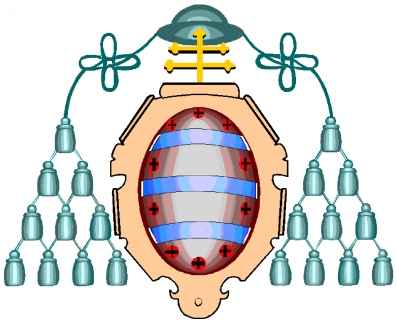
Universidad de Oviedo



Master in Soft Computing
and Intelligent Data Analysis

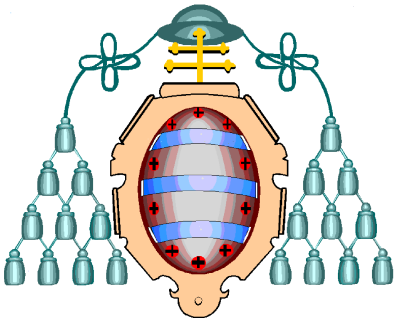
Proyecto Fin de Máster
Master Project

An R-Package for Generating Fractals Via Iterated Function Systems

by Elena Merce Rodríguez Pérez

July 2012

Universidad de Oviedo



Master in Soft Computing
and Intelligent Data Analysis

Proyecto Fin de Máster
Master Project

An R-Package for Generating Fractals Via Iterated Function Systems

by Elena Merce Rodríguez Pérez

Advisors:
Wolfgang Trutschnig
Gil González Rodríguez

July 2012

Acknowledgements

It is with immense gratitude that I acknowledge the support and help of my Professor, Wolfgang Trutschnig.

A special thanks to my family: To my sister, for her loyalty, love and never ending friendship. To my mother, for her everlasting love, all her sacrifices and for the unforgettable moments we share. To my father, for all his hard work and dedication to our family, for his unending love, and for his encouragement. I dedicate who I am and all of my accomplishments to my loving family.

Abstract

The aim of this master project is the development of an R-Package for generating fractals via iterated function systems (IFS). At first glance fractals are eye-catching, self-similar patterns that form a pretty or interesting picture. However, the inner workings of fractals are much more complex and can be extremely useful in applications for image compression, image animation, for identification and for modeling.

Extensive research on fractal theory was undertake to effectively define functions in R capable of handling diverse IFS. Two algorithms are analyzed and implemented in the generation of fractals, the Deterministic Algorithm and the Random Iteration Algorithm. Through the functions developed for an R-package to construct fractals, some of the major theorems and corollaries in fractal theory were successfully verified. The convergence of the Hutchinson operator is validated in the Hausdorff metric; while, experimentation with IFS with probabilities through the implementation of the Random Iteration Algorithm helps illustrate and define measures on fractals in $\mathbb{R}^2$. Furthermore, a comparison between the Deterministic Algorithm and the Random Iteration Algorithm for generating fractals proves the effective and time efficient use of the Random Iteration Algorithm.

The result of this master project is the development of multiple functions to compose an R-package for generating fractals via iterated function systems. Determ_R2 was created to generate fractals in the two-dimensional vector space of real numbers, $\mathbb{R}^2$, via the Deterministic Algorithm. The functions RandItl_R2, RandIt_R3 and RandIt_C were programed to build fractals in $\mathbb{R}^2$, $\mathbb{R}^3$ and the complex plane ($\mathbb{C}$), respectively, using the Random Iteration Algorithm. A function, IFS_Prob, was developed to randomly generate probabilities for a hyperbolic IFS. Additionally, IFSP_Gen_R2 was developed to recursively test random probabilities for a given IFS. The function FractalMeas_R2 calculates the measure of a Borel subset of $\mathbf{X}$, given the limiting coordinates of said Borel subset. Lastly, TransMat_R2 approximates the attractor for an IFS induced by a transformation matrix.

Keywords: Master Project, Soft Computing, Fractals, Iterated Function Systems, Measure, Attractor

Contents

List of Figures

List of Tables

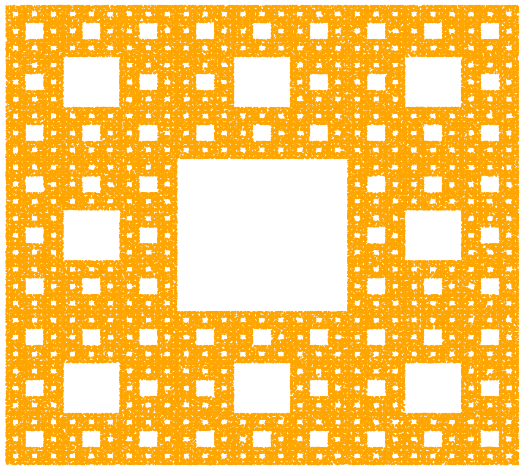
Chapter 1

Introduction

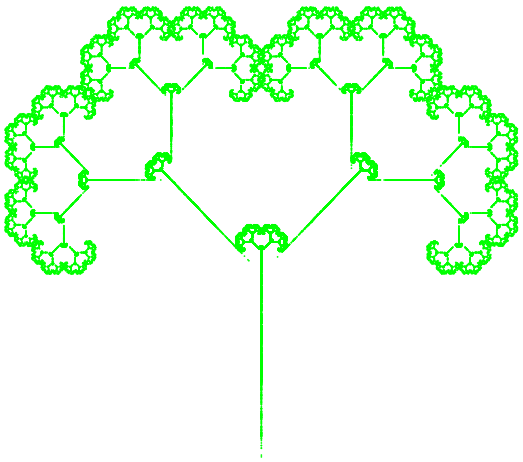
At first glance fractals are eye-catching, self-similar patterns that form a pretty or interesting picture. However, the inner workings of fractals are much more complex and can be extremely useful in real-world applications. Fractals can be generated mathematically, or observing closely, fractals can even be found in nature. The exact meaning of a fractal is difficult to condense into a short, simplistic definition. For the time begin, a fractal is a subset of a simple geometrical space, where said subset may be geometrically complex. This fractal subset can be built through an iterated function system (IFS). A more precise definition of a fractal and an IFS is given in Chapter 3; just the same, some visual examples of fractals are provided on the following page in Figure 1.1.

The detailed description of the master project herein was prepared as the final project for the Master in Soft Computing and Intelligent Data Analysis at the University of Oviedo. The central objective of this master project is the development of an R-Package for generating fractals via IFS. R was chosen for the development of this project because R is a free programming environment effective in data manipulation, calculations, and graphical display. R can be easily adapted and extended via the creation of new functions and/or new packages, such as those constructed in this master project. When considering the computational times mentioned throughout, note that the programming of the R functions was performed on a 2.7 GHz Intel Core i7 processor.

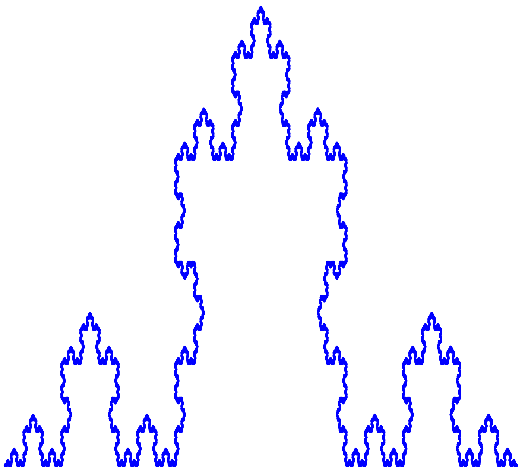
Clearly, Chapter 1 serves as a brief introduction to the development of this master project. Chapter 2 further details the objectives of the project. Extensive research on fractal theory was undertake to effectively define functions in R capable of handling diverse IFS. A summary of this research is presented in Chapter 3, the Materials and Methods section of this text, to give the reader an understanding of the basic concepts in fractal theory relevant to the formation of fractals via IFS. Chapter 4 describes the results of this master project, that is, each one of the functions developed in R, which compose the core of the R-package. This section summarizes how the functions were built, which are their inputs, their outputs and some possible uses of said functions. Finally, Chapter 5 enumerates the outcomes of this master project, draws important conclusions developed during the project and discusses relevant future work. Appendix A includes the IFS used and developed throughout the project, and Appendix B provides the R function programmed for the R-package.



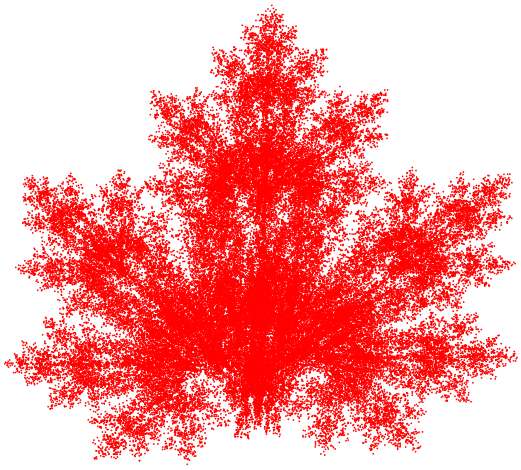
(a) Sierpinski Carpet



(b) Fractal Tree



(c) Koch Curve



(d) Maple Leaf

Figure 1.1: Fractals Generated Via IFS

What link exists between fractals and a Master in Soft Computing and Intelligent Data Analysis? Soft computing and intelligent data analysis combine both traditional and emerging problem solving methodologies, including fuzzy logic, artificial neural networks, evolutionary algorithms and probabilistic reasoning. The aim is to develop powerful, approximate reasoning systems with the ability to handle and model ill-define problems with large-scale solutions spaces that emerge in real-world situations. Fractal theory provides soft computing and intelligent data analysis researchers with mathematical tools capable of analyzing the geometrical complexity of natural and artificial objects, and can be used for identification and modeling [2]. Moreover, fractal theory provides the ability to interpolate between similar fractal images; this is particularly useful image animation. The capacity to control fractals by adjusting parameters in the IFS is extremely important in image compression applications [1]. Even though this master project does not specifically concern image compression, a glimpse of the connection between fractals and

image compression is illustrated in the following figure [6].

(a) Smiley Face Image Compression

(b) Letter "A" Image Compression

Figure 1.2: Fractals and Image Compression

Chapter 2

Objectives

Chapter 2 of this text details the objectives of this master project, whose principle objective is the development of an R-package for generating fractals via IFS.

2.1 Objectives

As previously mentioned the main goal of the master project describe herein is the creation of an R-package for generating fractals via IFS. Two algorithms for generating fractals will be analyzed and implemented through the developed R functions, the Deterministic Algorithm and the Random Iteration Algorithm. The functions developed for an R-package to construct fractals, should uphold the major theorems and corollaries in fractal theory.

The functions programmed must be robust and user friendly. More specifically the functions should be able to operate in various mathematical spaces, i.e., the two-dimensional Euclidean plane ($\mathbb{R}^2$), the three-dimensional Euclidean plane ($\mathbb{R}^3$) and the complex plane ($\mathbb{C}$).

Determ_R2 will be created to generate fractals in $\mathbb{R}^2$ via the Deterministic Algorithm. The functions RandItl_R2, RandIt_R3 and RandIt_C shall be programed to build fractals in $\mathbb{R}^2$, $\mathbb{R}^3$ and $\mathbb{C}$, respectively, using the Random Iteration Algorithm. A function, IFS_Prob, is to be developed to randomly generate probabilities for a hyperbolic IFS. Additionally, IFSP_Gen_R2 will recursively test random probabilities for a given IFS. The function FractalMeas_R2 will calculate the measure of a Borel subset of $\mathbf{X}$, given the limiting coordinates of said Borel subset. Lastly, TransMat_R2 will approximates the attractor for an IFS using a transformation matrix.

The package should be easy to use. The functions should have parameters the user can adjust to suit his or her needs; yet, inputs should have a standard form to ensure the correct execution of each function. Error checking shall be added to control the implementation of each function. Computational time will be minimized to the greatest extent possible, in turn maxing the efficiency of the R-package.

Chapter 3

Materials and Methods

Chapter 3 on materials and methods offers information on the R environment as well as essential background knowledge on fractal theory. It is assumed that the reader has a basic understanding of certain mathematical concepts, particularly: linear algebra, metric spaces and set theory. Unless otherwise noted, any definitions, theorems, and corollaries mention in this chapter are referenced from *Fractals Everywhere* by Michael F. Barnsley [1].

3.1 The R Environment

R is a free programming environment for data analysis and graphics. While R has many applications, it is widely used for its statistical techniques. The source code is accessible online at http://www.r-project.org/ and is available for download on all major platforms, i.e, Linux, MacOS, UNIX and Windows.

R is effective in data manipulation, calculation, and graphical display. Among its many features, the advantages of R arise from

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities [5].

The R environment is a network characterized by its fully planned and coherent system. R can be easily adapted and extended via the creation of new functions and/or "packages." Packages are groups of functions developed for a particular application or area of interest. There are approximately 25 packages supplied with the original download of R. However, thousands of packages are available through the CRAN family of Internet sites (http://cran.r-project.org/).

Because of its versatility, ease of access, effective array operation and superb graphical capabilities, the R environment was the obvious choice for generating fractals via iterated function systems.

