

The Emergence of Patterns in Nature and Chaos Theory

Ryan Greenup & James Guerra

October 22, 2020

Contents

1	Introduction	RYAN	3
1.1	A note on Images in this report		3
2	Fractals		3
2.1	Definition of a Fractal	RYAN	3
2.2	Fractals Generally	JAMES	4
2.3	Fractal Dimension		5
2.3.1	Topological Equivalence	RYAN	5
2.3.2	Hausdorff Measure	RYAN	8
2.3.3	Hausdorff Dimension	RYAN	11
2.3.4	Box Counting Dimension	JAMES	12
2.4	Generating Self Similar Fractals	RYAN	13
2.4.1	Vicsek Fractal		13
2.4.2	Turtle		19
2.4.3	Pascals Triangle and Sierpinski's Triangle	JAMES	23
2.5	Fractal Dimensions Sans Self Similarity	RYAN	26
2.5.1	Calculating the Dimension of Julia Sets		26
3	Connecting Fractals to Natural Processes	RYAN	30
3.1	Prove Fibonacci using Monotone Convergence Theorem		33
3.1.1	Show that the Series is Monotone		34
3.1.2	Show that the Series is Bounded		34
3.1.3	Find the Limit		34
3.1.4	Comments		34
3.1.5	Python		35
4	The Fibonacci Sequence		35
4.1	Introduction	RYAN	36
4.2	Computational Approach	RYAN	36
4.3	Exponential Generating Functions		37
4.4	Proving with the Monotone Convergence Theorem	RYAN	49
4.5	Fibonacci Sequence and the Golden Ratio	RYAN	49

CONTENTS

4.5.1	Fibonacci Sequence in Nature (This may be Removed)	RYAN .	50
4.6	Relating The Fibonacci Sequence to the Julia Set	RYAN	51
5	Julia Sets	RYAN	51
5.1	Introduction		51
5.2	Motivation		53
5.3	Plotting the Sets		54
6	MandelBrot Set	RYAN	61
7	Appendix		63
.1	Finding Material		64
.2	Code Listings		64
.2.1	Sierpinski's Carpet		64
.2.2	Cantor's Dust		64
.3	Font Lock		64
.4	Resources Used for the Hausdorff Dimension		67
.5	Functions for constructing the Julia Set		67

1 Introduction

RYAN

Fractals are complex shapes that often occur from natural processes, in this report we hope to investigate the emergence of patterns and complex structures from natural phenomena. We begin with an investigation into fractals and the concept of dimension and then discuss links between fractal patterns and natural processes.

1.1 A note on Images in this report

Although the images in this document may appear to be quite small, they are high quality PNG images and one of the luxuries of PDF is that the embedded media is near lossless, so to perceive greater detail in an image it is sufficient to simply zoom in and the greater detail should be rendered.¹

2 Fractals

2.1 Definition of a Fractal

RYAN

Benoît Mandelbrot coined the term fractal in 1975 [15] and defined it in his 1982 book *The Fractal Geometry of Nature* [33, p. 15] :

A fractal is by definition a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension.

*Every set with a noninteger D is a fractal.*²

The topological dimension is a strictly integer value that describes the natural dimension used to describe a shape [45], for example the *Koch Snowflake* (shown in figure 2) is composed of just a line, so its topological dimension would simply be 1, its *fractal dimension* however is shown to be $\frac{\ln(4)}{\ln(3)}$ at (8) in §2.3.1.

Many authors seem to accept this earlier definition (see e.g. [51, §2.2] and [49, §2.1]), this definition however does not capture many edge-case fractals [7, p. VII] and in reprintings of *The Fractal Geometry of Nature* Mandelbrot himself commented that in hindsight it may have been more appropriate [33, p. 459]:

to leave the term “fractal” without a pedantic definition, to use “fractal dimension” as a generic term applicable to all the variants in Chapter 39, and to use in each specific case whichever definition is the most appropriate.

¹Apparently it's also possible to embed live GIFs into PDF as well which would have been nice to do had time permitted, see for example the Animate package for L^AT_EX[6] and this discussion [40] generally.

²In this quote by Mandelbrot is indicating that by the earlier definition of a fractal, any shape (i.e. a subset of \mathbb{R}^n) with a non-integer dimension must be a fractal. This however is merely a sufficient, rather than a necessary, condition of a fractal, for example the *Dragon Curve* discussed in section 2.4.2 and the Mandelbrot set discussed at section 6 both have a dimension of 2 but are clearly fractals by this definition because their shape is constructed with a line, which has a “topological” dimension of 1.

Gerald Edgar, in his 2008 book *Measure, topology, and fractal geometry* rejected this view because “*a term without a ‘pedantic definition’ cannot be studied mathematically*” [7, p. VII] and presented a more robust definition in Ch. 6 of that book, it was however accepted that the loose definition of fractal dimension is convenient and was indeed adopted in that work.

Although reviewing the precise definition of a fractal would have been very interesting, without a cursory knowledge of fractals generally this would have been very time consuming and outside the scope of this report³.

Some authors simply define a fractal as a shape that *shows irregularities at all scales* [16, p. 1] and in his 2003 book, *Fractal Geometry*, Facloner suggested that it is more convenient to describe a fractal by a list of properties characteristic of such shapes⁴ because of the difficulty in defining a fractal in a way that can encompass all edge cases and provides the following characteristics [9, p. xxv]:

- Detail at all scales
- Cannot be described in a traditional geometric way
- May have some form of approximate self similarity
- Usually the fractal dimension is greater than its topological dimension
- In many cases defined very simply, perhaps recursively

This will be the approach adopted in this report.

It's interesting to note that many authors refer to complex natural shapes as fractals, such as coastlines (see e.g. [20, 54, 53]) much in the spirit of Mandelbrot's paper *How long is the Coastline of Britain* [32], although he coined the term fractal many years after this paper, presumably he might have had this in mind⁵ when framing the definition so this issue in clearly defining what a fractal is appears on the surface to be a purely mathematical one (as opposed to a practical or applied one).

The Wikipedia page on fractals [12] also points out that a fractal is nowhere differentiable, this would be because a fractal is nowhere smooth, which is certainly an important and distinguishing feature worth noting.

2.2 Fractals Generally

JAMES

Dimension is the main defining property of a fractal. As aforementioned above, the Hausdorff dimension is a unique number in that, if we take some shape in \mathbb{R}^n , and the Hausdorff dimension converges to some number, then the dimension of the shape is given by that number. Otherwise, it will equal 0 or ∞ . For example, if we want to evaluate the dimension of a square and we use a 1-Dimensional shape as the cover set

³Mandelbrot also discussed fractals of a Euclidean and Reimannian nature, [33, p. 361], this again is interesting but too specific for the broad nature of our investigation

⁴Much like the definition of life in the field of biology

⁵Mandelbrot also spent much time looking at the roughness of financial markets and so presumably may have had that in mind as well, see e.g. [15, 34]

to calculate the Hausdorff dimension, we will get ∞ . On the other hand, if we do the same with a 3-Dimensional shape, we will get 0. And finally if we use a 2-Dimensional shape, the Hausdorff dimension will evaluate to 2. This same notion is important when computing the dimension of a more complex shape such as the Koch snowflake.

To define a fractal, we must define it's dimension. Whilst some research states that a fractal has a non-integer dimension, this is not true for all fractals. Although, most fractals like the Koch snowflake do in fact have non-integer dimensions, we can easily find a counter example namely, the Mandelbrot set. The Mandelbrot set lies in the same dimension as a square, a 2-Dimensional shape. However, we give recognition to the complexity and roughness of the Mandelbrot set which clearly distinguishes itself from a square. Beneath the Mandelbrot set's complexity are exact replicates of the largest scaled Mandelbrot set, i.e a self similar shape. Furthermore, although the Mandelbrot set has an integer dimension, the self similarity and complexity is what also defines its fractal nature.

2.3 Fractal Dimension

The concept of a non-integer dimension may at first seem odd, particularly given that the familiar definition from linear algebra (concerned with the number of vectors within a basis for a vector space [26]) is strictly an integer value, but in the early 20th century mathematicians recognised the shortcomings of this definition [33, Ch. 3].

In this section we hope to convince the reader that there is grounds for extending the definition of dimension and as a matter of fact many definitions for non-integer dimensions of a shape have been proposed (see generally [33, Ch. 39] and [16, §1.3]) of these the Hausdorff Dimension (and corresponding Hausdorff Measure) is considered to be the most important and mathematically robust [9, p. 27], while the *Box-Counting Dimension* (introduced in §2.3.4) has the most practical applications in science [41, p. 192].

The Hausdorff dimension is more like counting balls than boxes and is identical to the Box-Counting Dimension in many cases, it's more general but harder to define [45]. An extension to the work of this report would be to show the mathematical connections between the Hausdorff Dimension and box counting dimension with respect to the fractals generated and measured.

2.3.1 Topological Equivalence

RYAN

Topology is an area of mathematics concerned with ideas of continuity through the study of figures that are preserved under homeomorphic transformations [14] , where two figures are said to be homeomorphic if there is a continuous bijective mapping between the two shapes [41, p. 105] . ⁶

So for example deforming a cube into a sphere would be homeomorphic, but deforming a sphere into a torus would not, because the the surface of the shape would have to be compromised to achieve that.

⁶For further reading on this topic see [41, p. 106]

As mentioned above, historically, the concept of dimension was a difficult problem with a tenuous definition. Although an intuitive definition related the dimension of a shape to the number of parameters needed to describe that shape, this definition is not sufficient to be preserved under a homeomorphic transform.

Consider the koch fractal and snowflake in figures 1 and 2, at each iteration (n) the perimeter is given by $p_n = \left(\frac{4}{3}\right) p_{n-1}$ and the number of edges by N_n :

$$N_n = N_{n-1} \cdot 4 \quad (1)$$

$$= 3 \cdot 4^n \quad (2)$$

If the length of any individual side was given by l and scaled by some value s then the length of each individual edge would be given by:

$$l = \frac{s \cdot l_0}{3^n} \quad (3)$$

The total perimeter would be given by:

$$p_n = N_n \times l \quad (4)$$

$$= 3 \cdot 4^n \times \frac{s \cdot l_0}{3^n} \quad (5)$$

$$= 3 \cdot s \cdot l_0 \left(\frac{4}{3}\right)^n \quad (6)$$

$$\implies p_n \cdot s \propto \left(\frac{4}{3}\right)^n \quad (7)$$

$$\implies n = \frac{\log(4)}{\log(3)} \approx 1.26 \quad (8)$$

This means that if the koch snowflake is scaled by any factor, the resulting perimeter of the snowflake will not be linearly proportional to the scaling factor, as would be the case with an ordinary shape such as a square or a circle, it will instead be proportional to 1.26, this should hopefully motivate the need to more clearly define both the concept of measure (in this case the perimeter⁷) and dimension.

To clarify the koch snowflake, is defined such that there are no edges, every point on the “curve” is the vertex of an equilateral triangle, this shape has no smooth edges.

See [47, p. 414] and [2, §5.4] for further reading on the self similar dimension of the *Koch Snowflake*.

This approach of considering the scaling factor of a deterministic fractal is known as the similarity dimension [47, p. 413] and should be equal to the Hausdorff and box counting dimensions for most fractals. For fractals that aren’t so obviously self similar

⁷Grant Sanderson equates the measure of a fractal as analogous to mass, which is a very helpful way comparison [45]

2.3 Fractal Dimension

it won't be feasible however [29, p. 393], for example with the julia set ⁸ or the outline of a coastline it is not immediately clear if the the dimension would be constant at all scales

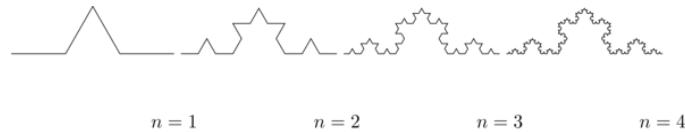


Figure 1: Progression of the Koch Snowflake

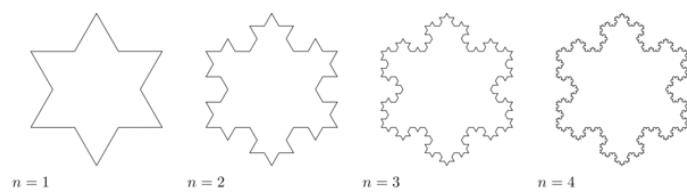


Figure 2: Progression of the Koch Snowflake

⁸It is indeed shown to be mostly constant at all scales in section §2.5.1

2.3.2 Hausdorff Measure

The Hausdorff dimension depends first on a rigorous definition of measure, this is distinct from the box counting approach in that it is more mathematically rigorous, it is however complex and in practice this report will be concerned with implementing the box counting dimension.⁹

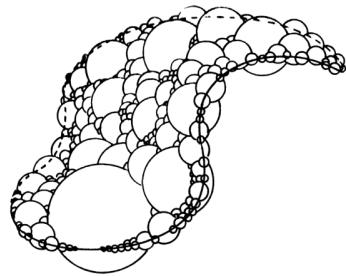


Figure 3: The Hausdorff Measure, in this case area, of an arbitrary surface approximated by the cross section of little balls of diameter $< \delta$, this is reproduced from *Measure, Topology and Fractal Geometry* [7, p. 166] because it so elegantly illustrates the concept of the Hausdorff Measure.

Let F be some arbitrary subset of euclidean space \mathbb{R}^n ,¹⁰

Let U be a subset of euclidean space \mathbb{R}^n such that the diameter is defined as the greatest distance between any of the points:

$$|U| = \sup (\{|x - y| : x, y \in U\})$$

Consider a collection of these sets, $G = \{U_i : i \in \mathbb{Z}^+\}$ such that each element has a diameter less than δ .

The motivating idea is that if the elements of G can be laid onto F then G is said to be a δ -cover of F , more rigorously, G is a δ -cover of F if:¹¹

$$F \subset \bigcup_{U \in G} (U) \quad : \quad 0 \leq |U| \leq \delta \quad (9)$$

An example of this covering is provided in figure 4, in that example the figure on the right is covered by squares, which each could be an element of $\{U_i\}$, it is important to note, by this definition, that the shapes represented by U could be any arbitrary figure [9, §2.1] the size of which may vary in size so long as the diameter is less than δ .

So for example:

- F could be some arbitrary 2D shape, and U_i could be a collection of identical squares, OR
- F could be the outline of a coastline and U_i could be a set of circles, OR

⁹See §.4 for further reading.

¹⁰A subset of euclidean space could be interpreted as an uncountable set containing all points describing that region

¹¹Falconer defines this as $\bigcup_{i=1}^{\infty}$ [9, §2.1], presumably treating any index value greater than the cardinality of the set as \emptyset , this is particularly ambiguous and we have avoided it, an alternative way to present that might be $\bigcup_{i=0}^{\#G}$ where $\#G$ denotes the cardinality of G (or ∞ if it is uncountable). The use of $\#\square$ to denote cardinality was introduced by Knuth in *Concrete Mathematics* [17] and is convenient in that it avoids any ambiguity with diameter ($|\square|$).

- F could be the surface of a sheet and U_i could be a set of spherical balls as shown in figure 3
 - Some authors suggest that the Haussdorff Measure is concerned primarily with round covering objects (see e.g. [45]), this is well illustrated by figure 3, however in truth it is merely more convenient to use round shapes for most fractals.
 - The use of balls is a simpler but equivalent approach to the theory [9, §2.4] because any set of diameter r can be enclosed in a ball of radius $\frac{r}{2}$ [8, p. 166]
- F could be a more abstracted figure like figures 4 or 6 and $\{U_i\}$ a collection of various different lines, shapes or 3d objects.

The Hausdorff measure is concerned with only the diameter of each element of $\{U_i\}$ and considers $\sum_{U \in G} [|U|^s]$ where each element $U \in G$ is arranged so as to minimize the value of the summation [9, p. 27], the δ -Hausdorff is hence defined, for various dimensions s :

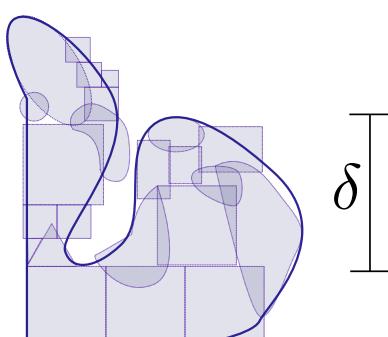
$$\mathcal{H}_\delta^s(F) = \inf \left\{ \sum_{U \in G} |U|^s : \{U_i\} \text{ is a } \delta\text{-cover of } F \right\}, \quad \delta, s > 0 \quad (10)$$

The value of s can be different regardless of the dimension of F , for example if F was an arbitrary 2D shape the value of $\mathcal{H}_\delta^2(F)$ is equivalent to considering the number of shapes $U \in G$ (e.g. boxes, discs etc.), of diameter $\leq \delta$ that will cover over a shape as shown in figure 4, the delta Haussendorf measure $\mathcal{H}_\delta^2(F)$ will be the area of the boxes when arranged in such a way that minimises the area.

As δ is made arbitrarily small \mathcal{H}_δ^s will approach some limit, in the case of figures 4 and 6 the value of \mathcal{H}_δ^2 will approach the area of the shape as $\delta \rightarrow 0$ and so the s^{th} dimensional Hausdorff measure is given by:

$$\mathcal{H}^s = \lim_{\delta \rightarrow 0} (\mathcal{H}_\delta^s) \quad (11)$$

This is defined for all subsets of \mathbb{R}^n for example the value of \mathcal{H}^2 corresponding to figure 6 will be limit that boxes would approach when covering that area, which would be the area of the shape ($4 \times 1^2 + 4 \times \pi \times \frac{1}{2^2} + \frac{1}{2} \times 1 \times \sin \frac{\pi}{3}$).



Lower Dimension Hausdorff Measurements

Examples Consider again the example of a 2D shape, the value of \mathcal{H}^1 would still be defined by (10), but unlike \mathcal{H}^2 in §2.3.2 the value of $|U_i|^1$ would be considered as opposed to $|U_i|^2$ (i.e.

Figure 4: The blue outline corresponds to some $F \subset \mathbb{R}^2$, covered by various grey objects, each of which represent an element from the set U_i . The grey shapes all have a diameter less than δ and so this $\sum |U_i|$

the diameter as opposed to the diameter squared).

