Simulation

Natural Computing Homework

Paul Blasi

May 1, 2015

Contents

List of Figures

List of Tables

List of Algorithms

Document Preparation and Updates

Current Version [1.1.0]

Prepared By:
Paul Blasi

Revision History

Date	***Author***	***Version***	***Comments***
4/29/15	*Paul Blasi*	*1.0.0*	*Wrote down problem set.*
4/29/15	*Paul Blasi*	*1.1.0*	*Finished Chapter 9 Problems*
4/30/15	*Paul Blasi*	*1.1.0*	*Finished 7.10 & 7.15*

1

Fractals - Text Chapter 7

1.1 Problem 10

Implement a bracketed OL-system and reproduce all plant-like structures of Figure 7.24 in the text. Change some derivation rules and see what happens. Make your own portfolio with at least ten plants.

The first step to solving this problem was to gather the necessary test input. The parameters from Figure 7.24 in the text were summarized into Figure **??**. The images were created using the program developed for the problem.

After collecting the necessary data, I saw two distinct parts to this problem. One was to create the L-system strings, and the other was to interpret them using Python's Turtle graphics (as suggested).

1.1.1 L-system generation

L-systems are rather straight forward so I made a simple class to encapsulate them. I tested this class against the example in section 7.4.3 in the text. The code can be found in Listing **??** and the results can be found in Figure **??**.

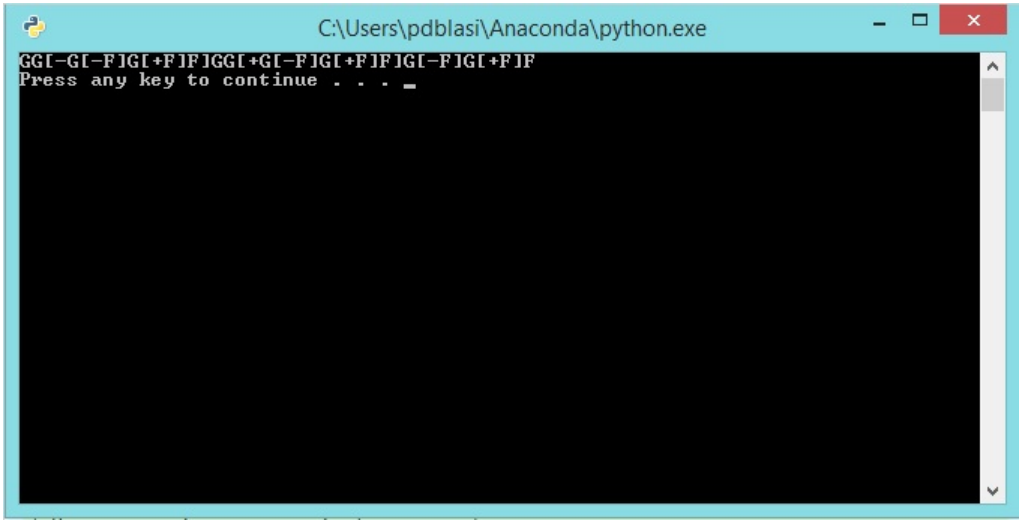


Figure 1.1: LSystem Results

1.1.2 Turtle Graphics representation

Generating the actual Python turtle code was also simple. I encapsulated the process in a class called LSystemTurtle that would initialize, run, and save the results from each L-system. This code can be found in Listing **??**. The default function run if this python file is run generated the six example images seen in Figure **??**. It's worth noting that even with the turtle speed set to 0 (meaning no animation), these plants took a very long time to reproduce[1].

After generating the given examples the task was to mess with the parameters to create a portfolio of ten different plants. Below I'll list my thought processes for each of the plants in the portfolio found in Appendix **??**. For the last two, I had to modify my code slightly. The modification amounted to checking if the d and delta variables were functions and if they were calling them instead of using them as normal. You can see this around lines 30, 47, and 52 in the code. The code to actually create the portfolio can be seen in Listing **??**.

Symmetrical G

For this plant, I decided to keep the $F \to FF$ parameter and just adapt the G parameter. I like symmetry so my thought was to make a string that is a palindrome and use that as the replacement in G. The full set of parameters I decided to go with was:

$$t = 4, \delta = 22.5^\circ$$
$$\omega = G$$
$$G \to GG + [GF-][-FG] + GG \quad F \to FF$$

This L-system generated a nest like structure with a few larger voids in it. It makes sense that a symmetrical pattern would create a somewhat circular result.