3.2 The Space of Fractals

Foremost, consider the ideal space in which to study fractal theory and fractal geometry. Begin by working in a complete metric space, for instance $\mathbb{R}^2$ with the Euclidean metric or the Riemann sphere with the spherical metric, denoted ($\mathbf{X}$,d). The space $\mathcal{H}$ is introduced to analyze pictures, drawings, "black-on-white" subsets of our complete metric space.

Definition 3.2.1. *Let $(\mathbf{X}, d)$ be a complete metric space. Then $\mathcal{H}(\mathbf{X})$ denotes the space whose points are the compact subsets of $\mathbf{X}$, other than the empty set.*

Definition 3.2.2. *Let $(\mathbf{X}, d)$ be a complete metric space, $x \in \mathbf{X}$, and $B \in \mathcal{H}(\mathbf{X})$. Define*

$$d(x, B) = min\{d(x, y) : y \in B\}.$$

$d(x, B)$ is called the **distance from** *the point x to the set B.*

Definition 3.2.3. *Let $(\mathbf{X}, d)$ be a complete metric space. Let $A, B \in \mathcal{H}(\mathbf{X})$. Define*

$$d(A, B) = max\{d(x, B) : x \in A\}.$$

$d(A, B)$ is called the **distance from** *the set $A \in \mathcal{H}(\mathbf{X})$ to the set $B \in \mathcal{H}(\mathbf{X})$.*

Definition 3.2.4. *Let $(\mathbf{X}, d)$ be a complete metric space. The the Hausdorff* **distance** *between points A and B in the $\mathcal{H}(\mathbf{X})$ is defined by*

$$h(A, B) = d(A, B) \vee d(B, A).$$

This metric space $(\mathcal{H}(\mathbf{X}), h)$ is referred to as the "space of fractals."

3.3 Affine Transformations

Definition 3.3.1. *A transformation $w : \mathbb{R}^2 \to \mathbb{R}^2$ of the form*

$$w(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f),$$

where a, b, c, d, e, and f are real numbers, is called a (two-dimensional) **affine** *transformation.*

An affine transformation is often represented as the multiplication of a matrix and a vector, plus a vector, with the following equivalent notations

$$w(\mathbf{x}) = w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = A\mathbf{x} + \mathbf{t}.$$

The general form of an affine transformation, $w(\mathbf{x}) = A\mathbf{x} + \mathbf{t}$, can be described in two parts. A, a 2×2 real matrix, is a linear transformation which deforms space relative to the origin. The multiplication of $A\mathbf{x}$ is followed by a translation or shift specified by the column vector $\mathbf{t}$. Furthermore, the matrix A can also take the following special form.

Definition 3.3.2. *A transformation $w : \mathbb{R}^2 \to \mathbb{R}^2$ is called a* **similitude** *if it is an affine transformation having one of the special forms*

$$w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

$$w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r\cos\theta & r\sin\theta \\ r\sin\theta & -r\cos\theta \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

for some translation $(e, f) \in \mathbb{R}^2$, some real number $r \neq 0$, and some angle θ, $0 \leq \theta < 2\pi$. θ is called the rotation angle while r is called the **scale factor** *or* **scaling**. *The linear transformation*

$$R_\theta\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

is a **rotation**. *The linear transformation*

$$R\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

is a **reflection**.

In the development of this package, it is convenient to treat the affine transformations as an operation of matrices and vectors as the R environment has a distinguished capability of efficiently manipulating such elements.

3.4 Analytic Transformations

Definition 3.4.1. *Let $(\mathbb{C}, d)$ denote the complex plane with the Euclidean metric. A transformation $f : \mathbb{C} \to \mathbb{C}$ is called* **analytic** *if for each $z_0 \in \mathbb{C}$ there is a similitude of the form*

$$w(z) = az + b, \qquad \text{for some pair of numbers } a, b, \in \mathbb{C},$$

such that $d(f(z), w(z))/d(z, z_0) \to 0$ as $z \to z_0$. The numbers a and b depend on z_0.

3.5 Iterated Function System

Definition 3.5.1. *A transformation $f : \mathbf{X} \to \mathbf{X}$ on a metric space $(\mathbf{X}, d)$ is called* **contractive** *or a* **contraction mapping** *if there is a constant $0 \leq s < 1$ such that*

$$d(f(x), f(y)) \leq sd(x, y)\forall x, \qquad y \in \mathbf{X}.$$

Any such number s is called a **contractivity factor** *for f.*

Definition 3.5.2. *A (hyperbolic)* **iterated function system** *consists of a complete metric space $(\mathbf{X}, d)$ together with a finite set of contraction mappings $w_n : \mathbf{X} \to \mathbf{X}$, with respective contractivity factors s_n, for $n = 1, 2, ..., N$.*

From this definition the notation for the IFS is develop as $\{\mathbf{X}; w_n, n = 1, 2, ..., N\}$ and its contractivity factor as $s = max\{s_n : n = 1, 2, ..., N\}$. In practice, the word "hyperbolic" is oftentimes dropped. Simply, IFS can be used to refer to a finite set of maps acting on a metric space, with no particular condition imposed upon the maps. A summary of hyperbolic IFS is presented in the subsequent theorem.

Theorem 3.5.1. *Let $\{\mathbf{X}; w_n, n = 1, 2, ..., N\}$ be a hyperbolic iterated function system with contractivity factor s. Then the transformation $W : \mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$ defined by*

$$W(B) = \bigcup_{n=1}^{n} w_n(B)$$

for all $B \in \mathcal{H}(\mathbf{X})$, is a contraction mapping on the complete metric space $(\mathcal{H}(\mathbf{X}), h(d))$ with contractivity factor s. That is

$$h(W(B), W(C)) \leq sh(B, C)$$

for all $B, C \in \mathcal{H}(\mathbf{X})$. Its unique fixed point, $A \in \mathcal{H}(\mathbf{X})$, obeys

$$A = W(A) = \bigcup_{n=1}^{n} w_n(A)$$

and is given by $A = \lim_{n\to\infty} W^{\circ n}(B)$ for any $B \in \mathcal{H}(\mathbf{X})$.

Definition 3.5.3. *The fixed point $A \in \mathcal{H}(\mathbf{X})$ described in the theorem is called the* **attractor** *of the IFS.*

In Theorem 3.5.1, $A = W(A) = \bigcup_{n=1}^{n} w_n(A)$ is commonly referred to as the Hutchinson operator.

3.6 The Deterministic Algorithm

Principally, there are two algorithms for computing fractals from IFS. First, we consider the Deterministic Algorithm.

Algorithm 3.6.1. The Deterministic Algorithm *Let $\{\mathbf{X}; w_1, w_2, ..., w_N\}$ be a hyperbolic IFS. Choose a compact set $A_0 \subset \mathbb{R}^2$. Then compute successively $A_n = W^{\circ n}(A)$ according to*

$$A_{n+1} = \bigcup_{j=1}^{n} w_j(A_n) \qquad for\ n = 1, 2, ...$$

The Deterministic Algorithm leads to the construction of a sequence $\{A_n : n = 0, 1, 2, 3, ...\} \subset \mathcal{H}(\mathbf{X})$, which by Theorem 3.5.1 converges to the attractor, A^*, of the IFS in the Hausdorff metric.

3.7 The Random Iteration Algorithm

Definition 3.7.1. *An iterated function system* **with probabilities** *consists of an IFS*

$$\{\mathbf{X}; w_1, w_2, ..., w_N\}$$

together with an ordered set of numbers $\{p_1, p_2, ..., p_N\}$, such that

$$p_1 + p_2 + p_3 + \cdots + p_N = 1 \text{ and } p_i > 0 \qquad \text{for } i = 1, 2, ..., N.$$

Thus, the notation for an iterated function system with probabilities (IFSP) is

$$\{\mathbf{X}; w_1, w_2, ..., w_N; p_1, p_2, ..., p_N\}.$$

The second algorithm for generating fractals from IFS is the Random Iteration Algorithm.

Algorithm 3.7.1. The Random Iteration Algorithm *Let* $\{\mathbf{X}; w_1, w_2, ..., w_N\}$ *be a hyperbolic IFS, where probability* $p_i > 0$ *has been assigned to* w_i *for* $i = 1, 2, ..., N$, *where* $\sum_{i=1}^{n} p_i = 1$. *Choose* $x_0 \in \mathbf{X}$ *and then choose recursively, independently,*

$$x_n \in \{w_1(x_{n-1}), w_2(x_{n-1}), ..., w_N(x_{n-1})\} \qquad \text{for } n = 1, 2, 3, ...$$

where the probability of the event $x_n = w_i(x_{n-1})$ *is* p_i. *Thus, construct a sequence* $\{x_n : n = 0, 1, 2, 3, ...\} \subset \mathbf{X}$.

The Random Iteration Algorithm constructs a sequence where x_0 is selected randomly, then $x_1 = w_i(x_0)$, $x_2 = w_i(x_1)$, $x_3 = w_i(x_2)$ and so forth, for $i = 1, 2, ..., N$. This process is repeated a number of times, resulting in a finite sequence of points $\{x_n : n = 0, 1, 2, 3, ...numits\} \subset \mathcal{H}(\mathbf{X})$, where *numits* is a positive integer. This sequence is known as the "orbit." For simplicity, assume that $x_0 \subset A^*$, the attractor. It follows that the $\{x_n : n = 0, 1, 2, 3, ...\}$ would lie in the attractor and converges to A^* of the IFS in the Hausdorff metric.

It is very important to note that, unlike with the Deterministic Algorithm, the initial points computed via the Random Iteration Algorithm are not necessarily located within the attractor of the IFS. It is necessary to eliminate these points which are not subsets of the attractor, from the generated sequence, before it can be concluded that the sequence $\{x_n\}$, of infinity many points, converges to the attractor. The Random Iteration Algorithm is more commonly known as the Chaos Game.

Since neither the Deterministic Algorithm nor the Random Iteration Algorithm can realistically generate infinitely many subsets. The functions defined for each algorithm will be able to generate a large, but finite, number of subsets in the attractor, which yield a good approximation of the attractor.

3.8 Measures

Definition 3.8.1. *Let* $\mathbf{X}$ *be a space. Let* $\mathcal{F}$ *denote a nonempty class of subsets of a space* $\mathbf{X}$, *such that*

$$(1) A, B \in \mathcal{F} \Rightarrow A \cup B \in \mathcal{F};$$
$$(2) A \in F \Rightarrow \mathbf{X} \backslash A \in \mathcal{F}.$$

Then $\mathcal{F}$ *is called a* **field**.

Definition 3.8.2. *A* **measure** μ, *on a field* $\mathcal{F}$, *is a real nonnegative function* $\mu : \mathcal{F} \to [0, \infty) \subset \mathbb{R}$, *such that whenever* $A_i \in \mathcal{F}$ *for* $i = 1, 2, 3, ...$, *with* $A_i \cap A_j = \emptyset$ *for* $i \neq j$ *and* $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$, *we have*

$$\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i).$$

Let $(\mathbf{X}, d)$ be a complete metric space. Let $\{\mathbf{X}; w_1, ..., w_N; p_1, ..., p_N\}$ be an IFSP. Let A denote the attractor of the IFS. Then there exists a concept called the *invariant measure* of the IFS, which we denote here by μ. μ assigns "mass" to many subsets of $\mathbf{X}$. For example, $\mu(A) = 1$ and $\mu(\emptyset) = 0$. That is, the "mass" of the attractor is one unit, and the "mass" of the empty set is zero. Also $\mu(\mathbf{X}) = 1$, which says that the whole space has the same "mass" as the attractor of the IFS; the "mass" is located on the attractor [1].

Not all subsets of $\mathbf{X}$ have a "mass" assigned to them. The subsets of $\mathbf{X}$ that do have a "mass" are called the *Borel subsets* of $\mathbf{X}$, denoted by $\mathcal{B}(\mathbf{X})$. The Borel subsets of $\mathbf{X}$ include the compact nonempty subsets of $\mathbf{X}$, so that $\mathcal{H}(\mathbf{X}) \subset \mathcal{B}(\mathbf{X})$ [1].

Definition 3.8.3. *Let ($\mathbf{X}$ be a metric space. Let $\mathcal{B}$ denote the Borel subsets of $\mathbf{X}$. Let μ be a measure on $\mathcal{B}$. Then μ is called a* **Borel measure**.

Theorem 3.8.1. *Let $(\mathbf{X}, d)$ be a compact metric space. Let*

$$\{\mathbf{X}; w_1, w_2, ..., w_N; p_1, p_2, ..., p_N\}$$

be a hyperbolic IFS with probabilities. Let $(\mathbf{X}, d)$ be a compact metric space. Let $\{x_n\}_{n=0}^{\infty}$ denote an orbit of the IFS produced by the Random Iteration Algorithm, starting at x_0. That is,

$$x_n = w_{\sigma_n} \circ w_{\sigma_{n-1}} \circ \cdots \circ w_{\sigma_1}(x_0).$$

where the maps are chosen independently according to the probabilities

$$p_1, p_2, ..., p_N, \qquad \text{for } n = 1, 2, 3, ...$$

Let μ be the unique invariant measure for the IFS. Then with probability one (that is, for all code sequences $\sigma_1, \sigma_2, ...$ except for a set of sequences having probability zero).

$$\lim_{n\to\infty} \frac{1}{n+1} \sum_{k=0}^{n} f(x_k) = \int_X f(x) d\mu(x)$$

for all continuous functions $f : \mathbf{X} \to \mathbb{R}$ and all x_0.