As δ is made arbitrarily small the boxes ¹² that cover the shape are made also to be arbitrarily small. Although the area of the boxes must clearly be bounded by the shape of F , if one imagines an infinite number of infinitely dense lines packing into a 2D shape with an infinite density it can be seen that the total length of those lines will be infinite

and so the limit in (11) will increase without bound.

To build on that same analogy, another way to imagine this is to pack a 2D shape with straight lines, the total length of all lines will approach the same value as the length of the lines of the squares as they are packed infinitely densely. Because lines cannot fill a 2D shape, as the density of the lines increases, the overall length will increase without bound.

This is consistent with fractals as well, consider the koch snowflake introduced in section 2.3.1 and shown in figure 1, the dimension of this shape, as shown in §2.3.1 is greater than 1, and the number of lines necessary to describe that shape is also infinite because every point of the “curve” is a point of an equilateral triangle.

Formally If the dimension of F is less than s , the Hausdorff Measure will be given by:

$$\dim(F) < s \implies \mathcal{H}^s(F) = \infty \quad (12)$$

Higher Dimension Hausdorff Dimension For small values of s (i.e. less than the dimension of F), the value of \mathcal{H}^s will be ∞ .

Consider some value s such that the Hausdorff measure is not infinite, i.e. values of s :

$$\mathcal{H}^s = L \in \mathbb{R}$$

Consider a dimensional value t that is larger than s and observe that:

¹²Even though U may contain a variety of shapes, (10) is concerned only with the power of there diameter, so in this sense the limit is concerned only with boxes corresponding to the diameter of the elements of U

$$\begin{aligned}
 0 < s < t \implies \sum_i [|U_i|^t] &= \sum_i [|U_i|^{t-s} \cdot |U_i|^s] \\
 &\leq \sum_i [\delta^{t-s} \cdot |U_i|^s] \\
 &= \delta^{t-s} \sum_i [|U_i|^s]
 \end{aligned}$$

Now if $\lim_{\delta \rightarrow 0} [\sum_i |U_i|^s]$ is defined as a non-infinite value:

$$\lim_{\delta \rightarrow 0} \left(\sum_i [|U_i|^t] \right) \leq \lim_{\delta} \left(\delta^{t-s} \sum_i [|U_i|^s] \right) \quad (13)$$

$$\leq \lim_{\delta \rightarrow 0} (\delta^{t-s}) \cdot \lim_{\delta \rightarrow 0} \left(\sum_i [|U_i|^s] \right) \quad (14)$$

$$\leq 0 \quad (15)$$

and so we have the following relationship:

$$\mathcal{H}^s(F) \in \mathbb{R}^+ \implies \mathcal{H}^t(F) = 0 \quad \forall t > s \quad (16)$$

Hence the value of the s -dimensional *Hausdorff Measure*, s is only a finite, non-zero value, when $s = \dim_H(F)$.

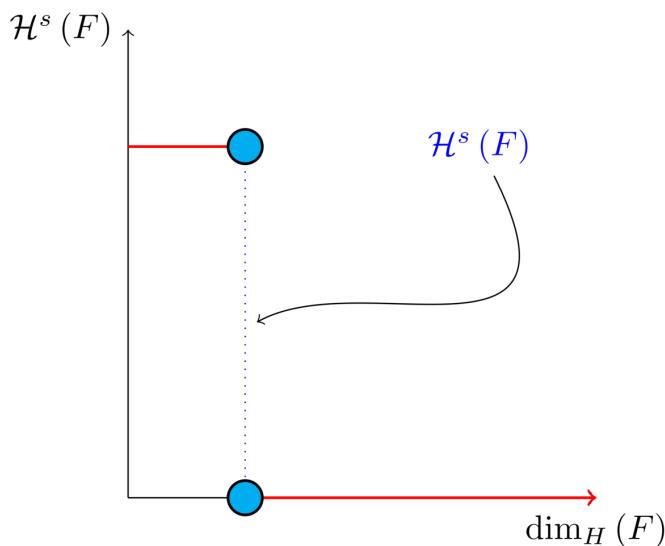


Figure 5: The value of the s -dimensional *Hausdorff Measure* of some subset of *Euclidean space* $F \in \mathbb{R}^n$ is 0 or ∞ when the dimension of F is not equal to s .

2.3.3 Hausdorff Dimension

RYAN

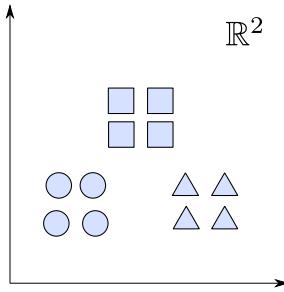


Figure 6: A disconnected subset of \mathbb{R}^2 , the squares have a diameter of $\sqrt{2}$, the circles 1 and the equilateral triangles 1.

The value s at which \mathcal{H}^s (16) changes from ∞ to 0, shown in figure 5, is the defined to be the *Hausdorff Dimension* [9, §2.2], it is a generalisation of the idea of dimension that is typically understood with respect to ordinary figures.

2.3.4 Box Counting Dimension JAMES

While the Hasudorff dimension is the first formal definition to measure the roughness of a fractal, there are several other definitions of dimension that have stemmed from this. Namely, the box-counting dimension. The box counting method is widely used as it is relatively easy to calculate [9, p. 41] and in many cases is equal to the *Hausdorff Dimension* [35, p. 11] (see generally [30]). The box-counting dimension is defined as the following from [10]:

Let F be any non-empty bounded subset of \mathbb{R}^n and let $N_\delta(F)$ be the smallest number of sets of diameter at most δ which can cover F . The *lower* and *upper* box-counting dimensions of F respectively are defined as

$$\dim_B F = \underline{\lim}_{\delta \rightarrow 0} \frac{\ln N_\delta(F)}{-\ln \delta}$$

$$\overline{\dim}_B F = \overline{\lim}_{\delta \rightarrow 0} \frac{\ln N_\delta(F)}{-\ln \delta}$$

When the *lower* and *upper* box-counting dimensions of F are equal, then

$$\dim_B F = \lim_{\delta \rightarrow 0} \frac{\ln N_\delta(F)}{-\ln \delta}$$

For example, suppose we had a square with side length 1 and we use smaller squares of side length $\frac{1}{\delta}$ to cover the larger square. This would mean that one side of the large square would need δ^{-1} small squares, and so to cover the entire square, one would need n^2 small squares, i.e. $N_{\frac{1}{\delta}}(F) = n^2$. Now, substituting these values into the box-counting definiton, we get:

$$\begin{aligned} \dim_B F &= \lim_{\frac{1}{\delta} \rightarrow 0} \frac{\ln(\delta^2)}{-\ln(\frac{1}{\delta})} \\ &= \lim_{\frac{1}{\delta} \rightarrow 0} \frac{\ln(\delta^2)}{\ln(\delta)} \\ &= \lim_{\frac{1}{\delta} \rightarrow 0} 2 \frac{\ln(\delta)}{\ln(\delta)} \\ &= 2 \end{aligned}$$

Which is expected, because we know that a square is a 2-Dimensional shape. We can apply this same concept to fractals. Consider another example, the Koch Curve, a self similar fractal which we can calculate its dimension and provide a measure of roughness of the curve. If we take a close look at the curve progression in figure 1, the pattern begins with one line segment and the middle third of the line is replaced with two sides of an equilateral triangle with side length $\frac{1}{3}$. After this first iteration, the line segment now becomes four line segments. Thus, if we use a square of length $\frac{1}{3^\delta}$ to cover the δ^{th} iteration of the curve, there will be 4^δ line segments covered.

Let F be the Koch Curve.

$$\begin{aligned}\dim_B F &= \lim_{\frac{1}{3^\delta} \rightarrow 0} \frac{\ln(4^\delta)}{-\ln(\frac{1}{3^\delta})} \\ &= \lim_{\frac{1}{3^\delta} \rightarrow 0} \frac{\ln(4^\delta)}{\ln(3^\delta)} \\ &= \lim_{\frac{1}{3^\delta} \rightarrow 0} \frac{\ln(4)}{\ln(3)} \\ &= \frac{\ln(4)}{\ln(3)}\end{aligned}$$

2.4 Generating Self Similar Fractals

In order to investigate the dimension of fractals, we intend to generate and measure a variety of figures by using of *R* [43], *Julia* [4] and *Python* [52].

Self Similar fractals have a self-similar dimension and so can be used to verify an approach implemented with a programming language.

2.4.1 Vicsek Fractal

The Vicsek Fractal [51, p. 12] involves a pattern of iterating boxes, to implement this consider the process¹³:

$$\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{B} & \mathbf{Z} & \mathbf{B} \\ \mathbf{Z} & \mathbf{B} & \mathbf{Z} \\ \mathbf{B} & \mathbf{Z} & \mathbf{B} \end{bmatrix} \quad (17)$$

where:

- $\mathbf{B} = [1]$
- $\mathbf{Z} = [0]$

If this is repeated many times a matrix of values will be created, such a matrix can be interpreted as a greyscale image and plotted as a heatmap to show the fractal (shown in figure).

¹³This was actually a fractal I came up with myself only to later find that somebody already had the same idea!

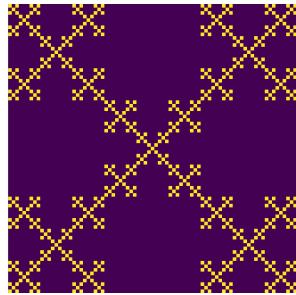


Figure 7: Vicsek fractal [51, p. 12] (also known as the *AntiCross-Stitch* [19]) produced by listing 1, at each iterative step the fractal itself is “copied” to the four corners of itself producing this complex shape.

At each step of the process, the number of elements of this fractal increases by a ratio of 5 while the height increases only by a factor of 3, hence the self similarity dimension is given by:

$$\begin{aligned} 5 &= 3^D \\ \implies D &= \frac{\ln 5}{\ln 3} \end{aligned} \tag{18}$$

By modifying listing 1 alternative fractals can get also be generated like *Cantor’s Dust* and *Sierpinski’s Carpet* shown in figures 9 and .

Upon review this is actually a variant on the *Cantor Dust* which should actually be represented by a 3×3 matrix:

$$\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{B} & \mathbf{Z} & \mathbf{B} \\ \mathbf{Z} & \mathbf{Z} & \mathbf{Z} \\ \mathbf{B} & \mathbf{Z} & \mathbf{B} \end{bmatrix} \tag{19}$$

and hence has the same dimension as the *Vicsek Fractal* as opposed to a dimension of 1.

Sierpinskis Carpet and Cantor’s Dust By modifying the approach provided in listing 1 other fractals such as *Sierpinski’s Carpet* and *Cantor’s Dust* can be produced, this is implemented in listings 21 22 and shown in figures 8 and 9 respectively.¹⁴

Sierpinski’s Triangle Not all fractal patterns can be produced by using recursive functions involving matrices, one such function is *Sierpinski’s Triangle*.

¹⁴See generally Ch. 2.7 of [41, §2.7] for further Reading.

```

1  #-----
2  #--- Function ---
3  #
4  function vicsek_matrix(ICMat, width)
5      B = ICMat
6      h  = size(B)[1]
7      w  = size(B)[2]
8      Z  = zeros(Int, h, w)
9      B = [B Z B ;
10         Z B Z ;
11         B Z B]
12      if (3*w)<width
13          B = vicsek_matrix(B, width)
14      end
15      return B
16  end
17  #
18  #--- Plot ---
19  #
20  #
21  (mat = vicsek_matrix(fill(1, 1, 1), 27)) |> size
22  GR.imshow(mat)
23  #
24  #-----
25  #--- Similarity Dimension ---
26  #
27  #
28  mat2 = vicsek_matrix(fill(1, 1, 1), 1000)

29  l2    = sum(mat2)
30  size2 = size(mat2)[1]
31  #
32  mat1 = vicsek_matrix(fill(1, 1, 1), 500)
33  l1    = sum(mat1)
34  size1 = size(mat1)[1]
35  #
36  #
37  julia> log(l2/l1)/log(size2/size1)
38  1.4649735207179269
39  julia> log(5)/log(3)
40  1.4649735207179269

```

Listing 1: Generating the Vicsek Fractal (shown in figure 7) and measuring the dimension using *Julia*, the measured dimension is consistent with the self similarity dimension shown in (18)

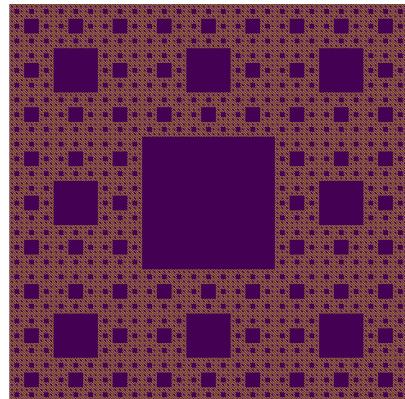


Figure 8: Sierpinski's Carpet, produced by listing .

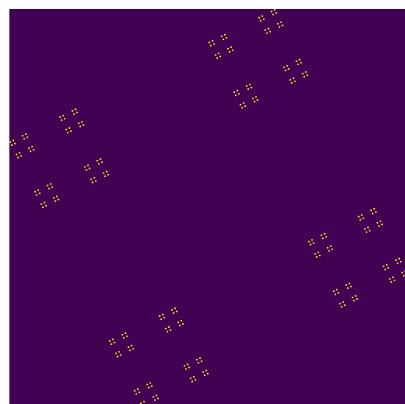


Figure 9: A shape this is similar, but different to, the *Cantor Dust*, the dimension of this fractal is 1, produced by listing 22.

Chaos Game The chaos game is a technique that can generate fractals, one of the advantages of this approach is that it can provide an estimate of the theoretical measure of a fractal without needing to iterate a function many times. The technique involves marking 3 points of an equilateral triangle and marking an arbitrary point, select one of these 3 points randomly with a uniform probability and create a new point halfway between the previous point and this point, repeat this process for as many points of detail are desired for the image.

This can be visualised by mapping the co-ordinates of an equilateral triangle to a cartesian plane:

- $A (0, 0)$
- $B (0, 1)$
- $C (0.5, \sin(\frac{\pi}{3}))$

The mean value of the x, y values for these co-ordinates is equal to the halfway point and using this the chaos game can be implemented as a program and visualised by plotting each point on a scatter plot. This is implemented in **R** in listing 2 and the output is shown in figure 10.

To measure the fractal dimension of this could be done by mapping the cartesian plane back to a matrix and taking the same approach as previous fractals presented, this however was not implemented, due to time constraints, the dimension was however measured using the method discussed at §2.4.1.

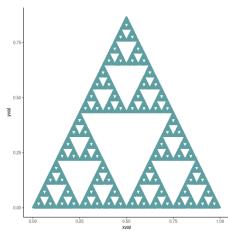


Figure 10: Sierpinski's Triangle created using the *Chaos Game* in listing 2.

Pascals Triangle

RYAN

The even and odd values in *Pascal's Triangle* demonstrate the same pattern as the *Sierpinski Triangle* this is discussed in greater detail in §2.4.3, implementing this to produce the sierpinski triangle is very simple, it is however significantly more resource intensive, even in *Julia* than using the chaos game and the the measured dimension converges to the self similar dimension very slowly.

The fractal produced is composed of right angle triangles, as opposed to equilateral triangles but interestingly the measured dimension is still the same as an equilateral *Sierpinski's Triangle*, it does however converge to this value slowly.

```
1 library(ggplot2)
2 # Parameters
3 n <- 50000
4 df <- data.frame("xval"=1:n, "yval"=1:n)
5 # Constants
6 x <- c(runif(1), runif(1))
7 A <- c(0, 0)
8 B <- c(1, 0)
9 C <- c(0.5, sin(pi/3))
10 # Loop
11 for (i in 1:n) {
12   dice = sample(1:3, 1)
13   if (dice == 1) {
14     x <- (x + A)/2
15     df[i,] <- x
16   } else if (dice == 2) {
17     x <- (x + B)/2
18     df[i,] <- x
19   } else {
20     x <- (x + C)/2
21     df[i,] <- x
22   }
23 }
24 # Plot
25 ggplot(df, aes(x = xval, y = yval)) +
26   geom_point(size = 1, col = "cadet blue") +
27   theme_classic()
```

Listing 2: R code to construct Sierpinski's triangle through the Chaos Game, shown in figure 10.

```

1  function pascal(n)
2      mat = [isodd(binomial(BigInt(j+i),BigInt(i))) for i in 0:n, j in 0:n]
3      return mat
4  end
5  GR.imshow(pascal(999))
6  GR.savefig("../Report/media/pascal-sierpinsky-triangle.png")
7
8  #-----
9  #-- Calculate Dimension --
10 #
11
12 mat2 = pascal(300)
13 l2   = sum(mat2)
14 size2 = size(mat2)[1]
15 mat1 = pascal(200)
16 l1   = sum(mat1)
17 size1 = size(mat1)[1]
18 log(l2/l1)/log(size2/size1)
19 # https://en.wikipedia.org/wiki/Sierpi%C5%84ski\_triangle
20 log(3)/log(2)
21 #
22 #
23 julia> log(l2/l1)/log(size2/size1)
24 1.8177195595512954
25 julia> log(3)/log(2)
26 1.5849625007211563

```

Listing 3: Julia code demonstrating Sierpinski's triangle, this converges to the self similar dimension very slowly, using the ratio between a 300^2 and 200^2 matrix gave the correct answer to 2 decimal places, using a 300^2 and 200^2 matrix produced a value far off as shown.

2.4.2 Turtle

Some Fractals cannot be well explained by using matrices or the chaos game, Turtle graphics are a programmatic way to draw a pen across a screen, these are implemented in *Julia* using the *Luxor* package [23].

We were unfortunately unable to implement a strategy to measure the dimension of such fractals, one such approach that looked promising but did not return consistent results was to export the generated image to a PNG and then import that file as a matrix using the *Python Pillow Library* [42] or the *Julia Images* library [24], when this was unsuccessful we also experimented with *ImageMagick* [31], *AstroPy* [1] and *JuliaAstro* [22]. Unfortunately the values returned by this approach were inconsistent and further investigation into this method is required.

