The Emergence of Patterns in Nature and Chaos Theory

Ryan Greenup & James Guerra

October 21, 2020

Contents

Section Attribution

```
1. Introduction                                          :Ryan:
2. Fractals
.. 1. Definition of a Fractal                            :Ryan:
.. 2. Fractals Generally                                :James:
.. 3. Fractal Dimension
..... 1. Hausdorff Dimension                             :Ryan:
..... 2. Box Counting                                   :James:
.. 4. Generating Self Similar Fractals                   :Ryan:
..... 1. Examples
........ 1. Vicsek Fractal
........ 2. Sierpinskis Carpet
........ 3. Triangle
.......... 1. Chaos Game
.......... 2. Pascals Triange                           :James:
..... 2. Turtle
.. 5. Fractal Dimensions                                 :Ryan:
3. Connecting Fractals to Natural Processes              :Ryan:
4. The Fibonacci Sequence
.. 1. Introduction                                       :Ryan:
.. 2. Computational Approach                             :Ryan:
.. 3. Exponential Generating Functions
```

1 Introduction RYAN

- Fractals occur in nature
  - The definition of a fractal is greatly concerned with its dimension so we discuss that
  - Linear Regression can be used to measure dimension so we implement that
- My Fractal is an example of why these things occur in nature
- Prove the Fibonacci Numbers, show the relationship between
  - MCT approach
  - ODEs
- The Fibonacci numbers (Veritasium), by logistic regression, are connected with:
- Julia Set
  - Mandelbrot
  - So we solve the fractal dimension of these

2 Fractals

2.1 Definition of a Fractal RYAN

Benoît Mandelbrot coined the term fractal in 1975 [14] and defined it his 1982 book *The Fractal Geometry of Nature* [32, p. 15] :

> *A fractal is by definition a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension.*
>
> *a Every set with a noninteger D is a fractal.*

The topoligical dimension is a strictly integer value that describes the natural dimension used to describe a shape [42], for example the *Koch Snowflake* (shown in figure 2) is composed of just a line, so it's topological dimension would simply be 1, it's *fractal dimension* however is shown to be $\frac{\ln(4)}{\ln(3)}$ at (**??**) in §2.3.1.

Many authors seem to accept this earlier definition (see e.g. [48, §2.2] and [46, §2.1]), this definition however does not capture many edge-case fractals [6, p. VII] and in reprintings of *The Fractal Geometry of Nature* Mandelbrot himself commented that in hindsight it may have been more appropriate [32, 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./

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*" [6, 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 report [1]. no strict definition will be used for fractals, but, guiding principles implemented by other authors will be used.

Some authors simply define a fractal as a shape that *shows irregularities at all scales* [15, p. 1] and in his 2003 book, *Fractal Geometry*, Facloner it is more convenient to describe a fractal by a list of properties characteristic of such shapes [2] because of the difficulty in defining a fractal in a way that can encompass all edge cases [8, 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 to refer to complex natural shapes as fractals, such as coastlines (see e.g. [19, 51, 50]) much in the spirit of Mandelbrot's paper *How long*

[1] Mandelbrot also discussed fractals of a Euclidean and Reimannian nature, [32, p. 361], this again is interesting but too specific for the broad nature of our investigation

[2] Much like the definition of life in the field of biology

is the Coastline of Britain [31], although he coined the term fractal many years after this paper, presumably he might have had this in mind [3] 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 [11] also points out that a fractal is nowhere differentiable, this would be because a fractal is nowhere smooth, which I think personally is quite a distinguishing feature.

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 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 distiguishes 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 [25]) is strictly an integer value, but in the early 20^{th} century mathematicians recognised the shortcomings of this definition [32, 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 [32, Ch. 39] and [15, §1.3]) of these the *Hausdorff Dimension* (and corresponding *Hausdorff Measure*) is considered to be the most

[3] Mandelbrot 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. [14, 33]

important and mathematically robust [8, p. 27], while the box counting dimension has the most practical applications in science [38, p. 192].

The Hausdorff dimension is more like counting balls than boxes and is identical to the Hausdorff dimension in many cases, it's more general but harder to define [42], 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

Topology is an area of mathematics concerned with ideas of continuity through the study of figures that are preserved under homeomorphic transformations [13] , where two figures are said to be homeomorphic if there is a continuous bijective mapping between the two shapes [38, p. 105] . [4]

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 acheive that.

As mentioned above, historically, the concept of dimension was a difficult problem with a tenuous definition. Although an inutitive 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 the perimeter is given by $p_n = \left(\frac{4}{3}\right) p_{n-1}$ and the number of edges by:

$$N_n = N_{n-1} \cdot 4 \tag{1}$$
$$= 3 \cdot 4^n \tag{2}$$

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:

$$l = \frac{s \cdot l_0}{3^n} \tag{3}$$

The total perimeter would be given by:

[4] For further reading on this topic see [38, p. 106]

$$p_n = N_n \times l \tag{4}$$
$$= 3 \cdot 4^n \times \frac{s \cdot l_o}{3^n} \tag{5}$$
$$= 3 \cdot s \cdot l_0 \left(\frac{4}{3}\right)^n \tag{6}$$
$$\implies p_n \cdot s \propto \left(\frac{4}{3}\right)^n \tag{7}$$
$$\implies n = \frac{\log(4)}{\log(3)} \approx 1.26 \tag{8}$$

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 by proportional to 1.26, this should hopefully motivate the need to more clearly define both the concept of measure (in this case the permiter [5]) 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 [44, 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 [44, 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 it won't be feasible however, [28, p. 393] for example with the julia set or the fractal or a coastline it is not immediately clear if the the dimension would be constant at all scales [6].

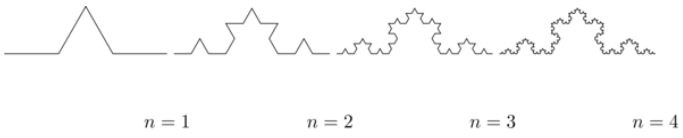


Figure 1: Progression of the Koch Snowflake

2.3.2 Hausdorff Dimension RYAN

2.3.3 Hausdorff Measure

[5] Grant Sanderson equates the measure of a fractal as analogous to mass, which is a very helpful way comparison [42]

[6] It is indeed shown to be mostly constant at all scales in section §

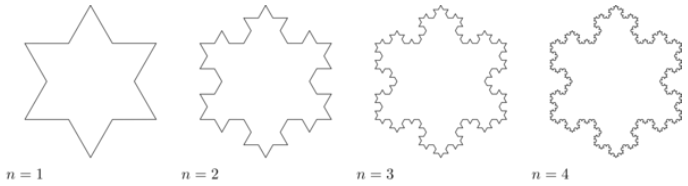


Figure 2: Progression of the Koch Snowflake

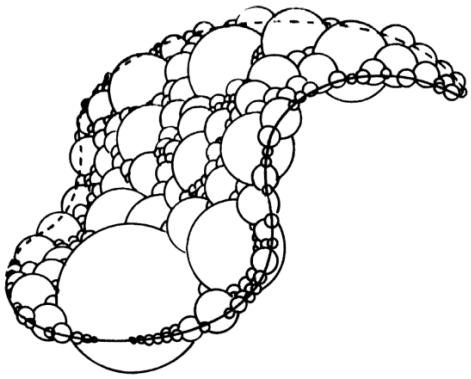


Figure 3: The Hausdorff Measure, in this case area, of an arbitrary surface approximated by the cross section of litte balls of diameter $< \delta$, this is reproduced from *Measure, Topology and Fractal Geometry* [6, p. 166] because it so elegantly illustrates the concept of the Hausdorff Measure.

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. [7]

Let F be some arbitrary subset of euclidean space $\mathbb{R}^n$, [8]

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

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

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 ontop of F then G is said to be a δ -cover of F, more rigorously, G is a δ-cover of F if: [9]

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

[7] See §7.3 for further reading.

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

[9] Falconer defines this as $\bigcup_{i=1}^{\infty}$ [8, §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 or G (or ∞ if it is uncountable). The use of $\#\square$ to denote cardinality was introduced by Knuth in *Concrete Mathematics* [16] and is convenenient in that it avoids any ambiguity with diamater ($|\square|$).

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 [8, §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
- 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. [42]), 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 [8, §2.4] because any set of diamater r can be enclosed in a ball of radius $\frac{r}{2}$ [7, 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 diamater of each element of $\{U_i\}$ and considers $\sum_{U\in G}\left[|U|^s\right]$ where each element $U \in G$ is arranged so as to minimize the value of the summation [8, p. 27] , the δ-Hausdorff is hence defined, for various dimensions s:

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

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 $\mathcal{H}^2_\delta(F)$ is equivalent to considering the number of shapes $U \in G$ (e.g. boxes, discs etc.), of diamater $\leq \delta$ that will cover over a shape as shown in figure 4, the delta Haussendorf measure $\mathcal{H}^2_\delta(F)$ will be the area of the boxes when arranged in such a way that minimises the area.

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

$$\mathcal{H}^s = \lim_{\delta\to 0}\left(\mathcal{H}^s_\delta\right) \tag{11}$$

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 $(4\times 1^2 + 4\times\pi\times\frac{1}{2^2} + \frac{1}{2}\times 1\times\sin\frac{\pi}{3})$.

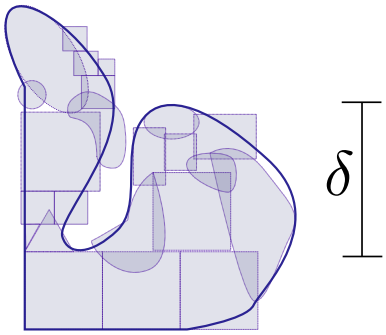


Figure 4: The blue outline corresponds to some $F \subset \mathbb{R}^2$, covered by various grey objects, each of which represent an element from the set U_i. The grey shapes all have a diameter less than δ and so this $\bigcup [U_i]$ would be a δ-covering of F.