Corollary 3.8.1. *Let B be a Borel subset of $\mathbf{X}$ and let $\mu(\text{boundary of} B) = 0$. Let $\mathcal{N}(B, n) =$ number of points in $\{x_0, x_1, x_2, x_3, ..., x_n\} \cap B$, for $n = 0, 1, 2, ...$ Then, with probability one,*

$$\mu(B) = \lim_{n\to\infty} \{\frac{\mathcal{N}(B,n)}{(n+1)}\}$$

for all starting points x_0. That is, the "mass" of B is the proportion of iteration steps, when running the Random Iteration Algorithm, which produces points in B.

Chapter 4

Results

The results section of this master project presents the functions developed for an R-package for generating fractal via iterated function systems. Each function is described in its respective section and important links are draw to the fractal theory described in the previous chapter.

4.1 Determ_2R

The Determ_R2 function (Section B.1 of Appendix B) generates an image of a fractal based on the Deterministic Algorithm (Algorithm 3.6.1) described in Chapter 3.

The function takes as input a previously defined IFS as well as the number of desired compact sets, n,to be computed from the initial set A_0. The initial set, A_0, consists of a single point generated randomly from the uniform distribution on the interval $[0, 1]$. In reality, this initial set can lie anywhere in $\mathbb{R}^2$ and consist of any finite number of points.

Next, we apply each of the mappings in the IFS to the set A_0, that is $w_j(A_0)$ for all w in the IFS. Take for instance the IFS defined for the black spleewort fern in Table A.2. This IFS consists of 4 mappings. The calculations below create the new compact set A_1.

$$A_1 = \{w_1(A_0), w_2(A_0), w_3(A_0), w_4(A_0)\}$$

Determ_R2 contains a loop that constructs and plots compact sets $A_0, A_1, A_2, ..., A_n$.

Note that for the black spleewort fern IFS the first iteration creates 4 points, the second iteration creates 16 points, the third creates 64 points, the forth creates 256 points and so on. By $n = 8$, 65,536 points have been generated. The following figure illustrates A_0 through A_8 for the fern.

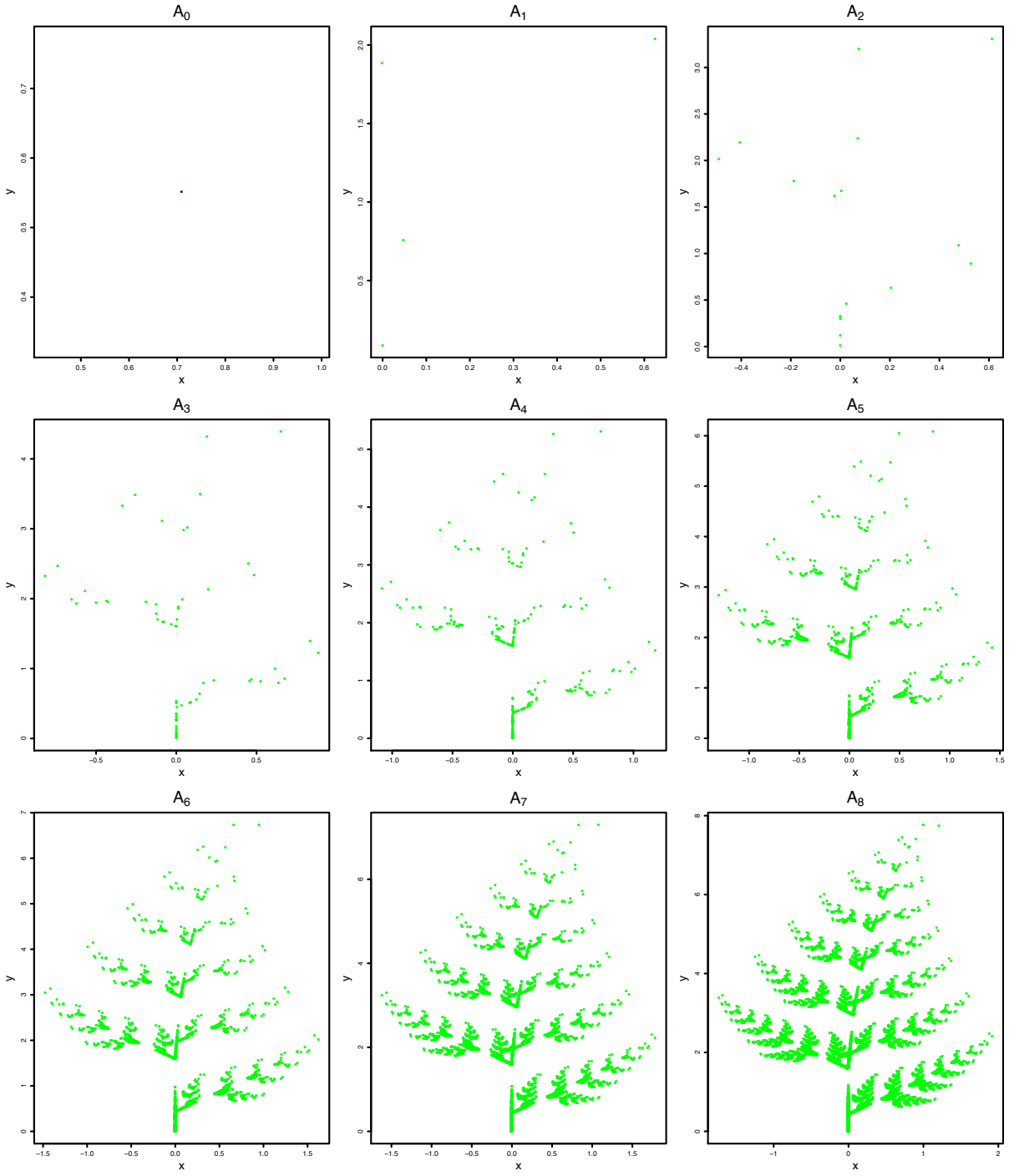


Figure 4.1: Progression of a Black Spleewort Fern in $\mathbb{R}^2$ by Implementing the Deterministic Algorithm

Implementing this function in the R environment requires approximately 24 seconds to create the above sequence of images. However, the greatest disadvantage of the Deterministic Algorithm is that the required number of mappings increases exponentially as n increases. Consider the IFS for the Sierpinski Triangle in $\mathbb{R}^2$, Table A.1. Let $n = 11$

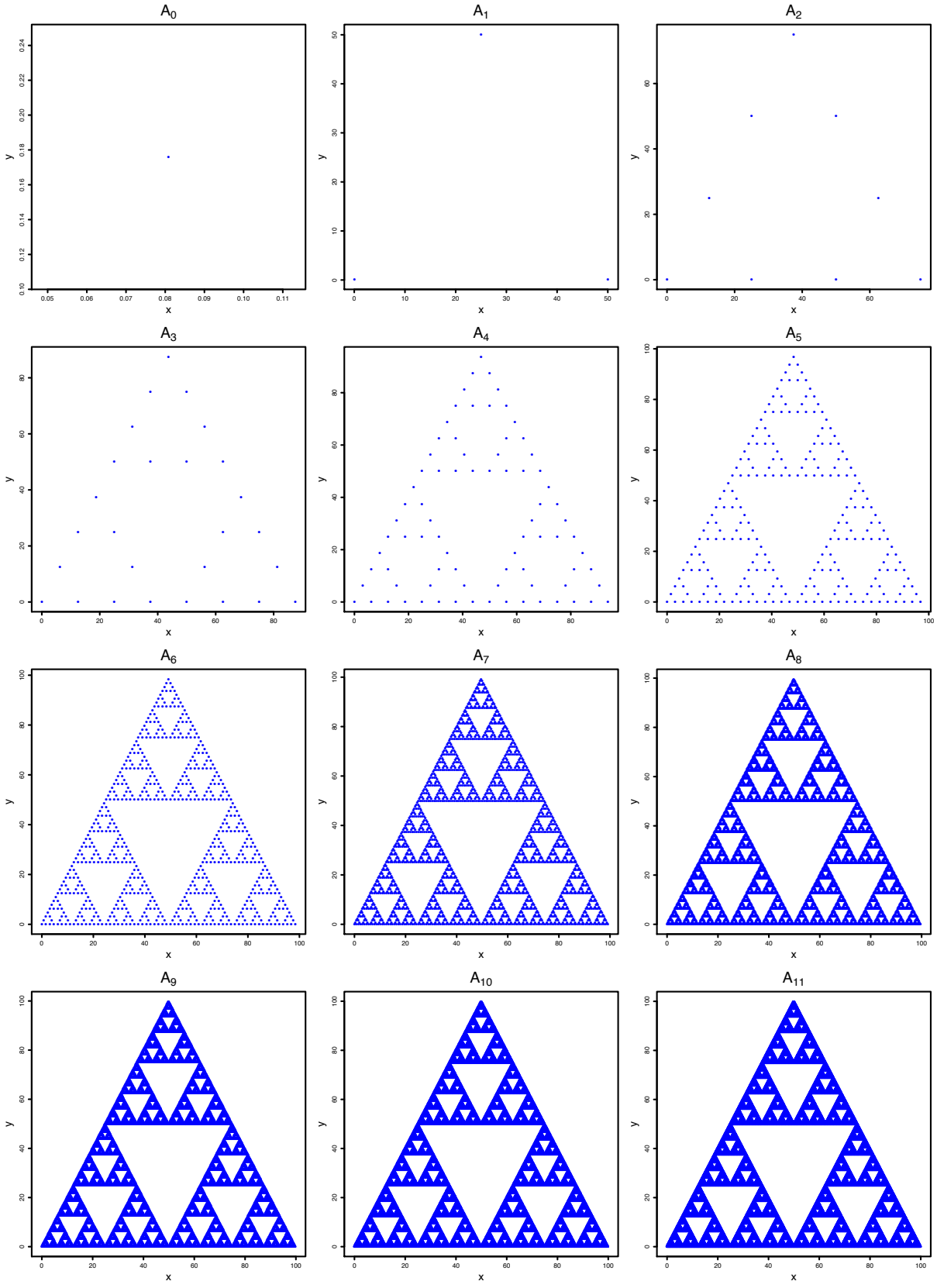


Figure 4.2: Progression of a Sierpinski Triangle in $\mathbb{R}^2$ by Implementing the Deterministic Algorithm

Determ_R2 generates A_{11}, with 177,147 points, in approximately 215 seconds, a significant increase in computational time. An even larger increase is seen when computing A_{12}, with 531,441 points. A_{12} requires approximately 2155 seconds.

This function can only generate an approximation of the attractor from a finite number of points. However, it is obvious that the attractor of IFS illustrated is the Sierpinski Triangle. It is observed that by $n = 8$ the estimation of attractor clearly indicates the Sierpinski Triangle fractal. This approximation is, more often than not, sufficient and significantly more time effective.

4.2 RandIt_R2

RandIt_R2 (Section B.2 of Appendix B) constructs an image of a fractal based on the Random Iteration Algorithm (Algorithm 3.7.1) as denoted in the previous chapter.

The function takes as input a previously defined IFSP as well as the number of desired points, n, to be computed from the initial point x_0. The initial point, x_0, consists of a single point in the form of a two-element column vector generated randomly from the uniform distribution on the interval $[0, 1]$. Again, this initial set can lie anywhere in $\mathbb{R}^2$.

Error checking in the RandIt_R2 ensures that the length of the probability vector defined for the IFSP is equivalent to the number of mappings in the IFS. The $\sum_{i=1}^{n} p_i = 1$. is also verified. If either of these conditions are not met, the user will receive an error message describing the discrepancy.

The function proceeds by computing n points using the Random iteration Algorithm. Recall, that the when implementing the Random Iteration algorithm it is not guaranteed that the initial points are a subset of the attractor. To make certain that the points plotted for the approximation of the attractor are indeed subsets of the attractor, the function RandIt_R2 plots point vectors $\{x_{5001}, x_{5002}, ..., x_n\}$.

If the function was not programmed to eliminate the first 5000 points of the sequence, the user would see points dispersed throughout the plot which do not lie in the attractor. Figure 4.3 depicts four iterations of the Random Iteration Algorithm for the Sierpinski Triangle attractor, without the elimination of the initial points of the constructed sequence. The first 20 points of the generated sequence of points are in blue. The reader can see that several of these points, in each case, are not located within the attractor.