Symmetrical G with +/- swapping

For this plant, I kept the same parameters as the previous one, but instead of being a true palindrome, I swapped the + and - signs on the second half of the G parameter. This gave me:

$$t = 4, \delta = 22.5^\circ$$
$$\omega = G$$
$$G \to GG + [GF-][+FG] - GG \quad F \to FF$$

[1]Like, "I could have grown a real tree faster than this" very long time

This L-system, unsurprisingly in hindsight, created a very tall plant that doesn't have almost any width to it. It ends up going off of the canvas regardless of the step size taken.

Symmetrical F only

In this parameter set, I removed the G parameter and went with only one parameter to choose from. Again, I wanted to experiment with symmetry so I went with a palindrome.

$$
\begin{aligned}
&t = 4, \delta = 22.5^\circ \\
&\omega =' F' \\
&F \to FF + [-F[--F][F--]F-] + FF
\end{aligned}
$$

Similarly to the symmetrical G pattern, this created a highly circular pattern. Because it was created from one rule it's far more regular than the symmetrical G pattern. It looks nearly like a buzz saw.

Symmetrical F only with +/- swapping

This time I used the same single parameter set as the previous plant but swapped the + and - signs in the second half of the palindrome.

$$
\begin{aligned}
&t = 4, \delta = 22.5^\circ \\
&\omega =' F' \\
&F \to FF + [-F[--F][F++]F+] - FF
\end{aligned}
$$

This turned out very similar to the symmetrical G with sign swapping. In the same way as the previous, it was more regular.

Turning F

For this plant I wanted to make a very asymmetrical plant. My plan for this was to have the F parameter constantly turn in one direction. This plant is directly modified from the plant in Figure **??**(c).

$$
\begin{aligned}
&t = 4, \delta = 22.5^\circ \\
&\omega = G \\
&G \to F[+FFG][G] - FG \\
&F \to FF - FFF
\end{aligned}
$$

This was a surprising plant to me. The Fs far overpowered the Gs and more or less made a circular scribble.

Symmetrical Turning F

This time I wanted to try having a palindrome with turns for the F value. This one is again adapted from Figure **??**(c).

$$
\begin{aligned}
&t = 4, \delta = 22.5^\circ \\
&\omega = G \\
&G \to F[+FFG][G] - FG \\
&F \to FF - F - FF
\end{aligned}
$$

This was very similar to the previous, but with two turns it's a much more dense scribble.

Symmetrical Turning F with +/- swapping

This one is adapted from the above plant with the second - sign being replaced with a +. The hope was that this would produce a "wavy" plant.

$$t = 4, \delta = 22.5^\circ$$
$$\omega = G$$
$$G \to F[+FFG][G] - FG$$
$$F \to FF - F + FF$$

This one turned out somewhat interesting. I don't think it looks like a plant, but it does look like it could be used to simulate river paths. The wavy effect did come through rather nicely and looks somewhat random.

Similar G & F

For this plant, I wanted to see what happens when F and G are similar, but with the Fs and Gs in them swapped.

$$t = 4, \delta = 22.5^\circ$$
$$\omega = G$$
$$G \to FF - G[+GF]G$$
$$F \to GG - F[+FG]F$$

Similarity seems to have an effect reminiscent of symmetry on L-systems. This plant came out looking like a nest, similarly to many of the symmetrical patterns.

Randomized distance

In an attempt to get a random plant, I decided to make the distance stepped at each point be determined randomly. The other parameters where taken from Figure **??**(b).

$$t = 4, \delta = 22.5^\circ$$
$$\omega = F$$
$$F \to FF + [+F - F - F] - [-F + F + F]$$

These last two turned out to be the more interesting ones. They ended up very similar to Figure 7.24(b) from which they were adapted. In the case of randomized distance, the entire fractal seemed to expand ever so slightly.

Randomized turn (δ)

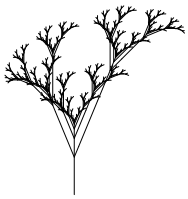
In another attempt to get a random plant, I decided to make the angle turned at each point be determined randomly. The other parameters where taken from Figure **??**(b).

$$t = 4, \delta_i = random_i(20.0, 30.0)$$
$$\omega = F$$
$$F \to FF + [+F - F - F] - [-F + F + F]$$

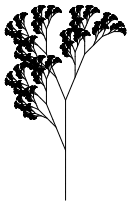
Again, this was similar to Figure 7.24(b) but in this case, the plant only seemed to widen, not to expand vertically.

