Analytic and Numerical Calculations of Fractal Dimensions

Arman Shamsgovara
arman.shamsgovara@minervaskolan.se

under the direction of
Gaultier Lambert
Department of Mathematics
Royal Institute of Technology, KTH

Research Academy for Young Scientists
July 11, 2012
Abstract

This study has focused on understanding the different features of fractals, their geometries and their applications. Hausdorff and Box-counting dimensions have been numerically estimated for a variety of fractals, mostly self-similar ones. Many fractals were plotted in MATLAB, but some fractals could not, due to technical difficulties paired with the project’s time limit, be plotted using IFS. They were recursively drawn with Turtle Graphics in PYTHON instead. All of the numerically calculated dimensions were compared to the analytical values and the topological dimensions, to experimentally verify that the box-counting, Hausdorff and the topological dimension are all different. Also, some analytic theory behind fractal geometry was studied, and some within the mathematical genre well-known theorems and lemmas are presented in the theory section.
Contents

1 Introduction - What is a fractal? .................................................. 3
   1.1 Lightweight theory - What is good to know about fractals ............... 5

2 How to plot a fractal ................................................................. 8

3 Random Fractals - A Brief Introduction ...................................... 12
   3.1 Percolation ............................................................................. 12
   3.2 Brownian Motion .................................................................. 13

4 Theory ....................................................................................... 14
   4.1 Definitions .............................................................................. 14
   4.2 Final proof - the dimension of selfsimilar fractals ......................... 18

5 Results of computer simulations .................................................. 21
   5.1 The Cantor set ........................................................................ 22
   5.2 The Sierpinski triangle ............................................................. 22
   5.3 The Sierpinski carpet ............................................................... 23
   5.4 The Vicsek fractal ..................................................................... 23
   5.5 The Hénon attractor ................................................................. 24
   5.6 A One-dimensional Brownian Motion ........................................ 25
   5.7 A Two-dimensional Percolation Cluster ...................................... 26

6 Discussion ................................................................................... 27

7 Acknowledgements ...................................................................... 29

A Python code .................................................................................. 31
   A.1 Koch curve ............................................................................. 31
1 Introduction - What is a fractal?

Picture the coast of Sweden before your eyes. Imagine that the swedish government has to measure it’s length, to decide how many guard posts that should be placed to guard all of the coastline. You take out your favorite ruler, and you start to measure the shore at the waterline. It is a simple method to measure its length, but it is not accurate. What would happen if you used a smaller ruler? Would the length be the same? In that case, you measure the shore in more details, you take the coastline’s unevenness into consideration.

Now, picture yourself doing the same procedure over and over again, with smaller and smaller rulers each time you measure. The coast gets longer and longer, right? Does that not strike you as an odd occurrence? Maybe the algorithm is wrong. However, it seems that the coastline fills out a little more space than it should have if it was a straight line, and therefore we cannot measure it. In fact, the method of measurement is not faulty, it is the concept of dimension that is not general enough: the coastline of Sweden makes turns and cracks and whatever could be called ”filling out more space than an ordinary line would have”, and must therefore be of dimension greater than one.

So, what exactly is the dimension of an object? The oldest and most natural definition of dimension is similar to the one from the theory of vector spaces, that is, an object visible from \(k\) orthogonal directions is of dimension \(k\). Another view is the topological one. For instance, a sphere has dimension 2, a line has dimension 1, and in general mathematicians say that a set has topological dimension \(n\) if there exists a mapping from the \(n\)-dimensional room to the set. But Sweden’s coast is too irregular to be properly described by the topological dimension.

A more general and complicated definition of dimension of sets was provided later in the early 19\(^{th}\) century by Felix Hausdorff. It takes into account how irregular the set is that we measure, and it is supposed to give us a feeling for which scale that is the most fitting to use
when we measure the geometric object.

Now, imagine the same concept on a rugged rock, or the bark-covered stem of a tree, or on a lap around the Earth. Do you get the same intuitive result? It seems as if we do not have definitions that are general enough to explain everything geometrically yet.

Other problems of relevance to fractals tormented mathematicians from many areas of mathematics, all over Europe. In analysis, a Swedish mathematician called Helge von Koch constructed a geometric figure that had a finite area but a boundary of infinite length. Referred to as the von Koch snowflake (fig. 1), it had, besides the features already mentioned, no tangent line to any point of the boundary. It consisted of broken lines, everywhere.

![The von Koch Snowflake](image)

**Figure 1: The von Koch Snowflake**

This implied that it was impossible to deal with the figure in any section of mathematics. Neither in Euclidean geometry, calculus nor even measure theory this shape could be dealt with. The roughness of the shape was the main hindrance. It was so detailed in roughness that there were no smooth lines, and that stopped all attempts to do anything with the snowflake. It is really hard to analyze, but is, due to its selfsimilarity, constructed by following a simple
algorithm, explained in the following section.

To solve the problems that appear with this kind of geometrical figure (there are plenty more of the kind), a mathematician called Benoît Mandelbrot used new concepts of dimension, and also introduced a whole new section of mathematics: fractal geometry. A fractal is basically one of these broken and problematic shapes, such as the snowflake. The name comes from the Latin word *fractus*, which means broken. The theory of fractals has become one of the most researched ones in modern times, and there are applications all over society. In technology, in physics, in computer graphics, in art, to understand the structure of nature, and so on, infinitely many times. Even the perfect mathematical fractals model some kind of phenomenon, so the work done in this specific area of mathematics has a lot of interesting applications.

This study will focus on plotting and numerically estimating the fractal dimension for different sets with a computer. Then, for some of the fractals, the calculated dimensions will be compared to the results one gets if the calculations are done analytically.