The koch snowflake can be implemented by recursively calling a function that

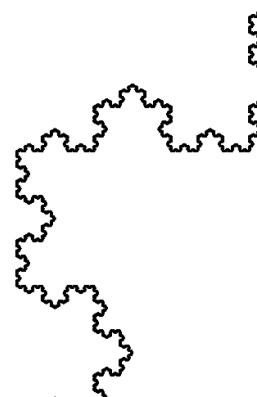


Figure 12: Portion of the Koch Snowflake Produced by the Turtle graphics in listing 4

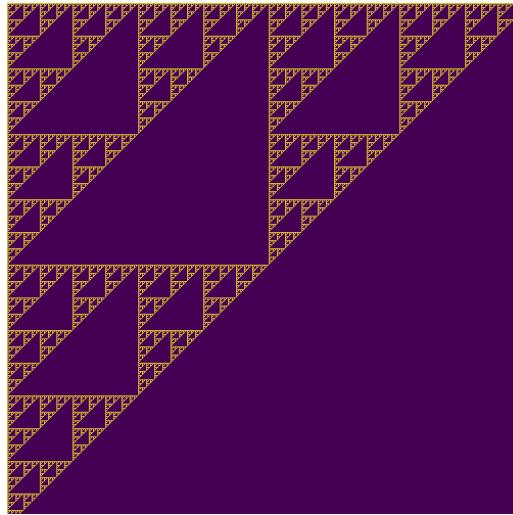


Figure 11: Sierpinski's triangle generated

draws the first level of a koch curve, if this function decrements a provided level and is defined to call itself for each arm of the curve unless the level has reached zero it will produce a koch snowflake at the specified level, this is implemented in listing 4 and shown in figure 12.

The dragon curve is slightly more complicated and can be implemented by two separate functions, one to turn and trigger a motion and the other to control in which direction to turn, this is implemented in *Julia* in listing 5 and shown in figure .

```

1  using Shapefile
2  using Luxor
3  using Pkg
4
5  #-----
6  #--- Round Snowflake Working
7  #
8  function snowflake(length, level, a)
9    if level == 0
10      Forward(a, length)
11      Circle(a, 1)
12      return
13  end
14  length = length/9
15  snowflake(length, level-1, a)
16  Turn(a, -60)
17  snowflake(length, level-1, a)
18  Turn(a, 2*60)
19  snowflake(length, level-1, a)
20  Turn(a, -60)
21  snowflake(length, level-1, a)
22 end
23
24 a = Turtle()
25 @png begin
26 for i in 1:3
27   levels = 9
28   Pendown(a)
29   snowflake(8^(levels-1), levels, a)
30   Turn(a, 120)
31 end
32 end 600 600 "snowCurve.png"

```

Listing 4: Generate a Koch Snowflake using a Turtle Diagram

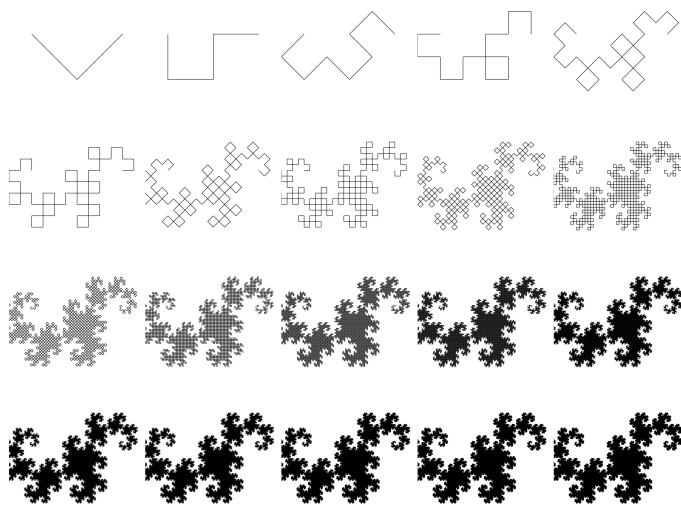


Figure 13: Progression of the Dragon Curve, this is known as a space filling curve [41, p. 350] which is a curve with a range that contains the entire 2-dimensional unit square [50], it has a dimension of two. For some historical background on the curve on the origins of this curve see [48].

```

1  using Shapefile
2  using Luxor
3
4  #-----
5  #--- Dragon ---
6  #-----
7  # Define the Parent Function
8  function dragon(®, order, length)
9      print(" ") # Don't remove this or code breaks, I don't know why?
10     Turn(®, order*45)
11     dragon_iterate(®, order, length, 1)
12 end
13 # Define the Helper Function
14 function dragon_iterate(®, order, length, sign)
15     if order==0
16         Forward(®, length)
17     else
18         rootHalf = sqrt(0.5)
19         dragon_iterate(®, order -1, length*rootHalf, 1)
20         Turn(®, sign * -90)
21         dragon_iterate(®, order -1, length*rootHalf, -1)
22     end
23 end
24 # Draw the Image
25 @png begin
26     ® = Turtle()
27     # Start from left to centre result
28     Turn(®, 180)
29     Penup(®)

30     Forward(®, 200)
31     Pendown(®)
32     Turn(®, 180)
33     # Create the Output
34     dragon(®, 15, 400)
35 end 1000 1000
36
37 # Create many images
38 ;mkdir /tmp/dragon
39 for i in range(1, 15)
40     name = string("/tmp/dragon/d", lpad(d, 5, "0"), ".png")
41     @png begin
42         ® = Turtle()
43         # Start from left to centre result
44         Turn(®, 180)
45         Penup(®)
46         Forward(®, 200)
47         Pendown(®)
48         Turn(®, 180)
49         # Create the Output
50         dragon(®, 15, 400)
51     end 1000 1000 name
52 end
53 montage -geometry 1000x1000 *.png dragon.png

```

Listing 5: Julia code to produce the Dragon Curve shown in figure 13

2.4.3 Pascals Triangle and Sierpinski's Triangle

JAMES

Motivation Over many centuries, mathematicians have been able to produce a range of patterns from Pascal's triangle. One of which is relevant to the emergence of Sierpinski's triangle. To construct Pascal's triangle it begins with a 1 in the 0th (top) row, then each row underneath is made up of the sum of the numbers directly above it, see figure 14. Alternatively, the n^{th} row and k^{th} column can be written in combinatorics form, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

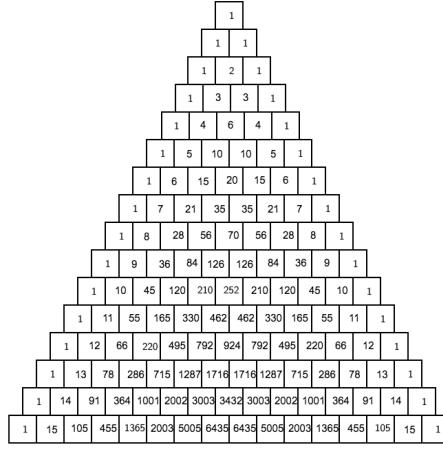
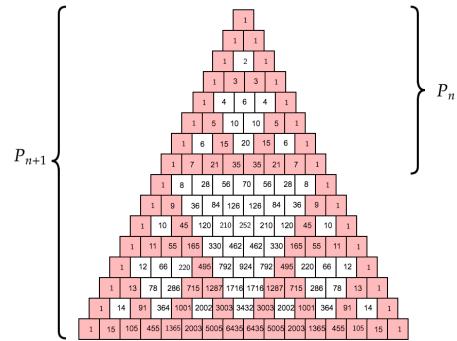


Figure 14: Pascal's triangle

The connection As mentioned before there is one pattern that produces the Sierpinski triangle, namely highlighting all odd numbers in Pascal's triangle. This is equivalent to considering all the numbers in the triangle modulo 2, shown in figure 2.4.3.



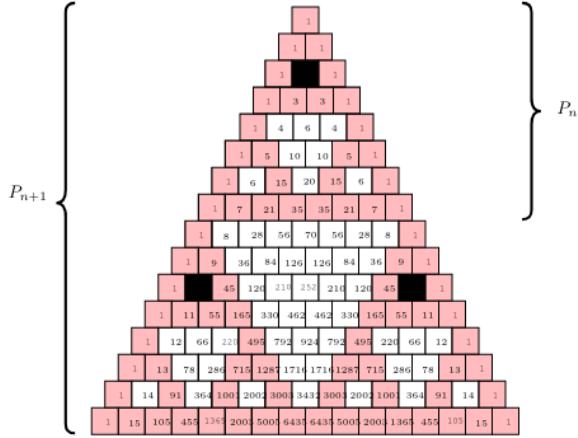


Figure 15: The black squares represent one example of a position on Pascal's triangle that are equivalent modulo 2

In figure 2.4.3, we can observe that all the highlighted odd numbers begin to form the Sierpinski triangle. Note that this is not the complete Sierpinski's triangle, that would require an infinite number of iterations. Now, we also notice that there are three identical Sierpinski triangles within the larger triangle, each containing the same value modulo 2, at each corresponding row and column.

To prove this, we need to split the triangle into two parts, P_n denoting the first 2^n rows, i.e. the top “Sierpinski triangle” in figure 2.4.3 and P_{n+1} representing the entire triangle. We must show that any chosen square in P_n is equal to the corresponding row and column in the lower two triangles of P_{n+1} , shown in figure 15. This requires an identity that allows us to work with combinations in modulo 2, namely Lucas' Theorem.

Lucas' Theorem Let $n, k \geq 0$ and for some prime p , we get:

$$\binom{n}{k} = \prod_{i=0}^m \binom{n_i}{k_i} \pmod{p} \quad (20)$$

where,

$$\begin{aligned} n &= n_m p^m + n_{m-1} p^{m-1} + \cdots + n_1 p + n_0, \\ k &= k_m p^m + k_{m-1} p^{m-1} + \cdots + k_1 p + k_0 \end{aligned}$$

are the expansions in radix p ¹⁵. This uses the convention that $\binom{n}{k} = 0$ if $k < n$

¹⁵Radix refers to a numerical system which uses some number of digits. Since we are working in modulo 2 for Pascal's triangle, we are only concerned with the numbers 0 or 1, i.e. a radix 2 or a binary numeric system.

Take some arbitrary row r and column c in the triangle P_n . If we add 2^n rows to r , we will reach the equivalent row and column in the lower left triangle of P_{n+1} , since there are 2^n rows in P_n . In the same way, if we add 2^n columns to c we reach the equivalent row and column in the lower right triangle of P_{n+1} , leaving us with:

$$\begin{aligned} \text{Top Triangle: } & \binom{r}{c} \\ \text{Bottom-left Triangle: } & \binom{r + 2^n}{c} \\ \text{Bottom-right Triangle: } & \binom{r + 2^n}{c + 2^n} \end{aligned}$$

Using Lucas' theorem, we can prove that the above statements are equivalent. We can rewrite r and c in base 2 notation as follows:

$$\begin{aligned} r &= r_i 2^i + r_{i-1} 2^{i-1} + \cdots + r_1 2 + r_0 = [r_i r_{i-1} \cdots r_1 r_0]_2 \\ c &= c_i 2^i + c_{i-1} 2^{i-1} + \cdots + c_1 2 + c_0 = [c_i c_{i-1} \cdots c_1 c_0]_2 \end{aligned}$$

$$\begin{aligned} \binom{2^n + r}{c} \pmod{2} &= \binom{1r_{i-1}r_{i-2}\cdots r_0}{0c_{i-1}c_{i-2}\cdots c_0} \pmod{2} \\ &= \binom{1}{0} \binom{r_{i-1}}{c_{i-1}} \binom{r_{i-2}}{c_{i-2}} \cdots \binom{r_0}{c_0} \pmod{2} \\ &= \binom{r_{i-1}}{c_{i-1}} \binom{r_{i-2}}{c_{i-2}} \cdots \binom{r_0}{c_0} \pmod{2} \\ &= \binom{r}{c} \pmod{2} \end{aligned}$$

$$\begin{aligned} \binom{2^n + r}{2^n + c} \pmod{2} &= \binom{1r_{i-1}r_{i-2}\cdots r_0}{1c_{i-1}c_{i-2}\cdots c_0} \pmod{2} \\ &= \binom{1}{1} \binom{r_{i-1}}{c_{i-1}} \binom{r_{i-2}}{c_{i-2}} \cdots \binom{r_0}{c_0} \pmod{2} \\ &= \binom{r_{i-1}}{c_{i-1}} \binom{r_{i-2}}{c_{i-2}} \cdots \binom{r_0}{c_0} \pmod{2} \\ &= \binom{r}{c} \pmod{2} \end{aligned}$$

Thus, $\binom{r}{c} = \binom{2^n+r}{c} = \binom{2^n+r}{2^n+c} \pmod{2}$, which concludes the proof

2.5 Fractal Dimensions Sans Self Similarity

RYAN

2.5.1 Calculating the Dimension of Julia Sets

A value on the complex plane can be associated with the julia set by iterating that value against a function of the form $z \rightarrow z^2 + \alpha + i\beta$ and measuring whether or not that value diverges or converges. This process is demonstrated in listing 6.

By associating each value on the complex plane with an element of a matrix an image of this pattern may be produced, by considering only values on the boundary between convergent and divergent an outline may be produced see for example figure 16, this outline is known as the *Julia Set*.

In order to measure the dimension of the *Julia Set* it is necessary to generate a representation of the fractal at two scales, compare them and then and then measure the corresponding dimension value as was done previously. The julia set is a non self-similar fractal and so it is not immediately clear whether or not the dimension will be constant at all scales, to determine whether or not the dimension is constant at various scales it can be convenient to plot the log transformed scaling factor and measures and inspect whether or not the points form a linear relationship, the slope of such a relationship will be the dimension [51, p. 30].

To implement this all the functions necessary to build the fractals were placed into a separate script `julia-set-functions.jl` which is shown in § and this script was included into a working script `julia-set-dimensions.jl` by using the following line:

```
1 @time include("./Julia-Set-Dimensions-functions.jl")
```

The julia set is defined as the boundary between values on the complex plane that converge and diverge under iteration of $z \leftarrow z^2 + a + ib$ to consider only the boundary of values, the `juliaSet` function shown in listing 6 can be modified to return only the value of 1 or 0 as opposed to the time taken to cross the threshold of divergence¹⁶ and then it can be determined whether or not a point is a boundary by considering whether or not the sum of all elements within the immediate neighbourhood of the element is greater than 1. This is shown in the `outline` function which is defined in § 5.

Comparing squared values rather than using `abs()` improved the performance of this function by about two fold.

So I run the code shown in listing ?? which calls a file `./Julia-Set-Dimensions-functions.jl` which is shown in § 5

which returns the values shown in table .

```
1 @time include("./Julia-Set-Dimensions-functions.jl")
2 #####
3 ###### Investigate Plot #####
4 #####
5 #####
6 test_mat = make_picture(800,800, z -> z^2 + -0.123+0.745*im)
7 #####
```

¹⁶if `abs(z) ≥ 2` the point is assured to diverge under iteration

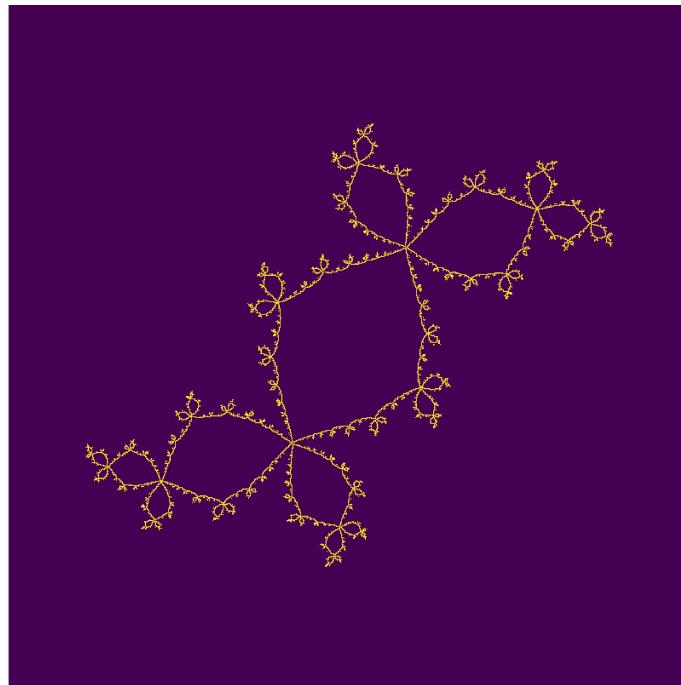


Figure 16: Image of the Douady Rabbit, the julia set corresponding to the iteration of $z \leftarrow z^2 - 0.123 + 0.745i$ produced by *Julia*

```

1  #!/bin/julia
2  function juliaSet(z, num, my_func, boolQ=true)
3      count = 1
4      # Iterate num times
5      while count ≤ num
6          # check for divergence
7          if real(z)^2+imag(z)^2 > 2^2
8              if (boolQ) return 0 else return Int(count) end
9          end
10         #iterate z
11         z = my_func(z) # + z
12
13         count=count+1
14     end
15     #if z hasn't diverged by the end
16     if(boolQ) return 1 else return Int(count) end
end

```

Listing 6: Function that returns how many iterations of a function of is necessary for a complex value to diverge, the julia set is concerned with the function $z \rightarrow z^2 + \alpha + i\beta$

```

8   #Inspect
9   GR.imshow(test_mat) # PyPlot uses interpolation = "None"
10  # Outline
11  test_mat = outline(test_mat)
12  #Inspect
13  GR.imshow(test_mat) # PyPlot uses interpolation = "None"
14  # GR.savefig("/home/ryan/Dropbox/Studies/2020Spring/QuantProject/Current/Python-Quant/Pr ]
15  ↪ oblems/fractal-dimensions/media/outline-Julia-set.png")
16
17  ## Return the perimeter
18  sum(test_mat)
19
20  # Take a measurement at a point
21
22  mat2 = outline(make_picture(9000,9000, f))
23  l2   = sum(mat2)
24  size2 = size(mat2)[1]
25  mat1 = outline(make_picture(10000,10000, f))
26  l1   = sum(mat1)
27  size1 = size(mat1)[1]
28  log(l2/l1)/log(size2/size1)
29  # https://en.wikipedia.org/wiki/Vicsek_fractal#Construction
30  # 1.3934 Douady Rabbit
31  #
32
33  # Take a measurement using LInear Regression
34  using CSV
35
36  @time data=scaleAndMeasure(900, 1000 , 4, f)
37  # CSV.read("./julia-set-dimensions.csv", data)
38  # data = CSV.read("./julia-set-dimensions.csv")
39  data.scale = [log(i) for i in data.scale]
40  data.mass  = [log(i) for i in data.mass]
41  mod   = lm(@formula(mass ~ scale), data)
42  p = Gadfly.plot(data, x=:scale, y=:mass, Geom.point)
43
44  print("the slope is $(round(coef(mod)[2], sigdigits=4))")
45  print(mod)
46  print("\n")
47  return mod
48
49  a = SharedArray{Float64}(10)
50  @distributed for i = 1:10
51    a[i] = i
52  end
53
54  #-----
55  # julia> return mod
56  # StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64, 1}}, GLM.DensePre ]
57  ↪ dChol{Float64,LinearAlgebra.Cholesky{Float64,Array{Float64,2}}}}, Array{Float64,2}}
58  #
59  # mass ~ 0 + scale
60  #
61  # Coefficients:
62  # _____
63  #      Coef.  Std. Error      t Pr(>|t|) Lower 95% Upper 95%
64  # _____
65  # scale  1.28358  0.000497296  2581.11     <1e-9    1.28199    1.28516
# _____