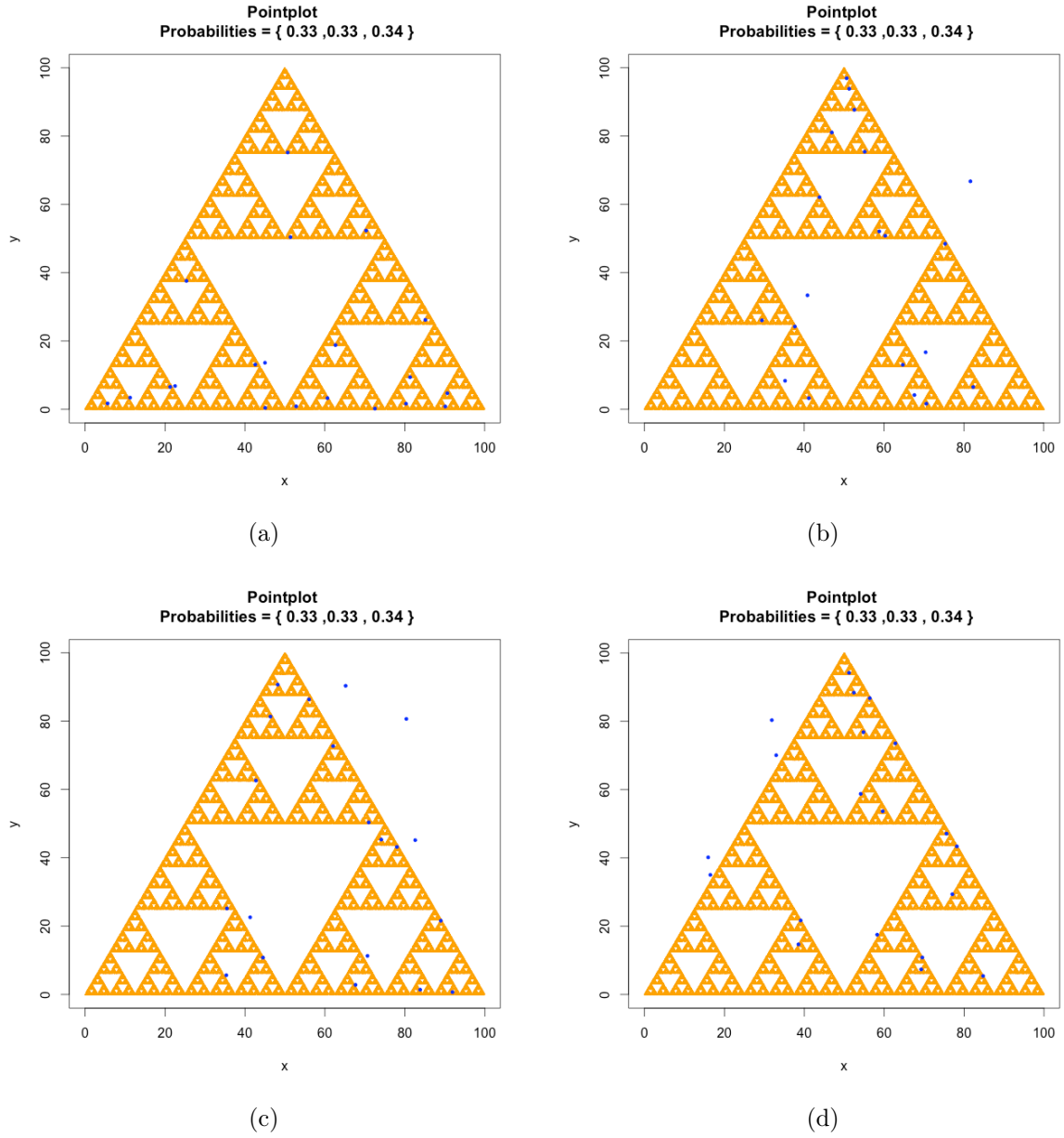


Figure 4.3: Random Iterated Function for the Sierpinski Triangle Attractor with Initial Points

Nevertheless, the goal is to provide fractal approximations where all points are in a subset of the attractor, so the first 5000 points are eliminated from the plots. The RandIt_R2 function outputs a pointplot of $n - 5000$ points to approximate the attractor. With the aid of the *ggplot* R-package, a second plot containing a two-dimensional histogram is drawn. This histogram supplies the user with information about the density of points in a specific region of the fractal image. The two-dimensional histogram in the RandIt_R2 function divides the plot region into 600 bins. These bins act as baskets collecting and counting the points that fall within and then consequently assigning each bin a color depending on its density. The fractal in Figure 4.4 is the attractor of the IFS

code for the Black Spleewort Fern (Table A.2) generated with the RandIt_R2 function and $n = 100,000$ in about 55 seconds. A close-up of the edge of the leaves at the tip of the blade is seen in Figure 4.5. Note there is a higher density at the tips of the leaves, evident from the magenta and yellow colors.

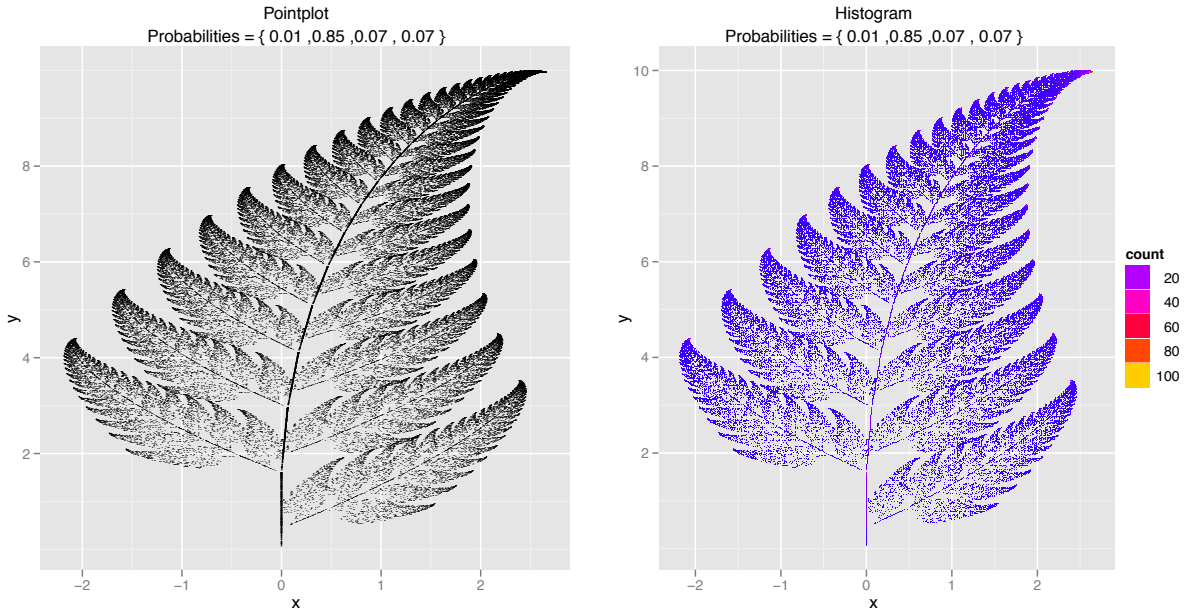


Figure 4.4: Black Spleewort Fern Generated by RandIt_R2

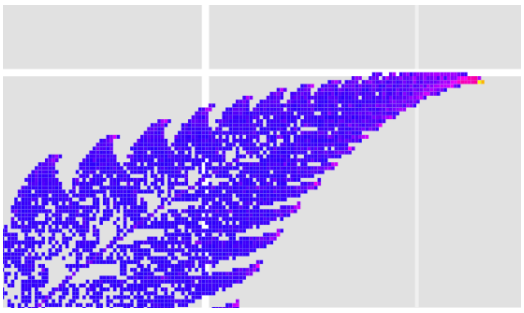


Figure 4.5: Close-Up of Black Spleewort Fern

The RandIt_R2 function is also capable of handling similitudes, or affine transformations of the special form

$$w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r\cos\theta & -s\sin\phi \\ r\sin\theta & s\cos\phi \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} h \\ k \end{pmatrix}$$

The IFS code in Table A.12 is for the same Black Spleewort Fern but in the scale and angle format, where the affine transformation is described as translations, rotations and

scalings. Figure 4.6 illustrates how RandIt_R2 is able to produce the same image when the IFS is input in scale and angle format.

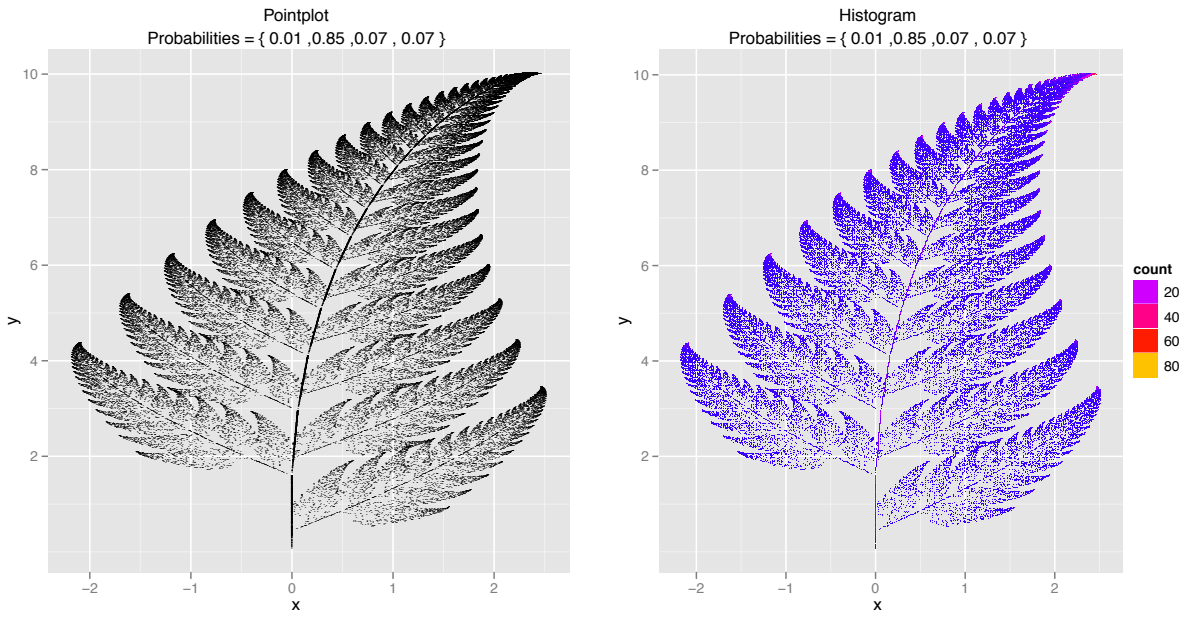
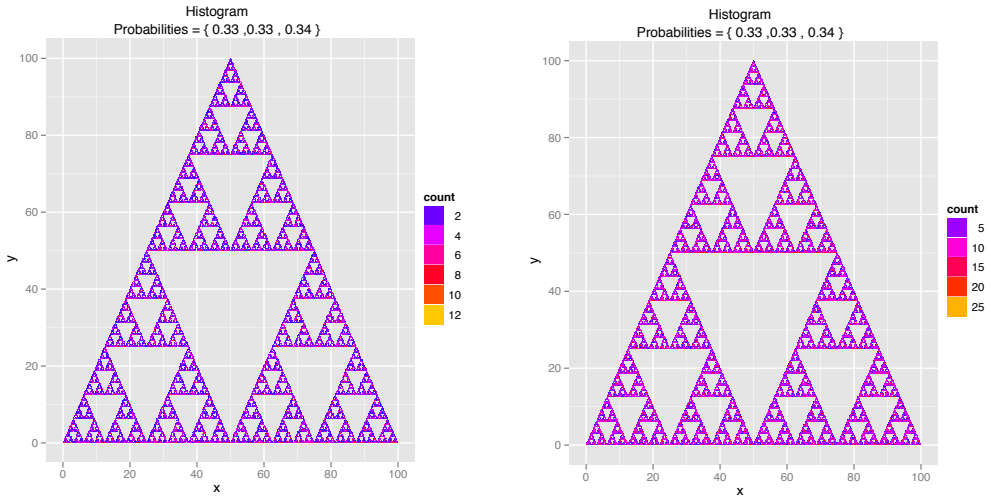


Figure 4.6: Fern Generated by RandIt_R2 in Scale and Angle Format

The application of the RandIt_R2 function, $n = 100,00$, yields a set of 95,001 points, plotted in Figure 4.7 (a). Figure 4.7 (b), the Hutchinson operator of this set is shown, which consists of $3 \times 95,001$ points in case of an this IFS with 3 functions.



(a) RandIt_R2 for the Sierpinski Triangle where $n = 100,000$

(b) RandIt_R2 Plus Hutchinson Operator for the Sierpinski Triangle

Figure 4.7: Random Iteration Algorithm vs. Deterministic Algorithm for a Triangle

Aside from a longer computational time and the greater number of points plotted in Figure 4.7 (b), the fractal approximations are identical, and it is evident that infinity many points in both images would converge to the same attractor, the Sierpinski Triangle. It is clear that the Random Iteration Algorithm is just as effective as the Deterministic Algorithm, but without the exponential growth in required calculations and with a much shorter computational time.

4.3 RandIt_R3

RandIt_R3 (Section B.3 of Appendix B) generates an image of a fractal based on the Random Iteration Algorithm (Algorithm 3.7.1) in $\mathbb{R}^3$.