### 1.1 Lightweight theory - What is good to know about fractals

A fractal is, as said before, a geometrical object consisting of mostly broken lines. They are created by iterating a set of operations infinitely many times, and they are often self-similar. That is, if you zoom in on e.g. the snowflake’s boundary, what you see is the same thing as you saw before you zoomed in. Each side of the original triangle looks like itself however much you zoom in on them. This is a key of many fractals. Of course there are fractals that aren’t self-similar too, but they can be handled in many cases. For a selfsimilar fractal, the so called Hausdorff dimension, named after the mathematician bearing the same name, is defined as:

\[
s^d = c
\]  

(1)
, where \( d \) is the Hausdorff dimension, \( c \) the amount of new copies you get after one iteration, and \( s \) the scaling factor.

Taking the snowflake as an example, let’s say that it is in a certain state of iteration. Iterate once, and count the amount of smaller copies of the past iteration that can be found in the current one. There are four smaller copies in the new iteration, right? That number would be \( c \) in the formula. Now, look for the scaling factor that each copy has been scaled with. Each small copy in the new iteration is one third the size of the past iteration, so they have been scaled down with the factor 3. Thus, \( s = 3 \). Inserting the values into equation 1, and using the logarithmic laws, we find that:

\[
\frac{d}{\log s} = \frac{\log 4}{\log 3} = 1.2618...
\]  

(2)

The dimension of the snowflake’s boundary is not of integral size! Fractals are special in the sense that they can have dimensions that are not integers. That was one of the modifications that had to be introduced in order to develop a working theory of fractals. Please note that this particular formula for calculating a given fractal’s dimension only applies to the selfsimilar fractals.

As for the nonselfsimilar fractals, there are other ways to calculate the dimension, and one way to do this is by "covering" all of the figure with smaller objects, usually circles or squares, or hypercubes or balls in whatever topological dimension the fractal set in question lives in. What is needed to calculate the dimension is the minimum number of objects needed to cover the fractal [1]. This is referred to as the box-counting dimension of a fractal.

By estimating the dimension of fractal structures that arise in nature and in other settings, we can get a better approximation of the length of Sweden’s coastline, the periphery of a tree, and the distance you have to hike to reach the peak of a mountain. It can also be used to research what happens to a laser if it bounces off a rugged surface, or to enhance
radar technology. Fractal structures arise everywhere in nature, for example the surface of
the human brain has a fractal dimension of 2.79 [2], so if we learn more about them and their
geometries, there is seemingly no limit of situations where we can apply them. Of course the
natural fractals are not as perfectly fractal as the mathematical ones (they cannot be scaled
down infinitely much because of physics), but their dimensions and other properties can be
determined with fractal theory.

For the interested reader there are brief introductions on percolation and Brownian mo-
tion in the later parts of the paper. These topics are important fractal-like structures, and
the theory of fractals is applied to study these phenomena.
2 How to plot a fractal

Let us look at how to simulate a fractal. To draw a fractal, one can use different types of algorithms. One is to draw a specific figure by recursive algorithms. It is generally an easy way to draw selfsimilar fractals, but useless when trying to draw e.g. the Mandelbrot set, or other fractals that are not completely selfsimilar. Since most of the fractals studied here are selfsimilar, this type of algorithm has been implemented to draw many of the fractals. The main idea of this method is to take a so called generator for a given set, a figure that represents the operations needed to iterate a fractal set one time. The generator shall be applied to each copy of the original set that exist in the current iteration. It is best explained by using a figure, so here is the generator for the Koch curve:

Figure 2: The generator for the Koch curve

I we are at an arbitrary iteration \( k \), this generator should replace each straight line in an arbitrary iteration of the fractal to iterate once, so that we arrive at iteration \( (k + 1) \). Regard the following iterations of the Koch curve, and notice how each line in the previous iteration has been replaced by the generator.

Figure 3: The first three iterations.

Another algorithm is called the chaos game algorithm, mentioned in [3]. If we want to
generate a fractal in $\mathbb{R}^n$, we pick a random point $P \in \mathbb{R}^n$. Create an IFS, an Iterated Function System, consisting of $k$ different mappings $g_1, g_2, \ldots, g_k$ of $\mathbb{R}^n$ to $\mathbb{R}^n$. The mappings should map a point in $[0, 1]$ to one of the generator copies when speaking of selfsimilar fractals. Choose some functions $f_1, f_2, f_3$, and so on, at random from the IFS. Repeat this until there are enough functions to map one point into virtually all points in the fractal set (that amount would be of $O(10000)$ functions in general), with $f_1, f_2, f_3, \ldots$ not necessarily distinct mappings. Create the function $f_1 \circ f_2 \circ f_3 \circ \ldots \circ f_n$, and put the coordinates of $P$ in the composite function. Mark each one of the points that you visit in $\mathbb{R}^n$ that has the coordinates of $P$ for each function that executes. It may sound difficult, but by compressing these functions with matrix notation and using program loops, programming languages like MATLAB can be used with ease to plot the fractals.

As an example, there is a fractal called the Sierpinski triangle, see picture below. The generator is an equilateral triangle, divided into four congruent triangles out of which the middle one is deleted. The fractal can be constructed by randomly choosing a point with coordinates in $[0, 1]$, such that the point in question lies within an equilateral triangle of side 1. Three functions are needed to map the point into all the points of the generator.

![Sierpinski triangle](image1.png)  
![Generator](image2.png)

Figure 4: The Sierpinski triangle with generator.

For a selfsimilar fractal, you look for the relationships between the triangles in the generator. The bottom left one has, just as the other two, half the side-length of the fractal in question, so an appropriate function would be one that halves the coordinate values of the
point $P$. One generator triangle has now been constructed. The other two are mappings of the bottom left triangle, so the other two functions should first halve the coordinate values of $P$, and then do two translations of the obtained coordinates. In the case of the Sierpinski triangle, the translations would be one that adds 0.5 to the horizontal coordinates (for the bottom right triangle in the generator), and one that adds 0.5 to the vertical and 0.25 to the horizontal coordinates (for the upper triangle). After applying these functions for some 50,000 times, the fractal, who lives in $[0, 1] \times [0, 1]$, should be considered to be constructed.