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 §2.3.3 the value of $|U_i|^1$ would be considered as opposed to $|U_i|^2$ (i.e. the diamater as opposed to the diameter squared).

As δ is made arbitrarily small the boxes [10] 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 factals 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:

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

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:

[10] Even 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

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

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

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

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

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

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

$$\leq 0 \tag{15}$$

and so we have the following relationship:

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

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

2.3.4 Hausdorff Dimension

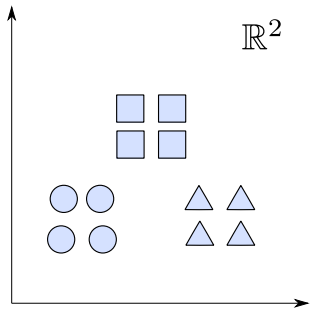


Figure 6: A disconnected subset of $\mathbb{R}^2$, the squares have a diameter of $\sqrt{2}$, the circles 1 and the equilateral triangles 1.

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* [8, §2.2], it is a generalisation of the idea of dimension that is typically understood with respect to ordinary figures.

2.3.5 Box Counting DimensionJames

While the Hasudorff dimension is the first formal definition to measure the

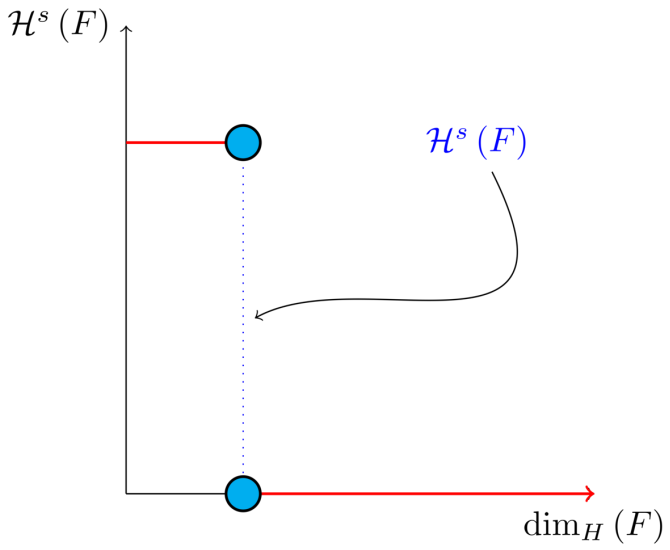


Figure 5: The value of the s-dimensional *Hausdorff Measure* of some subset of *Euclidean space* $F \in \mathbb{R}^n$ is 0 or ∞ when the dimension of F is not equal to s.

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 [8, p. 41] and in many cases is equal to the *Hausdorff Dimension* [34, p. 11] (see generally [29]). The box-counting dimension is defined as the following from [9]:

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

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

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

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

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

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 $\frac{1}{\delta}$ small squares, and so to cover the entire square, one would need n^2 small squares, i.e. $N_{\frac{1}{n}}(F) = n^2$. Now, substituting these values into the box-counting definiton, we get:

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

Which is expected, becuase 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.

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

2.4 Generating Self Similar Fractals — RYAN

In order to investigate the dimension of fractals, we intend to generate and measure a variety of figures by using of ***R*** [40], *Julia* [4] and *Python* [49].

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

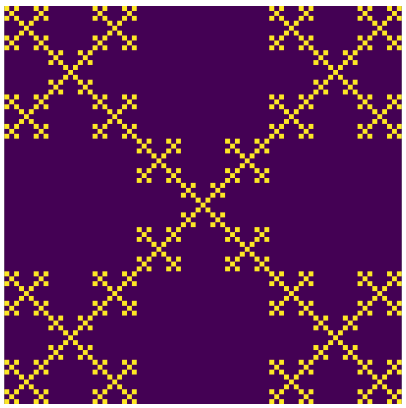


Figure 7: Vicsek fractal [48, p. 12] (also known as the *AntiCross-Stictch* [18]) produced by listing 1, at each iterative step the fractal itself is "copied" to the four corners of itself producing this complex shape.

The Vicsek Fractal [48, p. 12] involves a pattern of iterating boxes, to implement this consider the process[11]:

$$\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{B} & \mathbf{Z} & \mathbf{B} \\ \mathbf{Z} & \mathbf{B} & \mathbf{Z} \\ \mathbf{B} & \mathbf{Z} & \mathbf{B} \end{bmatrix} \tag{17}$$

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).

The iterative process shown in (17) is represented as a recursive function at line 5 of listing 1 and plotted immediately after. To measure the the dimension of this fractal a the sum of the matrix is taken to be the measure of the fractal, two fractals are generated and the change in size relative to the scale is compared and the log taken to return the value of the dimension:

$$\mathcal{D} = \frac{s}{m_2/m_1}$$

The recursive function begins with a 3x3 matrix, where the four corner squares and middle square are set to 1 and the rest are set to 0, a new matrix is built by joining together the past matrix following the rule described in (17). The function repeats until it reaches some arbitrary set width.

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:

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

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

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.

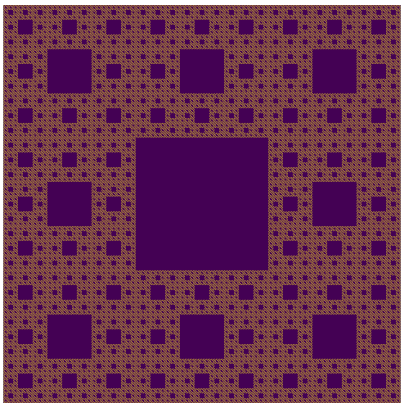


Figure 8: Sierpinksi's Carpet, produced by listing .

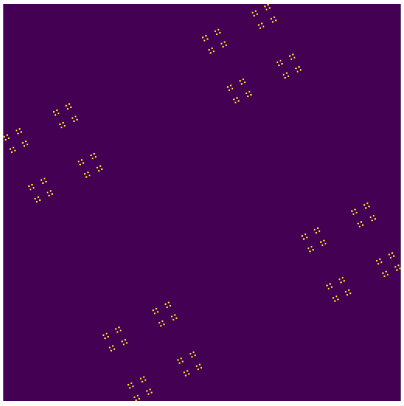


Figure 9: A shape this is similar, but different to, the *Cantor Dust*, the dimension of this fractal is 1, produced by listing 2.

Sierpinskis Carpet and Cantor's Dust

```julia
#-------------------------------------------------------------
#--- Function ------------------------------------------------
#-------------------------------------------------------------
function visek_matrix(ICMat, width)
    B = ICMat
    h  = size(B)[1]
    w  = size(B)[2]
    Z  = zeros(Int, h, w)
    B = [B Z B ;
         Z B Z ;
          B Z B]
    if (3*w)<width
        B = visek_matrix(B, width)
    end
    return B
end

#-------------------------------------------------------------
#-- Plot -----------------------------------------------------
#-------------------------------------------------------------
(mat = visek_matrix(fill(1, 1, 1), 27)) |> size
GR.imshow(mat)

#-------------------------------------------------------------
#-- Similarity Dimension -------------------------------------
#-------------------------------------------------------------

mat2  = visek_matrix(fill(1, 1, 1), 1000)
l2    = sum(mat2)
size2 = size(mat2)[1]

mat1  = visek_matrix(fill(1, 1, 1), 500)
l1    = sum(mat1)
size1 = size(mat1)[1]

#-------------------------------------------------------------
julia> log(l2/l1)/log(size2/size1)
1.4649735207179269
julia> log(5)/log(3)
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)

```
#-------------------------------------------------------------
#--- Function ------------------------------------------------
#-------------------------------------------------------------

function dust(ICMat, width)
    B = ICMat
    h  = size(B)[1]
    w  = size(B)[2]
    Z  = zeros(Int, h, w)
    B = [Z Z B Z;
         B Z Z Z;
         Z Z Z B;
         Z B Z Z]
    if (3*w)<width
        B = dust(B, width)
    end
    return B
end

#-------------------------------------------------------------
#-- Plot ---------------------------------------------------
#-------------------------------------------------------------
using GR, Plots
gr() # Set Plots backend as GR

(mat = dust(fill(1, 1, 1), 9^2)) |> size
p1 = GR.imshow(mat)


#-------------------------------------------------------------
#-- Dimension ----------------------------------------------
#-------------------------------------------------------------
mat2 = dust(fill(1, 1, 1), 1000)
l2   = sum(mat2)
size2 = size(mat2)[1]
mat1 = dust(fill(1, 1, 1), 500)
l1   = sum(mat1)
size1 = size(mat1)[1]

#-------------------------------------------------------------
## julia> log(l2/l1)/log(size2/size1)
## 1.0
```

Listing 2: Function to generate Cantor Dust, shown in 9

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

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:

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

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 3 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.

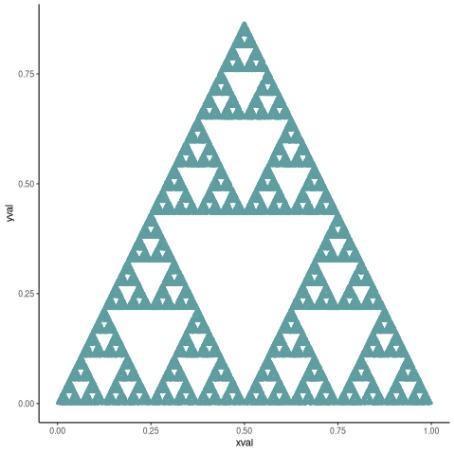


Figure 10: Sierpinski's Triangle created using the *Chaos Game* in listing 3.

Pascals Triange **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

```r
library(ggplot2)

n <- 50000
df <- data.frame("xval"=1:n, "yval"=1:n)

x <- c(runif(1), runif(1))
A <- c(0, 0)
B <- c(1, 0)
C <- c(0.5, sin(pi/3))
points <- list()
points <- list(points, x)


for (i in 1:n) {
    dice = sample(1:3, 1)
    if (dice == 1) {
        x <- (x + A)/2
        df[i,] <- x
    } else if (dice == 2) {
        x <- (x + B)/2
        df[i,] <- x
    } else {


        x <- (x + C)/2
        df[i,] <- x
    }
}

# df

ggplot(df, aes(x = xval, y = yval)) +
    geom_point(size = 1, col = "cadet blue") +
    theme_classic()
```

Listing 3: R code to construct Sierpinksi's triangle through the Choas Game, shown in figure 10.

equlateral *Sierpinski's Triangle*, it does however converge to this value slowly.