$t = 8, \delta = 22.5°$
$\omega = G$
$G \to F+[[G]-G]-F[-FG]+G$
$F \to FF$

$t = 4, \delta = 22.5°$
$\omega = F$
$F \to FF+[+F-F-F]-[-F+F+F]$



$t = 6, \delta = 22.5°$
$\omega = G$
$G \to F[+FFG][G]-FG$
$F \to FF$



$t = 9, \delta = 20°$
$\omega = G$
$G \to F[-G]F[+G]-G$
$F \to FF$

$t = 9, \delta = 25.7°$
$\omega = G$
$G \to F[-G][+G]FG$
$F \to FF$

$t = 5, \delta = 22.5°$
$\omega = G$
$G \to FG[-F[G]-G][G+G][+F[G]+G]$
$F \to FF$

Figure 1.2: Reproduction of figure 7.24 in the text

1.2 Problem 15

Implement a random iterated function system (RIFS) to generate all the fractals whose codes are presented in Table 7.3 in the text.

Again, the first step was to reproduce the data needed for the problem. Table 7.3 from the text has been reproduced in Table **??**.

With this data available I created a simple class to do the iterated functions, which would output a list of x and y values that I would plot after the full set was created. This is a slight (less graphics intensive) modification of Algorithm 7.3 in the text. This code can be seen in Listing **??**. The resulting fractals can be seen in Appendix **??**. The fractals produced do not contain different shades of grey depending on the chosen transform.

w	a	b	c	d^2	e	f	p
1	0.5	0	0	0.5	1	1	0.33
2	0.5	0	0	0.5	1	50	0.33
3	0.5	0	0	0.5	50	50	0.34

Sierpinski Gasket

w	a	b	c	d	e	f	p
1	0.5	0	0	0.5	1	1	0.25
2	0.5	0	0	0.5	50	1	0.25
3	0.5	0	0	0.5	1	50	0.25
4	0.5	0	0	0.5	50	50	0.25

Square

w	a	b	c	d	e	f	p
1	0	0	0	0.16	0	0	0.01
2	0.85	0.04	-0.04	0.85	0	1.6	0.85
3	0.2	-0.26	0.23	0.22	0	1.6	0.07
4	-0.15	0.28	0.26	0.24	0	0.44	0.07

Barnsley Fern

w	a	b	c	d	e	f	p
1	0	0	0	0.5	0	0	0.05
2	0.42	-0.42	0.42	0.42	0	0.2	0.40
3	0.42	0.42	-0.42	0.42	0	0.2	0.40
4	0.1	0	0	0.1	0	0.2	0.15

Tree

Table 1.1: Reproduction of Table 7.3 from the text

1.3 Problem 21

Implement the random midpoint displacement algorithm in 3D and generate some fractal landscapes. Study the influence of H on the landscapes generated.

2

Cellular Automata - Chapter 7

2.1 Problem 1 (from slides)

Modify the heat flow example to deal with insulated conditions on the top and bottom boundary. Insulation means zero flux or u[N][j] = u[N-1][j]. This implies that instead of fixed valued ghost points on the top and bottom, you modify the CA rule using the previous relation.

2.2 Problem 2 (from slides)

Reproduce patterns theta, lambda, mu, and alpha in the Gray-Scott Model CA. You don't need to follow their color scheme.

3

ALife - Text Chapter 8

3.1 Problem 3

Choose one of the sample projects of StarLogo and solve its exploration tasks (`http://education.mit.edu/starlogo/projects.html`). Write a brief report with the results obtained including any theoretical background knowledge that may eventually be necessary to perform the exploration.

3.2 Problem 4

Implement a bi-dimensional CA following the rules of 'The Game of Life'.

4

DNA Computing - Text Chapter 9

Most of this section of the assignment was paraphrased or expanded on from Wikipedia articles.

4.1 Problem 1

Name four problems that cannot be solved by a Turing machine.

Halting Problem

Determining a busy beaver[1] champion

The Mortality Problem

Determining whether a given machine computes a partial function with a nontrivial property of partial functions.

4.2 Problem 2

Name four NP-complete and four NP-hard problems.

NP-complete problems	NP-hard problems
SAT problem	Subset Sum Problem
Hamiltonian Path Problem (HPP)	Traveling Salesman Problem
Knapsack Problem	K Minimum-spanning tree
Partition Problem	Graph Coloring Problem

[1] "Busy Beaver" is my favorite program name of all time.

4.3 Problem 5

The two most basic DNA sequencing techniques are known as a) Maxam-Gilbert and b) Sanger, after their proponents. Explain how each of these techniques work and contrast them.