Like RandIt_R2 this function takes as input a previously defined IFS, a vector $\mathbf{p}$ of probabilities, as well as the number of desired points, n, to be computed from the initial point x_0. But in this case the IFS is of the type

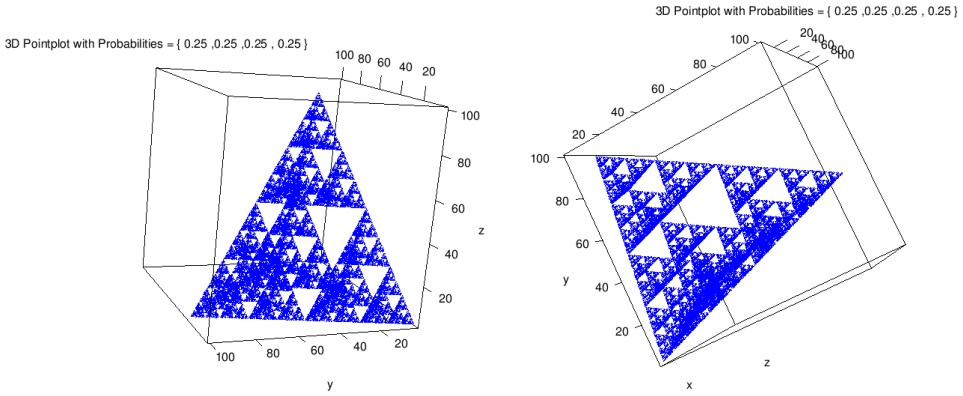
$$w\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} j \\ k \\ l \end{pmatrix} = A\mathbf{x} + \mathbf{t}$$

The initial point, x_0, consists of a single point in the form of a three-element column vector generated randomly from the uniform distribution on the interval $[0, 1]$. This initial point can lie anywhere in $\mathbb{R}^3$.

Error checking in the RandIt_R3 also guarantees that the length of the probability vector defined for the IFSP is equivalent to the number of mappings in the IFS and that $\sum_{i=1}^{n} p_i = 1$.

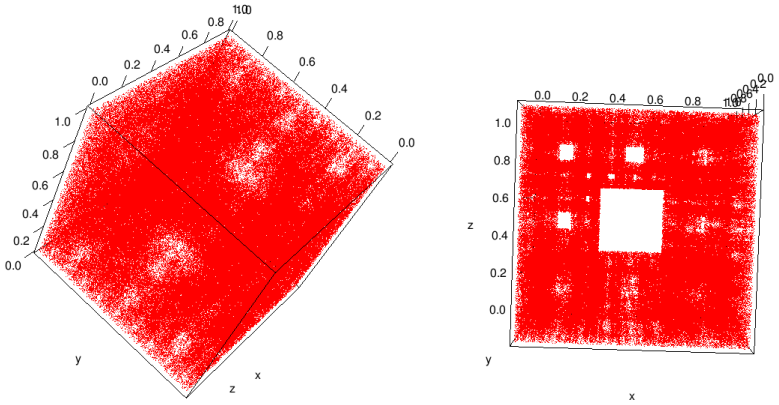
The function proceeds by computing n points using the Random iteration Algorithm. The function RandIt_R3 plots point vectors $\{x_{5001}, x_{5002}, ..., x_n\}$ to make certain that the points plotted for the approximation of the attractor are indeed subsets of the attractor. To obtain three-dimensional fractal approximations the *rgl* R-package is loaded.

Figure 4.8 depicts a pointplot for a Sierpinski Triangle in $\mathbb{R}^3$, IFSP in Table A.9. The approximation of the attractor was computed in 45 seconds, when $n = 100,000$. Figure 4.9 illustrates a pointplot for a Merger Sponge in $\mathbb{R}^3$, IFSP in Table A.11, with computational time equal to 677 seconds, when $n = 400,000$ Lastly, a three-dimensional fractal approximation is generated for the IFSP of a fern, in Table A.10, via the RandIt_R3 function in 171 seconds, for $n = 200,000$. The resulting image is shown in Figure 4.10.



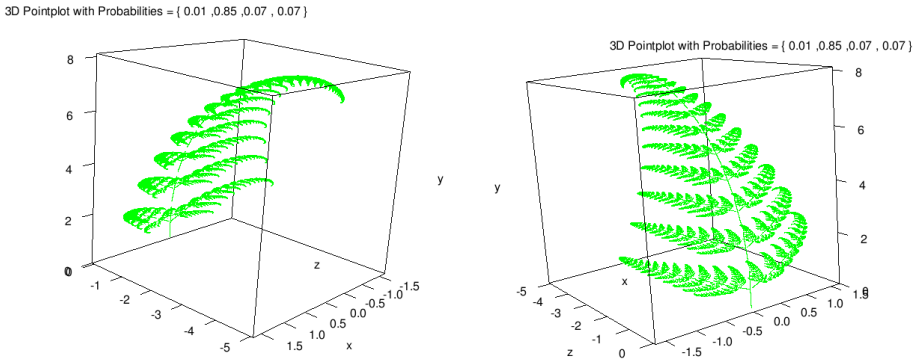
(a) (b)

Figure 4.8: Pointplot for a Sierpinski Triangle in $\mathbb{R}^3$ from Different Angles



(a) (b)

Figure 4.9: Pointplot for a Merger Sponge in $\mathbb{R}^3$ from Different Angles



(a) (b)

Figure 4.10: Pointplot for a Fern in $\mathbb{R}^3$ from Different Angles

4.4 RandIt_C

The RandIt_C function (Section B.4 of Appendix B) computes and plots the an image of a fractal based on the Random Iteration Algorithm (Algorithm 3.7.1) in $\mathbb{C}$.

In RandIt_C the IFSP are analytic transformations. The function inputs an IFSP of the type shown in sections A.12 and A.13 of Appendix A, as well as the number of points to be computed in three-dimensional space, n. Similarly to RandIt_R2, the initial point, z_0, consists of a single point in the form of complex number generated randomly from the uniform distribution on the interval $[0, 1]$. This initial point can lie anywhere in $\mathbb{C}$.

The function proceeds by computing n points using the Random iteration Algorithm. The function RandIt_C plots complex numbers $\{z_{5001}, z_{5002}, ..., z_n\}$ to make certain that the points plotted for the approximation of the attractor are indeed subsets of the attractor. Once more the *ggplot* R-package is implemented for plotting.

The result of applying the RandIt_C to the IFSP for a square in $\mathbb{C}$, as defined in Section A.13 of Appendix A, is shown below.

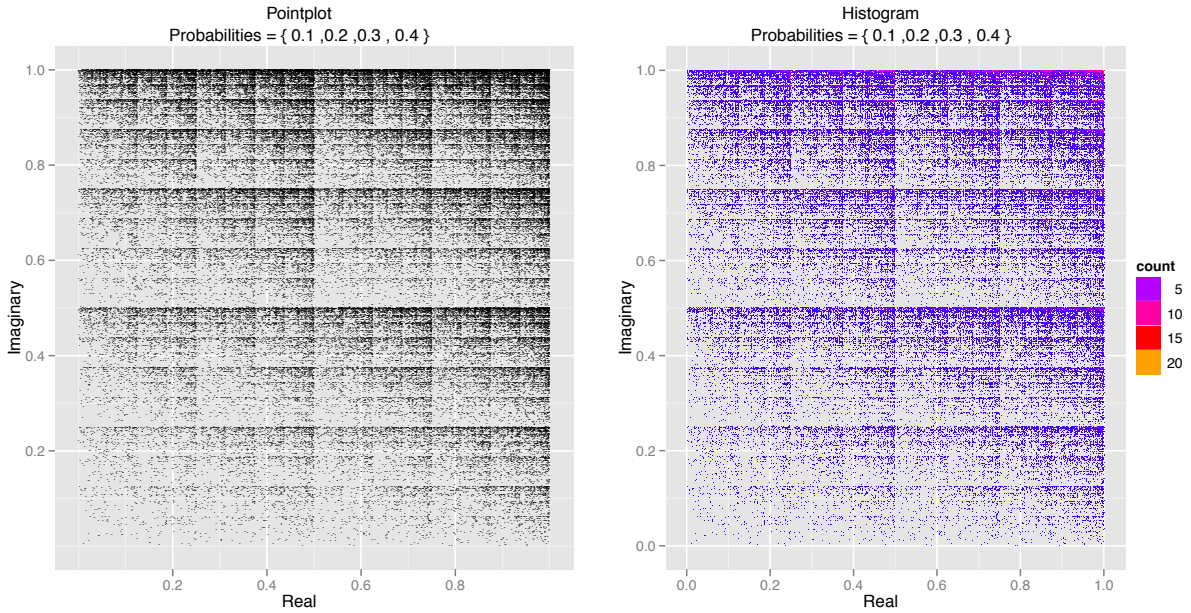


Figure 4.11: Square Attractor Approximation Generated by RandIt_C

The square depicted above contains 95,001 points in the complex plane and was generated in approximately 11 seconds. The RandIt_C function appears to be have a faster computational time than the RandIt_R2 function. This is due to R's capability of dealing with complex numbers. There are similarities between $\mathbb{R}^2$ and $\mathbb{C}$, however, the user must be careful when defining a metric, since vector operations are not the same as complex operations.

4.5 IFS_Prob

The IFS_Prob function randomly generates a vector p of probabilities for a given IFS, based on the number of transformations in said IFS.

Let n be the number of transformations in the given IFS. Once the input, n, is defined the function, internally, generates a random value p_1 from the uniform distribution on the interval $(0, 1]$. The next probability, p_2, is generated randomly from the uniform distribution on the interval $(0, 1 - p_1]$, p_3 is selected randomly from the interval $(0, 1-p_1-p_2]$, and so on until p_{n-1} is assigned. Lastly, define $p_n = 1 - \sum_{i=1}^{n-1} p_i$ to ensure the sum of the elements composing the probability vector p is 1, that is

$$\sum_{i=1}^{n} p_i = 1.$$

Note that $p_i > 0$ for all p_i. The function output is a probability vector, p, of the form

$$p = [\ p_1\ p_2\ \ldots\ p_n\ ].$$

4.6 IFSP_Gen_R2

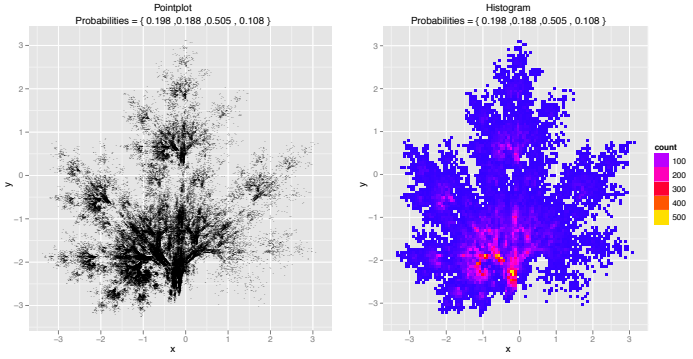
The user can generate a selected number, t, of fractal images with varying probabilities by implementing the single function IFSP_Gen_R2.

The variety probabilities for each iteration are selected randomly via the previously defined function, IFS_Prob. The IFSP_Gen_R2 function uses the same code as the Rand_R2 function to preform the Random Iteration Algorithm.

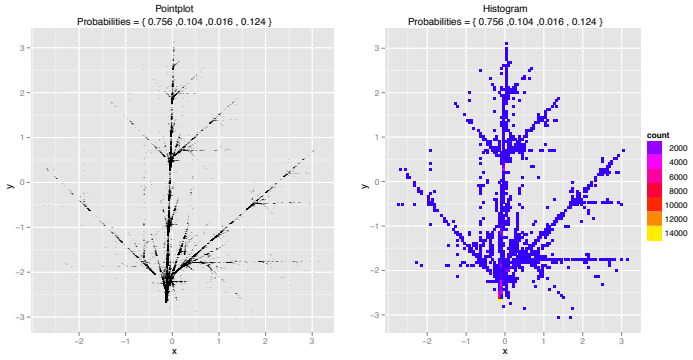
The *ggplot* package is used to generate the two-dimensional histograms for IFSP_Gen_R2. The bin size of the histograms was changed to 100 bins, so as to more clearly indicate variations in plot density. The output of four iterations, $t = 4$, on the Maple Leaf IFS (Table A.8) is shown figure 4.12.

Notice, as the probabilities of the probability vector change so do the depicted and stressed areas of the fractal approximation. Figure 4.12 (a) has a high p_3, which indicates that the area with the greatest density is the middle left portion of the maple leaf. p_1, in Figure 4.12 (b), clearly correspond to w_1 and the stem of the leaf. Figure 4.12 (c), illustrates how w_2 maps to the middle right portion of the maple leaf. The probabilities in Figure 4.12 (d), are more favorably balanced, as most of the maple leaf is visible.

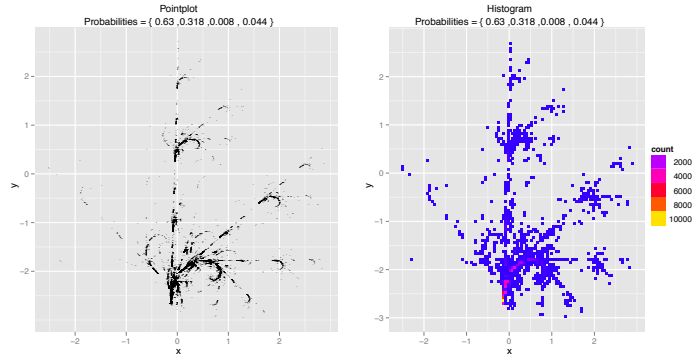
If the ideal probability vector of an IFS is unknown, the IFSP_Gen_R2 function is a quick method to iteratively test different probabilities and find relationships between the mappings and the image. Thus, an estimate of the ideal probability vector can be drawn for a given IFS.