For the curious readers, there is also another method of plotting fractals that uses a special type of IFS, in which case the fractals are called flame fractals [3], but they will not be covered in this study. Many fractals, preferably ones that include circles or use rotational mappings a lot, can also be plotted in $\mathbb{C}$ by using moebius transformations and other means of transformations.

Finally, many fractals that are not completely selfsimilar can be plotted in $\mathbb{C}$ by using certain kinds of algorithms. The famous fractal called the Mandelbrot set consists of all complex numbers $c$ in $\mathbb{C}$ such that the value of the recurrence relation

$$z_n = z_{n-1}^2 + c; z_0 = 0$$

(3)

does not diverge to infinity. The algorithm is quite different from other fractal algorithms, and so is the fractal; it is widely considered to be one of the most complicated object in mathematics today [4].
Figure 5: The Mandelbrot set.
3 Random Fractals - A Brief Introduction

In many situations, completely random patterns have some kind of fractal structures, and these can be studied to find out more about natural phenomena such as the motion of particles in the air, or the waterflow when a plant gets watered. Below we present a couple of the most studied areas of random fractals.

3.1 Percolation

One setting where fractal geometry experiences a lot of usage is the theory of percolation. Imagine pouring a liquid, supposedly water, onto the soil in a flower pot, to water a flower. Is there any chance that some leftover water seeps through the soil, out from the hole in the bottom of the pot, and onto the ground? This is a prime example of a situation in which knowledge about percolation can help us. What you do when you study percolation is that you simulate the following process:

1. Draw a basic shape, preferably a square, for simplification.
2. Divide the shape into a number of subsets (9 subsets are often used when studying percolation in squares).
3. Delete at random any number of the subsets.
4. Iterate steps 2-3 on the remaining subsets.
5. Iterate step 2-4 on all new subsets.
6. Iterate step 5 as many times as you would like.

The resulting image would be tiny fractal fragments of the original square. There are other algorithms for plotting percolation too, such as deleting lines in a grid at random and coloring each disjoint set of lines in certain colors, or randomly coloring hexagons in a honeycomb pattern and following a certain rule for walking through the hexagonal grid.

The structure of these fragments are explained with the theory of random fractals, and the
main question is whether a liquid of any sort would be able to pass through the figure without getting blocked. By studying this phenomenon, we can for example explain probabilistic events that occur when liquid passes through soil, sand etc. [5]

3.2 Brownian Motion

In 1905, ”anno mirabilis”, Albert Einstein, at the age of 26, published four papers that would play an important role in their respective fields. Aside from the special relativity theory, \( E = mc^2 \) and the photoelectric effect, he also described Brownian motion. The phenomenon in question describes the erratic motion of a small particle of pollen submitted to temperature (meaning chocs by atoms composing the air). His predictions were consistent with experimental observations and that marked the beginning of Kinetic Theory for the description of gases. Although mathematicians do not experiment with expensive equipment or other apparatus, they can simulate the motions with computer programs. The algorithm for generating Brownian motions is simple; you start with the particle at the origin and each step you move infinitesimally in one of the 6 direction at random. You stop when you think the trace of the particle is big enough, and voilà.

This type of motion is under study because it would be good to know as much about it as possible when speaking of e.g. the physics of gases, stock price development, and so on.
4 Theory

4.1 Definitions

We will present some of the theory of Hausdorff dimension and Hausdorff dimension for subset of the space [0,1]. The definition generalised easily to higher dimension. In the sequel $S$ is an arbitrary subset of [0,1], let us defined its dimension. To understand the following definitions, we now define the diameter of an interval, $\text{diam}(S)$:

Definition 4.1. The diameter of a set is:

$$\text{diam}(S) = \sup\{|x - y|, x, y \in S\}$$

The diameter of a set is, in simpler terms, the largest distance between two points in the set. It is crucial to memorize this definition and the next one of the content of a covering to even understand the content of the report.

Definition 4.2. We say that the collection $\{U_i\}_{i=1}^N$ for some $N \in \mathbb{N}$ is a cover of $S$ if $S \subset \bigcup U_i$. For any $\alpha \geq 0$ define the (Hausdorff) $\alpha$-content of such a covering to be

$$\sum_{i=1}^N (\text{diam}(U_i))^\alpha$$

We will only use covering by intervals and the diameter of an interval is equal to its length: $\text{diam} [a, b] = b - a$

If $\{U_i\}_{i=1}^N$ is a covering of $S$, then the $\alpha$-content of this covering is a weighted (by a factor $\alpha$) approximation of the size $S$. Next, to define the Hausdorff measure of $S$, we take the best approximation of the size of $S$ over all covering by intervals of size less than $\delta$.

We will also define measure, just to make things clearer later on.
**Definition 4.3.** For each set $A$, let $\mu(A)$ be 0 if $A$ is empty, $\infty$ if $A$ is infinite, and $|A|$ if $A$ is finite and nonempty.

**Definition 4.4.** For $\alpha \geq 0$ and $\delta > 0$ the Hausdorff measure of $S$ is denoted $H_\delta^\alpha(S)$ and equal to:

$$H_\delta^\alpha(S) = \inf \left\{ \sum_{i=1}^{N} (\text{diam}(U_i))^\alpha; \text{diam}(U_i) < \delta, \right\}$$

where the infimum is taken over all cover of $S$ by intervals ($S \subset \bigcup_{i=1}^{\infty} U_i$).

Simpler expressed, the Hausdorff measure is the smallest sum of the $\alpha$-powers of diameters of intervals so that those diameters are less than some value $\delta > 0$ and the intervals cover the sets of $S$.

As explained in the introduction as the size $\delta$ of our ruler shrinks, the measure of $S$ gets bigger because we capture more and more details of the geometry of $S$. Thus in order to take into account all details of $S$, we define its Hausdorff measure, $H^\alpha(S)$ by taking the limit as $\delta$ goes to 0.