Maxam-Gilbert Sequencing

Maxam-Gilbert Sequencing works by cleaving the DNA strands via four different solutions. The solutions are balanced in such a way that each strand will, on average, only be cleaved once. The four solutions cleave at different deoxynucleotides: (A + G), G, C, & (C + T).

After electrophoresing the leftover strands to sort them by size, you are left with a distribution of sizes in each solution. Reading these from shortest to longest, you can infer the deoxynucleotide at that position. Note: in the case of the (A + G) and (C + T) bands, the presence of A and T are infered by that band showing and a lack of the G and C bands respectively. An example graphic of Maxam-Gilbert Sequencing can be seen in Figure **??** (a).

Sanger Sequencing

Sanger Sequencing clones a sequence of DNA in four different solutions. Each solution contains 3 of the normal deoxynucleotides that make up DNA chains. The fourth deoxynucleotide is replaced with a corresponding *di*-deoxynucleotide which inhibits chaining due to it's lack of a 3'-OH group used to form phosphodiester bonds. The di-deoxynucleotides can be labeled through various methods including florescence or radioactivity.

Once the DNA is copied, the four strands are heat denatured and separated by length using gel electrophoresis. The length of the strands was limited by the di-deoxynucleotide which means the different lengths of the strands in each solution corresponds to places where the corresponding deoxynucleotide would reside. The sequence can then be read by reading the relative positions in the four lanes. An example of this type of sequencing can be found in Figure **??** (b).

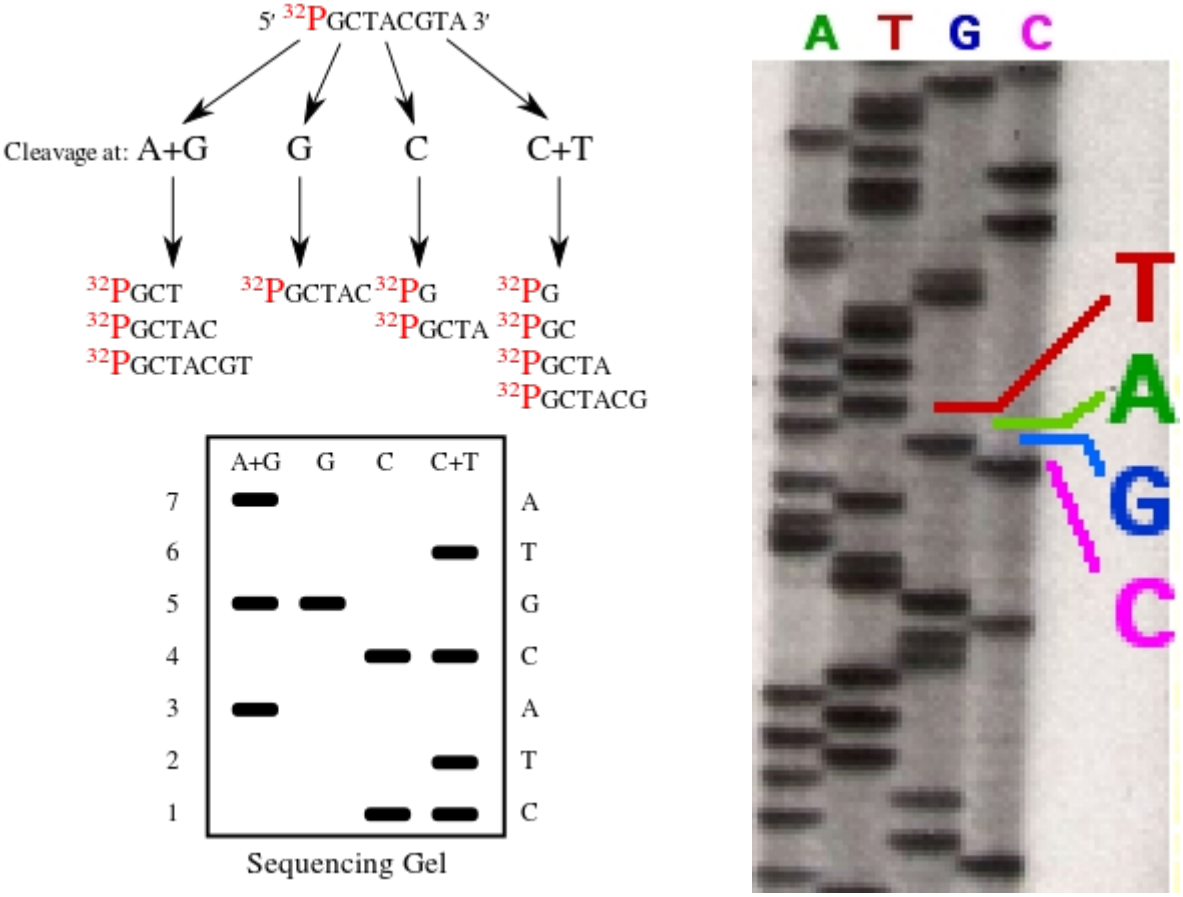
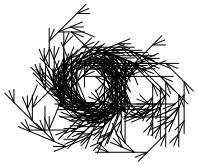