```

This converges very slowly and the code can take a very long time to run, and has a tendency to cause crashes, likely due to the large amounts of memory required, these

are things that could be improved, by running this code for scales from 9000 to 10000 and leaving it for an hour, the following table of values is returned:

scale	mass
500	4834.0
563	5754.0
625	6640.0
688	7584.0
750	8418.0
813	9550.0
875	10554.0
938	11710.0
1000	12744.0

Linear Regression can be performed against these values using **R**¹⁷

```

1 scale <- c(500, 563, 625, 688, 750, 813, 875, 938, 1000)
2 measure <- c( 4834, 5754, 6640, 7584, 8418, 9550, 10554, 11710, 12744)
3 data <- data.frame(scale, measure)
4
5 lm(log(measure) ~ 0 + log(scale), data)$coefficients # set 0 intercept
6
7 #-----
8 # 1.36720041333112

```

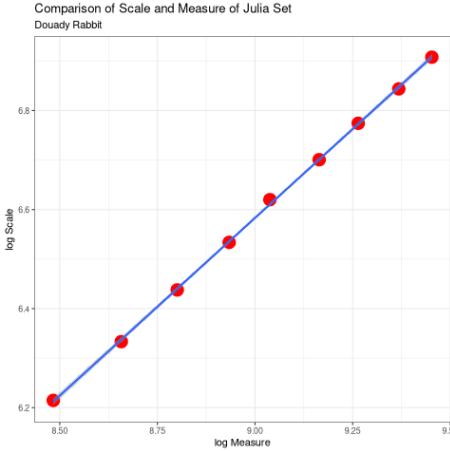
This shows that the value returned is 1.37 which is very close to the value of 1.39 which is reported in the literature [36], inspecting the behaviour of the log transformed scale and measure indicates that there is a very linear relationship between these variables and so even though this Julia set does not appear to demonstrate simple self-similarity, it appears to be a figure with a constant dimension across scales.

```

1 library(ggplot2)
2 ggplot(data, aes(x = log(measure), y = log(scale))) +
3   geom_point(size = 6, col = 'red') +
4   geom_smooth(method = 'lm') +
5   theme_bw() +
6   labs(x = "log Measure", y = "log Scale",
7        title = "Comparison of Scale and Measure of Julia Set", subtitle = "Douady Rabbit")
8

```

¹⁷The could just as well have been done inside Julia, R was chosen simply because ggplot produces very nice plots.



3 Connecting Fractals to Natural Processes

RYAN

In order to better understand the ways in which complex fractal patterns can emerge from simple processes, I have constructed a very simple process that leads to such a pattern.

Consider a fractal that begins simply with a square , imagine that this square is replicated, rotated and appended , then that shape, again is replicated rotated and appended , this process is illustrated in figure 18, and if perpetuated a pattern emerges, as shown in figure 17, this fractal can be generated by joining a joining two matrices together in a similar way to the pattern described, this is shown in listing 7.

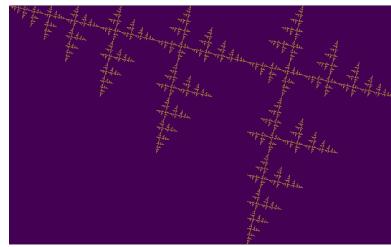
Figure 17: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well

The fractal produces an angle that is caused by the point where the longest and shortest points meet up, the length of these points actually follow the fibonacci numbers and so the angle can be given by the following relationship:

$$\theta = \tan^{-1} \left(\lim_{n \rightarrow \infty} \left(\frac{F_{n-2}}{F_n} \right) \right) \quad (21)$$

$$= \tan^{-1} \left(\lim_{n \rightarrow \infty} \left(\frac{F_{n-2}}{F_{n-1} + F_{n-2}} \right) \right) \quad (22)$$

$$= \tan^{-1} \left(\lim_{n \rightarrow \infty} \left(\frac{F_n}{F_{n+1} + F_n} \right) \right) \quad (23)$$



```

1   function matJoin(A, B)
2     function nrow(X)
3       return size(X)[1]
4     end
5     function ncol(X)
6       return size(X)[2]
7     end
8     emptymat = zeros(Bool, max(size(A)[1], size(B)[1]), sum(ncol(A) + ncol(B)) )
9     emptymat[1:nrow(A), 1:ncol(A)] = A
10    emptymat[1:nrow(B), (ncol(A)+1):ncol(emptymat)] = B
11    return emptymat
12  end
13
14  function mywalk(B, n)
15    for i in 1:n
16      B = matJoin(B, rotl90(B));
17    end
18
19    return B
20
21 GR.imshow(mywalk([1, 1], 9))

```

Listing 7: Generate the fractal described in §3 and shown in figure

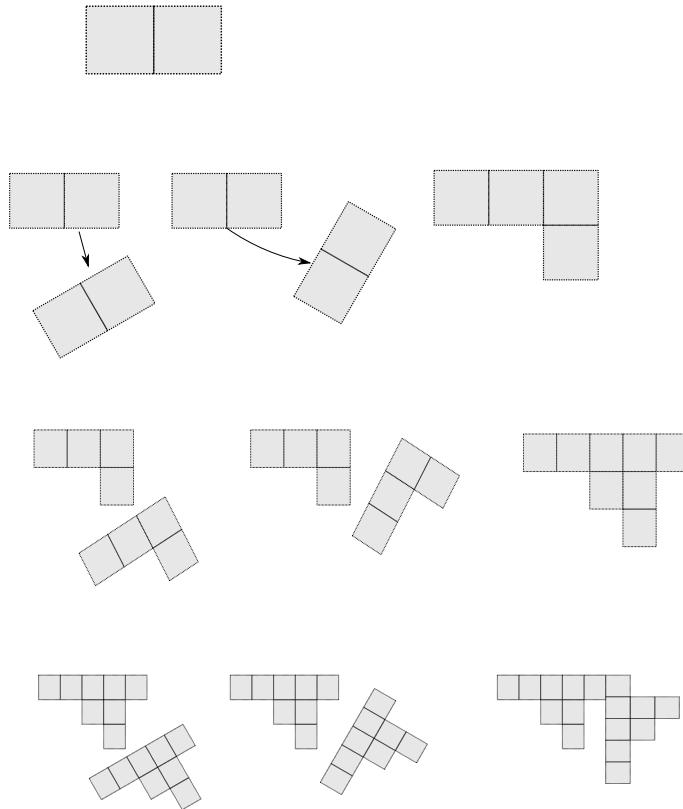


Figure 18: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well

Because $\lim_{n \rightarrow \infty} \left(\frac{F_n}{F_{n-1}} \right) = \varphi \approx 1.618$ we can substitute the values:

$$= \tan^{-1}(\varphi(\varphi+1)) \quad (24)$$

$$= \tan^{-1}\left(\frac{\psi}{\varphi+1}\right) \quad (25)$$

$$\approx 0.231^\circ \quad (26)$$

this is illustrated in figures 20 and 21. This angle can be contrasted to the golden angle visualised in figure 19.

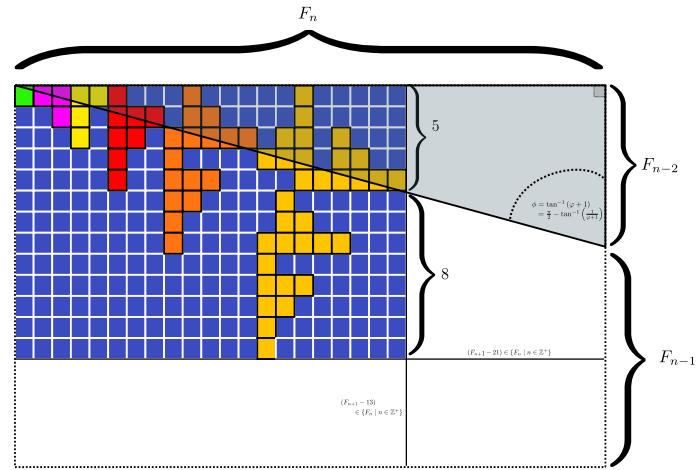


Figure 20: TODO

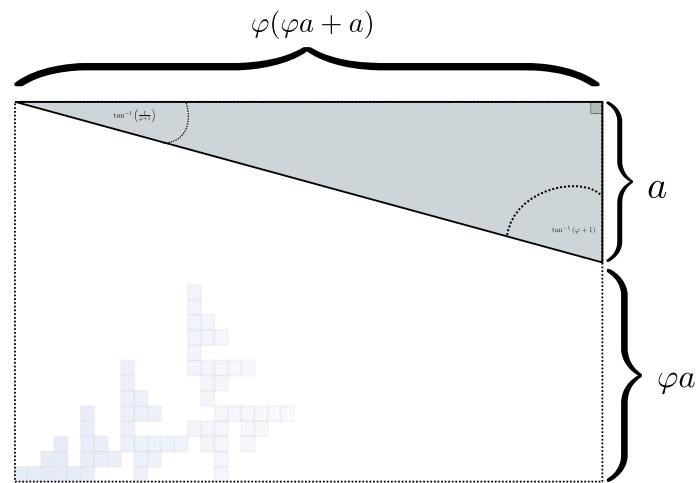


Figure 21: TODO

3.1 Prove Fibonacci using Monotone Convergence Theorem

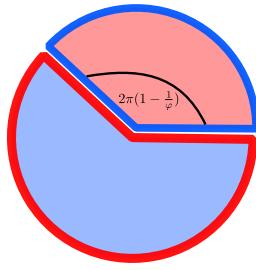


Figure 19: Diagram of the Golden Angle, an angle formed by having a ratio of edges equal to the golden ratio.

Each time this fractal is iterated the measure (i.e. the number of boxes) is doubled, but the scale of the fractal is increased by a ratio of the width and height, because the boxes add up this ratio will be a ratio of fibonacci numbers which is shown to be equal to the golden ratio ϕ in §4.5, hence the dimension of this fractal is given by:

$$\log_{\phi}(2) \approx 1.44042 \quad (27)$$

This value is consistent with measurements performed using *Julia* in listing 8.

```

1  using DataFrames
2  function returnDim()
3      mat2 = mywalk(fill(1, 1, 1), 10)
4      l2   = sum(mat2)
5      size2 = size(mat2)[1]
6      mat1 = mywalk(fill(1, 1, 1), 11)
7      l1   = sum(mat1)
8      size1 = size(mat1)[1]
9      df = DataFrame()
10     df.measure = [log(l2/l1)/log(size2/size1)]
11     df.actual  = [log(2)/log(1.618) ]
12     return df
13 end
14
15 returnDim()
16
17 #-----
18 # 1x2 DataFrame
19 # | Row | measure | actual |
20 # |    | Float64 | Float64 |
21 # |---|---|---|
22 # | 1 | 1.44052 | 1.44048 |

```

Listing 8: Measure the fractal dimension of the fractal described in §3

3.1 Prove Fibonacci using Monotone Convergence Theorem

Consider the series:

$$G_n = \frac{F_n}{F_{n-1}}$$

Such that:

$$F_n = F_{n-1} + F_{n-2}; \quad F_1 = F_2 = 1$$

3.1.1 Show that the Series is Monotone

$$\begin{aligned} F_n &> 0 \\ 0 &< F_n \\ \implies 0 &< F_{n-2} + F_{n-1} \quad \forall n > 2 \\ F_{n-2} &< F_{n-1} \\ \implies F_n &< F_{n+1} \end{aligned}$$

$$\begin{aligned} F_n &> 0 \\ 0 &< F_n \\ \implies 0 &< F_{n-2} + F_{n-1} \quad \forall n > 2 \\ F_{n-2} &< F_{n-1} \\ \implies F_n &< F_{n+1} \end{aligned}$$

3.1.2 Show that the Series is Bounded

3.1.3 Find the Limit

$$\begin{aligned} G &= \frac{F_n + F_{n+1}}{F_{n+1}} \\ &= 1 + \frac{F_{n-1}}{F_n} \end{aligned}$$

Recall that $F_n > 0 \forall n$

$$\begin{aligned} &= 1 + \frac{1}{|G|} \\ \implies 0 &= G^2 - G + 1; \quad G > 0 \\ \implies G &= \varphi = \frac{\sqrt{5} - 1}{2} \quad \square \end{aligned}$$

3.1.4 Comments

The Fibonacci sequence is quite unique, observe that:

This can be rearranged to show that the Fibonacci sequence is itself when shifted in either direction, it is the sequence that does not change during recursion.

$$F_{n+1} - F_n = F_{n-1} \quad \forall n > 1$$

This is analogous to how e^x doesn't change under differentiation:

$$\frac{d}{dx} (e^x) \dots$$

or how 0 is the additive identity and it shows why generating functions are so useful.

Observe also that

$$\lim_{n \rightarrow \infty} \left[\frac{F_n}{F_{n-1}} \right] = \varphi$$

$$\lim_{n \rightarrow \infty} \left[\frac{F_n}{F_{n-1}} \right] = \psi$$

$$\varphi - \psi = 1$$

$$\varphi \times \psi = 1$$

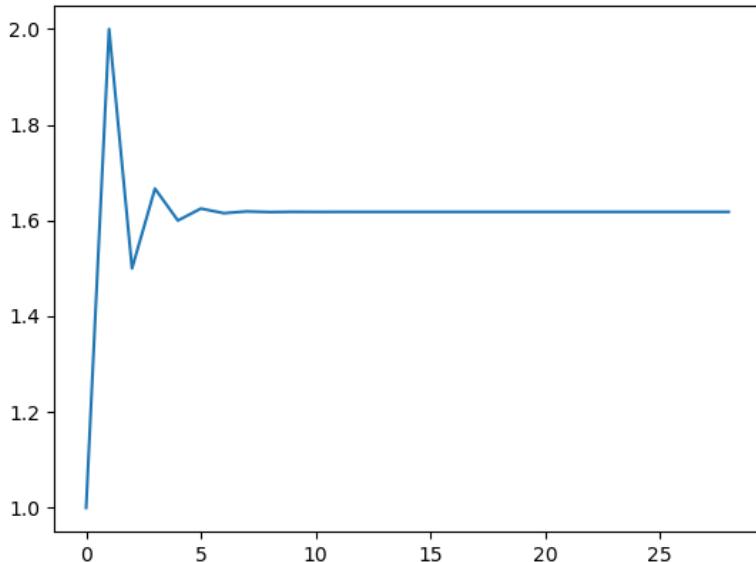
$$\frac{\psi}{\varphi} = \frac{1}{\varphi^2} = \frac{1}{1-\varphi} = \frac{1}{2-\varphi} = \frac{2}{3-\sqrt{5}}$$

3.1.5 Python

```

1  #+begin_src python
2  import matplotlib.pyplot as plt
3  import sympy
4
5  plt.plot([sympy.N(sympy.fibonacci(n+1)/sympy.fibonacci(n)) for n in range(1, 30)])
6  plt.savefig("./a.png")

```



4 The Fibonacci Sequence

The Fibonacci Sequence occurs in my example from the §3, let's investigate it

4.1 Introduction

RYAN

The *Fibonacci Sequence* and *Golden Ratio* share a deep connection¹⁸ and occur in patterns observed in nature very frequently (see [46, 3, 37, 38, 25, 44]), an example of such an occurrence is discussed in section 4.5.1.

In this section we lay out a strategy to find an analytic solution to the *Fibonacci Sequence* by relating it to a continuous series and generalise this approach to any homogenous linear recurrence relation.

This details some open mathematical work for the project and our hope is that by identifying relationships between discrete and continuous systems generally we will be able to draw insights with regard to the occurrence of patterns related to the *Fibonacci Sequence* and *Golden Ratio* in nature.

4.2 Computational Approach

RYAN

Given that much of our work will involve computational analysis and simulation we begin with a strategy to solve the sequence computationally.

The *Fibonacci Numbers* are given by:

$$F_n = F_{n-1} + F_{n-2} \quad (28)$$

This type of recursive relation can be expressed in *Python* by using recursion, as shown in listing 9, however using this function will reveal that it is extraordinarily slow, as shown in listing 10, this is because the results of the function are not cached and every time the function is called every value is recalculated¹⁹, meaning that the workload scales in exponential as opposed to polynomial time.

The `functools` library for python includes the `@functools.lru_cache` decorator which will modify a defined function to cache results in memory [13], this means that the recursive function will only need to calculate each result once and it will hence scale in polynomial time, this is implemented in listing 11.

```

1  def rec_fib(k):
2      if type(k) is not int:
3          print("Error: Require integer values")
4          return 0
5      elif k == 0:
6          return 0
7      elif k <= 2:
8          return 1
9      return rec_fib(k-1) + rec_fib(k-2)

```

Listing 9: Defining the *Fibonacci Sequence* (28) using Recursion

¹⁸See section

¹⁹Dr. Hazrat mentions something similar in his book with respect to *Mathematica®* [18, Ch. 13]

4.3 Exponential Generating Functions

```

1 start = time.time()
2
3 print(str(round(time.time() - start, 3)) + "seconds")
4
5 ## 2.245seconds

```

Listing 10: Using the function from listing 9 is quite slow.

```

1 start = time.time()
2 rec_fib(6000)
3 print(str(round(time.time() - start, 9)) + "seconds")
4
5 ## 8.3923e-05seconds

```

Restructuring the problem to use iteration will allow for even greater performance as demonstrated by finding F_{10^6} in listing 12. Using a compiled language such as *Julia* however would be thousands of times faster still, as demonstrated in listing 13.