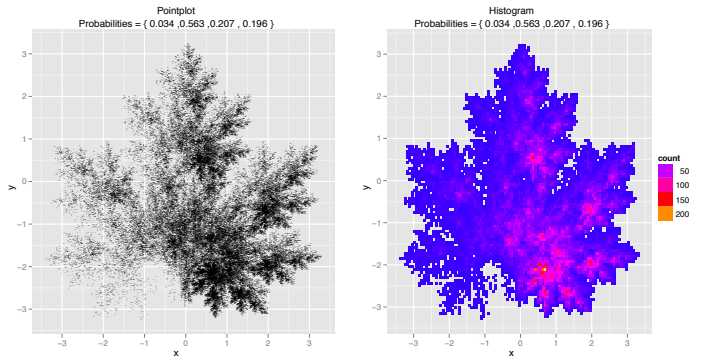
(a)



(b)



(c)



(d)

Figure 4.12: Four Iterations with Varying Probability Vectors

4.7 FractalMeas_R2

The FractalMeas_R2 function computes fractal images in much the same way the RandIt_R2 function does, using the Random Iteration Algorithm. However, instead of analyzing the density of the points with a two-dimensional histogram, the measure, μ, is calculated according to Corollary 3.8.1.

$$\mu(B) = \lim_{n\to\infty}\left\{\frac{\mathcal{N}(B,n)}{(n+1)}\right\}$$

The user must input a previously define IFS (ifs), the number of points to be generated (n), a vector of probabilities (p), and the limits of x ($xlim$) and y ($ylim$) to denote the area of interest, where the user wishes to calculate the measure. The limits of x and y are to be designated as vectors of length two, such that

$$xlim = [x_1, x_2] \quad \text{and} \quad ylim = [y_1, y_2],$$

where

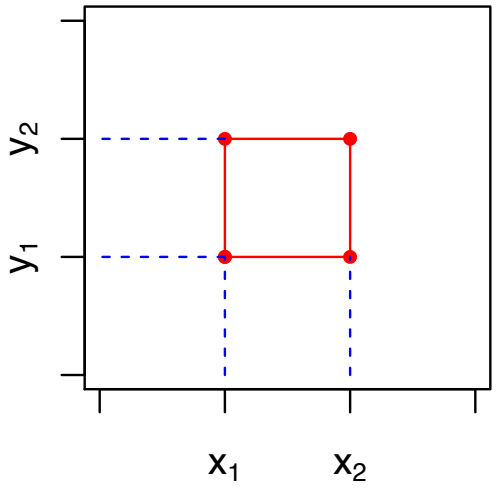


Figure 4.13: Limits of Borel Subset

Firstly, the orbit is generated via the Random Iteration Algorithm. Next, given the limits of the Borel subset, the points within the subset are counted and the measure, $\mu(B)$, is determined. The final output includes a pointplot of the fractal, a red rectangle delimiting the Borel subset and the calculated measure as shown in the next figure. The IFSP used is defined in Section A.4 of Appendix A.

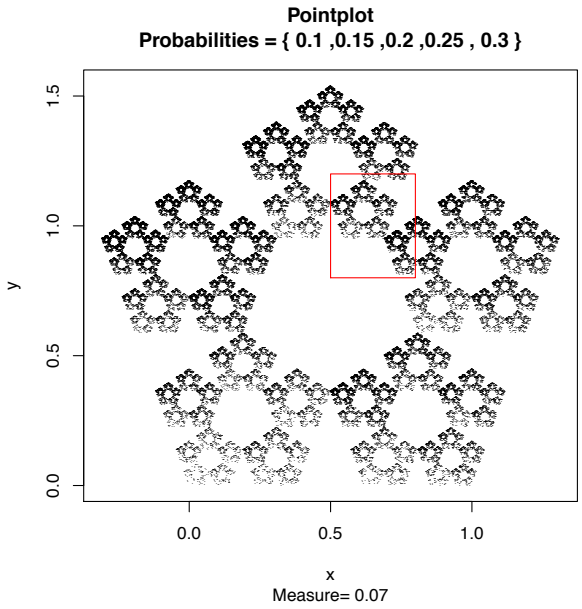
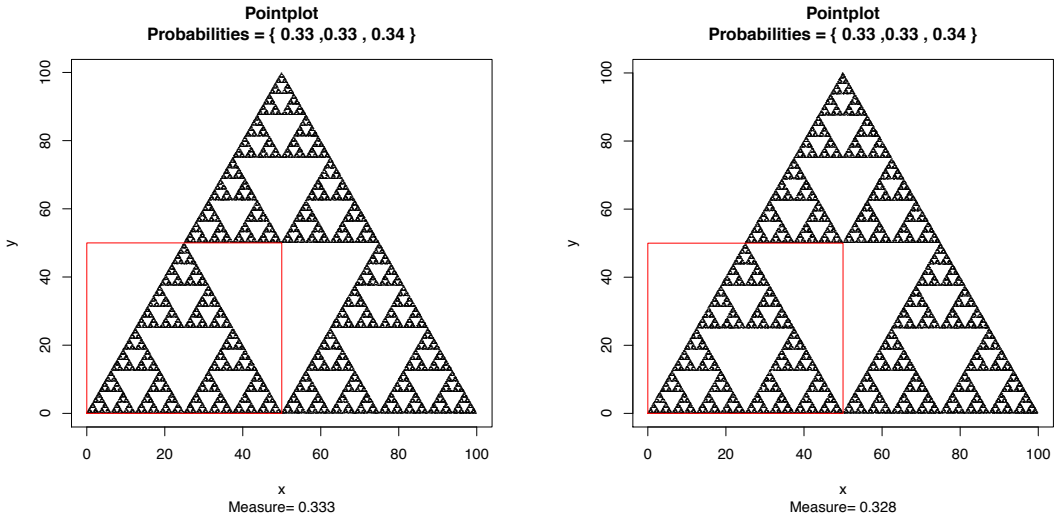


Figure 4.14: Measure of a Borel Subset of $\mathbf{X}$

Remember that the measure, μ, assigns a "mass" to the subsets of $\mathbf{X}$. Thus, if the probabilities of an IFSP are equivalent for each function in the IFSP, it can be expected that selecting a Borel subset of $\mathbf{X}$ which encompasses one-third of the attractors area will yield $\mu(B) = 0.333$. This fact is evident from the pointplot of the Sierpinski Triangle shown on the next page.



(a) (b)

Figure 4.15: Limits of Borel Subset

Figure 4.15 (a) yields a measure of 0.333 while Figure 4.15 (b) calculates a measure of 0.328. This small inconstancy is due to the randomness of the Random Iteration algo-

rithm. For the above images, 100,000 points were generated. The Hutchinson operator assures that the larger the value of n, the smaller the variation among iterations of the measure of the same Borel subset because the union of the generated subsets of points coverages to the attractor.

It can be shown that the measure of the attractor is one, that is $\mu(A) = 1$, see Figure 4.16.

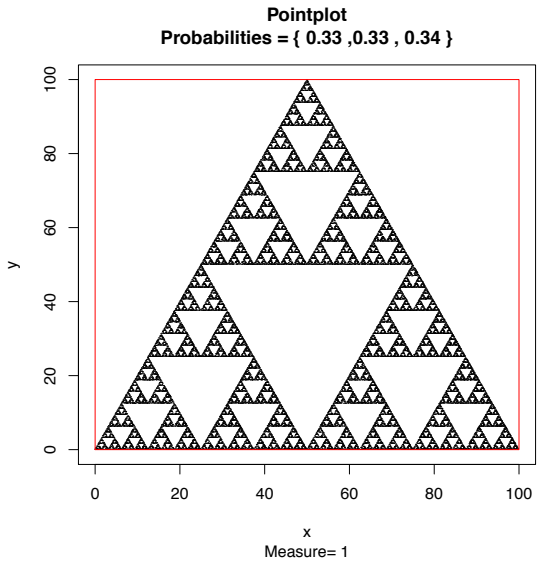


Figure 4.16: $\mu(A) = 1$

Furthermore, it can be shown that the measure of the empty set is zero, that is $\mu(\emptyset) = 0$, see Figure 4.17.

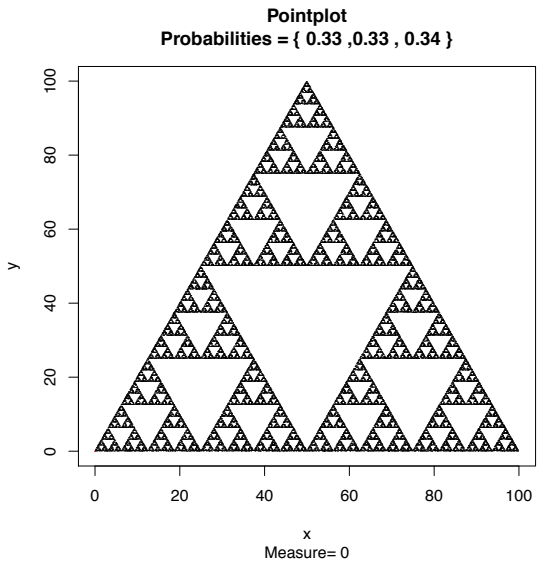


Figure 4.17: $\mu(\emptyset) = 0$

4.8 TransMat_R2

Finally a very special case of the chaos game can be analyzed with the TransMat_R2 function.

The input for the function TransMat_R2 is a matrix, M, which has nonnegative entries, has no row or column that has all entries equal to zero and the sum of the probabilities is one, i.e., $\sum_{i=1}^{n} p_i = 1$ [3]. This matrix is referred to as a transformation matrix. For this particular function the matrix M, additionally is quadratic and has at least 2 rows. Error checking for each of these conditions is added to the function to ensure correct execution.

The affine mapping is defined $w_{ji} : [0,1]^2 \to [a_{j-1}, a_j] \times [b_{i-1}, b_i]$, $1 \leq i, j \leq 3$, by

$$w_{ji}(x, y) = (a_{j-1} + x(a_j - a_{j-1}), b_{i-1} + x(b_i - b_{i-1}))$$

where a and b are the cumulative column and row sums, respectively, of M.

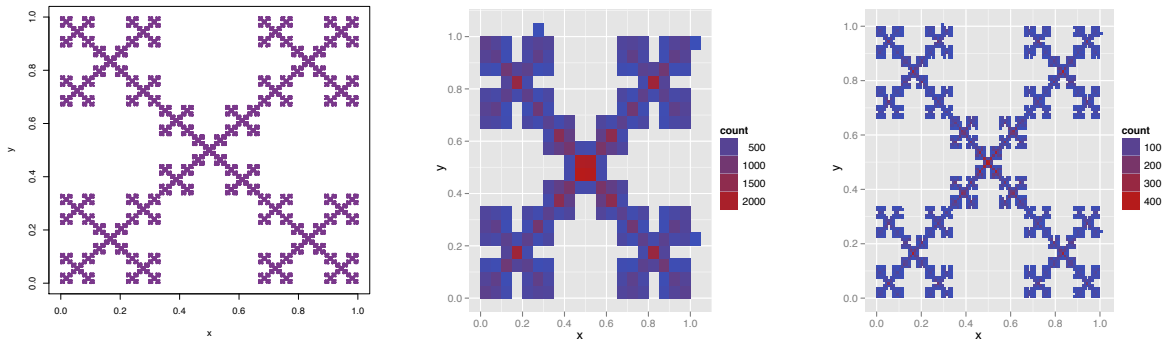
Previously, the Random Iteration Algorithm was preformed on a constant IFS and the user could vary the probability vector using the IFSP_Gen_R2 function. Now, the affine transformation will vary dependent to the input transformation matrix.

The number of points to be generated is input as n. And like before, the function preforms the Random Iteration Algorithm and uses the *ggplot* R-package to output useful plots. This function allows the user to better understand the effects of the values of p_i on the fractal.

Let

$$M = \begin{pmatrix} 1/6 & 0 & 1/6 \\ 0 & 1/3 & 0 \\ 1/6 & 0 & 1/6 \end{pmatrix}.$$

The result with $n = 100,000$ is presented in Figure 4.18.



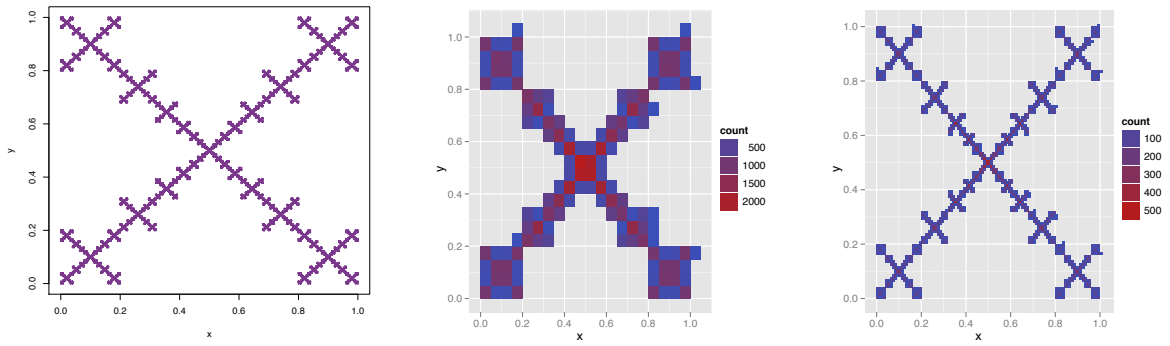
(a) Pointplot (b) 2-D Histogram with 20 Bins (c) 2-D Histogram with 100 Bins

Figure 4.18: Transformation Matrix with Changing IFS

Notice that the center square has the greatest frequency since it corresponds to the largest selection probability, $\frac{1}{3}$, in matrix M. The corners have a smaller density corresponding to the corner entries of M, $\frac{1}{6}$. Now change the transformation matrix M and observe how the image changes.