Figure 4.1: DNA Sequencing Methods. (a)Maxam-Gilbert Sequencing (b)Sanger Sequencing

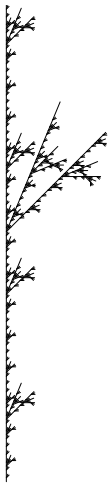
A

Supporting Materials

A.1 Plant Portfolio



Symmetrical G



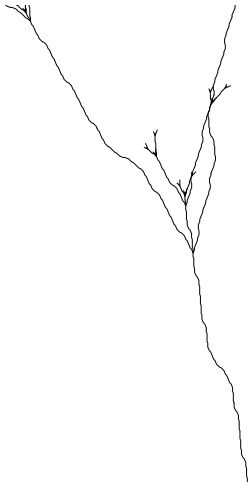
Symmetrical G with +/- swapping

Symmetrical F only

Symmetrical F only with +/- swapping

Turning F

Symmetrical Turning F



Symmetrical Turning F with +/- swapping

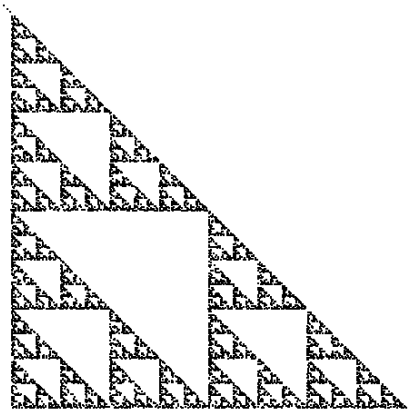
Similar G & F

Randomized distance (d)

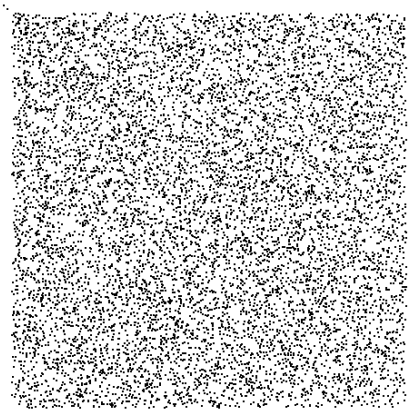
7.24(b) for comparison

Randomized turn (δ)

A.2 RIFS Results

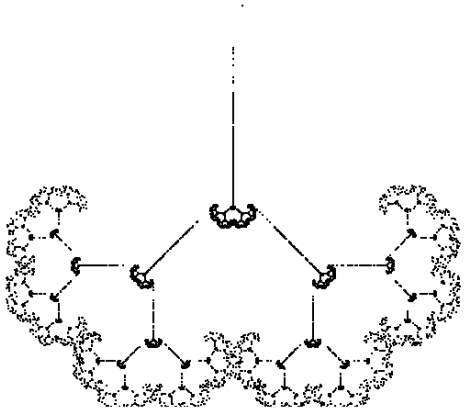


Sierpinski Gasket



Square

Barnsley Fern[1]



Tree

B

Code

Listing B.1: LSystem.py

```
class LSystem:
    def __init__(self, productions_dict):
        self.productions_dict = productions_dict;

    def generate_word(self, iters, omega):
        word = omega
        for i in range(iters):
            word = self.rewrite(word)
        return word

    def rewrite(self, word):
        new = ""
        for c in word:
            if c in self.productions_dict:
                new = new + self.productions_dict[c]
            else:
                new = new + c
        return new

if __name__ == '__main__':
    system = LSystem({'F':'G[-F]G[+F]F', 'G':'GG'})
    print system.generate_word(2, 'F')
```