**Definition 4.5.** The Hausdorff measure of $S$ is given by:

$$H^\alpha(S) = \lim_{\delta \to 0} H_\delta^\alpha(S)$$

Those definition are hard to grasp, because they involve infimums and limits, but they are the most natural from a theoretical point of view. They are widely used in the theory of fractals and chaos, because the Hausdorff dimension encodes how irregular is a set and it allows us to treat them as mathematical objects in an easier way. As a function of $\alpha$ the Hausdorff measure $H$ has a very special graph. It jumps from $\infty$ to 0 for a single value of $\alpha$:

We define the Hausdorff dimension $d=\dim(S)$ to be that special value of the weight parameter $\alpha$. Whenever the value $\alpha$ is larger than the dimension, the Hausdorff measure is equal to zero, and whenever $\alpha$ is less than the dimension, the Hausdorff measure is infinite.
Proposition 4.6. It follows directly from the graphical representation of $H$ that if $0 < H^\alpha(S) < \infty$ then $\dim_H(S) = \alpha$

Next up, we will define a special type of function, the Lipschitz functions.

Definition 4.7. A mapping $f$ that is Lipschitz has the property that $\exists k$, such that $|f(x) - f(y)| = |x - y| \cdot k$

This property of certain functions is handy to know about when proving things later on.

Now, for the main section of the definitions, the definitions of fractal dimensions. There are numerous definitions of dimension, and here are three most used ones:

At first, the one type of dimension everybody was using was the topological dimension.

Definition 4.8. 1. The topological dimension of a point (or $R^0$) is equal to $0$.

2. An object $S$ that can be embedded in $R^n$ but not in $R^{n-1}$ has $(\dim_T(S)) = n$.

The Hausdorff dimension has already been mentioned above.

The third and final dimension that is soon to be introduced is the one used for the numerical computations. It is called the box-counting dimension.

Definition 4.9. For a fractal set $S$, cover all the points with congruent squares of size-length $\epsilon$. Denote the number of squares to cover the fractal set completely with $N$. Then the box-counting dimension of the fractal set is: $\dim_B(S) = \frac{\log N}{\log \epsilon}$
Please note that the different types of dimensions have different values for different objects.

**Theorem 4.10.** If $f$ is a Lipschitz function, then $k^\alpha H^\alpha(S) \geq H^\alpha(f(S))$, for any set $S \neq \emptyset$, and some positive number $k$.

**Proof** - To proceed with this proof, we need to show that $k^\alpha H^\alpha(S) \geq H^\alpha_k(f(S))$. Then we should find the limits of both sides of the equation as $\delta$ tends to zero, to prove the original statement.

Now, as $f$ is a Lipschitz function, we know that $|f(x) - f(y)| \leq k|x - y|$, it is the definition of such a function. This also applies to the diameters of the sets covering $S$, because the diameter of a set is the length between two extreme points, then $\text{diam}(f(U_i)) \leq k \cdot \text{diam}(U_i)$. In particular if $\text{diam}(U_i) \leq \delta$ then $\text{diam}(f(U_i)) \leq k\delta$. If $\{U_i\}$ is a cover of $S$ then the collection $\{f(U_i)\}$ is obviously a cover of $f(S)$.

We can establish summing the diameter of all the sets of a cover the following inequality which is valid for all covers of $S$ and all $\delta > 0$

$$
\sum_{i=1}^{N} (k \cdot \text{diam}(U_i))^{\alpha} \geq \sum_{i=1}^{N} (\text{diam}(f(U_i)))^{\alpha}
$$

$$
k^\alpha \sum_{i=1}^{N} \text{diam}(U_i)^{\alpha} \geq \sum_{i=1}^{N} (\text{diam}(f(U_i)))^{\alpha}
$$

Taking the infimum

$$
k^\alpha H^\alpha_\delta(S) \geq H^\alpha_k(f(S))
$$

taking the limit as $\delta$ tends to zero,

$$
k^\alpha H^\alpha(S) \geq H^\alpha(f(S))
$$

□
Corollary 4.11. (i) If \( f(x) = ax \), for some \( a > 0 \), then for any \( \alpha \geq 0 \) we have \( a^\alpha H^\alpha(S) = H^\alpha(f(S)) \). (ii) If \( f(x) = x + b \) for some \( b \in \mathbb{R} \), then for any \( \alpha \geq 0 \) we have \( H^\alpha(S) = H^\alpha(f(S)) \).

It means that translation preserve the Hausdorff measure of a set, while a dilation by some factor \( a > 0 \) rescale the Hausdorff measure by a factor \( a^\alpha \).

4.2 Final proof - the dimension of selfsimilar fractals

We will now prove the formula for the Hausdorff dimension of selfsimilar fractals. We will begin with a definition and a lemma:

Definition 4.12. A set \( F \subset \mathbb{R} \) is selfsimilar if \( \exists \) maps \( g_1, g_2, g_3, \ldots, g_k : \mathbb{R}^n \to \mathbb{R}^n \), \( k \) the amount of smaller copies of the set, such that

\[
F = \bigcup_{i=1}^{k} g_i(F)
\]

Lemma 4.13. Let \( f(x) = ax \) be a contraction. Then \( f \) is \( a \)-Lipschitz for all sets \( S \):

\[
H^d(f(S)) \leq a^d H^d(S)
\]

, just as proved in theorem 4.9. Now, \( f^{-1} = a^{-1}x \), so \( f^{-1} \) is \( a^{-1} \)-Lipschitz for all sets \( S \):

\[
H^d(S) \leq a^{-d} H^d(f(S))
\]

Multiply with \( a^d \), which is positive, on both sides of the equation:

\[
a^d H^d(S) \leq H^d(f(S))
\]
\[ a^d H^d(S) \leq H^d(f(S)) \leq a^d H^d(S) \]

Since we cannot have an inequality,

\[ H^d(f(S)) = a^d H^d(S) \]