Let

$$M = \begin{pmatrix} 0.1 & 0 & 0.1 \\ 0 & 0.6 & 0 \\ 0.1 & 0 & 0.1 \end{pmatrix}.$$



(a) Pointplot (b) 2-D Histogram with 20 Bins (c) 2-D Histogram with 100 Bins

Figure 4.19: Transformation Matrix with Changing IFS

Chapter 5

Conclusions

The final chapter summarizes the outcome and results of this master project. Additionally, the conclusion considers possible improvements and future work.

5.1 Outcome

Throughout the master project a deeper understanding of fractal theory was achieved. The programming of new functions and R-packages was learned, and a deeper and more proficient use of the R environment was gained.

The outcome of this master project is the development of eight functions to compose an R-package for generating fractals via iterated function systems. Determ_R2 was created to generate fractals in the two-dimensional vector space of real numbers, $\mathbb{R}^2$, via the Deterministic Algorithm. The functions RandItl_R2 builds an approximation of the attractor in $\mathbb{R}^2$ via an affine transformation and the chaos game. RandIt_R3 generates a fractal in $\mathbb{R}^3$ and plots a three-dimensional model of the fractal approximation, also implementing the chaos game and an IFS of affine transformations. RandIt_C was programed to build fractals in complex plane ($\mathbb{C}$), using analytic transformations and the Random Iteration Algorithm. A function, IFS_Prob, was developed to randomly generate probabilities for a hyperbolic IFS. Additionally, IFSP_Gen_R2 was developed to recursively test random probabilities for a given IFS. This function is useful in determining the ideal probability vector $\mathbf{p}$, if the probabilities are unknown. The function FractalMeas_R2 calculates the measure of a Borel subset of $\mathbf{X}$, given the limiting coordinates of said Borel subset. Lastly, TransMat_R2 approximates the attractor for an changing IFS induced by a transformation matrix.

Experimentation with the developed functions makes it clear that the Random Iteration Algorithm is more effective and time efficient than the Deterministic Algorithm. Since only a large finite number of iterations can be performed it is necessary to note that the fractals developed are simply approximations of the attractor, to which the infinite series of subset would converge to in the Hausdorff metric. Implementations of theses functions provides a more complete of fractal theory that can be applied to problem solving and applications.

5.2 Future Work

Overall, a giant stride has been taken in developing an R-package for generating fractals via iterated function systems. However, the completion of this R-package might possibly include more functions to manipulate and experiment with fractals in $\mathbb{R}^2$ but more particularly in $\mathbb{R}^3$ and $\mathbb{C}$. The inclusion of additional error checking is necessary. Further research in fractal theory and the R environment will lead to a more efficient, robust and computationally time optimal R-package. Additionally, it is essential to develop user help documentation as well as a manual for the R-package for generating fractals via iterated function systems.

Appendix A

Iterated Function Systems

Appendix A contains a complete collect the IFS implemented and developed during this master project.

A.1 IFS Code for a Sierpinski Triangle in $\mathbb{R}^2$ [1]

Table A.1: IFS Code for a Sierpinski Triangle in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.5	0	0	0.5	0	0	0.33
2	0.5	0	0	0.5	25	50	0.33
3	0.5	0	0	0.5	50	0	0.34

A.2 IFS Code for a Black Spleewort Fern in $\mathbb{R}^2$ [1]

Table A.2: IFS Code for a Black Spleewort Fern in $\mathbb{R}^2$

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

A.3 IFS Code for a Sierpinski Carpet in $\mathbb{R}^2$ [5]

Table A.3: IFS Code for a Sierpinski Carpet in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.333	0	0	0.333	0	0	0.125
2	0.333	0	0	0.333	0	0.333	0.125
3	0.333	0	0	0.333	0	0.667	0.125
4	0.333	0	0	0.333	0.333	0	0.125
5	0.333	0	0	0.333	0.333	0.667	0.125
6	0.333	0	0	0.333	0.667	0	0.125
7	0.333	0	0	0.333	0.667	0.333	0.125
8	0.333	0	0	0.333	0.667	0.667	0.125

A.4 IFS Code for a Sierpinski Pentagon in $\mathbb{R}^2$ [5]

Table A.4: IFS Code for a Sierpinski Pentagon in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.382	0	0	0.382	0	0	0.1
2	0.382	0	0	0.382	0.618	0	0.15
3	0.382	0	0	0.382	0.809	0.588	0.2
4	0.382	0	0	0.382	0.309	0.951	0.25
5	0.382	0	0	0.382	-0.191	0.588	0.3

A.5 IFS Code for a Square in $\mathbb{R}^2$ [1]

Table A.5: IFS Code for a Square in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.5	0	0	0.5	1	1	0.1
2	0.5	0	0	0.5	50	1	0.2
3	0.5	0	0	0.5	1	50	0.3
4	0.5	0	0	0.5	50	50	0.4

A.6 IFS Code for a Koch Curve in $\mathbb{R}^2$ [5]

Table A.6: IFS Code for a Koch Curve in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.333	0	0	0.333	0	0	0.25
2	0.167	-0.289	0.289	0.167	0.333	0	0.25
3	0.167	0.289	-0.289	0.167	0.5	0.289	0.25
4	0.333	0	0	0.333	0.667	0	0.25

A.7 IFS Code for a Fractal Tree in $\mathbb{R}^2$ [1]

Table A.7: IFS Code for a Fractal Tree in $\mathbb{R}^2$

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	0.4
3	0.42	0.42	-0.42	0.42	0	0.2	0.4
4	0.1	0	0	0.1	0	0.2	0.15

A.8 IFS Code for a Maple Leaf in $\mathbb{R}^2$ [4]

Table A.8: IFS Code for a Maple Leaf in $\mathbb{R}^2$

w	a	b	c	d	e	f	p
1	0.14	0.01	0	0.51	-0.08	-1.31	0.1
2	0.43	0.52	-0.45	0.5	1.49	-0.75	0.35
3	0.45	-0.49	0.47	0.47	-1.62	-0.74	0.35
4	0.49	0	0	0.51	0.02	1.62	0.2

A.9 IFS Code for a Sierpinski Triangle in $\mathbb{R}^3$

Table A.9: IFS Code for a Sierpinski Triangle in $\mathbb{R}^3$

w	a	b	c	d	e	f	g	h	i	j	k	l	p
1	0.5	0	0	0	0.5	0	0	0	0.5	1	1	1	0.25
2	0.5	0	0	0	0.5	0	0	0	0.5	13.3975	50	1	0.25
3	0.5	0	0	0	0.5	0	0	0	0.5	50	13.3975	1	0.25
4	0.5	0	0	0	0.5	0	0	0	0.5	21.1325	21.1325	50	0.25

A.10 IFS Code for a Fern in $\mathbb{R}^3$ [1]

Table A.10: IFS Code for a Fern in $\mathbb{R}^3$

w	a	b	c	d	e	f	g	h	i	j	k	l	p
1	0	0	0	0	0.18	0	0	0	0	0	0	0	0.01
2	0.85	0	0	0	0.85	0.1	0	-0.1	0.85	0	1.6	0	0.85
3	0.2	-0.2	0	0.2	0.2	0	0	0	0.3	0	0.8	0	0.07
4	-0.2	0.2	0	0.2	0.2	0	0	0	0.3	0	0.8	0	0.07

A.11 IFS Code for a Merger Sponge in $\mathbb{R}^3$

Table A.11: IFS Code for a Merger sponge in $\mathbb{R}^3$

w	a	b	c	d	e	f	g	h	i	j	k	l	p
1	0.333	0	0	0	0.333	0	0	0	0.333	0	0	0	0.05
2	0.333	0	0	0	0.333	0	0	0	0.333	0	0.333	0	0.05
3	0.333	0	0	0	0.333	0	0	0	0.333	0	0.667	0	0.05
4	0.333	0	0	0	0.333	0	0	0	0.333	0.333	0	0	0.05
5	0.333	0	0	0	0.333	0	0	0	0.333	0.333	0.667	0	0.05
6	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0	0	0.05
7	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0.333	0	0.05
8	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0.667	0	0.05
9	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0.667	0.667	0.05
10	0.333	0	0	0	0.333	0	0	0	0.333	0	0	0.333	0.05
11	0.333	0	0	0	0.333	0	0	0	0.333	0	0	0.667	0.05
12	0.333	0	0	0	0.333	0	0	0	0.333	0	0.667	0.667	0.05
13	0.333	0	0	0	0.333	0	0	0	0.333	0	0.667	0.333	0.05
14	0.333	0	0	0	0.333	0	0	0	0.333	0	0.333	0.667	0.05
15	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0	0.667	0.05
16	0.333	0	0	0	0.333	0	0	0	0.333	0.333	0	0.667	0.05
17	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0	0.333	0.05
18	0.333	0	0	0	0.333	0	0	0	0.333	0.333	0.667	0.667	0.05
19	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0.667	0.333	0.05
20	0.333	0	0	0	0.333	0	0	0	0.333	0.667	0.333	0.667	0.05

A.12 IFS for a Sierpinski Triangle in $\mathbb{C}$ [1]

$$w_1(z) = 0.5z + 24 + 24i$$
$$w_2(z) = 0.5z + 24i$$
$$w_3(z) = 0.5z$$

$$p = \{0.25, 0.25, 0.5\}$$

A.13 IFS for a Square in $\mathbb{C}$ [1]

$$w_1(z) = 0.5z$$
$$w_2(z) = 0.5z + 0.5$$
$$w_3(z) = 0.5z + (0.5)i$$
$$w_4(z) = 0.5z + 0.5 + (0.5)i$$

$$p = \{0.1, 0.2, 0.3, 0.4\}$$

A.14 IFS Code for a Fern in $\mathbb{R}^2$ in Scale and Angle Format [1]

Table A.12: FS Code for a Fern in $\mathbb{R}^2$ in Scale and Angle Format

w	h	k	θ	ϕ	e	f	p
1	0	0	0	0	0	0.16	0.01
2	0	1.6	-2.5	-2.5	0.85	0.85	0.85
3	0	1.6	49	49	0.3	0.34	0.07
4	0	0.44	120	-50	0.3	0.37	0.07

Appendix B

R Functions

Appendix B contains the R-code written for the R-Package for generating fractals via iterated function systems developed during this master project.

B.1 Determ_R2

```
Determ_R2<-function(ifs,n){

  # initialize list a to hold vectors of points
  a<-list()

  # generate the first set of points, a(0)
  # start with a set consisting of a single point
  a[[1]]<-rbind(runif(1,0,1),runif(1,0,1))

  # plot settings
  pdf("FractalDeterm.pdf",paper="special",width=6,height=9)
  par(mfrow=c(ceiling((n+1)/3),3),cex=0.2,pch=16,cex.main=2,
        cex.lab=1.5,cex.axis=1.2)

  # extract x and y coordinates from a(0)
  x<-a[[1]][1]
  y<-a[[1]][2]

  # pointplot with a(0)
  title<-expression(A[0])
  plot(x,y,col="blue",main=title)

  # affine transformation for mapping new set of points, a(n), from a(n-1)
  for(j in 1:n){

    m<-1
    # initialize new set of points a(n)
    a_new<-list()
    g<-length(a)
    for(k in 1:g){
      for(r in 1:length(ifs)){
        a_new[[m]]<-ifs[[r]](a[[k]])
        m<-m+1
```

```
      }
    }

  # the new set of points becomes a(n-1)
  a<-a_new

  # extract x and y coordinates from set a(n)
  x<-rep(0,length(a))
  y<-rep(0,length(a))
  for(i in 1:length(a)){x[i]<-a[[i]][1]}
  for(i in 1:length(a)){y[i]<-a[[i]][2]}

  # pointplot of points a(n)
  title<-bquote(expression(A[.(j)]))
  plot(x,y,col="blue",main=eval(title))

  }

  dev.off()

}
```

B.2 RandIt_R2

```
RandIt_R2<-function(ifs,p,n){

  library(ggplot2)
  library(gridExtra)

  # check if the number of functions in the IFS equals the number
  # of probabilities
  if(length(ifs)!=length(p)){
    print("Error: Number of functions in the IFS does not equal the number
        of probabilities.")
  }

  # check that the sum of the probabilities is one
  if(sum(p)!=1){
    print("Error: Sum of the probabilities is not equal to 1.")
  }

  # sample of mappings determined by the probabilities in vector p
  w<-sample(ifs,n,replace=TRUE,prob=p)

  # initialize list v to store point vectors
  v<-list()

  # initialize the first point, v(0), randomly
  v[[1]]<-rbind(runif(1,0,1),runif(1,0,1))

  # affine mapping
  for(i in 1:n){
    v[[i+1]]<-w[[i]](v[[i]])
  }
```

```
  # points to plot eliminating first 5000 points
  x<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){x[i-5000]<-v[[i]][1]}
  y<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){y[i-5000]<-v[[i]][2]}

  fractal<-data.frame(x,y)

  # pointplot
  d<-round(p,3)
  pstring<-paste(d[1:length(d)-1],",",collapse="")
  title1<-paste("Pointplot\nProbabilities={",pstring,d[length(d)],"}")
  p1<-ggplot(fractal,aes(x,y))+opts(title=title1,plot.title=
      theme_text(size=12))+geom_point(size=0.3)

  # 2d histogram
  title2<-paste("Histogram\nProbabilities={",pstring,d[length(d)],"}")
  colors<-rainbow(100,start=.7,end=.17)
  p2<-ggplot(fractal,aes(x,y))+opts(title=
      title2,plot.title=theme_text(size=12))+stat_bin2d(bins=
      600)+scale_fill_gradientn(colour=colors,legend=TRUE)

  pdf("FractalRandItR2.pdf",paper="special",width=12,height=6.3)
  grid.arrange(p1,p2,ncol=2)
  dev.off()

}
```