Listing B.2: LSystemTurtle.py

```
from LSystem import *
from turtle import *
from Tkinter import *
from inspect import isfunction

class LSystemTurtle:
    def __init__(self, lsystem, d, delta):
        self.lsystem = lsystem
        self.d = d
        self.delta = delta
        self.pos_stack = []
        self.turtle = Turtle()
        self.turtle.pu()
        self.turtle.setheading(90)
        starty = (-1 * (self.turtle.getscreen().window_height() // 2)) + 80
        self.turtle.sety(starty)
        self.turtle.speed(0)
        self.turtle.ht()
        self.turtle.pd()

    def set_center(self,x,y):
        self.turtle.pu()
        self.turtle.setx(x)
        self.turtle.sety(y)
        self.turtle.pd()

    def run(self, iters, omega):
        word = self.lsystem.generate_word(iters, omega)
        for c in word:
            if c in self.lsystem.productions_dict:
                if isfunction(self.d):
                    self.turtle.forward(self.d())
                else:
                    self.turtle.forward(self.d)
            elif c == '[':
                self.pos_stack.append((self.turtle.xcor(),
                                       self.turtle.ycor(),
                                       self.turtle.heading()))
            elif c == ']':
                pos = self.pos_stack.pop()
                self.turtle.pu()
                self.turtle.setx(pos[0])
                self.turtle.sety(pos[1])
                self.turtle.setheading(pos[2])
                self.turtle.pd()
            elif c == '-':
                if isfunction(self.delta):
                    self.turtle.left(self.delta())
                else:
                    self.turtle.left(self.delta)
            elif c == '+':
                if isfunction(self.delta):
```

```
                    self.turtle.right(self.delta())
                else:
                    self.turtle.right(self.delta)
            else:
                SyntaxError("Invalid character in LSystem word.")

    def save(self, filename):
        ts = self.turtle.getscreen()
        ts.getcanvas().postscript(file=filename)
        bye()


if __name__== '__main__':
    #system = LSystem({'G':'F+[[G]-G]-F[-FG]+G', 'F':'FF'})
    #lturtle = LSystemTurtle(system, 1, 22.5)
    #lturtle.run(8, 'G')
    #lturtle.save('lsystem_a.eps')

    #system = LSystem({'F':'FF+[+F-F-F]-[-F+F+F]'})
    #lturtle = LSystemTurtle(system, 5, 22.5)
    #lturtle.run(4, 'F')
    #lturtle.save('lsystem_b.eps')

    #system = LSystem({'G':'F[+FFG][G]-FG', 'F':'FF'})
    #lturtle = LSystemTurtle(system, 3, 22.5)
    #lturtle.run(6, 'G')
    #lturtle.save('lsystem_c.eps')

    #system = LSystem({'G':'F[-G]F[+G]-G', 'F':'FF'})
    #lturtle = LSystemTurtle(system, .5, 22.5)
    #lturtle.run(9, 'G')
    #lturtle.save('lsystem_d.eps')

    system = LSystem({'G':'F[-G][+G]FG', 'F':'FF'})
    lturtle = LSystemTurtle(system, .5, 22.5)
    lturtle.run(9, 'G')
    lturtle.save('lsystem_e.eps')

    system = LSystem({'G':'FG[-F[G]-G][G+G][+F[G]+G]', 'F':'FF'})
    lturtle = LSystemTurtle(system, 3, 22.5)
    lturtle.run(5, 'G')
    lturtle.save('lsystem_f.eps')
```