Likewise, if we let \( g(x) = x + b \) be a translation. Then for all \( \alpha > 0 \):

\[ H^d(f(S)) = H^d(S) \]

Now, for the main proof:

**Proof** - Let \( F \) be a set in \( \mathbb{R}^n \). If \( g_1(F), g_2(F), \ldots, g_k(F) \) are disjoint sets and if \( g_i(x) = g_1(x) + t_i \), \( t_i \) a translation for some \( t_1, t_2, t_3, \ldots, t_k \in \mathbb{R}^n \), then, from the above lemma,

\[ H^d(g_i(F)) = H^d(g_1(F)) \]

If also \( g_1(x) = \frac{x}{a}, a > 0, \) then

\[ \dim_H(F) = \frac{\log k}{\log a} \]

Since \( g_1(F), g_2(F), \ldots, g_k(F) \) are disjoint,

\[ H^d(F) = \sum_{i=1}^{k} H^d(g_i(F)) = k H^d(g_1(F)) \]

Since \( g_1(x) = \frac{x}{a}, \) we have

\[ H^d(g_1(F)) = \frac{1}{a^d} H^d(F) \]

And from the biLipschitz property,

\[ H^d(F) = \frac{k}{a^d} H^d(F) \]

If \( H^d(F) \) is finite and positive, division by \( H^d(F) \) is defined, and that happens at only
one value of $d$, the jump in fig. 6. At that point,

$$d = \frac{\log k}{\log a}$$

Thus we have the expression for the dimension of all selfsimilar fractals. □
5 Results of computer simulations

The data extracted from the computer simulations are presented below. Note that the coordinates in the fractal images represent length units. The horizontal coordinates in the diagrams represent the natural logarithms \((\log_e)\) of the amount of squares needed to cover the fractal set. The vertical coordinates in the diagrams represent the natural logarithms of the lengths of the sides of the squares.

The studied fractals are, in the presented order, the Cantor set, the Sierpinski triangle, the Sierpinski carpet, the Vicsek fractal, the Hénon mapping, a onedimensional Brownian motion, and a twodimensional percolation cluster. For comparison, we can say that all of these fractals have \(dim_T = 2\), except for the Cantor set, which has a topological dimension of one. All the studied fractals have a lot of use in modeling of nature-like systems, for example, the Hénon mapping is used to model atmospheric dynamics, and the Cantor set models dust at all scales, even in space. We have done linear interpolations of \(\log N vs. \log \epsilon\), and from the slopes of the lines determined the box-counting dimension of the plotted sets.
5.1 The Cantor set

The cantor set is a model for a dust of uncountably many points in $[0,1]$. Our IFS simulation is one dimensional, but in physics literature there are models of the universe where galaxies are described as threedimensional cantor dust.

![The Cantor set](image1)

![The linear approximation](image2)

Figure 7: The simulated Cantor set with linear interpolation, $\dim_B(\text{Cantor set}) = 0.6386$.

5.2 The Sierpinski triangle

![The Sierpinski triangle](image3)

![The linear approximation](image4)

Figure 8: The simulated Sierpinski triangle with linear interpolation, $\dim_B(\text{Sierpinski triangle}) = 1.584$. 
5.3 The Sierpinski carpet

Figure 9: The simulated Sierpinski carpet with linear interpolation, \( \dim_B(\text{Sierpinski carpet}) = 1.877 \).

5.4 The Vicsek fractal

Figure 10: The simulated Vicsek fractal with linear interpolation, \( \dim_B(\text{Vicsek fractal}) = 1.453 \).
5.5 The Hénon attractor

The Hénon attractor is created by using an IFS with only one single function \( (x_{t+1}, y_{t+1}) = f(x_t, y_t) \):

\[
\begin{align*}
x_{t+1} &= y_t + 1 - 1.4x_t^2 \\
y_{t+1} &= 0.3x_t
\end{align*}
\]

This attractor is the time evolution, also called trajectory of a point whose dynamics is given by iterating the function \( f \). The trajectory is fractal, because the map \( f \) is chaotic. The Hausdorff-dimension of the attractor is related to the intensity of the chaos in the dynamics. The Hénon map and its attractor were introduced as a toy model to describe the motion of particles in the upper atmosphere.

![Figure 11: The simulated Hénon attractor with linear interpolation, \( \dim_B(\text{Henonattractor}) = 1.236 \).](image)
5.6 A Onedimensional Brownian Motion

The one-dimensional brownian motion was introduced in mathematics in 1900 as a model for time evolution of auction prices, by a mathematician called Bachelier. From a theoretical point of view, it corresponds to a function which is chosen uniformly at random from the space $C([0, 1] \rightarrow R)$ of continuous real-valued function on $[0, 1]$. As a boundary of the Koch snowflakes, the path of the Brownian motion is very irregular. Precisely, it is nowhere differentiable and not monotonic on any intervals. In addition, all Brownian path are self-similar in a weak sens and have Hausdorff dimension $3/2$.

Below the retrieved picture of a brownian motion over time, that my mentor simulated on his computer, is displayed:

Figure 12: The simulated Brownian motion with linear interpolation, $dim_B(1D - Brownianmotion) = 1.497$. 

(a) A Brownian motion over time (one dimension)  
(b) The linear approximation
5.7 A Twodimensional Percolation Cluster

Below the retrieved picture of a percolation cluster, made by my mentor, is displayed:

![A twodimensional percolation cluster](image)

![The linear approximation](image)

Figure 13: The simulated Brownian motion with linear interpolation, $dim_B(2D - percolation cluster) = 1.744$.