```
function pascal(n)
    mat = [isodd(binomial(BigInt(j+i),BigInt(i))) for i in 0:n, j in 0:n]
    return mat
end
GR.imshow(pascal(999))
GR.savefig("../../Report/media/pascal-sierpinsky-triangle.png")

#-------------------------------------------------------------
#-- Calculate Dimension ------------------------------------
#-------------------------------------------------------------

mat2 = pascal(300)
l2   = sum(mat2)
size2 = size(mat2)[1]
mat1 = pascal(200)
l1   = sum(mat1)
size1 = size(mat1)[1]
log(l2/l1)/log(size2/size1)
# https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle
log(3)/log(2)

#-------------------------------------------------------------
julia> log(l2/l1)/log(size2/size1)
1.8177195595512954
julia> log(3)/log(2)
1.5849625007211563
```

Listing 4: Julia code demonstrating Sierpinksi's triangle, this converges to the self similar dimension very slowly, using the ratio between a 3000^2 and 2000^2 matrix gave the correct answer to 2 decimal places, using a 300^2 and 200^2 matrix produced a value far of as shown.

2.4.2 Turtle

Some Fractals cannot be well explained by using matrices or the chaos game, Turtle graphics are a programatic way to draw a pen across a screen, these are implemented in *Julia* using the *Luxor* package [22].

We were unfourtunately 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* [39] or the *Julia Images* library [23], when this was unsucessful we also experimented with *ImageMagick* [30], *AstroPy* [1] and *JuliaAstro* [21]. Unfourtunately 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 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 shown in figure 13 and can be implemented in julia like so:

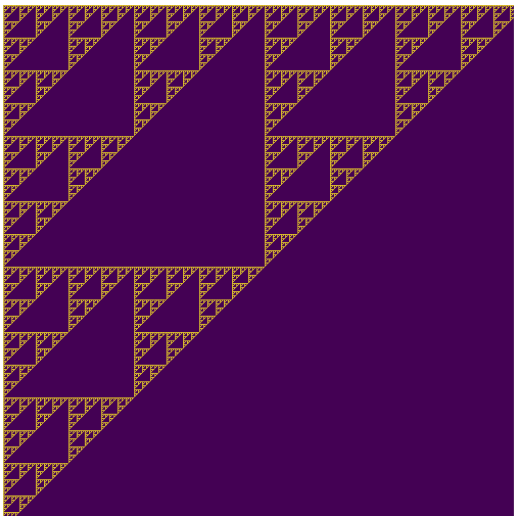


Figure 11: Sierpinski's triangle generated

The dragon curve is slightly more complicated and can be implemented by two seperate functions, one to turn and trigger a motion and the other to control in which direction to turn, this is shown in figure 12 and implemented by the following *Julia* code:

```
using Shapefile
using Luxor

#-------------------------------------------------------------
#--- Dragon ------------------------------------------------
#-------------------------------------------------------------
# Define the Parent Function
function dragon(♘, order, length)
    print(" ") # Don't remove this or code breaks, I don't know why?
    Turn(♘, order*45)
    dragon_iterate(♘, order, length, 1)
end
# Define the Helper Function
function dragon_iterate(♘, order, length, sign)
    if order==0
        Forward(♘, length)
    else
        rootHalf = sqrt(0.5)
        dragon_iterate(♘, order -1, length*rootHalf, 1)
        Turn(♘, sign * -90)
        dragon_iterate(♘, order -1, length*rootHalf, -1)
    end
end
# Draw the Image
@png begin
    ♘ = Turtle()
    # Start from left to centre result
    Turn(♘, 180)
    Penup(♘)
```

```julia
using Shapefile
using Luxor
using Pkg

#-------------------------------------------------------------
#--- Round Snowflake Working ---------------------------------
#-------------------------------------------------------------
function snowflake(length, level, ♘)
    if level == 0
        Forward(♘, length)
        Circle(♘, 1)
        return
    end
    length = length/9
    snowflake(length, level-1, ♘)
    Turn(♘, -60)
    snowflake(length, level-1, ♘)
    Turn(♘, 2*60)
    snowflake(length, level-1, ♘)
    Turn(♘, -60)
    snowflake(length, level-1, ♘)
end

♘ = Turtle()
@png begin
for i in 1:3
    levels = 9
    Pendown(♘)
    snowflake(8^(levels-1), levels, ♘)
    Turn(♘, 120)
end
end 600 600 "snowCurve.png"
```

Listing 5: Generate a Koch Snowflake using a Turtle Diagram

```
    Forward(♘, 200)
    Pendown(♘)
    Turn(♘, 180)
    # Create the Output
    dragon(♘, 15, 400)
end 1000 1000

# Create many images
;mkdir /tmp/dragon
for i in range(1, 15)
name = string("/tmp/dragon/d", lpad(d, 5, "0"), ".png")
    @png begin
        ♘ = Turtle()
        # Start from left to centre result
        Turn(♘, 180)
        Penup(♘)
        Forward(♘, 200)
        Pendown(♘)
        Turn(♘, 180)
        # Create the Output
        dragon(♘, 15, 400)
    end 1000 1000 name
end
montage -geometry 1000x1000 *.png dragon.png
```

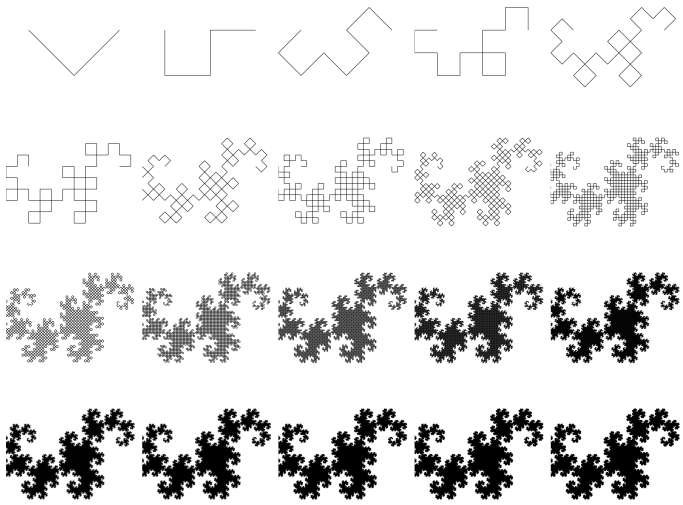


Figure 12: Progression of the Dragon Curve, this is known as a space filling curve [38, p. 350] which is a curve with a range that contains the entire 2-dimensional unit square [47], it has a dimension of two. For some historical background on the curve on the origins of this curve see [45].

2.4.3 Pascals Triangle and Sierpinski's Triangle JAMES

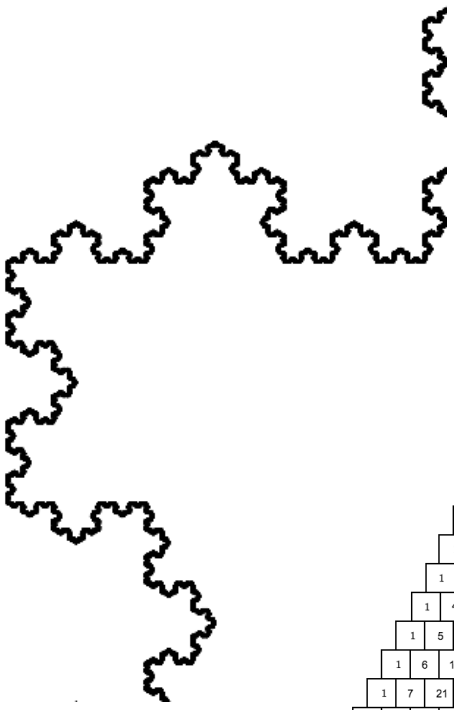


Figure 13: Portion of the Koc
Produced by the Turtle graphi

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 0^{th} (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, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

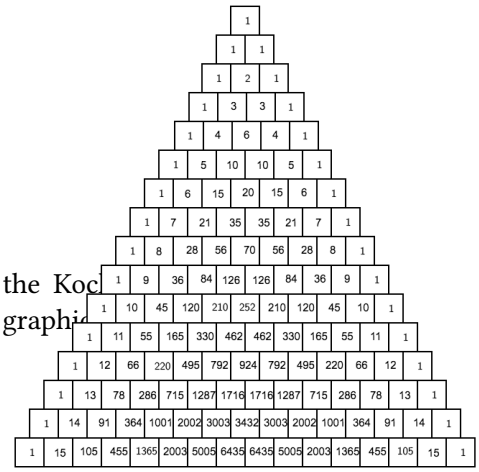
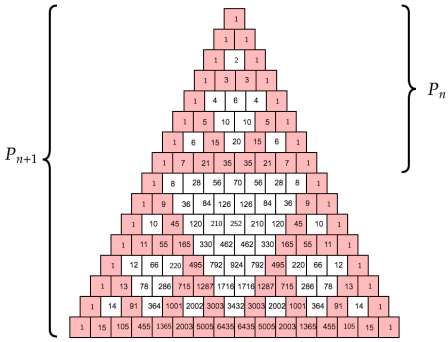


Figure 14: Pascal's triangle

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.



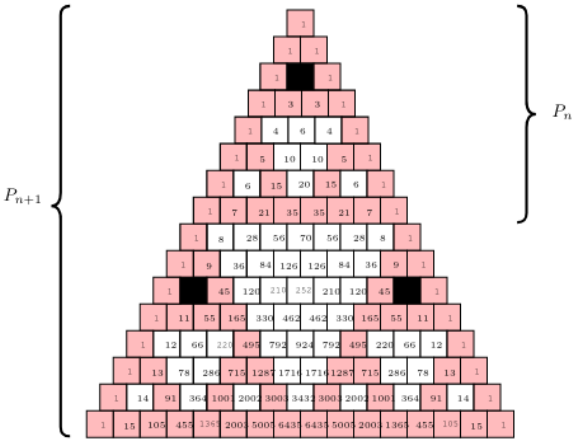


Figure 15: The black squares represent one example of a position on Pascal's triangle that are equivalent modulo 2

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:

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

where,

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

are the expansions in radix p [12]. This uses the convention that $\binom{n}{k} = 0$ if $k < n$

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:

$$\text{Top Triangle:} \quad \binom{r}{c}$$
$$\text{Bottom-left Triangle:} \quad \binom{r+2^n}{c}$$
$$\text{Bottom-right Triangle :} \quad \binom{r+2^n}{c+2^n}$$

[12] Radix 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.

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

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

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

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

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 Set

A value on the complex plane can be associated with the julia set by iterating that value against a function of the form $z \to z^2 + \alpha + i\beta$ 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 convervgent 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 [48, p. 30].

To implement this all the functions necessary to build the fractals were placed into a seperate 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:

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

The julia set is defined as the boundary between values on the complex plane that converge and diferge 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 divergence [13] 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 elemnt is greater than 1. This is shown in the `outline` function which is defined in §2.5.1.

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 §2.5.1

which returs the values shown in table .

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

############################################################
#### Investigate Plot ######################################
############################################################
test_mat = make_picture(800,800, z -> z^2 + -0.123+0.745*im)

#Inspect
GR.imshow(test_mat) # PyPlot uses interpolation = "None"
# Outline
test_mat = outline(test_mat)
#Inspect
GR.imshow(test_mat) # PyPlot uses interpolation = "None"
# GR.savefig("/home/ryan/Dropbox/Studies/2020Spring/QuantProject/Current/Python-Quant/Problems/fractal-dimensions/media/outline-Julia-set.png")