Listing B.3: PlantPortfolio.py

```
from LSystem import *
from LSystemTurtle import *
import random

def d_rand():
    return random.uniform(4.0,7.0)

def delta_rand():
    return random.uniform(20.0, 30.0)

#Symmetrical G
system = LSystem({'G':'GG+[GF-][-FG]+GG', 'F':'FF'})
lturtle = LSystemTurtle(system, 10, 22.5)
lturtle.set_center(0,0)
lturtle.run(4, 'G')
lturtle.save('plant_1.eps')

#Symmetrical G with +/- swapping
system = LSystem({'G':'GG+[GF-][+FG]-GG', 'F':'FF'})
lturtle = LSystemTurtle(system, 3, 22.5)
lturtle.run(4, 'G')
lturtle.save('plant_2.eps')

#Symmetrical F
system = LSystem({'F':'FF+[-F[--F][F--]F-]+FF'})
lturtle = LSystemTurtle(system, 5, 22.5)
lturtle.run(4, 'F')
lturtle.save('plant_3.eps')

#Symmetrical F with +/- swapping
system = LSystem({'F':'FF+[-F[--F][F++]F+]-FF'})
lturtle = LSystemTurtle(system, 5, 22.5)
lturtle.run(4, 'F')
lturtle.save('plant_4.eps')

#Turning F
system = LSystem({'G':'F[+FFG][G]-FG', 'F':'FF-FFF'})
lturtle = LSystemTurtle(system, 5, 22.5)
lturtle.set_center(0,0)
lturtle.run(4, 'G')
lturtle.save('plant_5.eps')

#Symmetrical Turning F
system = LSystem({'G':'F[+FFG][G]-FG', 'F':'FF-F-FF'})
lturtle = LSystemTurtle(system, 3, 22.5)
lturtle.set_center(0,0)
lturtle.run(4, 'G')
lturtle.save('plant_6.eps')

#Symmetrical Turning F with +/- swapping
system = LSystem({'G':'F[+FFG][G]-FG', 'F':'FF-F+FF'})
lturtle = LSystemTurtle(system, 3, 22.5)
```

```
lturtle.run(4, 'G')
lturtle.save('plant_7.eps')

#Swapping Gs and Fs
system = LSystem({'G':'FF-G[+GF]G', 'F':'GG-F[+FG]F'})
lturtle = LSystemTurtle(system, 3, 22.5)
lturtle.run(4, 'G')
lturtle.save('plant_8.eps')

#Random distance (d)
system = LSystem({'F':'FF+[+F-F-F]-[-F+F+F]'})
lturtle = LSystemTurtle(system, d_rand, 22.5)
lturtle.run(4, 'F')
lturtle.save('plant_9.eps')

#Random angle (delta)
system = LSystem({'F':'FF+[+F-F-F]-[-F+F+F]'})
lturtle = LSystemTurtle(system, 5, delta_rand)
lturtle.run(4, 'F')
lturtle.save('plant_10.eps')
```

Listing B.4: RIFS.py

```
import random
import matplotlib.pyplot as plt
from math import ceil

class Transform:
    def __init__(self, a, b, c, d, e, f):
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.e = e
        self.f = f

    def transform(self, pt):
        return (self.a * pt[0] + self.b * pt[1] + self.e,
                self.c * pt[0] + self.d * pt[1] + self.f)

class RIFS:
    def __init__(self, w, p, x):
        self.w = w
        self.p = p
        self.x = x

    def generate_points(self, iters):
        x = []
        y = []
        for i in range(iters):
            x.append(self.x[0])
            y.append(self.x[1])
            self.x = self.select().transform(self.x)

        return x,y

    def select(self):
        i = 0
        sum = self.p[0]
        rand = random.uniform(0, 1)
        while i < len(self.w) - 1 and sum < rand:
            i = i + 1
            sum = sum + self.p[i]
        return self.w[i]

    def plot(self, iters, name):
        x,y = self.generate_points(iters)
        plt.scatter(x,y, s=1, marker=',', facecolor='0', lw=0)
        plt.gca().invert_yaxis()
        plt.axis('equal')
        plt.axis('off')
        plt.savefig(name, bbox_inches='tight')
        plt.clf()

if __name__ == '__main__':
```

```
#Sierpinski Gasket
w = [ Transform(0.5, 0, 0, 0.5, 1, 1),
      Transform(0.5, 0, 0, 0.5, 1, 50),
      Transform(0.5, 0, 0, 0.5, 50, 50) ]
p = [0.33, 0.33, 0.34]
RIFS(w, p, (0.01, 0.01)).plot(10000, 'rifs_sierpinski.png')

#Square
w = [ Transform(0.5, 0, 0, 0.5, 1, 1),
      Transform(0.5, 0, 0, 0.5, 1, 50),
      Transform(0.5, 0, 0, 0.5, 50, 1),
      Transform(0.5, 0, 0, 0.5, 50, 50) ]
p = [0.25, 0.25, 0.25, 0.25]
RIFS(w, p, (0.01, 0.01)).plot(10000, 'rifs_square.png')

#Barnsley Fern
w = [ Transform(0, 0, 0, 0.16, 0, 0),
      Transform(0.85, 0.04, -0.04, 0.85, 0, 1.6),
      Transform(0.2, -0.26, 0.23, 0.22, 0, 1.6),
      Transform(-0.15, 0.28, 0.26, 0.24, 0, 0.44) ]
p = [0.01, 0.85, 0.07, 0.07]
RIFS(w, p, (0.01, 0.01)).plot(10000, 'rifs_barnsleyfern.png')

#Tree
w = [ Transform(0, 0, 0, 0.5, 0, 0),
      Transform(0.42, -0.42, 0.42, 0.42, 0, 0.2),
      Transform(0.42, 0.42, -0.42, 0.42, 0, 0.2),
      Transform(0.1, 0, 0, 0.1, 0, 0.2) ]
p = [0.05, 0.40, 0.40, 0.15]
RIFS(w, p, (0.01, 0.01)).plot(10000, 'rifs_tree.png')
```