In this case however an analytic solution can be found by relating discrete mathematical problems to continuous ones as discussed below at section .

4.3 Exponential Generating Functions

Motivation

RYAN Consider the *Fibonacci Sequence* from (28):

$$\begin{aligned} a_n &= a_{n-1} + a_{n-2} \\ \iff a_{n+2} &= a_{n+1} + a_n \end{aligned} \tag{29}$$

from observation, this appears similar in structure to the following *ordinary differential equation*, which would be fairly easy to deal with:

$$f''(x) - f'(x) - f(x) = 0$$

By ODE Theory we have $y \propto e^{m_i x}$, $i = 1, 2$:

$$f(x) = e^{mx} = \sum_{n=0}^{\infty} \left[r^m \frac{x^n}{n!} \right]$$

So using some sort of a transformation involving a power series may help to relate the discrete problem back to a continuous one.

4 THE FIBONACCI SEQUENCE

```
1  from functools import lru_cache
2
3  @lru_cache(maxsize=9999)
4
5  def rec_fib(k):
6
7      if type(k) is not int:
8
9          print("Error: Require Integer Values")
10
11         return 0
12
13     elif k == 0:
14
15         return 0
16
17     elif k <= 2:
18
19         return 1
20
21     return rec_fib(k-1) + rec_fib(k-2)
22
23
24
25     start = time.time()
26
27     rec_fib(35)
28
29     print(str(round(time.time() - start, 3)) + "seconds")
30
31     ## 0.0seconds
```

Listing 11: Caching the results of the function previously defined 10

4.3 Exponential Generating Functions

```

1   def my_it_fib(k):
2       if k == 0:
3           return k
4       elif type(k) is not int:
5           print("ERROR: Integer Required")
6           return 0
7       # Hence k must be a positive integer
8
9       i = 1
10      n1 = 1
11      n2 = 1
12
13      # if k <=2:
14      #     return 1
15
16      while i < k:
17          no = n1
18          n1 = n2
19          n2 = no + n1
20          i = i + 1
21      return (n1)
22
23      start = time.time()
24      my_it_fib(10**6)
25      print(str(round(time.time() - start, 9)) + "seconds")
26
27      ## 6.975890398seconds

```

Listing 12: Using Iteration to Solve the Fibonacci Sequence

Example RYAN Consider using the following generating function, (proof of the generating function derivative as in (31) and (32) is provided in section 4.3)

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \cdot \frac{x^n}{n!} \right] \quad (30)$$

$$\implies f'(x) = \sum_{n=0}^{\infty} \left[a_{n+1} \cdot \frac{x^n}{n!} \right] \quad (31)$$

$$\implies f''(x) = \sum_{n=0}^{\infty} \left[a_{n+2} \cdot \frac{x^n}{n!} \right] \quad (32)$$

So the Fibonacci recursive relation from (29) could be expressed :

$$\begin{aligned} a_{n+2} &= a_{n+1} + a_n \\ \frac{x^n}{n!} a_{n+2} &= \frac{x^n}{n!} (a_{n+1} + a_n) \\ \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} a_{n+2} \right] &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} a_{n+1} \right] + \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} a_n \right] \end{aligned}$$

And hence by applying (30), (31) and (32):

4 THE FIBONACCI SEQUENCE

```
1  function my_it_fib(k)
2      if k == 0
3          return k
4      elseif typeof(k) != Int
5          print("ERROR: Integer Required")
6          return 0
7      end
8      # Hence k must be a positive integer
9
10     i = 1
11     n1 = 1
12     n2 = 1
13
14     # if k <=2:
15     #     return 1
16     while i < k
17         no = n1
18         n1 = n2
19         n2 = no + n2
20         i = i + 1
21     end
22     return (n1)
23
24
25 @time my_it_fib(10^6)
26
27 ## my_it_fib (generic function with 1 method)
28 ## 0.000450 seconds
```

Listing 13: Using Julia with an iterative approach to solve the 1 millionth fibonacci number

$$f''(x) = f'(x) + f(x) \quad (33)$$

Using the theory of higher order linear differential equations with constant coefficients it can be shown:

$$f(x) = c_1 \cdot \exp\left[\left(\frac{1-\sqrt{5}}{2}\right)x\right] + c_2 \cdot \exp\left[\left(\frac{1+\sqrt{5}}{2}\right)x\right]$$

By equating this to the power series:

$$f(x) = \sum_{n=0}^{\infty} \left[\left(c_1 \left(\frac{1-\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1+\sqrt{5}}{2}\right)^n\right) \cdot \frac{x^n}{n!} \right]$$

Now given that:

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right]$$

We can conclude that:

$$a_n = c_1 \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n + c_2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n$$

By applying the initial conditions:

$$\begin{aligned} a_0 &= c_1 + c_2 \implies c_1 = -c_2 \\ a_1 &= c_1 \left(\frac{1+\sqrt{5}}{2}\right) - c_1 \left(\frac{1-\sqrt{5}}{2}\right) \implies c_1 = \frac{1}{\sqrt{5}} \\ &\therefore c_1 = \frac{1}{\sqrt{5}}, c_2 = -\frac{1}{\sqrt{5}} \end{aligned}$$

And so finally we have the solution to the *Fibonacci Sequence* 29:

$$\begin{aligned} a_n &= \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right] \\ &= \frac{\varphi^n - \psi^n}{\sqrt{5}} \\ &= \frac{\varphi^n - \psi^n}{\varphi - \psi} \end{aligned} \quad (34)$$

where:

- $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.61\dots$
- $\psi = 1 - \varphi = \frac{1-\sqrt{5}}{2} \approx 0.61\dots$

Derivative of the Exponential Generating Function

BaseRYAN Differentiating the exponential generating function has the effect of shifting the sequence once to the left: [27]

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \quad (35)$$

$$\begin{aligned} f'(x) &= \frac{d}{dx} \left(\sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \right) \\ &= \frac{d}{dx} \left(a_0 \frac{x^0}{0!} + a_1 \frac{x^1}{1!} + a_2 \frac{x^2}{2!} + a_3 \frac{x^3}{3!} + \dots \frac{x^k}{k!} \right) \\ &= \sum_{n=0}^{\infty} \left[\frac{d}{dx} \left(a_n \frac{x^n}{n!} \right) \right] \\ &= \sum_{n=0}^{\infty} \left[\frac{a_n}{(n-1)!} x^{n-1} \right] \\ \implies f'(x) &= \sum_{n=1}^{\infty} \left[\frac{x^n}{n!} a_{n+1} \right] \end{aligned} \quad (36)$$

Bridge JAMES This can be shown for all derivatives by way of induction, for

$$f^{(k)}(x) = \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^n}{n!} \quad \text{for } k \geq 0 \quad (37)$$

Assume that $f^{(k)}(x) = \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^n}{n!}$

Using this assumption, prove for the next element $k+1$

We need $f^{(k+1)}(x) = \sum_{n=k+1}^{\infty} \frac{a_{n+k+1} \cdot x^n}{n!}$

$$\begin{aligned} \text{LHS} &= f^{(k+1)}(x) \\ &= \frac{d}{dx} (f^{(k)}(x)) \\ &= \frac{d}{dx} \left(\sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^n}{n!} \right) \quad \text{by assumption} \\ &= \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot n \cdot x^{n-1}}{n!} \\ &= \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^{n-1}}{(n-1)!} \\ &= \sum_{n=k+1}^{\infty} \frac{a_{n+k+1} \cdot x^n}{n!} \\ &= \text{RHS} \end{aligned}$$

4.3 Exponential Generating Functions

Therefore, by mathematical induction $f^{(k)}(x) = \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^n}{n!}$ for $k \geq 0$

Furthermore, if the first derivative of the exponential generating function shown in (36) shifts the sequence across, then every derivative thereafter does so as well.

Homogeneous Proof

RYAN:JAMES An equation of the form:

$$\sum_{i=0}^n [c_i \cdot f^{(i)}(x)] = 0 \quad (38)$$

is said to be a homogenous linear ODE: [55, Ch. 2]

Linear because the equation is linear with respect to $f(x)$

Ordinary because there are no partial derivatives (e.g. $\frac{\partial}{\partial x}(f(x))$)

Differential because the derivates of the function are concerned

Homogenous because the **RHS** is 0

- A non-homogeneous equation would have a non-zero RHS

There will be k solutions to a k^{th} order linear ODE, each may be summed to produce a superposition which will also be a solution to the equation, [55, Ch. 4] this will be considered as the desired complete solution (and this will be shown to be the only solution for the recurrence relation (39)). These k solutions will be in one of two forms:

1. $f(x) = c_i \cdot e^{m_i x}$
2. $f(x) = c_i \cdot x^j \cdot e^{m_i x}$

where:

- $\sum_{i=0}^k [c_i m^{k-i}] = 0$
 - This is referred to as the characteristic equation of the recurrence relation or ODE [28]
 - $\exists i, j \in \mathbb{Z}^+ \cap [0, k]$
 - These are often referred to as repeated roots [28, 56] with a multiplicity corresponding to the number of repetitions of that root [39, §3.2]

Unique Roots of Characteristic Equation

RYAN

1. Example An example of a recurrence relation with all unique roots is the fibonacci sequence, as described in section 4.3.

2. Proof Consider the linear recurrence relation (39):

$$\sum_{i=0}^n [c_i \cdot a_i] = 0, \quad \exists c \in \mathbb{R}, \quad \forall i < k \in \mathbb{Z}^+$$

This implies:

$$\sum_{n=0}^{\infty} \left[\sum_{i=0}^k \left[\frac{x^n}{n!} c_i a_n \right] \right] = 0 \quad (39)$$

$$\sum_{n=0}^{\infty} \sum_{i=0}^k \frac{x^n}{n!} c_i a_n = 0 \quad (40)$$

$$\sum_{i=0}^k c_i \sum_{n=0}^{\infty} \frac{x^n}{n!} a_n = 0 \quad (41)$$

By implementing the exponential generating function as shown in (30), this provides:

$$\sum_{i=0}^k \left[c_i f^{(i)}(x) \right] \quad (42)$$

Now assume that the solution exists and all roots of the characteristic polynomial are unique (i.e. the solution is of the form $f(x) \propto e^{m_i x} : m_i \neq m_j \forall i \neq j$), this implies that [55, Ch. 4] :

$$f(x) = \sum_{i=0}^k [k_i e^{m_i x}], \quad \exists m, k \in \mathbb{C}$$

This can be re-expressed in terms of the exponential power series, in order to relate the solution of the function $f(x)$ back to a solution of the sequence a_n , (see section for a derivation of the exponential power series **#TODO make section on to prove exponential power series using taylor series expansion if we get time**):

$$\begin{aligned}
 \sum_{i=0}^k [k_i e^{m_i x}] &= \sum_{i=0}^k \left[k_i \sum_{n=0}^{\infty} \frac{(m_i x)^n}{n!} \right] \\
 &= \sum_{i=0}^k \sum_{n=0}^{\infty} k_i m_i^n \frac{x^n}{n!} \\
 &= \sum_{n=0}^{\infty} \sum_{i=0}^k k_i m_i^n \frac{x^n}{n!} \\
 &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} \sum_{i=0}^k [k_i m_i^n] \right], \quad \exists k_i \in \mathbb{C}, \forall i \in \mathbb{Z}^+ \cap [1, k]
 \end{aligned} \tag{43}$$

Recall the definition of the generating function from (30), by equating this to (43):

$$\begin{aligned}
 f(x) &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} a_n \right] \\
 &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} \sum_{i=0}^k [k_i m_i^n] \right] \\
 \implies a_n &= \sum_{i=0}^k [k_i m_i^n]
 \end{aligned}$$

□

This can be verified by the fibonacci sequence as shown in section 4.3, the solution to the characteristic equation is $m_1 = \varphi, m_2 = (1 - \varphi)$ and the corresponding solution to the linear ODE and recursive relation are:

$$\begin{aligned}
 f(x) &= c_1 e^{\varphi x} + c_2 e^{(1-\varphi)x}, \quad \exists c_1, c_2 \in \mathbb{R} \subset \mathbb{C} \\
 \iff a_n &= k_1 n^\varphi + k_2 n^{1-\varphi}, \quad \exists k_1, k_2 \in \mathbb{R} \subset \mathbb{C}
 \end{aligned}$$

Repeated Roots of Characteristic Equation

RYAN

- Example Consider the following recurrence relation:

$$\begin{aligned}
 a_{n+2} - 10a_{n+1} + 25a_n &= 0 \tag{44} \\
 \implies \sum_{n=0}^{\infty} \left[a_{n+2} \frac{x^n}{n!} \right] - 10 \sum_{n=0}^{\infty} \left[a_{n+1} \frac{x^n}{n!} \right] + 25 \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] &= 0
 \end{aligned}$$

By applying the definition of the exponential generating function at (30) :

$$f''(x) - 10f'(x) + 25f(x) = 0 \quad (45)$$

By implementing the already well-established theory of linear ODE's, the characteristic equation for (45) can be expressed as:

$$\begin{aligned} m^2 - 10m + 25 &= 0 \\ (m - 5)^2 &= 0 \\ m &= 5 \end{aligned} \quad (46)$$

Herein lies a complexity, in order to solve this, the solution produced from (46) can be used with the *Reduction of Order* technique to produce a solution that will be of the form [56, §4.3].

$$f(x) = c_1 e^{5x} + c_2 x e^{5x} \quad (47)$$

(47) can be expressed in terms of the exponential power series in order to try and relate the solution for the function back to the generating function, observe however the following power series identity (proof in section 3):

$$x^k e^x = \sum_{n=k}^{\infty} \left[\frac{x^n}{(n-k)!} \right], \quad \exists k \in \mathbb{Z}^+ \quad (48)$$

by applying identity (48) to equation (47)

$$\begin{aligned} \implies f(x) &= \sum_{n=0}^{\infty} \left[c_1 \frac{(5x)^n}{n!} \right] + \sum_{n=1}^{\infty} \left[c_2 n \frac{(5x)^n}{n(n-1)!} \right] \\ &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} (c_1 5^n + c_2 n 5^n) \right] \end{aligned}$$

Given the definition of the exponential generating function from (30)

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \\ \iff a_n &= c_1 5^n + c_2 n 5^n \end{aligned}$$

□

4.3 Exponential Generating Functions

2. Proof Consider a recurrence relation of the form:

$$\begin{aligned} \sum_{n=0}^k [c_i a_n] &= 0 \\ \implies \sum_{n=0}^{\infty} \sum_{i=0}^k c_i a_n \frac{x^n}{n!} &= 0 \\ \sum_{i=0}^k \sum_{n=0}^{\infty} c_i a_n \frac{x^n}{n!} \end{aligned}$$

By substituting for the value of the generating function from (30):

$$\sum_{i=0}^k [c_i f^{(k)}(x)] \quad (49)$$

Assume that (49) corresponds to a characteristic polynomial with only 1 root of multiplicity k , the solution would hence be of the form:

$$\begin{aligned} \sum_{i=0}^k [c_i m^i] &= 0 \wedge m = B, \quad \exists! B \in \mathbb{C} \\ \implies f(x) &= \sum_{i=0}^k [x^i A_i e^{mx}], \quad \exists A \in \mathbb{C}^+, \quad \forall i \in [1, k] \cap \mathbb{N} \end{aligned} \quad (50)$$

If we assume the identity from (48):

$$x^k e^x = \sum_{n=k}^{\infty} \left[\frac{x^n}{(n-k)!} \right]$$

See section 3 for proof.

We can apply identity (48) to (50), which gives:

$$\begin{aligned} f(x) &= \sum_{i=0}^k \left[A_i \sum_{n=i}^{\infty} \left[\frac{(xm)^n}{(n-i)!} \right] \right] \\ &= \sum_{n=0}^{\infty} \left[\sum_{i=0}^k \left[\frac{x^n}{n!} \frac{n!}{(n-i)!} A_i m^n \right] \right] \\ &= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} \sum_{i=0}^k \left[\frac{n!}{(n-i)!} A_i m^n \right] \right] \end{aligned}$$

Recall the generating function that was used to get (49):

$$\begin{aligned}
 f(x) &= \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \\
 \implies a_n &= \sum_{i=0}^k \left[A_i \frac{n!}{(n-i)!} m^n \right] \\
 &= \sum_{i=0}^k \left[m^n A_i \prod_0^k [n - (i-1)] \right]
 \end{aligned} \tag{51}$$

$\because i \leq k$

$$= \sum_{i=0}^k \left[A_i^* m^n n^i \right], \quad \exists A_i \in \mathbb{C}, \quad \forall i \leq k \in \mathbb{Z}^+$$

□

3. Proof

JAMES In this section the proof of

(a) Motivation

Consider the function $f(x) = xe^x$. Using the taylor series formula we get the following:

$$\begin{aligned}
 xe^x &= 0 + \frac{1}{1!}x + \frac{2}{2!}x^2 + \frac{3}{3!}x^3 + \frac{4}{4!}x^4 + \frac{5}{5!}x^5 + \dots \\
 &= \sum_{n=0}^{\infty} \frac{nx^n}{n!} \\
 &= \sum_{n=1}^{\infty} \frac{x^n}{(n-1)!}
 \end{aligned}$$

Similarly, $f(x) = x^2 e^x$ will give:

$$\begin{aligned}
 x^2 e^x &= \frac{0}{0!} + \frac{0x}{1!} + \frac{2x^2}{2!} + \frac{6x^3}{3!} + \frac{12x^4}{4!} + \frac{20x^5}{5!} + \dots \\
 &= \frac{2 \cdot 1x^2}{2!} + \frac{3 \cdot 2x^3}{3!} + \frac{4 \cdot 3x^4}{4!} + \frac{5 \cdot 4x^5}{5!} + \dots \\
 &= \sum_{n=2}^{\infty} \frac{n(n-1)x^n}{n!} \\
 &= \sum_{n=2}^{\infty} \frac{x^n}{(n-2)!}
 \end{aligned}$$

We conjecture that if we continue this on, we get:

$$x^k e^x = \sum_{n=k}^{\infty} \frac{x^n}{(n-k)!} \quad \text{for } k \in \mathbb{Z}^+ \cap 0$$

General Proof In sections 4.3 and 4.3 it was shown that a recurrence relation can be related to an ODE and then that solution can be transformed to provide a solution for the recurrence relation. This was shown in two separate cases, one with unique roots and the other with repeated roots. However, in many circumstances the solutions to the characteristics equation are a combination of both unique and repeated roots. Hence, in general the solution to a linear ODE will be a superposition of solutions for each root, repeated or unique and so a goal of our research will be to put this together to find a general solution for homogenous linear recurrence relations.

