive or negative angle specification. Characters indicating a line segment imply a basic length specification to the power of the number of iterations. This is done using the built-in *pow()* function. Each calculated result is appended to an array initialized blank at the start.

The same applies to Dragon Curve, except that the L-string for a Dragon Curve contains some filler characters which don't affect the length or angle specifications. These are ignored with the help of

if-statements.

```
    #Starting character A or B identifies the string as a Sierpinski Triangle
    elif LindenmayerString[0] == 'A' or LindenmayerString[0]=='B':

        for characterIndex in range(0, len(LindenmayerString)):
           #Treat all characters with even indexes (including 0)
           #as length specification
           if characterIndex%2 == 0:
               lineSegmentLength = pow(1/2,N)
               turtleCommands.append(lineSegmentLength)
           #Treat all remaining characters as angle specifications.
           else:
               if(LindenmayerString[characterIndex])=='L':
                   angle=(1/3)*np.pi
                   turtleCommands.append(angle)
               elif(LindenmayerString[characterIndex])=='R':
                   angle=-(1/3)*np.pi
                   turtleCommands.append(angle)
```

Listing 2: Example of a block in the turtleGraph function where the L-system type is identified from the L-string and the necessary values for length and angles specifications are calculated.

### 2.4 turtlePlot

The turtlePlot function interprets the instructions provided by the turtleGraph function to calculate points for the final plot. turtlePlot accepts the turtleCommands array from the turtleGraph function as an argument.

The turtlePlot function assumes that the array it receives has length and angle specifications placed alternately, starting with length. It thereby creates a new array including only angle specifications, by separating the elements with odd indexes (by checking if index modulo 2 is not 0).

The first element of the turtle commands array is assumed to be the length specification, and is stored in a variable. This is because the length specification remains constant throughout the array once calculated for a certain N iterations. Because of how the graph is supposed to be symmetrical, and as the problem is described in the project description, an array of arrays (or an n x 2 matrix) to hold direction vectors and an array of arrays (or an n x 2 matrix) to hold points are initialized with a starting direction vector ¡1,0¿ and origin point (0,0). The first point from the origin in direction ¡1,0¿ is calculated using the provided formula. This point is added to the n x 2 matrix that holds points. Then, the next direction vector is calculated; the dot product of a matrix of cosinuses and sinuses and the latest direction vector is calculated to decide the next directional vector. This continues in a for-loop until all the elements of the array of angle specifications are processed. In the end, another point is added to the matrix holding points; it is calculated using the last point and the initial direction ¡1,0¿.

The matrix of points is then transposed, resulting in a 2xn matrix with the first element/column being x-values and the second being y-values. These can be used directly by matplotlib.pyplot.plot to plot the desired graph. Once the graph is shown, the user is given an option to save it to a file. It is saved in a lossless svg file called "LindenmayerSystem.svg".

## 3 Usage

The user runs the program. The main menu is displayed, and the user is asked to pick an option. The user must enter the number next to the option to pick it. If the user picks option 1, a sub-menu is displayed, prompting the user to pick an L-system. Once the user inputs a number to pick the L-system, the user is asked to specify the number of iterations the chose L-system must run for. After accepting the user's input, the user is sent back to the main menu after being instructed to pick option 2. Generate Plots from the main menu to display plots. Once the user inputs 2, the desired plot is generated and displayed. The user is then given an option to save the plot in an svg file. If the user accepts, the user is

asked to provide a filename. The file is then saved, and the user is sent back to the main menu. Here the user can choose to generate a new diagram, change the L-system being investigated, or quit the program. If the user inputs 3, the program exits.

# 4 Known Issues

Generating plots more than once without changing settings in between makes the plot generated the second time not carry over the graph settings from the main script. That is, the axis labels and the graph title isn't displayed anymore.

Dragon Curves are affected by how the points are calculated from angle specifications by the turtlePlot function, and get an undesired "tilt" over iterations when drawn by the pyplot.

Sierpinski triangles at 3 iterations look very strange. At odd number of iterations, Sierpinski triangles are generated "flipped", starting from 0 and ending at -1 on the y axis. This has been quick-fixed by flipping the y axis and having it run from 0 to -1.

The file saving has no checks for existing files with the same name in place, and there is a risk of overwriting a previously saved image.

There is no check for the length of input for the file name. There is also no check for the length of input or the size of the number of iterations, which might be a good idea as the program takes exponentially longer to calculate the points for a graph per increment in the number of iterations.

# 5 Discussion

The Lindenmayer Systems project is a great introduction to learning the principles behind generating fractal patterns using code. By not using a library designed exactly for the task (e.g. turtle), designing a more general-purpose method to solve the problem is encouraged. It would be ideal to use a graphics library directly to generate the diagrams. Because of an incomplete understanding of how L-systems work, it has been difficult understand why the Dragon curve plot looks tilted and compressed.