```

[13] abs(z) exceeds to the point is assured to diverge under iteration

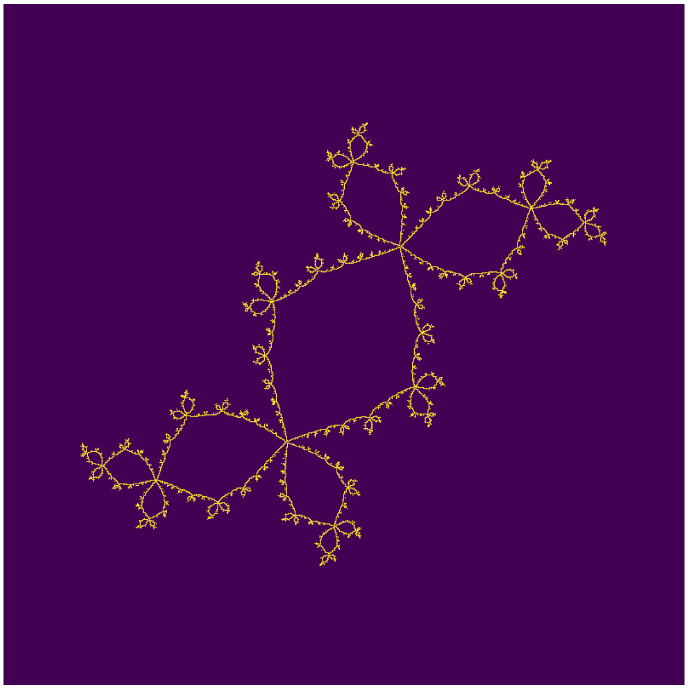


Figure 16: Image of the Doudy Rabbit, the julia set corresponding to the iteration of $z \leftarrow z^2 - 0.123 + 0.745i$ produced by *Julia*

```
#!/bin/julia
function juliaSet(z, num, my_func, boolQ=true)
    count = 1
    # Iterate num times
    while count ≤ num
        # check for divergence
        if real(z)^2+imag(z)^2 > 2^2
            if(boolQ) return 0 else return Int(count) end
        end
        #iterate z
        z = my_func(z) # + z
        count=count+1
    end
        #if z hasn't diverged by the end
    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 $z \to z^2 + \alpha + i\beta$

```
## Return the perimeter
sum(test_mat)


# Take a measurement at a point

mat2 = outline(make_picture(9000,9000, f))
l2   = sum(mat2)
size2 = size(mat2)[1]
mat1 = outline(make_picture(10000,10000, f))
l1   = sum(mat1)
size1 = size(mat1)[1]
log(l2/l1)/log(size2/size1)
# https://en.wikipedia.org/wiki/Vicsek_fractal#Construction
# 1.3934 Douady Rabbit
#

# Take a measurement using LInear Regression
using CSV

@time data=scaleAndMeasure(900, 1000 , 4, f)
# CSV.read("./julia-set-dimensions.csv", data)
# data = CSV.read("./julia-set-dimensions.csv")
data.scale = [log(i) for i in data.scale]
data.mass  = [log(i) for i in data.mass]
mod   = lm(@formula(mass ~ scale), data)
p = Gadfly.plot(data, x=:scale, y=:mass, Geom.point)

print("the slope is $(round(coef(mod)[2], sigdigits=4))")
print(mod)
print("\n")
return mod

a = SharedArray{Float64}(10)
@distributed for i = 1:10
    a[i] = i
end

#-------------------------------------------------------------
julia> return mod
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}}
↪  ,GLM.DensePredChol{Float64,LinearAlgebra.Cholesky{Float64,Array{Float
↪  64,2}}}},Array{Float64,2}}

mass ~ 0 + scale

Coefficients:
─────────────────────────────────────────────────────────────────────────
         Coef.   Std. Error        t  Pr(>|t|)  Lower 95%  Upper 95%
─────────────────────────────────────────────────────────────────────────
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*** [14]

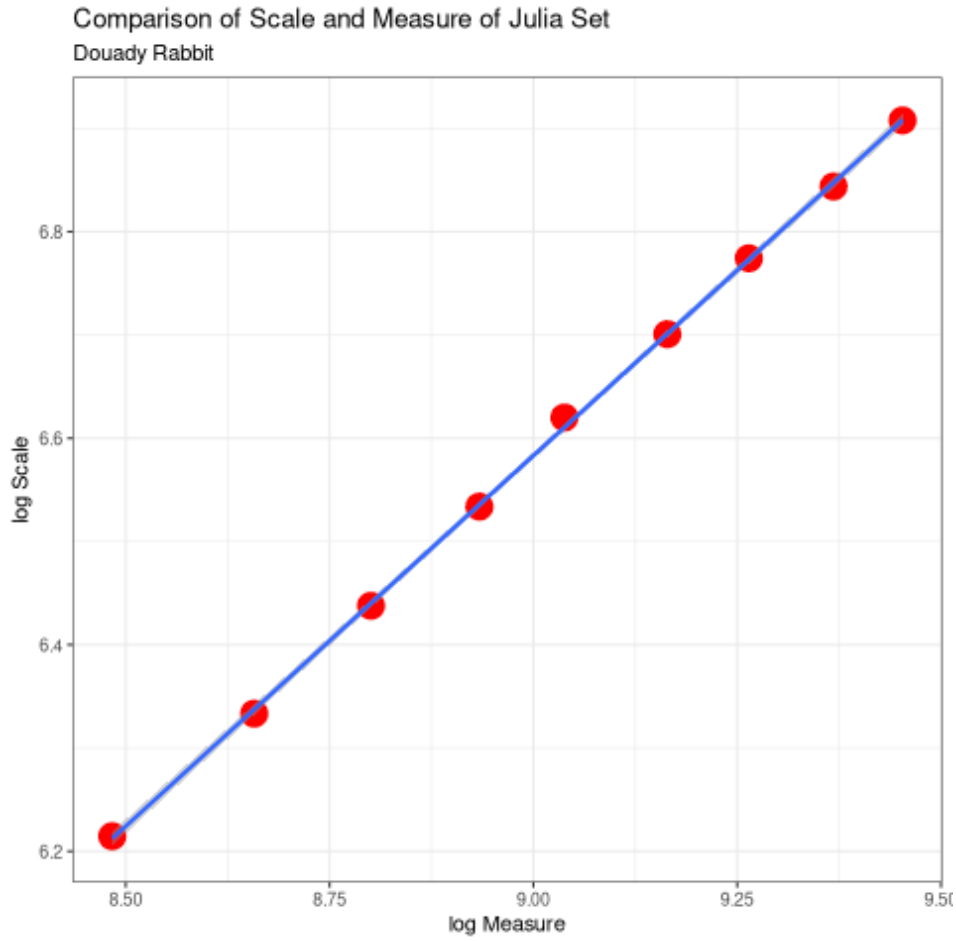
```
scale   <- c(500, 563, 625, 688, 750, 813, 875, 938, 1000)
measure <- c(  4834, 5754, 6640, 7584, 8418, 9550, 10554, 11710, 12744)
data <- data.frame(scale, measure)

lm(log(measure) ~ 0 + log(scale), data)$coefficients # set 0 intercept
```

This shows that the value returned is 1.37 which is very close to the value provided by Wikipedia at 1.39, inspecting the behaviour ofthe log transformed scale and measure indicates that there is a very linear relationship between these variables.