Sketching out an approach for this:

- Use the Generating function to get an ODE
 - The ODE will have a solution that is a combination of the above two forms
 - The solution will translate back to a combination of both above forms
1. Power Series Combination

4.4 Proving with the Monotone Convergence Theorem

By Solving the Fibonacci Sequence using the Monotone Converge Theorem we can show that it is related to e

4.5 Fibonacci Sequence and the Golden Ratio

The *Fibonacci Sequence* is actually very interesting, observe that the ratios of the terms converge to the *Golden Ratio*:

$$\begin{aligned} F_n &= \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\varphi^n - \psi^n}{\sqrt{5}} \\ \iff \frac{F_{n+1}}{F_n} &= \frac{\varphi^{n+1} - \psi^{n+1}}{\varphi^n - \psi^n} \\ \iff \lim_{n \rightarrow \infty} \left[\frac{F_{n+1}}{F_n} \right] &= \lim_{n \rightarrow \infty} \left[\frac{\varphi^{n+1} - \psi^{n+1}}{\varphi^n - \psi^n} \right] \\ &= \frac{\varphi^{n+1} - \lim_{n \rightarrow \infty} [\psi^{n+1}]}{\varphi^n - \lim_{n \rightarrow \infty} [\psi^n]} \end{aligned}$$

because $|\psi| < 0$ $n \rightarrow \infty \implies \psi^n \rightarrow 0$:

$$\begin{aligned} &= \frac{\varphi^{n+1} - 0}{\varphi^n - 0} \\ &= \varphi \end{aligned}$$

We'll come back to this later on when looking at spirals and fractals.

We hope to demonstrate this relationship between the ratio of successive terms of the fibonacci sequence without relying on ODEs and generating functions and by instead using limits and the *Monotone Convergence Theorem*, the hope being that this will reveal deeper underlying relationships between the *Fibonacci Sequence*, the *Golden Ratio* and there occurrences in nature (such as the example in section 4.5.1 given that the both appear to occur in patterns observed in nature).

We also hope to find a method to produce the the diagram shown in figure computationally, ideally by using the Turtle function in *Julia*.

4.5.1 Fibonacci Sequence in Nature (This may be Removed)

RYAN

The distribution of sunflower seeds is an example of the *Fibonacci Sequence* occuring in a pattern observed in nature (see Figure 24).

Imagine that the process a sunflower follows when placing seeds is as follows:²⁰

1. Place a seed
2. Move some small unit away from the origin
3. Rotate some constant angle θ (or 0) from the previous seed (with respect to the origin).
4. Repeat this process until a seed hits some outer boundary.

This process can be simulated in Julia [4] as shown in listing 14,²¹ which combined with *ImageMagick* (see e.g. 28), produces output as shown in figure 22 and 23.

A distribution of seeds under this process would be optimal if the amount of empty space was minimised, spirals, stars and swirls contain patterns compromise this.

To minimize this, the proportion of the circle traversed in step 3 must be an irrational number, however this alone is not sufficient, the decimal values must also be not to approximated by a rational number, for example [38]:

- $\pi \bmod 1 \approx \frac{1}{7} = 0.7142857142857143$
- $e \bmod 1 \approx \frac{5}{7} = 0.14285714285714285$

It can be seen by simulation that ϕ and ψ (because $\phi \bmod 1 = \psi$) are solutions to this optimisation problem as shown in figure 23, this solution is unstable, a very minor change to the value will result in patterns re-emerging in the distribution.

²⁰This process is simply conjecture, other than seeing a very nice example at MathIsFun.com [38], we have no evidence to suggest that this is the way that sunflowers distribute their seeds.

However the simulations performed within *Julia* are very encouraging and suggest that this process isn't too far off.

²¹Emojis and UTF8 were used in this code, and despite using *xelatex* with *fontspec* they aren't rendering properly, we intend to have this rectified in time for final submission.

Another interesting property is that the number of spirals that appear to rotate clockwise and anti-clockwise appear to be fibonacci numbers. Connecting this occur with the relationship between the *Fibonacci Sequence* as discussed in section 4.5 is something we hope to look at in this project. Illustrating this phenomena with *Julia* by finding the mathematics to colour the correct spirals is also something we intend to look at in this project.

The bottom right spiral in figure 22 has a ratio of rotation of $\frac{1}{\pi}$, the spirals look similar to one direction of the spirals occurring in figure 23, it is not clear if there is any significance to this similarity.

```

1  φ = 1.61803398875
2  ψ = φ^-1
3  ψ = 0.61803398875
4  function sfSeeds(ratio)
5  let t = Turtle()
6  for θ in [(ratio*2*π)*i for i in 1:3000]
7    gsave()
8    scale(0.05)
9    rotate(θ)
10   #      Pencolor(0, rand(1)[1], rand(1)[1], rand(1)[1])
11   Forward(t, 1)
12   Rectangle(t, 50, 50)
13   grestore()
14 end
15   label = string("Ratio = ", round(ratio, digits = 8))
16   textcentered(label, 100, 200)
17 end
18 @svg begin
19   sfSeeds(ψ)
20 end 600 600

```

Listing 14: Simulation of the distribution of sunflowers as described in section 4.5.1

4.6 Relating The Fibonacci Sequence to the Julia Set

RYAN

See [this link](#).

5 Julia Sets

RYAN

The julia set is the outline [41, Ch. 14].

The mandelbrot has to do with whether or not it's connected. ** TODO The math behind it ,*** TODO Like Escaping after 2 I cannot figure this out, I need more time, look around Ch. 12 of falconer [10] There is a relationship between the fibonacci sequence, modelling population growth and the mandelbrot curve, I would like to use that to tie some of the discussion together, see [this video from Veritasium to get an idea of what i mean](#).

5.1 Introduction

Julia sets are a very interesting fractal and we hope to investigate them further in this project.

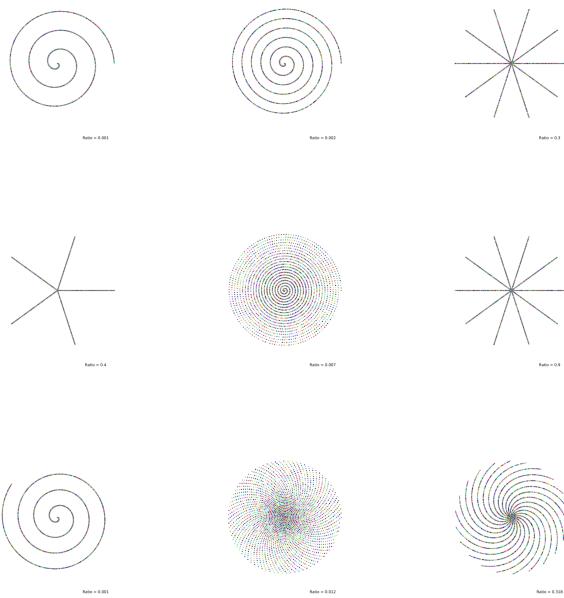


Figure 22: Simulated Distribution of Sunflower seeds as described in section 4.5.1 and listing 14

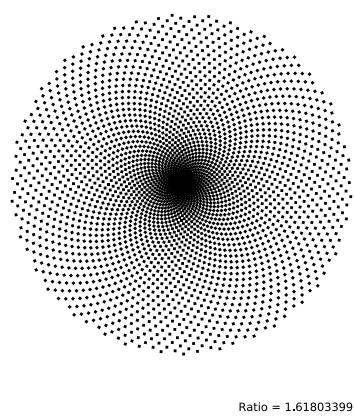
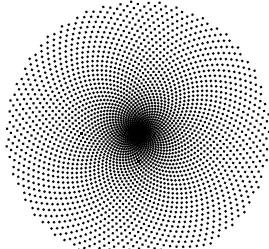


Figure 23: Optimisation of simulated distribution of Sunflower seeds occurs for $\theta = 2\varphi\pi$ as described in section 4.5.1 and listing 14



Ratio = 1.61803399

Figure 24: Distribution of the seeds of a sunflower (see [5] licenced under CC)

5.2 Motivation

Consider the iterative process $x \rightarrow x^2$, $x \in \mathbb{R}$, for values of $x > 1$ this process will diverge and for $x < 1$ it will converge.

Now Consider the iterative process $z \rightarrow z^2$, $z \in \mathbb{C}$, for values of $|z| > 1$ this process will diverge and for $|z| < 1$ it will converge.

Although this seems trivial this can be generalised.

Consider:

- The complex plane for $|z| \leq 1$
- Some function $f_c(z) = z^2 + c$, $c \leq 1 \in \mathbb{C}$ that can be used to iterate with

Every value on that plane will belong to one of the two following sets

- P_c
 - The set of values on the plane that converge to zero (prisoners)
 - Define $Q_c^{(k)}$ to be the set of values confirmed as prisoners after k iterations of f_c
 - * this implies $\lim_{k \rightarrow \infty} [Q_c^{(k)}] = P_c$
- E_c
 - The set of values on the plane that tend to ∞ (escapees)

In the case of $f_0(z) = z^2$ all values $|z| \leq 1$ are bounded with $|z| = 1$ being an unstable stationary circle, but let's investigate what happens for different iterative functions like $f_1(z) = z^2 - 1$, despite how trivial this seems at first glance.

5.3 Plotting the Sets

Although the convergence of values may appear simple at first, we'll implement a strategy to plot the prisoner and escape sets on the complex plane.

Because this involves iteration and *Python* is a little slow, We'll denote complex values as a vector²² and define the operations as described in listing 15.²³

To implement this test we'll consider a function called `escape_test` that applies an iteration (in this case $f_0 : z \rightarrow z^2$) until that value diverges or converges.

While iterating with f_c once $|z| > \max(\{c, 2\})$, the value must diverge because $|c| \leq 1$, so rather than record whether or not the value converges or diverges, the `escape_test` can instead record the number of iterations (k) until the value has crossed that boundary and this will provide a measurement of the rate of divergence.

Then the `escape_test` function can be mapped over a matrix, where each element of that matrix is in turn mapped to a point on the cartesian plane, the resulting matrix can be visualised as an image²⁴, this is implemented in listing 16 and the corresponding output shown in 25.

with respect to listing 16:

- Observe that the `magnitude` function wasn't used:
1. This is because a `sqrt` is a costly operation and comparing two squares saves an operation

```

1  from math import sqrt
2  def magnitude(z):
3      # return sqrt(z[0]**2 + z[1]**2)
4      x = z[0]
5      y = z[1]
6      return sqrt(sum(map(lambda x: x**2, [x, y])))
7
8  def cAdd(a, b):
9      x = a[0] + b[0]
10     y = a[1] + b[1]
11     return [x, y]
12
13
14  def cMult(u, v):
15      x = u[0]*v[0]-u[1]*v[1]
16      y = u[1]*v[0]+u[0]*v[1]
17      return [x, y]
```

Listing 15: Defining Complex Operations with vectors

This is precisely what we expected, but this is where things get interesting, consider now the result if we apply this same procedure to $f_1 : z \rightarrow z^2 - 1$ or something

²²See figure for the obligatory XKCD Comic

²³This technique was adapted from Chapter 7 of *Math adventures with Python* [11]

²⁴these cascading values are much like brightness in Astronomy

5.3 Plotting the Sets

```
1 %matplotlib inline
2 %config InlineBackend.figure_format = 'svg'

3 import numpy as np
4 def escape_test(z, num):

5     ''' runs the process num amount of times and returns the count of
6     divergence'''

7     c = [0, 0]
8     count = 0

9     z1 = z #Remember the original value that we are working with
10    # Iterate num times

11    while count <= num:
12        dist = sum([n**2 for n in z1])

13        distc = sum([n**2 for n in c])
14        # check for divergence

15        if dist > max(2, distc):
16            #return the step it diverged on

17            return count
18            #iterate z

19            z1 = cAdd(cMult(z1, z1), c)
20            count+=1

21            #if z hasn't diverged by the end
22            return num

23
24

25
26 p = 0.25 #horizontal, vertical, pinch (zoom)

27 res = 200
28 h = res/2

29 v = res/2
30

31 pic = np.zeros([res, res])
32 for i in range(pic.shape[0]):

33     for j in range(pic.shape[1]):
34         x = (j - h)/(p*res)

35         y = (i-v)/(p*res)
36         z = [x, y]

37         col = escape_test(z, 100)
38         pic[i, j] = col
```

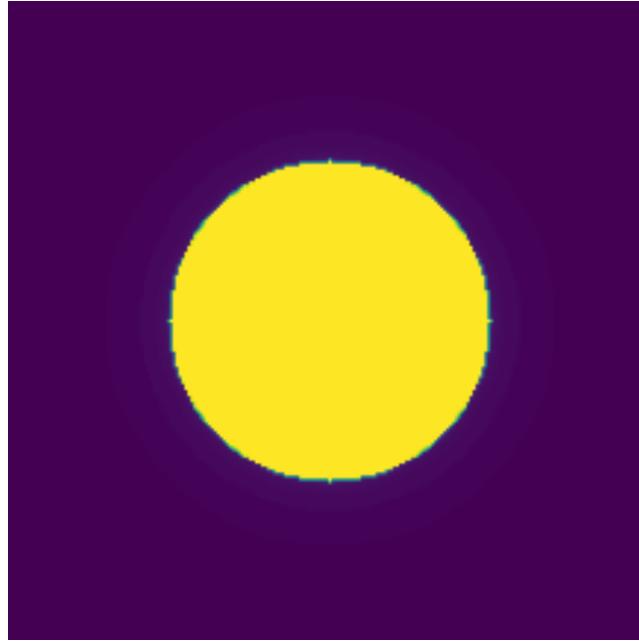


Figure 25: Circle of Convergence for $f_0 : z \rightarrow z^2$

arbitrary like $f_{\frac{1}{4} + \frac{i}{2}} : z \rightarrow z^2 + (\frac{1}{4} + \frac{i}{2})$, the result is something remarkably unexpected, as shown in figures 26 and 27.

To investigate this further consider the more general function $f_{0.8e^{\pi i \tau}} : z \rightarrow z^2 + 0.8e^{\pi i \tau}$, $\tau \in \mathbb{R}$, many fractals can be generated using this set by varying the value of τ ²⁵.

Python is too slow for this, but the *Julia* programming language, as a compiled language, is significantly faster and has the benefit of treating complex numbers as first class citizens, these images can be generated in *Julia* in a similar fashion as before, with the specifics shown in listing 17. The *GR* package appears to be the best plotting library performance wise and so was used to save corresponding images to disc, this is demonstrated in listing 18 where 1200 pictures at a 2.25 MP resolution were produced.
²⁶

A subset of these images can be combined using *ImageMagick* and bash to create a collage, *ImageMagick* can also be used to produce an animation but it often fails and a superior approach is to use *ffmpeg*, this is demonstrated in listing 19, the collage is shown in figure 28 and a corresponding animation is [available online](#)²⁷.

²⁵This approach was inspired by an animation on the *Julia Set* Wikipedia article [21]

²⁶On my system this took about 30 minutes.