B.3 RandIt_R3

```
RandIt_R3<-function(ifs,p,n){

  library(rgl)

  # check if the number of functions in the IFS equals the number
  # of probabilities
  if(length(ifs)!=length(p)){
    print("Error: Number of functions in the IFS does not equal the number
        of probabilities.")
  }

  # check that the sum of the probabilities is one
  if(sum(p)!=1){
    print("Error: Sum of the probabilities is not equal to 1.")
  }

  # sample of mappings determined by the probabilities in vector p
  w<-sample(ifs,n,replace=TRUE,prob=p)

  # intilalize list v to store point vectors
  v<-list()

  # intilalize the first point, v(0), randomly
  v[[1]]<-rbind(runif(1,0,1),runif(1,0,1),runif(1,0,1))
```

```
  # affine mapping
  for(i in 1:n){
    v[[i+1]]<-w[[i]](v[[i]])
  }

  # points to plot eliminating first 5000 points
  x<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){x[i-5000]<-v[[i]][1]}
  y<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){y[i-5000]<-v[[i]][2]}
  z<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){z[i-5000]<-v[[i]][3]}

  # 3d pointplot
  d<-round(p,3)
  pstring<-paste(d[1:length(d)-1],",",collapse="")
  title<-paste("3D Pointplot with Probabilities={",pstring,d[length(d)],"}")
  plot3d(x,y,z,size=0.3,main=title)

}
```

B.4 RandIt_C

```
RandIt_C<-function(ifs,p,n){

  # check if the number of functions in the IFS equals the number
  # of probabilities
  if(length(ifs)!=length(p)){
    print("Error: Number of functions in the IFS does not equal the number
        of probabilities.")
  }

  # check that the sum of the probabilities is one
  if(sum(p)!=1){
    print("Error: Sum of the probabilities is not equal to 1.")
  }

  library(ggplot2)
  library(gridExtra)

  # sample of mappings determined by the probabilities in vector p
  w<-sample(ifs,n,replace=TRUE,prob=p)

  # initialize the first point, z(0), randomly
  z<-rep(0,n+1)
  x<-runif(1,0,1)
  y<-runif(1,0,1)

  z[1]<-complex(real=x,imaginary=y)

  # analytic mapping
  for(m in 1:n){
    z[m+1]<-w[[m]](z[m])
  }
```

```
  # points to plot eliminating first 5000 points
  x<-Re(z)
  x<-x[5001:(n+1)]
  y<-Im(z)
  y<-y[5001:(n+1)]

  fractal<-data.frame(x,y)

  # pointplot
  d<-round(p,3)
  pstring<-paste(d[1:length(d)-1],",",collapse="")
  title1<-paste("Pointplot\nProbabilities = {",pstring,d[length(d)],"}")
  p1<-ggplot(fractal,aes(x,y))+opts(title=title1,plot.title=
        theme_text(size=12))+geom_point(size=0.3)+
        scale_x_continuous('Real')+scale_y_continuous('Imaginary')

  # 2d histogram
  title2<-paste("Histogram\nProbabilities = {",pstring,d[length(d)],"}")
  colors<-rainbow(100,start=.7,end=.17)
  p2<-ggplot(fractal,aes(x,y))+opts(title=title2,
        plot.title=theme_text(size=12))+stat_bin2d(bins=600)+
        scale_fill_gradientn(colour=colors,legend=TRUE)+
        scale_x_continuous('Real')+scale_y_continuous('Imaginary')

  pdf("FractalRandItC.pdf",paper="special",width=12,height=6.3)
  grid.arrange(p1,p2,ncol=2)
  dev.off()

}
```

B.5 IFS_Prob

```
IFS_Prob<-function(n){

  # initialize vector of probabilities p
  p<-rep(0,n)

  # randomly generate p(1) through p(n-1)
  for(i in 1:n-1){
    p[i]<-runif(1,0,1-sum(p))
  }

  # calculate p(n)
  p[n]<-1-sum(p)
  p

}
```

B.6 IFSP_Gen_R2

```
IFSP_Gen_R2<-function(ifs,n,t){

  library(ggplot2)
  library(gridExtra)
```

```
pdf("FractalGen.pdf",paper="special",width=12,height=6.3)

for (m in 1:t) {

  # generate a vector of probabilities p using the IFS_Prob
  # function
  p<-IFS_Prob(length(ifs))

  # check if the number of functions in the IFS equals the number of
  # probabilities
  if(length(ifs)!=length(p)){
    print("Error: Number of functions in the IFS does not equal the number
      of probabilities.")
  }

  # check that the sum of the probabilities is one
  if(sum(p)!=1){
    print("Error: Sum of the probabilities is not equal to 1.")
  }

  # sample of mappings determined by the probabilities in vector p
  w<-sample(ifs,n,replace=TRUE,prob=p)

  # initialize list v to store point vectors
  v<-list()

  # initialize the first point, v(0), randomly
  v[[1]]<-rbind(runif(1,0,1),runif(1,0,1))

  # affine mapping
  for(i in 1:n){
    v[[i+1]]<-w[[i]](v[[i]])
  }

  # points to plot eliminating first 5000 points
  x<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){x[i-5000]<-v[[i]][1]}
  y<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){y[i-5000]<-v[[i]][2]}

  fractal<-data.frame(x,y)

  # pointplot
  d<-round(p,3)
  pstring<-paste(d[1:length(d)-1],",",collapse="")
  title1=paste("Pointplot\nProbabilities={",pstring,d[length(d)],"}")
  p1<-ggplot(fractal,aes(x,y))+opts(title=title1,plot.title=
      theme_text(size=12))+geom_point(size=0.3)

  # 2d histogram
  title2=paste("Histogram\nProbabilities={",pstring,d[length(d)],"}")
  colors<-rainbow(100,start=.7,end=.17)
  p2<-ggplot(fractal,aes(x,y))+opts(title=
      title2,plot.title=theme_text(size=12))+
```

```
        stat_bin2d(bins=100)+scale_fill_gradientn(colour=colors,
        legend=TRUE)

    grid.arrange(p1,p2,ncol=2)

  }

  dev.off()

}
```

B.7 FractalMeas_R2

```
FractalMeas_R2<-function(ifs,p,n,xlim,ylim){

  # check if the number of functions in the IFS equals the number
  # of probabilities
  if(length(ifs)!=length(p)){
    print("Error: Number of functions in the IFS does not equal the number
        of probabilities.")
  }

  # check that the sum of the probabilities is one
  if(sum(p)!=1){
    print("Error: Sum of the probabilities is not equal to 1.")
  }

  # sample of mappings determined by the probabilities in vector p
  w<-sample(ifs,n,replace=TRUE,prob=p)

  # intilalize list v to store point vectors
  v<-list()

  # intilalize the first point, v(0), randomly
  v[[1]]<-rbind(runif(1,0,1),runif(1,0,1))

  # affine mapping
  for(i in 1:n){
    v[[i+1]]<-w[[i]](v[[i]])
  }

  # points to plot eliminating first 5000 points
  x<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){x[i-5000]<-v[[i]][1]}
  y<-rep(0,(n+1-5000))
  for(i in 5001:(n+1)){y[i-5000]<-v[[i]][2]}

  mass<-data.frame(x,y)

  # check that the input vector v1 and v2 delimit a rectangle
  if(xlim[1]==xlim[2]){
    print("Error: xlim and ylim do not delimit a rectangular subset.")
  }
  if(ylim[1]==ylim[2]){
    print("Error: xlim and ylim do not delimit a rectangular subset.")
```

```
  }

  # eliminate points not in the specified subset
  mass<-subset(mass,mass$x>xlim[1])
  mass<-subset(mass,mass$x<xlim[2])
  mass<-subset(mass,mass$y>ylim[1])
  mass<-subset(mass,mass$y<ylim[2])

  # calculate measure mu
  mu<-nrow(mass)/(length(x)+1)
  mu<-round(mu,3)

  # pointplot
  pdf("FractalMeas.pdf",paper="special",width=6,height=6)

  d<-round(p,3)
  pstring<-paste(d[1:length(d)-1],",",collapse="")
  title<-paste("Pointplot\nProbabilities={",pstring,d[length(d)],"}")
  sub<-paste("Measure=",mu)
  plot(x,y,main=title,sub=sub,cex=0.1,pch=16)
  x2<-c(xlim[1],xlim[2],xlim[2],xlim[1],xlim[1])
  y2<-c(ylim[1],ylim[1],ylim[2],ylim[2],ylim[1])
  lines(x2,y2,col="red")

  dev.off()

}
```

B.8 TransMat_R2

```
TransMat_R2<-function(M,n){

  library(ggplot2)

  # check if M is quadratic
  if(nrow(M)!=ncol(M)) print("Error: Matrix of selection
      probabilities is not quadratic.")

  # check if M is contains a negative value
  for(j in 1:nrow(M)){
    for(i in 1:ncol(M)){
      if(M[j,i]<0) print("Error: Matrix of selection probabilities
          contains negative value.")
    }
  }

  # check if M has at least 2 rows and 2 columns
  if(nrow(M)<2) print("Error: Matrix of selection probabilities
          has less than 2 rows.")
  if(ncol(M)<2) print("Error: Matrix of selection probabilities
          has less than 2 columns.")

  # check that no column or row of M has all 0 entries
  if(all(colSums(M)>0)==FALSE) print("Error: Matrix of selection
          probabilities contains column with all 0 entries.")
```

```
if(all(rowSums(M)>0)==FALSE) print("Error: Matrix of selection
        probabilities contains row with all 0 entries.")

# define column sums, a, and row sums, b, of M
a<-c(0,cumsum(colSums(M)))
b<-c(0,cumsum(rowSums(M)))

# check that the sum of the selection probabilities is 1
if(a[length(a)]!=1) print("Error: Sum of selection probabilities
    is not 1.")

# create vector of selection probabilities
t<-rep(0,nrow(M)*ncol(M))
m<-1
for(j in 1:nrow(M)){
  for(i in 1:ncol(M)){
    t[m]<-M[j,i]
    m<-m+1
  }
}

# create vector of indices
index<-rep(0,nrow(M)*ncol(M))
m<-1
for(j in 1:nrow(M)){
  for(i in 1:ncol(M)){
    index[m]<-j*10+i
    m<-m+1
  }
}

index_sample<-sample(index,n,replace=TRUE,prob=t)

# create vectors x and y
x<-rep(0,n+1)
y<-rep(0,n+1)

# intilalize x0 and y0 randomly
x[1]<-runif(1,-2,2)
y[1]<-runif(1,-2,2)

# affine mapping
for(k in 1:n){
  j<-index_sample[k]%/%10
  i<-index_sample[k]%%10
  x[k+1]<-a[j]+x[k]*(a[j+1]-a[j])
  y[k+1]<-b[i]+y[k]*(b[i+1]-b[i])
}

# points to plot
x<-x[5000:(n+1)]
y<-y[5000:(n+1)]

# point plot
pdf("Points.pdf")
```

```
  plot(x,y,pch=20,col="mediumorchid4",cex=0.3)
  dev.off()

  # 2d histogram with 20 bins
  ChaosGame<-data.frame(x,y)
  d<-ggplot(ChaosGame,aes(x,y))
  d+stat_bin2d(bins=20)
  ggsave("Bins20.pdf")

  # 2d histogram with 1000 bins
  d+stat_bin2d(bins=100)
  ggsave("Bins100.pdf")

}
```

Bibliography

[1] Barnsley, M. F. (1993). *Fractals everywhere.* (2 ed.). San Francisco: Morgan Kaufmann.

[2] Castillo, O., & Melin, P. (2003). *Studies in fuzziness and soft computing: Soft computing and fractal theory for intelligent manufacturing.* New York: Physica-Verlag Heidelberg.

[3] Fredricks, G. A., Nelsen, R. B., & Rodríguez-Lallena, J. A. (2005). Copulas with fractal supports. *Insurance: Mathematics and Economics, 37*(1), 42-48.

[4] *Hidden dimension galleries: Mathematics of iterated function system (IFS) fractals.* (n.d.). Retrieved from http://www.hiddendimension.com/FractalMath/IFS_Fractals_Main.html

[5] Riddle, L. (2010, January 25). *Classic iterated function systems.* Retrieved from http://ecademy.agnesscott.edu/ lriddle/ifs/ifs.htm

[6] Texas Instruments Europe. (1997, October). *An introduction to fractal image compression.* Retrieved from http://www.ti.com/lit/an/bpra065/bpra065.pdf

[7] Venables, W. N., & Smith, D. M. (2010). *An introduction to R.* R Development Core Team.

[8] Wickham, H. (2012). *ggplot.* Retrieved from http://had.co.nz/ggplot/