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

[14] The could just as well have been done inside Julia



Functions Julia Set The following functions were saved in a file called `@time include("./Julia-Set-D` this file was loaded into the current workspace by using `@time include("./Julia-Set-Dimensions-fu` at the top of a script.

```
using GR
using DataFrames
using Gadfly
using GLM
using SharedArrays
using Distributed

#############################################################
### Julia / MandelBrot Functions ############################
#############################################################

"""
# Julia Set
```

```
Returns how many iterations it takes for a value on the complex plane to diverge
under recursion. if `boolQ` is specified as true a 1/0 will be returned to
indicate divergence or convergence.

## Variables
- `z`
  - A value on the complex plane within the unit circle
- `num`
  - A number of iterations to perform before conceding that the value is not
    divergent.
- `my_func`
  - A function to perform on `z`, for a julia set the function will be of the
    form `z -> z^2 + a + im*b`
    - So for example the Douady Rabbit would be described by `z -> z^2 -0.123+0.745*im`
"""
function juliaSet(z, num, my_func, boolQ=true)
    count = 1
    # Define z1 as z
    z1 = z
    # Iterate num times
    while count ≤ num
        # check for divergence
        if real(z1)^2+imag(z1)^2 > 2^2
            if(boolQ) return 0 else return Int(count) end
        end
        #iterate z
        z1 = my_func(z1) # + z
        count=count+1
    end
        #if z hasn't diverged by the end
    if(boolQ) return 1 else return Int(count) end
end


"""
# Mandelbrot Set
Returns how many iterations it takes for a value on the complex plane to diverge
under recursion of \$z \\rightarrow z^2 + z_0\$.

Values that converge represent constants of the julia set that lead to a
connected set. (TODO: Have I got that Vice Versa?)


## Variables
- `z`
  - A value on the complex plane within the unit circle
- `num`
  - A number of iterations to perform before conceding that the value is not
    divergent.
- `boolQ`
  - `true` or `false` value indicating whether or not to return 1/0 values
```

```
    indicating divergence or convergence respecitvely or to return the number of
   iterations performed before conceding no divergence.
"""
function mandelbrot(z, num, boolQ = true)
    count = 1
    # Define z1 as z
    z1 = z
    # Iterate num times
    while count ≤ num
        # check for divergence
        if real(z1)^2+imag(z1)^2 > 2^2
            if(boolQ) return 0 else return Int(count) end
        end
        #iterate z
        z1 = z1^2 + z
        count=count+1
    end
        #if z hasn't diverged by the end
    return 1 # Int(num)
    if(boolQ) return 1 else return Int(count) end
end

function test(x, y)
    if(x<1) return x else return y end
end


############################################################
##### Build a Matrix Image #################################
############################################################

"""
# Make a Picture

This maps a function on the complex plane to a matrix where each element of the
matrix corresponds to a single value on the complex plane. The matrix can be
interpreted as a greyscale image.

Inside the function is a `zoom` parameter that can be modified for different
fractals, fur the julia and mandelbrot sets this shouldn't need to be adjusted.

The height and width should be interpreted as resolution of the image.

- `width`
  - width of the output matrix
- `height`
  - height of the output matrix
- `myfunc`
  - Complex Function to apply across the complex plane
"""
function make_picture(width, height, my_func)
    pic_mat = zeros(width, height)
    zoom = 0.3
```

```
    for j in 1:size(pic_mat)[2]
        for i in 1:size(pic_mat)[1]
            x = (j-width/2)/(width*zoom)
            y = (i-height/2)/(height*zoom)
            pic_mat[i,j] = juliaSet(x+y*im, 256, my_func)
        end
    end
    return pic_mat
end

#############################################################
### Make the Outline ########################################
#############################################################

"""
# Outline

Sets all elements with neighbours on all sides to 0.

- `mat`
  - A matrix
    - If this matrix is the convergent values corresponding to a julia set the
      output will be the outline, which is the definition of the julia set.
"""
function outline(mat)
    work_mat = copy(mat)
    for col in 2:(size(mat)[2]-1)
        for row in 2:(size(mat)[1]-1)
            ## Make the inside 0, we only want the outline
            neighbourhood = mat[row-1:row+1,col-1:col+1]
            if sum(neighbourhood) >= 9 # 9 squares
                work_mat[row,col] = 0
            end
        end
    end
    return work_mat
end


#############################################################
###### Return many Scaled Values ############################
#############################################################



function scaleAndMeasure(min, max, n, func)
    # The scale is equivalent to the resolution, the initial resolution could be
    # set as 10, 93, 72 or 1, it's arbitrary (previously I had res and scale)
    # #TODO: Prove this

    scale = [Int(ceil(i)) for i in range(min, max, length=n) ]
    mass = pmap(s -> sum(outline(make_picture(Int(s), Int(s), func))) , scale)

```

```
171         data = DataFrame(scale = scale, mass = mass)
172         return data
173     end
```

3 Connecting Fractals to Natural Processes RYAN

My fractal really shows many unique patterns

If it is scaled by φ then the boxes increase two fold.

We know the dimension will be constant because the figure is self similar, so we have:

$$\dim(\texttt{my\_fractal}) = \log_{\varphi} = \frac{\log \varphi}{\log 2}$$

3.1 Graphics

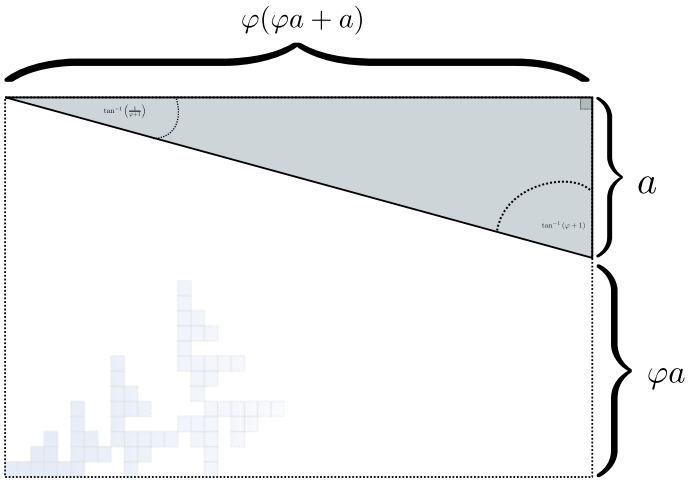


Figure 17: TODO

3.2 Discuss Pattern shows Fibonacci Numbers

3.2.1 Angle Relates to Golden Ratio

3.3 Prove Fibonacci using Monotone Convergence Theorem

Consider the series:

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

Such that:

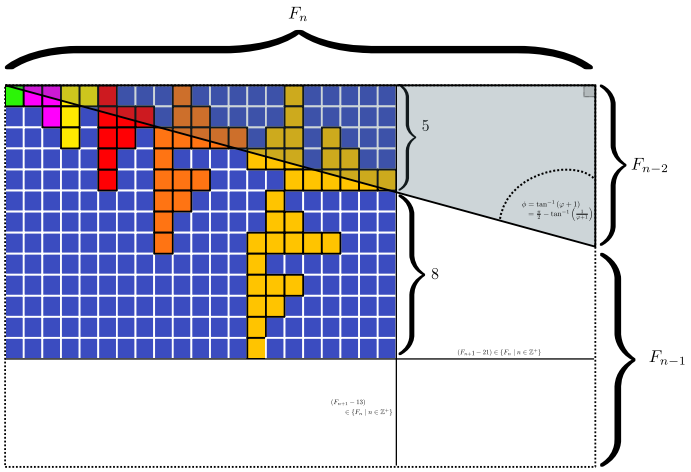


Figure 18: TODO

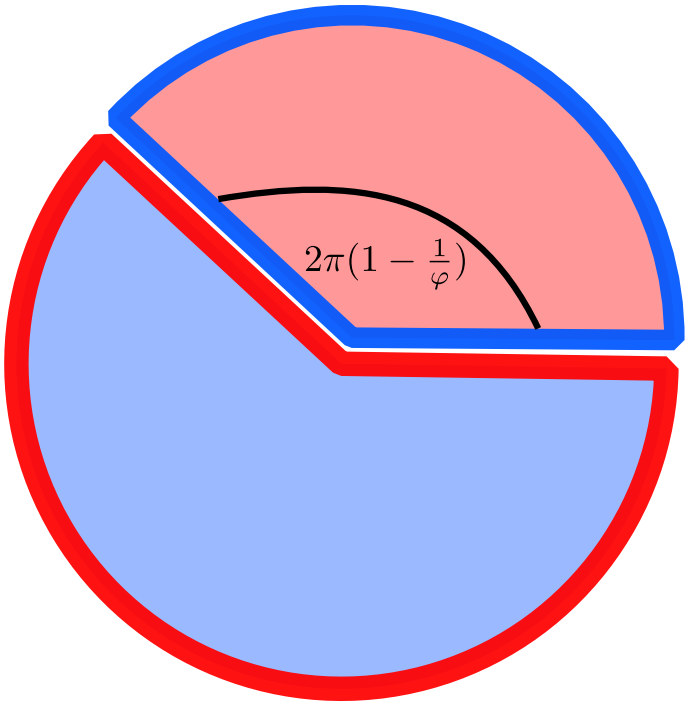


Figure 19: TODO

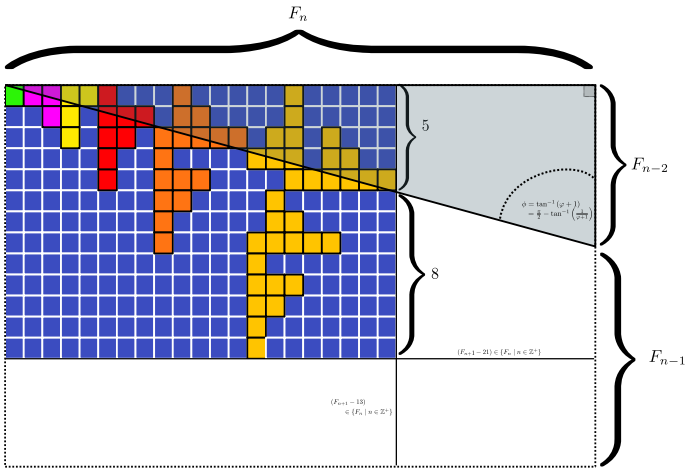


Figure 20: TODO

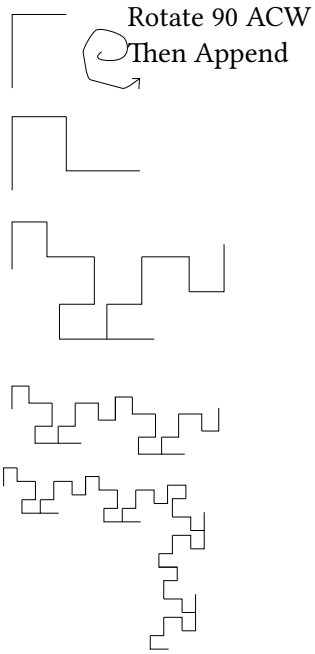


Figure 21: TODO

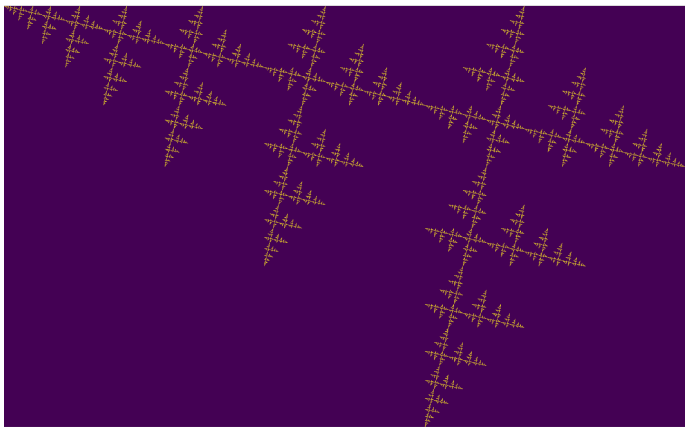


Figure 22: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well

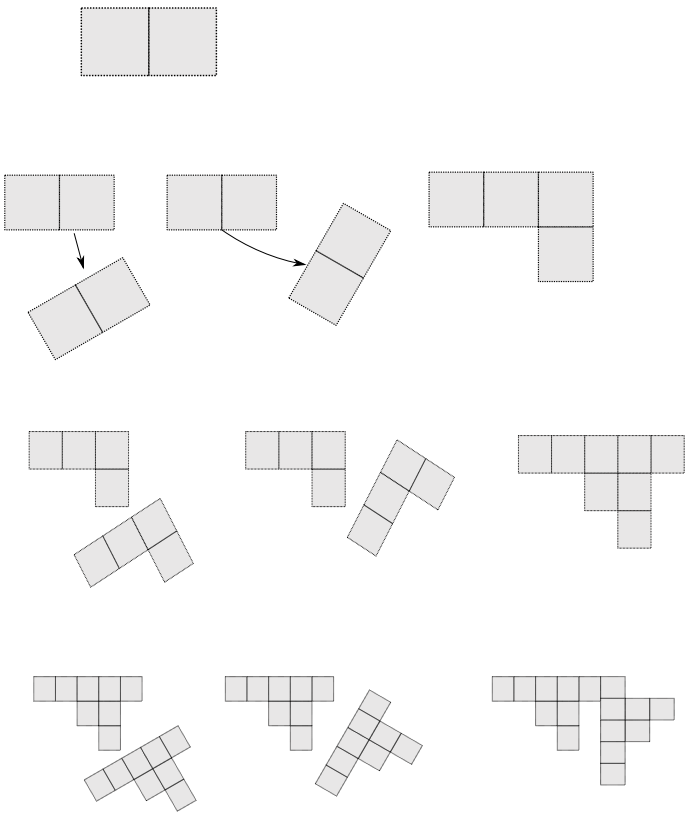
$$F_n = F_{n-1} + F_{n-2}; \quad F_1 = F_2 = 1$$

3.3.1 Show that the Series is Monotone

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

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

3.3.2 Show that the Series is Bounded



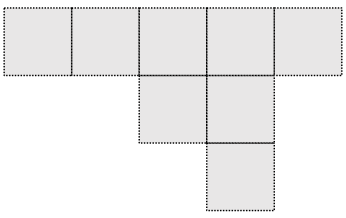


Figure 23: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well

3.3.3 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 $F_n > 0 \forall n$

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

3.3.4 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:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(e^x\right) \ldots$$

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

Observe also that

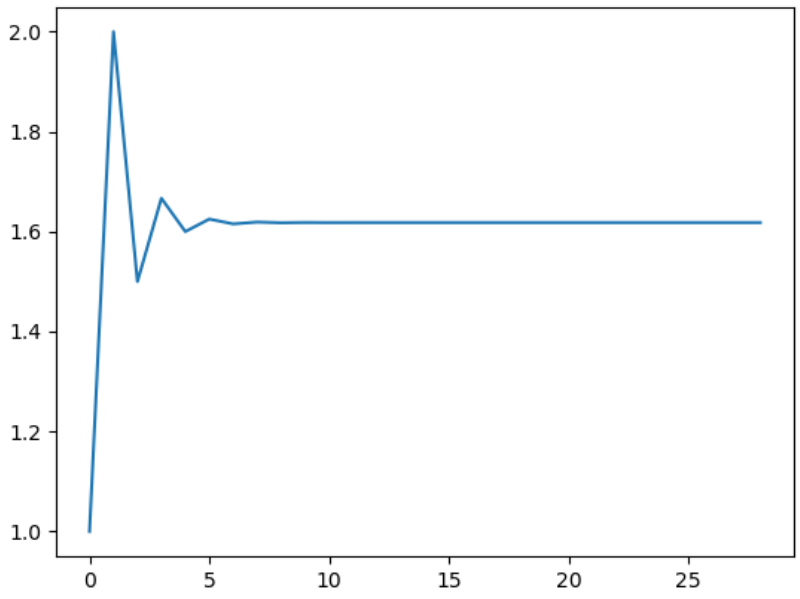
$$\begin{aligned} \lim_{n\to\infty}\left[\frac{F_n}{F_{n-1}}\right] &= \varphi \\ \lim_{n\to\infty}\left[\frac{F_n}{F_{n-1}}\right] &= \psi \\ \varphi - \psi &= 1 \\ \varphi \times \psi &= 1 \\ \frac{\psi}{\varphi} = \frac{1}{\varphi^2} = \frac{1}{1-\varphi} &= \frac{1}{2-\varphi} = \frac{2}{3-\sqrt{5}} \end{aligned}$$

3.3.5 Python

```
#+begin_src python
import matplotlib.pyplot as plt
import sympy

```

```
plt.plot([ sympy.N(sympy.fibonacci(n+1)/sympy.fibonacci(n)) for n in
↪  range(1, 30)])
plt.savefig("./a.png")
```