²⁷<https://dl.dropboxusercontent.com/s/rbu25urfg8sbwfu/out.gif?dl=0>

5.3 Plotting the Sets

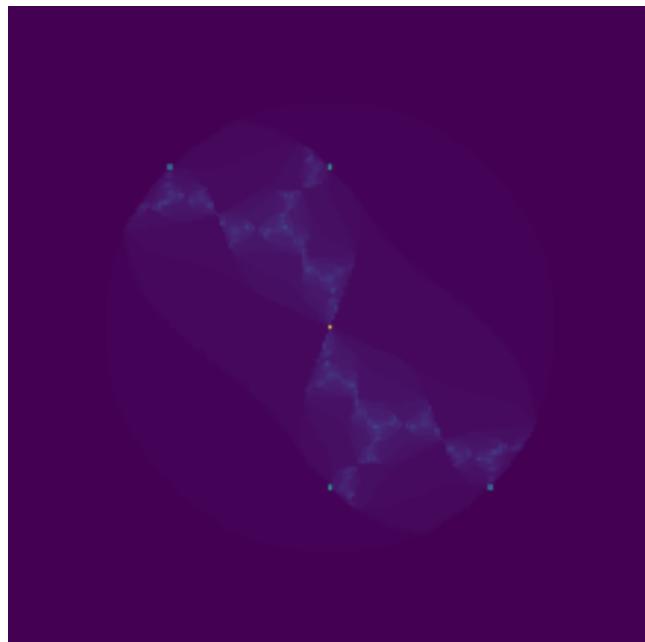


Figure 26: Circle of Convergence for $f_0 : z \rightarrow z^2 - 1$

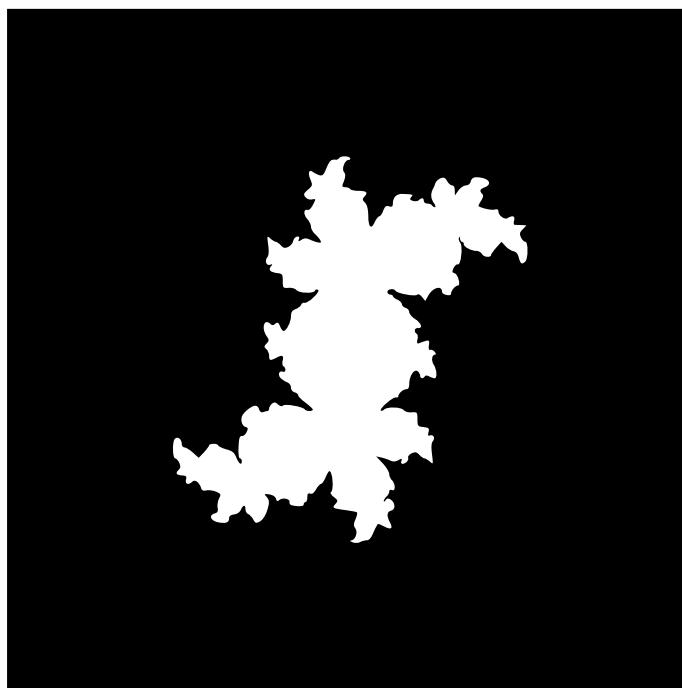


Figure 27: Circle of Convergence for $f_{\frac{1}{4} + \frac{i}{2}} : z \rightarrow z^2 + \frac{1}{4} + \frac{i}{2}$

```

1 # * Define the Julia Set
2 """
3
4 Determine whether or not a value will converge under iteration
5 """
6
7
8 function juliaSet(z, num, my_func)
9     count = 1
10
11    # Remember the value of z
12    z1 = z
13
14    # Iterate num times
15    while count ≤ num
16
17        # check for divergence
18        if abs(z1)>2
19
20            return Int(count)
21        end
22
23        #iterate z
24        z1 = my_func(z1) # + z
25
26        count=count+1
27    end
28
29    #if z hasn't diverged by the end
30    return Int(num)
31
32 end
33
34
35 # * Make a Picture
36 """
37
38 Loop over a matrix and apply apply the julia-set function to
39 the corresponding complex value
40
41 """
42
43 function make_picture(width, height, my_func)
44
45     pic_mat = zeros(width, height)
46     zoom = 0.3
47
48     for i in 1:size(pic_mat)[1]
49         for j in 1:size(pic_mat)[2]
50
51             x = (j-width/2)/(width*zoom)
52             y = (i-height/2)/(height*zoom)
53
54             pic_mat[i,j] = juliaSet(x+y*im, 256, my_func)
55         end
56     end
57
58     return pic_mat
59
60 end

```

5.3 Plotting the Sets

```
1  # * Use GR to Save a Bunch of Images
2  ## GR is faster than PyPlot

3  using GR
4  function save_images(count, res)

5  try
6      mkdir("/tmp/gifs")

7  catch
8  end

9  j = 1
10 for i in (1:count)/(40*2*pi)

11     j = j + 1

12     GR.imshow(make_picture(res, res, z -> z^2 + 0.8*exp(i*im*9/2))) # PyPlot uses
13      $\hookrightarrow$  interpolation = "None"

14     name = string("/tmp/gifs/j", lpad(j, 5, "0"), ".png")
15     GR.savefig(name)
16
17 end
18 end

17
18 save_images(1200, 1500) # Number and Res
```

Listing 18: Generate and save the images with GR

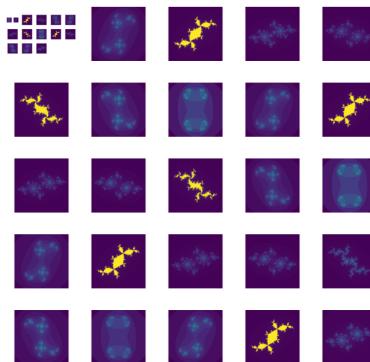


Figure 28: Various fractals corresponding to $f_{0.8e^{\pi i \tau}}$

```
1 # Use montage multiple times to get recursion for fun
2 montage (ls *png | sed -n '1p;0~600p') 0a.png

3 montage (ls *png | sed -n '1p;0~100p') a.png
4 montage (ls *png | sed -n '1p;0~50p') -geometry 1000x1000 a.png

5
6 # Use ImageMagick to Produce a gif (unreliable)

7 convert -delay 10 *.png 0.gif
8

9 # Use FFmpeg to produce a Gif instead
10 ffmpeg \
11     -framerate 60 \
12     -pattern_type glob \
13     -i '*.png' \
14     -r 15 \
15     out.mov
```

Listing 19: Using bash, `ffmpeg` and *ImageMagick* to combine the images and produce an animation.

6 MandelBrot Set

RYAN

Investigating these fractals, a natural question might be whether or not any given c value will produce a fractal that is an open disc or a closed disc.

So pick a value $|\gamma| < 1$ in the complex plane and use it to produce the julia set f_γ , if the corresponding prisoner set P is closed we this value is defined as belonging to the *Mandelbrot* set.

It can be shown (and I intend to show it generally), that this set is equivalent to re-implementing the previous strategy such that $z \rightarrow z^2 + z_0$ where z_0 is unchanging or more clearly as a sequence:

$$z_{n+1} = z_n^2 + c \quad (52)$$

$$z_0 = c \quad (53)$$

This strategy is implemented in listing and produces the output shown in figure 29.

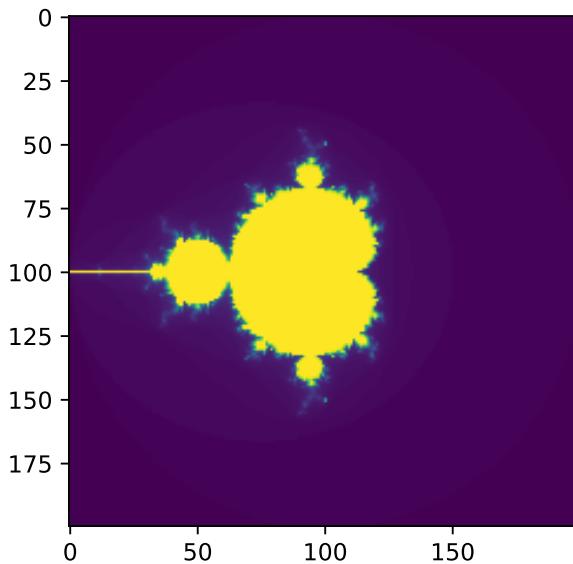


Figure 29: Mandelbrot Set produced in *Python* as shown in listing 20

This output although remarkable is however fairly undetailed, by using *Julia* a much larger image can be produced, in *Julia* producing a 4 GB, 400 MP image can be done in little time (about 10 minutes on my system), this is demonstrated in listing and the corresponding FITS image is [available-online](#).²⁸

²⁸<https://www.dropbox.com/s/jd5qf1pi2h68f2c/mandelbrot-400mpx.fits?dl=0>

```

1  %matplotlib inline
2  %config InlineBackend.figure_format = 'svg'
3  def mandelbrot(z, num):
4      ''' runs the process num amount of times and returns the count of
5      divergence'''
6      count = 0
7      # Define z1 as z
8      z1 = z
9      # Iterate num times
10     while count <= num:
11         # check for divergence
12         if magnitude(z1) > 2.0:
13             #return the step it diverged on
14             return count
15         #iterate z
16         z1 = cAdd(cMult(z1, z1),z)
17         count+=1
18         #if z hasn't diverged by the end
19     return num
20
21 import numpy as np
22
23
24 p = 0.25 # horizontal, vertical, pinch (zoom)
25 res = 200
26 h = res/2
27 v = res/2
28
29 pic = np.zeros([res, res])
30 for i in range(pic.shape[0]):
31     for j in range(pic.shape[1]):
32         x = (j - h)/(p*res)
33         y = (i-v)/(p*res)
34         z = [x, y]
35         col = mandelbrot(z, 100)
36         pic[i, j] = col
37
38 import matplotlib.pyplot as plt
39 plt.imshow(pic)
40 # plt.show()

```

Listing 20: All values of c that lead to a closed *Julia-set*

```

1   function mandelbrot(z, num, my_func)
2     count = 1
3     # Define z1 as z
4     z1 = z
5     # Iterate num times
6     while count <= num
7       # check for divergence
8       if abs(z1)>2
9         return Int(count)
10      end
11      #iterate z
12      z1 = my_func(z1) + z
13      count=count+1
14    end
15    #if z hasn't diverged by the end
16    return Int(num)
17  end
18
19  function make_picture(width, height, my_func)
20    pic_mat = zeros(width, height)
21    for i in 1:size(pic_mat)[1]
22      for j in 1:size(pic_mat)[2]
23        x = j/width
24        y = i/height
25        pic_mat[i,j] = mandelbrot(x+y*im, 99, my_func)
26      end
27    end
28    return pic_mat
29  end
30
31
32 using FITSIO
33 function save_picture(filename, matrix)
34   f = FITS(filename, "w");
35   # data = reshape(1:100, 5, 20)
36   # data = pic_mat
37   write(f, matrix) # Write a new image extension with the data
38
39   data = Dict("col1"=>[1., 2., 3.], "col2"=>[1, 2, 3]);
40   write(f, data) # write a new binary table to a new extension
41
42   close(f)
43 end
44
45 # * Save Picture
46 -----
47 my_pic = make_picture(20000, 20000, z -> z^2) 2000^2 is 4 GB
48 save_picture("/tmp/a.fits", my_pic)

```

7 Appendix

So unless code contributes directly to the discussion we'll put it in the appendix.

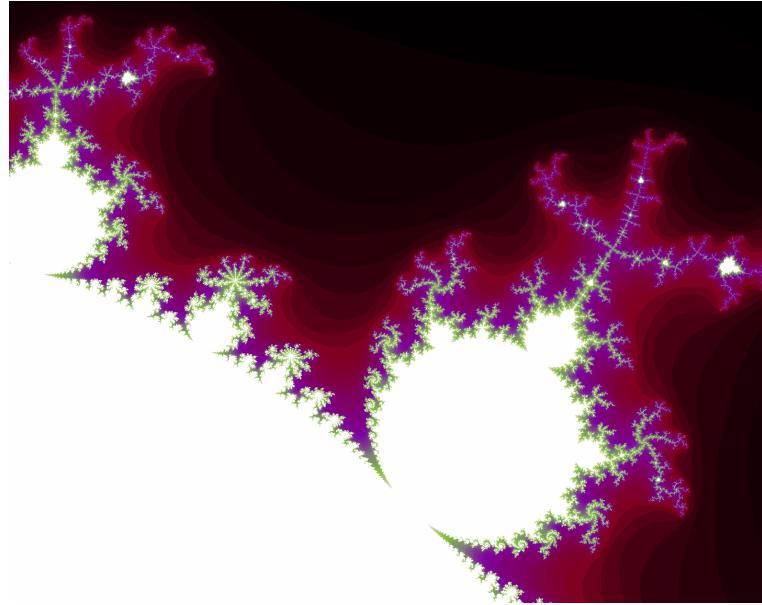


Figure 30: Screenshot of Mandelbrot FITS image produced by listing

.1 Finding Material

```
1  recoll -c /home/ryan/Dropbox/Books/Textbooks/Mathematics/Chaos_Theory/chaos_books_recoll
2  ↵ & disown
```

.2 Code Listings

.2.1 Sierpinski's Carpet

.2.2 Cantor's Dust

.3 Font Lock

```
1  ;; match:
2  ;;; \scite{key}s
3  (add-to-list 'font-lock-extra-managed-props 'display)
4  (font-lock-add-keywords nil
5  '("\\\\(cite:[a-z0-9A-Z]+\\\\)" 1 '(face nil display "□")))
6
7
8  ;; match
9  ;;; [[cite:key][p. num]]
10 (add-to-list 'font-lock-extra-managed-props 'display)
11 (font-lock-add-keywords nil
12 '("\\\\(\\\\[\\\\[cite:[a-z0-9A-Z]+\\\\]]\\\\[\\.\\*\\\\]\\]\\)\\)" 1 '(face nil display "□")))
```

.3 Font Lock

```
1 #-----
2 #--- Function -----
3 #
4
5 function carpet(ICMat, width)
6     B = ICMat
7     h = size(B)[1]
8     w = size(B)[2]
9     Z = zeros(Int, h, w)
10    B = [B B B;
11          B Z B;
12          B B B]
13    if (3*w)<width
14        B = carpet(B, width)
15    end
16    return B
17 end
18
19 #
20 #--- Plot -----
21 #
22 using GR, Plots

23 gr() # Set Plots backend as GR
24
25 (mat = carpet(fill(1, 1, 1), 9^2)) |> size
26 p1 = GR.imshow(mat)
27
28 #
29 #--- Dimension -----
30 #
31 mat2 = carpet(fill(1, 1, 1), 1000)
32 l2 = sum(mat2)
33 size2 = size(mat2)[1]
34 mat1 = carpet(fill(1, 1, 1), 500)
35 l1 = sum(mat1)
36 size1 = size(mat1)[1]
37
38 #
39 ## julia> log(l2/l1)/log(size2/size1)
40 ## 1.89
```

Listing 21: Function to produce Sierpinski's carpet, shown in figure 8

```
1 #-----
2 #--- Function -----
3 #
4
5 function dust(ICMat, width)
6     B = ICMat
7     h = size(B)[1]
8     w = size(B)[2]
9     Z = zeros(Int, h, w)
10    B = [Z Z B Z;
11          B Z Z Z;
12          Z Z Z B;
13          Z B Z Z]
14    if (3^w) < width
15        B = dust(B, width)
16    end
17    return B
18 end
19
20 #
21 #--- Plot -----
22 #
23 using GR, Plots
24 gr() # Set Plots backend as GR
25
26 (mat = dust(fill(1, 1, 1), 9^2)) |> size
27 pl = GR.imshow(mat)
28
29 #
30 #--- Dimension -----
31 #
32
33 mat2 = dust(fill(1, 1, 1), 1000)
34 l2 = sum(mat2)
35 size2 = size(mat2)[1]
36 mat1 = dust(fill(1, 1, 1), 500)
37 l1 = sum(mat1)
38 size1 = size(mat1)[1]
39
40 #
41 ## julia> log(l2/l1)/log(size2/size1)
42 ## 1.0
```

Listing 22: Function to generate Cantor Dust, shown in 9

4 Resources Used for the Hausdorff Dimension

Research for §on the Hausdorff Dimension proved actually to be quite difficult, while much information is available online, precice and clear explanations of the Hausdorff dimension are difficult to find without scouring texts, the following proved very helpful generally in preparing for this topic and I would strongly recommend these chapters as a starting point for further reading on this topic:

- Edgar, G. A., *Measure, topology, and fractal geometry* [7, Ch. 6]
- Falconer, K. J., *Fractal geometry: mathematical foundations and applications* [9, Ch. 2]
- Gouyet, J., *Physics and fractal structures* [16, §1.3]
- Vicsek, T., *Fractal Growth Phenomena* [51, Ch. 4]
 - See also p 14 specifically
- Tél, T., Gruiz, M., & Kulacsy, K., *Chaotic dynamics: an introduction based on classical mechanics* [49, §2.1.2]
- Peitgen, H., Jürgens, H., & Saupe, D., *Chaos and fractals: new frontiers of science* [41, §4.3]

5 Functions for constructing the Julia Set

The following functions were saved in a file called:

- `@time include("./Julia-Set-Dimensions-functions.jl")`

This file was loaded into the current workspace by using the following at the top of a *Julia* script:

- `@time include("./Julia-Set-Dimensions-functions.jl")`

```
1  using GR
2  using DataFrames
3  using Gadfly
4  using GLM
5  using SharedArrays
6  using Distributed
7
8  #####
9  ### Julia / MandelBrot Functions #####
10 #####
11 """
12 # Julia Set
13 Returns how many iterations it takes for a value on the complex plane to diverge
14 under recursion. if `boolQ` is specified as true a 1/0 will be returned to
15 indicate divergence or convergence.
16
17 ## Variables
```

```

19     - `z`
20         - A value on the complex plane within the unit circle
21     - `num`
22         - A number of iterations to perform before conceding that the value is not
23             divergent.
24     - `my_func`
25         - A function to perform on `z`, for a julia set the function will be of the
26             form `z -> z^2 + a + im*b`
27             - So for example the Douady Rabbit would be described by `z -> z^2 -0.123+0.745*im`
28 """
29 function juliaSet(z, num, my_func, boolQ=true)
30     count = 1
31     # Define z1 as z
32     z1 = z
33     # Iterate num times
34     while count ≤ num
35         # check for divergence
36         if real(z1)^2+imag(z1)^2 > 2^2
37             if(boolQ) return 0 else return Int(count) end
38         end
39         #iterate z
40         z1 = my_func(z1) # + z
41         count=count+1
42     end
43     #if z hasn't diverged by the end
44     if(boolQ) return 1 else return Int(count) end
45 end
46
47 """
48
49 # Mandelbrot Set
50 Returns how many iterations it takes for a value on the complex plane to diverge
51 under recursion of \$z \rightarrow z^2 + z_0\$.
52
53 Values that converge represent constants of the julia set that lead to a
54 connected set. (TODO: Have I got that Vice Versa?)
55
56
57 ## Variables
58 - `z`
59     - A value on the complex plane within the unit circle
60 - `num`
61     - A number of iterations to perform before conceding that the value is not
62         divergent.
63 - `boolQ`
64     - `true` or `false` value indicating whether or not to return 1/0 values
65         indicating divergence or convergence respectively or to return the number of
66         iterations performed before conceding no divergence.
67 """
68 function mandelbrot(z, num, boolQ = true)
69     count = 1
70     # Define z1 as z
71     z1 = z
72     # Iterate num times
73     while count ≤ num
74         # check for divergence
75         if real(z1)^2+imag(z1)^2 > 2^2
76             if(boolQ) return 0 else return Int(count) end
77         end
78         #iterate z
79         z1 = z1^2 + z
80         count=count+1
81     end
82     #if z hasn't diverged by the end