<table>
<thead>
<tr>
<th>Fractal</th>
<th>$dim_B$</th>
<th>Confidence Bound</th>
<th>$dim_H$</th>
<th>Error wrt $dim_H$</th>
<th>Measure Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cantor set</td>
<td>0.6386</td>
<td>0.6334, 0.6439</td>
<td>0.6309</td>
<td>1.2%</td>
<td>1.6%</td>
</tr>
<tr>
<td>Sierpinski Triangle</td>
<td>1.584</td>
<td>1.583, 1.585</td>
<td>1.585</td>
<td>0.063%</td>
<td>0.126%</td>
</tr>
<tr>
<td>Sierpinski Carpet</td>
<td>1.877</td>
<td>1.841, 1.913</td>
<td>1.893</td>
<td>0.83%</td>
<td>3.8%</td>
</tr>
<tr>
<td>Vicsek Fractal</td>
<td>1.453</td>
<td>1.436, 1.47</td>
<td>1.465</td>
<td>0.817%</td>
<td>2.34%</td>
</tr>
<tr>
<td>Hénon mapping</td>
<td>1.236</td>
<td>1.232, 1.24</td>
<td>1.26</td>
<td>1.9%</td>
<td>0.65%</td>
</tr>
<tr>
<td>1-D Brownian Motion</td>
<td>1.497</td>
<td>1.487, 1.506</td>
<td>1.5</td>
<td>0.2%</td>
<td>0.6%</td>
</tr>
<tr>
<td>2-D Percolation Cluster</td>
<td>1.744</td>
<td>1.73, 1.758</td>
<td>1.75</td>
<td>0.34%</td>
<td>1.61%</td>
</tr>
</tbody>
</table>

Table 1: Complete table of computations. Note that wrt = “with respect to”.

In table 1 all the computed values will conclude the results. The measurement error is represented in the rightmost column, and is computed by taking the absolute value of the confidence bound’s length and dividing it by the computed value of $dim_B$ for each set. The confidence bound is in turn the range of values that the computed values are in, with respect to error bounds. All Hausdorff dimensions for the selfsimilar fractals have been calculated from the simple formula in the introduction. The others are standards.
6 Discussion

What can be read from table 1 is that, primarily, the box-counting dimension of an object need not be exactly equal to the Hausdorff dimension, even though they can be really close at times. That is a main fact one has to have in mind when simulating fractals. What one does when computing the box-counting dimension is to numerically approximate the Hausdorff dimension of a given set. The lesser the side-length of the square is the more squares we need, and the more squares we compute with the closer we get to the theoretical values of a set’s dimension. The fact that the computed results are so close to the theoretic results just goes to show that the used algorithm was well constructed. We did have some really good results from the Sierpinski triangle, the Hénon attractor, the Brownian motion and the percolation cluster, in the latter case a nigh-on perfect result, which, according to my mentor, is rarely achieved.

The linear interpolation is inaccurate when we do not plot enough points in a set, and therefore the six rightmost points in each diagram have been excluded from the interpolation. To slim down the error bounds one can plot more points in each fractal set, it would for example give a better error bound for the Sierpinski carpet if we plotted about 500 000 points or more.

Future studies may, for example, study more fractals in a similar way, perhaps with more non-self-similar fractals, or fractals that live in the complex plane. It would be interesting to study how fractal structures can appear in chaotic systems, and to simulate these to make some relevant computations.

Another exciting project would be to scan different natural objects such as leaves, small collections of water on the ground (believe it or not, they have exotic boundaries at times, in which case they in my opinion resemble the structure of the Mandelbrot set), rocks or anything irregular. Then one could experimentally determine their box-counting dimensions
by e.g. selecting points at their boundaries and running the same simulation as I have done in this study. By numerically estimating the dimension of a set, one has an easier time doing that than struggling with the complicated definition of Hausdorff dimension. Fractal structures appear everywhere, so there are surely a lot of things to measure while waiting for the new geometry to arrive and explain nature thoroughly.
7 Acknowledgements

We would like to thank the organizers of Research Academy for Young Scientists, Rays*, for enabling such a huge opportunity for me to learn about the mathematical science. We would also like to thank Gaultier Lambert for being our mentor during this project. Also Stockholms Matematikcentrum and Institut Mittag-Leffler deserves credit for making this research possible, and Europaskolan for hosting us during our stay in Strangnas. Finally we would like to express our gratitude towards Mathworks for allowing us to use their product, MATLAB. Without the license to work with the software, we would have had some slight complications with the box-counting of the fractals. We thank you all.
References


A Python code

The following codes, written in PYTHON v.3.2, were used to draw some of the fractals. Most fractals that I could not plot with IFS with MATLAB, I plotted with recursive algorithms and turtle graphics.

A.1 Koch curve

# Program som ritar upp Von Kochs snflinga, det 1,261859506...
# (lg4/lg3)-dimensionella objekt som gav upphov till reformeringen av
# dimensionsbegrepet samt definieringen av begreppet Hausdorff-dimension.

from turtle import*
from math import*

def triangle(length,n): # Funktion som ska rita upp en kochkurva.
    if n==1:
        fd(length)
        return()
    else:
        triangle((length/3),(n-1))
        rt(60)
        triangle((length/3),(n-1))
        lt(120)
        triangle((length/3),(n-1))
        rt(60)
        triangle((length/3),(n-1))
    return()

try:
    speed(0) # Vi vill att det ska g snabbt att rita upp kurvan.
    ht()
    n=int(input("Ange vilken iteration du vill se.")) # Instllningar anges.
    length=int(input("Ange sidlngd p ursprungstriangeln."))
    if n<=0 or length<=0: # Kontroll av indata sker.
        raise ValueError
    pu()
    goto(length*cos(7*pi/6)/1.5,length*sin(7*pi/6)/1.5) # Vi flyttar oss till start.
    pendown()
    triangle(length,n) # Vi kr samma procedur tre gnger, en fr var sida.
    lt(120) # Skldpaddan vrids 120 mellan varje funktionsanrop.
    triangle(length,n) # Den andra "sidan" i flingan ritas ut med detta anrop.
    lt(120)
triangle(length,n) # Den tredje "sidan" ritas sledes ut hr.
a=input("Klart!") # Vi mste kunna beskda flingan nr den vl r uppritad.
bye()
except ValueError:
    print("Du mste ange positiva heltal.")