3.4 Angle is $\tan^{-1}\left(\frac{1}{1-\varphi}\right)$

3.4.1 Similar to Golden Angle $2\pi\left(\frac{1}{1-\varphi}\right)$

3.5 Dimension of my Fractal

$\log_{\varphi}(2)$

3.6 Code should be split up or put into appendix

```
function matJoin(A, B)
    function nrow(X)
        return size(X)[1]
    end
    function ncol(X)
        return size(X)[2]
    end
    emptymat = zeros(Bool, max(size(A)[1], size(B)[1]) ,sum(ncol(A) +
    ↪  ncol(B)) )
    emptymat[1:nrow(A), 1:ncol(A)] = A
```

```
    emptymat[1:nrow(B), (ncol(A)+1):ncol(emptymat)] = B
    return emptymat
end

function mywalk(B, n)
    for i in 1:n
        B = matJoin(B, rotl90(B));
    end
    return B
end

################################################################
##### Use Plot for themes ######################################
################################################################

using Plots
# SavePlot
## Docstring
    """
# MakePlot
Saveplot will save a plot of the fractals

- `n`
  - Is the number of iterations to produce the fractal
    - ``\\frac{n!}{k!(n - k)!} = \\binom{n}{k}``
- `filename`
  - Is the File name
- `backend`
  - either `gr()` or `pyplot()`
    - Gr is faster
    - pyplot has lines
    - Avoiding this entirely and using `GR.image()` and
     `GR.savefig` is even faster but there is no support
     for changing the colour schemes

    """
function makePlot(n, backend=pyplot())
    backend
    plt = Plots.plot(mywalk([1 1], n),
                     st=:heatmap, clim=(0,1),
                     color=:coolwarm,
                    colorbar_title="", ticks = true, legend = false,
                    ↪  yflip = true, fmt = :svg)
    return plt
end
plt = makePlot(5)

"""
# savePlot
Saves a Plot created with `Plots.jl` to disk (regardless of backend) as
↪  both an
svg, use ImageMagick to get a PNG if necessary

- `filename`
  - Location on disk to save image
- `plt`
  - A Plot object created by using `Plot.jl`
"""
```

```
function savePlot(filename, plt)
    filename = replace(filename, " " => "_")
    path = string(filename, ".svg")
    Plots.savefig(plt, path)
    print("Image saved to ", path)
end

#-------------------------------------------------------------
#-- Dimension ------------------------------------------------
#-------------------------------------------------------------
# Each time it iterates the image scales by phi
# and the number of pixels increases by 2
# so log(2)/log(1.618)
# lim(F_n/F_n-1)
# but the overall dimensions of the square increases by a factor of 3
# so 3^D=5 ==> log_3(5) = log(5)/log(3) = D
using DataFrames
function returnDim()
    mat2 = mywalk(fill(1, 1, 1), 10)
    l2   = sum(mat2)
    size2 = size(mat2)[1]
    mat1 = mywalk(fill(1, 1, 1), 11)
    l1   = sum(mat1)
    size1 = size(mat1)[1]
    df = DataFrame
    df.measure = [log(l2/l1)/log(size2/size1)]
    df.actual  = [log(2)/log(1.618) ]
    return df
end

##############################################################
### Main Functions ###########################################
##############################################################
# Usually Main should go into a seperate .jl filename
# Then a compination of import, using, include will
# get the desired effect of top down programming.
# Combine this with using a tmp.jl and tst.jl and you're set.
# See https://stackoverflow.com/a/24935352/12843551
# http://ryansnotes.org/mediawiki/index.php/Workflow_Tips_in_Julia

# Produce and Save a Plot
#=
filename = "my-self-rep-frac";
filename = string(pwd(), "/", filename);
savePlot(filename, makePlot(5))
;convert $filename.svg $filename.png
makePlot(5, pyplot())
=#
# Return the Dimensions
returnDim()


##############################################################
#### Render Image ############################################
##################yellow and purple############################
using GR
GR.imshow(mywalk([1 1], 5))
```

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 connection[15] and occur in patterns observed in nature very frequently (see [43, 3, 35, 36, 24, 41]), an example of such an occurence 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 generall 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:

$$F_n = F_{n-1} + F_{n-2} \tag{21}$$

This type of recursive relation can be expressed in *Python* by using recursion, as shown in listing 7, however using this function will reveal that it is extraordinarily slow, as shown in listing 8, this is because the results of the function are not cached and every time the function is called every value is recalculated[16], 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 [12], 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 9.

```
  start = time.time()
  rec_fib(6000)
  print(str(round(time.time() - start, 9)) + "seconds")

## 8.3923e-05seconds
```

[15] See section

[16] Dr. Hazrat mentions something similar in his book with respect to *Mathematica*® [17, Ch. 13]

```
  def rec_fib(k):
      if type(k) is not int:
          print("Error: Require integer values")
          return 0
      elif k == 0:
          return 0


      elif k <= 2:
          return 1
      return rec_fib(k-1) + rec_fib(k-2)
```

Listing 7: Defining the *Fibonacci Sequence* (21) using Recursion

```
  start = time.time()
  rec_fib(35)
  print(str(round(time.time() - start, 3)) + "seconds")

## 2.245seconds
```

Listing 8: Using the function from listing 7 is quite slow.

```
  from functools import lru_cache

  @lru_cache(maxsize=9999)
  def rec_fib(k):
      if type(k) is not int:
          print("Error: Require Integer Values")


          return 0
      elif k == 0:
          return 0
      elif k <= 2:
          return 1
      return rec_fib(k-1) + rec_fib(k-2)



start = time.time()
rec_fib(35)
print(str(round(time.time() - start, 3)) + "seconds")
## 0.0seconds
```

Listing 9: Caching the results of the function previously defined 8

Restructuring the problem to use iteration will allow for even greater performance as demonstrated by finding F_{10^6} in listing 10. Using a compiled language such as *Julia* however would be thousands of times faster still, as demonstrated in listing 11.

```
  def my_it_fib(k):
      if k == 0:
          return k
      elif type(k) is not int:
          print("ERROR: Integer Required")
          return 0
      # Hence k must be a positive integer

      i  = 1
      n1 = 1
      n2 = 1

      # if k <=2:
      #     return 1

      while i < k:
         no = n1
         n1 = n2
         n2 = no + n2
         i = i + 1
      return (n1)

  start = time.time()
  my_it_fib(10**6)
  print(str(round(time.time() - start, 9)) + "seconds")

## 6.975890398seconds
```

Listing 10: Using Iteration to Solve the Fibonacci Sequence

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 (21):

aLign a_n&= $a_{n-1} + a_{n-2}$
$\iff an + 2\& = an + 1 + an$

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

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

```
function my_it_fib(k)
    if k == 0
        return k
    elseif typeof(k) != Int
        print("ERROR: Integer Required")
        return 0
    end
    # Hence k must be a positive integer

    i  = 1
    n1 = 1
    n2 = 1

    # if k <=2:
    #     return 1
    while i < k
       no = n1
       n1 = n2
       n2 = no + n2
       i = i + 1


    end
    return (n1)
end

@time my_it_fib(10^6)

##  my_it_fib (generic function with 1 method)
##    0.000450 seconds
```

Listing 11: Using Julia with an iterative approach to solve the 1 millionth fibonacci number

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.

Example **Ryan** Consider using the following generating function, (proof of the generating function derivative as in (23) and (24) is provided in section 4.3)

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \cdot \frac{x^n}{n!} \right] \tag{22}$$

$$\implies f'(x) = \sum_{n=0}^{\infty} \left[a_{n+1} \cdot \frac{x^n}{n!} \right] \tag{23}$$

$$\implies f''(x) = \sum_{n=0}^{\infty} \left[a_{n+2} \cdot \frac{x^n}{n!} \right] \tag{24}$$

So the Fibonacci recursive relation from (4.3) 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 (22), (23) and (24):

$$f''(x) = f'(x) + f(x) \tag{25}$$

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* 4.3:

$$\begin{aligned} a_n &= \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right] \\ &= \frac{\varphi^n - \psi^n}{\sqrt{5}} \\ &= \frac{\varphi^n - \psi^n}{\varphi - \psi} \end{aligned} \tag{26}$$

where:

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

Derivative of the Exponential Generating Function

Base **Ryan** Differentiating the exponential generating function has the effect of shifting the sequence once to the left: [26]

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \tag{27}$$

$$\begin{aligned} f'(x) &= \frac{\mathrm{d}}{\mathrm{d}x} \left(\sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \right) \\ &= \frac{\mathrm{d}}{\mathrm{d}x} \left(a_0 \frac{x^0}{0!} + a_1 \frac{x^1}{1!} + a_2 \frac{x^2}{2!} + a_3 \frac{x^3}{3!} + \ldots \frac{x^k}{k!} \right) \\ &= \sum_{n=0}^{\infty} \left[\frac{\mathrm{d}}{\mathrm{d}x} \left(a_n \frac{x^n}{n!} \right) \right] \\ &= \sum_{n=0}^{\infty} \left[\frac{a_n}{(n-1)!} x^{n-1} \right] \\ \implies f'(x) &= \sum_{n=1}^{\infty} \left[\frac{x^n}{n!} a_{n+1} \right] \end{aligned} \tag{28}$$

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

$$f^{(k)}(x) = \sum_{n=k}^{\infty} \frac{a_{n+k} \cdot x^n}{n!} \quad \text{for } k \geq 0 \tag{29}$$

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

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

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 (28) shifts the sequence across, then every derivative thereafter does so as well.

Homogeneous Proof **RYAN:JAMES** An equation of the form:

$$\sum_{i=0}^{n}\left[c_i \cdot f^{(i)}(x)\right] = 0 \tag{30}$$

is said to be a homogenous linear ODE: [52, 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-homogeous 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, [52, 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 (31). 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:

- $\sum_{i=0}^{k} \left[c_i m^{k-i} \right] = 0$
  - This is referred to the characteristic equation of the recurrence relation or ODE [27]
- $\exists i, j \in \mathbb{Z}^+ \cap [0, k]$
  - These are often referred to as repeated roots [27, 53] with a multiplicity corresponding to the number of repetitions of that root [37, §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 (31):

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

This implies:

$$\sum_{n=0}^{\infty} \left[\sum_{i=0}^{k} \left[\frac{x^n}{n!} c_i a_n \right] \right] = 0 \tag{31}$$

$$\sum_{n=0}^{\infty} \sum_{i=0}^{k} \frac{x^n}{n!} c_i a_n = 0 \tag{32}$$

$$\sum_{i=0}^{k} c_i \sum_{n=0}^{\infty} \frac{x^n}{n!} a_n = 0 \tag{33}$$

By implementing the exponential generating function as shown in (22), this provides:

$$\sum_{i=0}^{k} \left[c_i f^{(i)}(x) \right] \tag{34}$$

Now assume that the solution exists and all roots of the characteristic polynomial are unique (i.e. the solution is of the form $f(x) \propto e^{m_i x} : \quad m_i \neq m_j \forall i \neq j$), this implies that [52, 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)**:

$$\begin{aligned}
\sum_{i=0}^{k} [k_i e^{m_i x}] &= \sum_{i=0}^{k} \left[k_i \sum_{n=0}^{\infty} \frac{(m_i x)^n}{n!} \right] \\
&= \sum_{i=0}^{k} \sum_{n=0}^{\infty} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty} \sum_{i=0}^{k} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} \sum_{i=0}^{k} [k_i m_i^n] \right], \quad \exists k_i \in \mathbb{C}, \ \ \forall i \in \mathbb{Z}^+ \cap [1, k] \qquad (35)
\end{aligned}$$