```

.5 Functions for constructing the Julia Set

```
83     return 1 # Int(num)
84     if(boolQ) return 1 else return Int(count) end
85 end
86
87 function test(x, y)
88     if(x<1) return x else return y end
89 end
90
91 #####
92 ##### Build a Matrix Image #####
93 #####
94 #####
95 """
96 # Make a Picture
97
98 This maps a function on the complex plane to a matrix where each element of the
99 matrix corresponds to a single value on the complex plane. The matrix can be
100 interpreted as a greyscale image.
101
102 Inside the function is a `zoom` parameter that can be modified for different
103 fractals, fur the julia and mandelbrot sets this shouldn't need to be adjusted.
104
105 The height and width should be interpreted as resolution of the image.
106
107 - `width`
108     - width of the output matrix
109 - `height`
110     - height of the output matrix
111 - `myfunc`
112     - Complex Function to apply across the complex plane
113 """
114 function make_picture(width, height, my_func)
115     pic_mat = zeros(width, height)
116     zoom = 0.3
117     for j in 1:size(pic_mat)[2]
118         for i in 1:size(pic_mat)[1]
119             x = (j-width/2)/(width*zoom)
120             y = (i-height/2)/(height*zoom)
121             pic_mat[i,j] = juliaSet(x+y*im, 256, my_func)
122         end
123     end
124     return pic_mat
125 end
126
127 #####
128 ### Make the Outline #####
129 #####
130 #####
131 """
132 # Outline
133
134 Sets all elements with neighbours on all sides to 0.
135
136 - `mat`
137     - A matrix
138     - If this matrix is the convergent values corresponding to a julia set the
139         output will be the outline, which is the definition of the julia set.
140 """
141 function outline(mat)
142     work_mat = copy(mat)
143     for col in 2:(size(mat)[2]-1)
144         for row in 2:(size(mat)[1]-1)
145             ## Make the inside 0, we only want the outline
```

```
147     neighbourhood = mat[row-1:row+1,col-1:col+1]
148     if sum(neighbourhood) >= 9 # 9 squares
149         work_mat[row,col] = 0
150     end
151     end
152     return work_mat
153 end
154
155
156 #####
157 ##### Return many Scaled Values #####
158 #####
159 #####
160
161
162
163 function scaleAndMeasure(min, max, n, func)
164     # The scale is equivalent to the resolution, the initial resolution could be
165     # set as 10, 93, 72 or 1, it's arbitrary (previously I had res and scale)
166     # #TODO: Prove this
167
168     scale = [Int(ceil(i)) for i in range(min, max, length=n) ]
169     mass = pmap(s -> sum(outline(make_picture(Int(s), Int(s), func))) , scale)
170
171     data = DataFrame(scale = scale, mass = mass)
172     return data
173 end
```

References

- [1] *Astropy*. URL: <https://www.astropy.org/> (visited on 10/20/2020) (cit. on p. 19).
- [2] Michael Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Texts in Computational Science and Engineering 9. 1. Two motivating examples: sequential orders on quadtrees and multidimensional data structures – 2. How to construct space-filling curves – 3. Grammar-based description of space-filling curves – 4. Arithmetic representation of space-filling curves – 5. Approximating polygons – 6. Sierpinski curves – 7. Further space-filling curves – 8. Space-filling curves in 3D – 9. Refinement trees and space-filling curves – 10. Parallelisation with space-filling curves – 11. Locality properties of space-filling curves – 12. Sierpinski curves on triangular and tetrahedral meshes – 13. Case study: cache efficient algorithms for matrix operations – 14. Case study: numerical simulation on spacetree grids using space-filling curves – 15. Further applications of space-filling curves: references and readings – A. Solutions to selected exercises. Heidelberg ; New York: Springer, 2013. 278 pp. ISBN: 978-3-642-31045-4 (cit. on p. 6).
- [3] Benedetta Palazzo. *The Numbers of Nature: The Fibonacci Sequence*. June 27, 2016. URL: <http://www.eniscuola.net/en/2016/06/27/the-numbers-of-nature-the-fibonacci-sequence/> (visited on 08/28/2020) (cit. on p. 36).
- [4] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: *SIAM Review* 59.1 (Jan. 2017), pp. 65–98. ISSN: 0036-1445, 1095-7200. DOI: [10.1137/141000671](https://doi.org/10.1137/141000671). URL: <https://pubs.siam.org/doi/10.1137/141000671> (visited on 08/28/2020) (cit. on pp. 13, 50).
- [5] Simon Brass. *CC Search*. 2006, September 5. URL: <https://search.creativecommons.org/> (visited on 08/28/2020) (cit. on p. 53).
- [6] *CTAN: Package Animate*. URL: <https://ctan.org/pkg/animate?lang=en> (visited on 10/21/2020) (cit. on p. 3).
- [7] Gerald A. Edgar. *Measure, Topology, and Fractal Geometry*. 2nd ed. Undergraduate Texts in Mathematics. New York: Springer-Verlag, 2008. 268 pp. ISBN: 978-0-387-74748-4 (cit. on pp. 3, 4, 8, 67).
- [8] Gerald A. Edgar. *Measure, Topology, and Fractal Geometry*. 2nd ed. Undergraduate Texts in Mathematics. New York: Springer-Verlag, 2008. 268 pp. ISBN: 978-0-387-74748-4 (cit. on p. 9).
- [9] K. J. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. 2nd ed. Chichester, England: Wiley, 2003. 337 pp. ISBN: 978-0-470-84861-6 978-0-470-84862-3 (cit. on pp. 4, 5, 8, 9, 12, 67).
- [10] K. J. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. 2nd ed. Chichester, England: Wiley, 2003. 337 pp. ISBN: 978-0-470-84861-6 978-0-470-84862-3 (cit. on pp. 12, 51).

REFERENCES

- [11] Peter Farrell. *Math Adventures with Python: An Illustrated Guide to Exploring Math with Code*. Drawing polygons with Turtle – Doing arithmetic with lists and loops – Guessing and checking with conditionals – Solving equations graphically – Transforming shapes with geometry – Creating oscillations with trigonometry – Complex numbers – Creating 2D/3D graphics using matrices – Creating an ecosystem with classes – Creating fractals using recursion – Cellular automata – Solving problems using genetic algorithms
Includes index. San Francisco: No Starch Press, 2019. 276 pp. ISBN: 978-1-59327-867-0 (cit. on p. 54).
- [12] *Fractal*. In: *Wikipedia*. Sept. 15, 2020. URL: <https://en.wikipedia.org/w/index.php?title=Fractal&oldid=978464038> (visited on 10/19/2020) (cit. on p. 4).
- [13] *functools – Higher-Order Functions and Operations on Callable Objects – Python 3.8.5 Documentation*. URL: <https://docs.python.org/3/library/functools.html> (visited on 08/25/2020) (cit. on p. 36).
- [14] Robert Gilmore and Marc Lefranc. *The Topology of Chaos: Alice in Stretch and Squeezeland*. New York: Wiley-Interscience, 2002. 495 pp. ISBN: 978-0-471-40816-1 (cit. on p. 5).
- [15] Ralph Gomory. “Benoît Mandelbrot (1924–2010)”. In: *Nature* 468.7322 (7322 Nov. 2010), pp. 378–378. ISSN: 1476-4687. DOI: [10.1038/468378a](https://doi.org/10.1038/468378a). URL: <https://www.nature.com/articles/468378a> (visited on 10/19/2020) (cit. on pp. 3, 4).
- [16] Jean-François Gouyet. *Physics and Fractal Structures*. Published with the support of ministère de l’Enseignement supérieur et de la Recherche (France) : Direction de l’information scientifique et technique et des bibliothèques-t.p. verso. Paris : New York: Masson ; Springer, 1996. 234 pp. ISBN: 978-0-387-94153-0 978-3-540-94153-8 978-2-225-85130-8 (cit. on pp. 4, 5, 67).
- [17] Ronald L. Graham, Donald Ervin Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd ed. Reading, Mass: Addison-Wesley, 1994. 657 pp. ISBN: 978-0-201-55802-9 (cit. on p. 8).
- [18] Roozbeh Hazrat. *Mathematica®: A Problem-Centered Approach*. 2nd ed. 2015. Springer Undergraduate Mathematics Series. Introduction – Basics – Defining functions – Lists – Changing heads! – A bit of logic and set theory – Sums and products – Loops and repetitions – Substitutions, Mathematica rules – Pattern matching – Functions with multiple definitions – Recursive functions – Linear algebra – Graphics – Calculus and equations – Worked out projects – Projects – Solutions to the Exercises – Further reading – Bibliography – Index. Cham: Springer International Publishing : Imprint: Springer, 2015. 1 p. ISBN: 978-3-319-27585-7. DOI: [10.1007/978-3-319-27585-7](https://doi.org/10.1007/978-3-319-27585-7) (cit. on p. 36).
- [19] jan wassenaar. *Cantor Dust*. Jan. 7, 2005. URL: <http://www.2dcurves.com/fractal/fractald.html> (visited on 10/20/2020) (cit. on p. 14).
- [20] Junwei Jiang and Roy E. Plotnick. “Fractal Analysis of the Complexity of United States Coastlines”. In: *Mathematical Geology* 30.5 (July 1, 1998), pp. 535–546. ISSN: 1573-8868. DOI: [10.1023/A:1021790111404](https://doi.org/10.1023/A:1021790111404). URL: <https://doi.org/10.1023/A:1021790111404> (visited on 10/19/2020) (cit. on p. 4).

REFERENCES

- [21] *Julia Set*. In: *Wikipedia*. July 12, 2020. URL: https://en.wikipedia.org/w/index.php?title=Julia_set&oldid=967264809 (visited on 08/25/2020) (cit. on p. 56).
- [22] *JuliaAstro · JuliaAstro*. URL: <https://juliaastro.github.io/dev/index.html> (visited on 10/20/2020) (cit. on p. 19).
- [23] *JuliaGraphics/Luxor.jl*. JuliaGraphics, Oct. 19, 2020. URL: <https://github.com/JuliaGraphics/Luxor.jl> (visited on 10/20/2020) (cit. on p. 19).
- [24] *JuliaImages/Images.jl*. JuliaImages, Oct. 13, 2020. URL: <https://github.com/JuliaImages/Images.jl> (visited on 10/20/2020) (cit. on p. 19).
- [25] Robert Lamb. *How Are Fibonacci Numbers Expressed in Nature?* June 24, 2008. URL: <https://science.howstuffworks.com/math-concepts/fibonacci-nature.htm> (visited on 08/28/2020) (cit. on p. 36).
- [26] Ron Larson and Bruce H. Edwards. *Elementary Linear Algebra*. 2nd ed. Includes index. Lexington, Mass: D.C. Heath, 1991. 592 pp. ISBN: 978-0-669-24592-9 (cit. on p. 5).
- [27] Eric Lehman, Tom Leighton, and Albert Meyer. *Readings / Mathematics for Computer Science / Electrical Engineering and Computer Science / MIT OpenCourseWare*. Sept. 8, 2010. URL: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/readings/> (visited on 08/10/2020) (cit. on p. 42).
- [28] Oscar Levin. *Solving Recurrence Relations*. Jan. 29, 2018. URL: http://discrete.openmathbooks.org/dmoi2/sec_recurrence.html (visited on 08/11/2020) (cit. on p. 43).
- [29] Zhong Li, Wolfgang A. Halang, and G. Chen, eds. *Integration of Fuzzy Logic and Chaos Theory*. Studies in Fuzziness and Soft Computing v. 187. Berlin ; New York: Springer, 2006. 625 pp. ISBN: 978-3-540-26899-4 (cit. on p. 7).
- [30] *List of Fractals by Hausdorff Dimension*. In: *Wikipedia*. Sept. 8, 2020. URL: https://en.wikipedia.org/w/index.php?title=List_of_fractals_by_Hausdorff_dimension&oldid=977401154 (visited on 09/24/2020) (cit. on p. 12).
- [31] ImageMagick Studio LLC. *ImageMagick*. URL: <https://imagemagick.org/> (visited on 10/20/2020) (cit. on p. 19).
- [32] Benoit Mandelbrot. “How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension”. In: *Science* 156.3775 (May 5, 1967), pp. 636–638. ISSN: 0036-8075, 1095-9203. doi: [10.1126/science.156.3775.636](https://doi.org/10.1126/science.156.3775.636). pmid: [17837158](https://pubmed.ncbi.nlm.nih.gov/17837158/). URL: <https://science.sciencemag.org/content/156/3775/636> (visited on 10/19/2020) (cit. on p. 4).
- [33] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. Includes index. San Francisco: W.H. Freeman, 1982. 460 pp. ISBN: 978-0-7167-1186-5 (cit. on pp. 3–5).
- [34] Benoit B. Mandelbrot and Richard L. Hudson. *The (Mis)Behaviour of Markets: A Fractal View of Risk, Ruin, and Reward*. Pbk. ed. London: Profile, 2008. 326 pp. ISBN: 978-1-84668-262-9 (cit. on p. 4).

REFERENCES

- [35] Mark Pollicott. *Fractals and Dimension Theory*. 2005-May. URL: <https://warwick.ac.uk/fac/sci/math/peop/ma...> (cit. on p. 12).
- [36] Curtis T. McMullen. “Hausdorff Dimension and Conformal Dynamics, III: Computation of Dimension”. In: *American Journal of Mathematics* 120.4 (1998), pp. 691–721. ISSN: 0002-9327. JSTOR: [25098619](https://www.jstor.org/stable/25098619) (cit. on p. 29).
- [37] Nikoletta Minarova. “The Fibonacci Sequence: Nature’s Little Secret”. In: *CRIS - Bulletin of the Centre for Research and Interdisciplinary Study* 2014.1 (2014), pp. 7–17. ISSN: 1805-5117 (cit. on p. 36).
- [38] *Nature, The Golden Ratio and Fibonacci Numbers*. 2018. URL: <https://www.mathsisfun.com/numbers/nature-golden-ratio-fibonacci.html> (visited on 08/28/2020) (cit. on pp. 36, 50).
- [39] Olympia Nicodemi, Melissa A. Sutherland, and Gary W. Towsley. *An Introduction to Abstract Algebra with Notes to the Future Teacher*. Includes bibliographic references (S. 391-394) and index. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. 436 pp. ISBN: 978-0-13-101963-8 (cit. on p. 43).
- [40] *Pdftex - Is There Any Way to Include an Animated GIF Directly?* URL: <https://tex.stackexchange.com/questions/5396/is-there-any-way-to-include-an-animated-gif-directly> (visited on 10/21/2020) (cit. on p. 3).
- [41] Heinz-Otto Peitgen, H. Jürgens, and Dietmar Saupe. *Chaos and Fractals: New Frontiers of Science*. 2nd ed. New York: Springer, 2004. 864 pp. ISBN: 978-0-387-20229-7 (cit. on pp. 5, 14, 21, 51, 67).
- [42] *Pillow—Pillow (PIL Fork) 8.0.0 Documentation*. URL: <https://pillow.readthedocs.io/en/stable/> (visited on 10/20/2020) (cit. on p. 19).
- [43] R Core Team. *R: A Language and Environment for Statistical Computing*. manual. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: <https://www.R-project.org/> (cit. on p. 13).
- [44] Ron Knott. *The Fibonacci Numbers and Golden Section in Nature - 1*. Sept. 25, 2016. URL: <http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibnat.html> (visited on 08/28/2020) (cit. on p. 36).
- [45] Grant Sanderson, director. *Fractals Are Typically Not Self-Similar*. Jan. 27, 2017. URL: https://www.youtube.com/watch?v=gB9n2gHsHN4&lc=Ugj8mIhm_S17y3gCoAEC (visited on 10/19/2020) (cit. on pp. 3, 5, 6, 9).
- [46] Shelly Allen. *Fibonacci in Nature*. URL: <https://fibonacci.com/nature-golden-ratio/> (visited on 08/28/2020) (cit. on p. 36).
- [47] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Second edition. Overview – One-dimensional flows – Flows on the line – Bifurcations – Flows on the circle – Two-dimensional flows – Linear systems – Phase plane – Limit cycles – Bifurcations revisited – Chaos – Lorenz equations – One-dimensional maps – Fractals – Strange attractors. Boulder, CO: Westview Press, a member of the Perseus Books Group, 2015. 513 pp. ISBN: 978-0-8133-4910-7 (cit. on p. 6).

REFERENCES

- [48] Sergei Tabachnikov. “Dragon Curves Revisited”. In: *The Mathematical Intelligencer* 36.1 (Feb. 2014), pp. 13–17. ISSN: 0343-6993, 1866-7414. DOI: [10.1007/s00283-013-9428-y](https://doi.org/10.1007/s00283-013-9428-y). URL: <http://link.springer.com/10.1007/s00283-013-9428-y> (visited on 10/20/2020) (cit. on p. 21).
- [49] Tamás Tél, Márton Gruiz, and Katalin Kulacsy. *Chaotic dynamics: an introduction based on classical mechanics*. Cambridge: Cambridge University Press, 2006. ISBN: 9780511335044 9780511334467 9780511333125 9780511803277 9780511333804 9781281040114 9786611040116 9780511567216. URL: <https://doi.org/10.1017/CBO9780511803277> (visited on 08/28/2020) (cit. on pp. 3, 67).
- [50] Jeffrey Ventrella. *Space-Filling Curves Are Not Squares*. Nov. 16, 2014. URL: <https://spacefillingcurves.wordpress.com/2014/11/16/space-filling-curves-are-not-squares/> (visited on 10/20/2020) (cit. on p. 21).
- [51] Tamás Vicsek. *Fractal Growth Phenomena*. 2nd ed. Singapore ; New Jersey: World Scientific, 1992. 488 pp. ISBN: 978-981-02-0668-0 978-981-02-0669-7 (cit. on pp. 3, 13, 14, 26, 67).
- [52] *Welcome to Python.Org*. URL: <https://www.python.org/> (visited on 10/20/2020) (cit. on p. 13).
- [53] Xiaojing Zhong, Peng Yu, and Shenliang Chen. “Fractal Properties of Shoreline Changes on a Storm-Exposed Island”. In: *Scientific Reports* 7.1 (1 Aug. 15, 2017), p. 8274. ISSN: 2045-2322. DOI: [10.1038/s41598-017-08924-9](https://doi.org/10.1038/s41598-017-08924-9). URL: <https://www.nature.com/articles/s41598-017-08924-9> (visited on 10/19/2020) (cit. on p. 4).
- [54] Xiao-hua Zhu and Jian Wang. “On Fractal Mechanism of Coastline”. In: *Chinese Geographical Science* 12.2 (June 1, 2002), pp. 142–145. ISSN: 1993-064X. DOI: [10.1007/s11769-002-0022-z](https://doi.org/10.1007/s11769-002-0022-z). URL: <https://doi.org/10.1007/s11769-002-0022-z> (visited on 10/19/2020) (cit. on p. 4).
- [55] Dennis G Zill and Michael R Cullen. *Differential Equations*. 7th ed. Brooks/Cole, 2009 (cit. on pp. 43, 44).
- [56] Dennis G. Zill and Michael R. Cullen. “8.4 Matrix Exponential”. In: *Differential Equations with Boundary-Value Problems*. 7th ed. Includes index. Belmont, CA: Brooks/Cole, Cengage Learning, 2009. ISBN: 978-0-495-10836-8 (cit. on pp. 43, 46).