### A.2 Cantor set - recursive algorithm

from turtle import*

def cantor(n,r):
    if n==1:
        fd(r)
        return()
    else:
        cantor(n-1,r/3)
        pu()
        fd(r/3)
        pd()
        cantor(n-1,r/3)
    return()

ht()
speed(0)
n=int(input("Hur mnga itereringar?"))
r=int(input("Ange ursprungslngd: "))
if n>=10:
    tracer(1000,0)
elif n>=5:
    tracer(30,0)
pu() goto(-(r/2),0)
pd()
cantor(n,r)
k=input("Klar!")

### A.3 Pythagoras’ tree

from turtle import*
from math import*

def square(n):
    begin_fill()
for i in range(4):
    fd(n)
    lt(90)
    end_fill()
    return()

def iterator(n,r):
    if r==1:
        square(n)
        return()
    else:
        square(n)
        lt(90)
        fd(n)
        rt(45)
        fd(n/sqrt(2))
        (x,y)=pos()
        bk(n/sqrt(2))
        z=heading()+90
        iterator(n/sqrt(2),r-1)
        pu()
        goto(x,y)
        pd()
        seth(z)
        rt(180)
        iterator(n/sqrt(2),r-1)
    return()

r=int(input("Ange iteration: "))
n=int(input("Ange lngd: "))
speed(0)
h(0)
pu()
goto(-n/2,-250)
pd()
iterator(n,r)
k=input("Klar!")
bye()

A.4 Harter-Heighway’s Dragoncurve
rom turtle import*
from math import*
def drawTheDragon(n,r,par):
    if not n:
        fd(r)
        return()
    else:
        if not par:
            lt(45)
            a=heading()
            r/=sqrt(2)
            drawTheDragon(n-1,r,1)
            seth(a)
            rt(90)
            drawTheDragon(n-1,r,0)
        else:
            rt(45)
            a=heading()
            r/=sqrt(2)
            drawTheDragon(n-1,r,1)
            seth(a)
            lt(90)
            drawTheDragon(n-1,r,0)
        return()

ht()
speed(0)
n=int(input("Ange iteration: "))
r=int(input("Ange grundlngd: "))
pu()
goto(-r/2,0)
pd()
drawTheDragon(n,r,1)
k=input("Klar!")
bye()

B MATLAB code for plotting fractals

B.1 The Mandelbrot Set

function [ ] = mandelbrot3( )
%Matlab function for plotting the Mandelbrot set.
% Searches for points belonging to the fractal within the circle of
radius 2 and center in origo.
lista=spdiags(speye(62800)); % Two arrays with room for all elements searched
lista2=spdiags(speye(62800));
a=2;
b=2;
index=1;
while a>0
    theta=0;
    while theta<=pi;
        k=a;
        l=b;
        lista(index)=mandelbrot(k*cos(theta)+i*l*sin(theta)); % The other function
        lista2(index)=imag(lista(index)); % checks whether a point is in the set.
        lista(index)=real(lista(index));
        index=index+1;
        lista(index)=lista(index-1); % Symmetry taken to consideration.
        lista2(index)=(-1)*lista2(index-1);
        theta=theta+0.01;
        index=index+1;
    end
    a=a-0.01;
    b=b-0.01;
end
scatter(lista,lista2,1)
end

function [ in ] = mandelbrot( c )
%Function for checking if a given point is in the Mandelbrot set.
% Checks if the recurrence relation diverges to infinity if iterated 1000 times.
z=0;
in=c;
for i=1:1000 % Iterates the recurrence formula 1000 times at most.
    z=z*z+c;
    if abs(z)>2
        in=0; % We see that c does not belong to the set, and we return 0 instead.
        return; % (Zero belongs to the set, so it acts as a dummy variable)
        break % We do not need to check anymore.
    end
end
return; % We return the value of in (c or 0)
end
B.2 The Cantor Set

function [ A ] = cantor( iterations )
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
lista=spdiags(speye(iterations-30));
lista2=spdiags(speye(iterations-30));
lista2=lista2*0;
x=rand;
y=rand;
for i=1:30
    p=randint;
    if p==0
        x=x/3;
    else
        x=x/3;
        x=x+2/3;
    end
end
for i=31:iterations
    p=randint;
    if p==0
        x=x/3;
        lista(i-30)=x;
    else
        x=x/3;
        x=x+2/3;
        lista(i-30)=x;
    end
end
A=[lista lista2];
scatter(lista,lista2,1);
return;
end

B.3 Sierpinski’s Triangle

function [ A ] = sierpinski( iterations )
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
A=[];
x=rand;
if(x<=0.5)
```matlab
y = rand*sqrt(3)*x;
else
    y = rand*2*(1-x);
end
for i = 1:iterations
    p = randi(3);
    if p == 1;
        x = x/2;
        y = y/2;
    elseif p == 2;
        x = x/2 + 1/2;
        y = y/2;
    else p == 3;
        x = x/2 + 1/4;
        y = y/2 + 1/2;
    end
    A(i,1) = x;
    A(i,2) = y;
end
scatter(A(:,1),A(:,2),1);
return;
end
```

### B.4 Sierpinski’s Carpet

```matlab
function [ A ] = carpet( iterations )
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
A=[];
x=rand;
y=rand;
for i = 1:iterations
    p = randi(8);
    if p == 1;
        x = x/3;
        y = y/3;
    elseif p == 2;
        x = x/3 + 1/3;
        y = y/3;
    elseif p == 3;
        x = x/3 + 2/3;
        y = y/3;
    elseif p == 4;
```
The Vicsek Fractal

function [ A ] = vicsek( iterations )