Listing B.5: 3D_RMD.py

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

class RMD_3D:
    def __init__(self, iters, H, d):
        self.iters = iters
        self.width = self.height = int(2**iters + 1)
        self.H = H
        self.d = d
        self.used = []

    def __stack(self, map):
        stack = []
        stack.append((0, 0)), stack.append((self.height-1, self.width-1)), stack.append(sel

        while len(stack) != 0:
            right = stack.pop(0)
            left = stack.pop(0)
            d = stack.pop(0)
            mid = ((left[0] + right[0]) // 2,
                   (left[1] + right[1]) // 2 )

            map[left[0]][mid[1]] = (map[left[0]][left[1]] + map[left[0]][right[1]]) / 2
            map[mid[0]][right[1]] = (map[left[0]][right[1]] + map[right[0]][right[1]]) / 2
            map[right[0]][mid[1]] = (map[right[0]][right[1]] + map[right[0]][left[1]]) / 2
            map[mid[0]][left[1]] = (map[right[0]][left[1]] + map[left[0]][left[1]]) / 2
            map[mid[0]][mid[1]] = max((((  map[left[0]][left[1]] + map[left[0]][right[1]] +
                                           map[right[0]][right[1]] + map[right[0]][left[1]])
                                           rand.uniform(-1*d, d)), 0))

            if right[0] - mid[0] != 1:
                d = d / 2**self.H
                stack.append(left), stack.append(mid), stack.append(d)
                stack.append((left[0], mid[1])), stack.append((mid[0], right[1])), stack.ap
                stack.append(mid), stack.append(right), stack.append(d)
                stack.append((mid[0], left[1])), stack.append((right[0], mid[1])), stack.ap

    def create_map(self):
        self.map = np.zeros((self.height, self.width))
        self.__recurse((0, 0), (self.height, self.width), 0, self.d)
        #self.__stack(self.map)

        self.__plot(self.map)

    def __plot(self, map):
        print map
        ax = plt.figure().add_subplot(111, projection='3d')
        x, y = np.meshgrid(range(self.width), range(self.height))
        ax.contourf(x, y, map, cmap=cm.terrain, vmin=1, vmax=100)
```

```
        plt.axis('equal')
        plt.axis('off')
        plt.show()

    def __recurse(self, min_corner, max_corner, depth, d):
        min_x = min([min_corner[0], max_corner[0]])
        max_x = max([min_corner[0], max_corner[0]])
        min_y = min([min_corner[1], max_corner[1]])
        max_y = max([min_corner[1], max_corner[1]])
        op = [ (min_x, min_y),
               (min_x, max_y),
               (max_x, min_y),
               (max_x, max_y) ]
        center = self.__center(op[0], op[3])
        if depth <= self.iters and not center in self.used:
            self.used.append(center)
            self.map[center[0]][center[1]] = self.__average(op)
            self.__perturb(center)

            self.__recurse(op[0], center, depth + 1, d - (1 / 2**self.H))
            self.__recurse(center, op[1], depth + 1, d - (1 / 2**self.H))
            self.__recurse(op[2], center, depth + 1, d - (1 / 2**self.H))
            self.__recurse(center, op[3], depth + 1, d - (1 / 2**self.H))

    def __center(self, min_corner, max_corner):
        #i = int(round((min_corner[0] + max_corner[0]) / 2.0))
        #j = int(round((min_corner[1] + max_corner[1]) / 2.0))
        i = (min_corner[0] + max_corner[0]) // 2
        j = (min_corner[1] + max_corner[1]) // 2
        return (i,j)

    def __average(self, op):
        ave = 0
        count = 0
        for i in range(4):
            if op[i][0] < self.width and op[i][1] < self.height:
                count = count + 1
                ave = ave + self.map[op[i][0]][op[i][1]]
        return ave / count

    def __perturb(self, c):
        r = rand.uniform(1, self.d)
        self.map[c[0]][c[1]] = self.map[c[0]][c[1]] + r
        if self.map[c[0]][c[1]] < 0:
            self.map[c[0]][c[1]] = 0


if __name__ == '__main__':
    RMD_3D(8, .9, 30).create_map()
```

Listing B.6: HeatFlowCA.py

Listing B.7: GrayScottCA.py

Listing B.8: GameOfLife.py