%VICSEK Summary of this function goes here
% Detailed explanation goes here

A=[];
x=rand;
y=0;
for i=1:iterations
    p=randi(5);
    if p==1
        x=x/3;
y=y/3;
x=x+1/3;
    elseif p==2
        x=x/3+2/3;
y=y/3+1/3;
    elseif p==3
        x=x/3+1/3;
y=y/3+1/3;
    elseif p==4
        x=x/3+2/3;
y=y/3+2/3;
    elseif p==5
        x=x/3+2/3;
y=y/3+1/3;
    elseif p==6
        x=x/3+2/3;
y=y/3+2/3;
    elseif p==7
        x=x/3+1/3;
y=y/3+2/3;
    elseif p==8
        x=x/3;
y=y/3+2/3;
    end
    A(i,1)=x;
    A(i,2)=y;
end
scatter(A(:,1),A(:,2),1);
return;
end
x=x/3;
y=y/3+1/3;
else
    x=x/3+1/3;
y=y/3+2/3;
end
A(i,1)=x;
A(i,2)=y;
end
scatter(A(:,1),A(:,2),1);
return;
end

B.6 The Hénon Mapping

function [ A ] = henon( iterations )
%UNTITLED3 Summary of this function goes here
% Detailed explanation goes here
A=[];
a=1.4;
b=0.3;
x=0;
y=0;
for i=1:iterations
    xbef=x;
x=y+1-a*(xbef*xbef);
y=b*xbef;
A(i,1)=x;
A(i,2)=y;
end
scatter(A(:,1),A(:,2),1);
return;
end

C Box-Counting Programs

The following two MATLAB functions were used to calculate the box-counting dimensions of all the fractals plotted in MATLAB.

function [ M ] = box(A,n)
%Sums up all the boxes of size n x n needed to cover all points in a set A
% The coordinates (they are in [0,1]x[0,1] as input) are floored, and then
% the cells in a boolean matrix that has at least one floored point in it are marked.
% The amount of marked cells are returned at the end.
N=size(A(:,1));
H=0*speye(n);
M=0;
for i=1:N
    a=floor(A(i,1)*n)+1;
    b=floor(A(i,2)*n)+1;
    if H(a,b)==0
        H(a,b)=1;
        M=M+1;
    end
end
end

function [X,Y] = paket(B)
% Function that repeatedly activates box with different values of n
% All return values are saved for linear regression at a later stage.
i=1;
N=size(B,1)/10;
X(i)=10;
Y(i)=box(B,X(i));
while X(i)<=N
    X(i+1)=2*X(i)
    i=i+1;
    Y(i)=box(B,X(i));
end
X=log(X); % Necessary steps to calculate dimension.
Y=log(Y);
end

D Gallery of generators

For the interested readers, here are some generators for the most common selfsimilar fractals:

D.1 The Pythagorean Tree

This treelike fractal is built upon a generator that looks like one of the most well-known figures used to prove the Pythagorean theorem, hence the name. To calculate this fractal’s dimension, we need to know how many new copies of the previous iteration that appear (2), and their scaling factors ($\sqrt{2}$). Inserting into the equation for Hausdorff dimension for
selfsimilar fractals,\n\[ \dim_H(\text{Tree}) = \frac{\log 2}{\log \sqrt{2}} = 2 \]

![Figure 14: The Pythagorean Tree with generator.](image)

(a) The tree (15th iteration).  
(b) The generator.

D.2 Harter-Heighways’ dragon curve

This fractal has an interesting shape, and supposedly is found in the "Jurassic Park"-book[6] where each iteration is shown at the beginning of every chapter. Observe that the generator should be placed alternately at one side and alternately at the other side of a line when iterating more than once. Looking at the generator, the lines are rotated 45 degrees from the original line, thus making them scaled down by the squareroot of two. Inserting into the formula for a selfsimilar fractal’s dimension together with the number of copies (2) we get that
\[ \dim_H(\text{Dragoncurve}) = \frac{\log 2}{\log \sqrt{2}} = 2 \]

![Figure 15: The Dragon curve with generator.](image)

(a) The dragon curve (15th iteration).  
(b) The generator.
D.3 The Sierpinski Triangle

This fractal has already been mentioned before, but it fits in here too, so here you go.

\[ \dim_H = \frac{\log 3}{\log 2} = 1.5845... \]

(a) The Sierpinski triangle.  
(b) The generator.

Figure 16: The Sierpinski Triangle with generator.

D.4 The Sierpinski Carpet

This fractal has already been mentioned before, but it fits in here too, so here you go.

\[ \dim_H = \frac{\log 8}{\log 3} = 1.893... \]

(a) The Sierpinski Carpet.  
(b) The generator.

Figure 17: The Sierpinski Carpet with generator.
D.5 The Cantor set

This fractal has already been mentioned before, but it fits in here too, so here you go.

\[ \dim_H = \frac{\log 2}{\log 3} = 0.6309... \]

(a) The Cantor set.  
(b) The generator.

Figure 18: The Cantor set with generator.

D.6 The Vicsek fractal

This fractal has already been mentioned before, but it fits in here too, so here you go.

\[ \dim_H = \frac{\log 5}{\log 3} = 1.465... \]

(a) The Vicsek fractal.  
(b) The generator.

Figure 19: The Vicsek fractal with generator.
D.7  A line-based fractal

I am not sure what the name of this fractal is, but it is based on a line.

\[ \dim_H = \frac{\log 5}{\log 3} = 1.465... \]

(a) The fractal.  (b) The generator.

Figure 20: The fractal with generator.

D.8  Another line-based fractal

I am not sure what the name of this fractal is either, but it is also based on a line.

\[ \dim_H = \frac{\log 8}{\log 3} = 1.893... \]

(a) The fractal.  (b) The generator.

Figure 21: The fractal with generator.
D.9 The von Koch curve

Last but not least, we are going to end the paper with the first fractals introduced in this paper, the von Koch curve and the snowflake.

\[ \dim_H(\text{Kochcurve}) = \frac{\log 4}{\log 3} = 1.26... \]

(a) The fractal. 
(b) The generator.

Figure 22: The fractal with generator.

Figure 23: The snowflake.