Recall the definition of the generating function from (22), by equating this to (35):

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

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

1. Example Consider the following recurrence relation:

$$\begin{aligned} a_{n+2} - 10a_{n+1} + 25a_n &= 0 \qquad (36) \\ \implies \sum_{n=0}^{\infty} \left[a_{n+2} \frac{x^n}{n!} \right] - 10 \sum_{n=0}^{\infty} \left[a_{n+1} \frac{x^n}{n!} \right] + 25 \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] &= 0 \end{aligned}$$

By applying the definition of the exponential generating function at (22) :

$$f''(x) - 10f'(x) + 25f(x) = 0 \qquad (37)$$

By implementing the already well-established theory of linear ODE's, the characteristic equation for (37) can be expressed as:

$$\begin{aligned} m^2 - 10m + 25 &= 0 \\ (m-5)^2 &= 0 \\ m &= 5 \qquad (38) \end{aligned}$$

Herein lies a complexity, in order to solve this, the solution produced from (38) can be used with the *Reduction of Order* technique to produce a solution that will be of the form [53, §4.3].

$$f(x) = c_1 e^{5x} + c_2 x e^{5x} \qquad (39)$$

(39) 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):

$$x^k e^x = \sum_{n=k}^{\infty} \left[\frac{x^n}{(n-k)!} \right], \quad \exists k \in \mathbb{Z}^+ \qquad (40)$$

by applying identity (40) to equation (39)

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

Given the defenition of the exponential generating function from (22)

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

□

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 (22):

$$\sum_{i=0}^{k}\left[c_i f^{(k)}(x)\right] \tag{41}$$

Assume that (41) corresponds to a charecteristic polynomial with only 1 root of multiplicity k, the solution would hence be of the form:

$$\begin{aligned}\sum_{i=0}^{k}\left[c_i m^i\right] &= 0 \wedge m = B,\ \ \exists! B \in \mathbb{C} \\ \implies f(x) &= \sum_{i=0}^{k}\left[x^i A_i e^{mx}\right],\quad \exists A \in \mathbb{C}^+,\ \ \forall i \in [1,k]\cap\mathbb{N}\end{aligned} \tag{42}$$

If we assume the identity from (40):

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

See section 3 for proof.

We can apply identity (40) to (42), 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 (41):

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \left[a_n \frac{x^n}{n!} \right] \\ \implies a_n &= \sum_{i=0}^{k} \left[A_i \frac{n!}{(n-i)!} m^n \right] \\ &= \sum_{i=0}^{k} \left[m^n A_i \prod_{0}^{k} [n-(i-1)] \right] \end{aligned} \tag{43}$$

$\because \quad i \leq k$

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

□

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

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

We conjecture thatIf 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 RYAN

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 RYAN

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

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

because $\mid \psi \mid < 0 \; n \to \infty \implies \psi^n \to 0$:

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

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 26).

Imagine that the process a sunflower follows when placing seeds is as follows: [17]

1. Place a seed
2. Move some small unit away from the origin
3. Rotate some constant angle θ (or θ) 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 12,[18] which combined with *ImageMagick* (see e.g. 30), produces output as shown in figure 24 and 25.

A distribution of seeds undder 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 sufficent, the decimal values must also be not to approximated by a rational number, for example [36]:

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

It can be seen by simulation that ϕ and ψ (because $\phi \mod 1 = \psi$) are solutions to this optimisation problem as shown in figure 25, this solution is unstable, a very minor change to the value will result in patterns re-emerging in the distribution.

Another interesting property is that the number of spirals that appear to rotate clockwise and anti-clockwise appear to be fibonacci numbers. Connecting this occure 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 24 has a ratio of rotation of $\frac{1}{\pi}$, the spirals look similar to one direction of the spirals occuring in figure 25, it is not clear if there is any significance to this similarity.

4.6 Relating The Fibonacci Sequence to the Julia Set RYAN

See this link.

[17] This process is simply conjecture, other than seeing a very nice example at *MathIsFun.com* [36], we have no evidence to suggest that this is the way that sunflowers distribute there seeds.

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

[18] Emojis 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.

```
φ = 1.61803398875
ψ = φ^-1


ψ = 0.61803398875
function sfSeeds(ratio)


♘ = Turtle()
    for θ in [(ratio*2*π)*i for i in 1:3000]


        gsave()
        scale(0.05)


        rotate(θ)
#        Pencolor(□, rand(1)[1], rand(1)[1], rand(1)[1])


        Forward(♘, 1)
        Rectangle(♘, 50, 50)


        grestore()
    end


    label = string("Ratio = ", round(ratio, digits = 8))
    textcentered(label, 100, 200)


end
@svg begin


    sfSeeds(φ)
end 600 600
```

Listing 12: Simulation of the distribution of sunflowers as described in section 4.5.1

Figure 24: Simulated Distribution of Sunflower seeds as described in section 4.5.1 and listing 12

Figure 25: Optimisation of simulated distribution of Sunflower seeds occurs for $\theta = 2\varphi\pi$ as described in section 4.5.1 and listing 12

Figure 26: Distribution of the seeds of a sunflower (see [5] licenced under CC)

5 Julia Sets

RYAN

The julia set is the outline [38, 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 [9] There is a relatioship between the fibonacicci 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.

5.2 Motivation

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

Now Consider the iterative process $z \to z^2, \quad z \in \mathbb{C}$, 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 $f_c(z) = z^2 + c, \quad c \leq 1 \in \mathbb{C}$ 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 the set of values confirmed as prisoners after k iterations of f_c
    - this implies $\lim_{k\to\infty}\left[Q_c^{(k)}\right] = P_c$
- 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 vector[19] and define the operations as described in listing 13.[20]

To implement this test we'll consider a function called `escape_test` that applies an iteration (in this case $f_0 : z \to z^2$) 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 image [21], this is implemented in listing 14 and the corresponding output shown in 27.

with respect to listing 14:

- 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

Figure 27: Circle of Convergence for $f_0 : z \to z^2$

This is precisely what we expected, but this is where things get interesting, consider now the result if we apply this same procedure to $f_1 : z \to z^2 - 1$ or something arbitrary like $f_{\frac{1}{4}+\frac{i}{2}} : z \to z^2 + (\frac{1}{4} + \frac{i}{2})$, the result is something remarkebly unexpected, as shown in figures 28 and 29.

Figure 28: Circle of Convergence for $f_0 : z \to z^2 - 1$

Figure 29: Circle of Convergence for $f_{\frac{1}{4}+\frac{i}{2}} : z \to z^2 + \frac{1}{4} + \frac{i}{2}$

[19] See figure for the obligatory *XKCD* Comic

[20] This technique was adapted from Chapter 7 of *Math adventures with Python* [10]

[21] these cascading values are much like brightness in Astronomy

```python
from math import sqrt
def magnitude(z):
    # return sqrt(z[0]**2 + z[1]**2)
    x = z[0]
    y = z[1]
    return sqrt(sum(map(lambda x: x**2, [x, y])))

def cAdd(a, b):
    x = a[0] + b[0]
    y = a[1] + b[1]
    return [x, y]


def cMult(u, v):
    x = u[0]*v[0]-u[1]*v[1]
    y = u[1]*v[0]+u[0]*v[1]
    return [x, y]
```

Listing 13: Defining Complex Operations with vectors

To investigate this further consider the more general function $f_{0.8e^{\pi i\tau}} : z \to z^2 + 0.8e^{\pi i\tau}, \ \tau \in \mathbb{R}$, many fractals can be generated using this set by varying the value of τ[22].

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 15. 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 16 where 1200 pictures at a 2.25 MP resolution were produced. [23]

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 17, the collage is shown in figure 30 and a corresponding animation is available online[24]].

Figure 30: Various fracals corresponding to $f_{0.8e^{\pi i\tau}}$

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.

[22]This approach was inspired by an animation on the *Julia Set* Wikipedia article [20]
[23]On my system this took about 30 minutes.
[24]https://dl.dropboxusercontent.com/s/rbu25urfg8sbwfu/out.gif?dl=0

```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
import numpy as np
def escape_test(z, num):
    ''' runs the process num amount of times and returns the count of
    divergence'''
    c = [0, 0]
    count = 0
    z1 = z  #Remember the original value that we are working with
    # Iterate num times
    while count <= num:
        dist = sum([n**2 for n in z1])
        distc = sum([n**2 for n in c])
        # check for divergence
        if dist > max(2, distc):
            #return the step it diverged on
            return count
        #iterate z
        z1 = cAdd(cMult(z1, z1), c)
        count+=1
        #if z hasn't diverged by the end


    return num



p = 0.25 #horizontal, vertical, pinch (zoom)
res = 200
h = res/2
v = res/2

pic = np.zeros([res, res])
for i in range(pic.shape[0]):
    for j in range(pic.shape[1]):
        x = (j - h)/(p*res)
        y = (i-v)/(p*res)
        z = [x, y]
        col = escape_test(z, 100)
        pic[i, j] = col

import matplotlib.pyplot as plt

plt.axis('off')


plt.imshow(pic)
# plt.show()
```

Listing 14: Circle of Convergence of z under recursion

```julia
# * Define the Julia Set
"""
Determine whether or not a value will converge under iteration
"""
function juliaSet(z, num, my_func)
    count = 1
    # Remember the value of z
    z1 = z
    # Iterate num times
    while count ≤ num
        # check for divergence


        if abs(z1)>2
            return Int(count)
        end
        #iterate z
        z1 = my_func(z1) # + z
        count=count+1
    end
        #if z hasn't diverged by the end
    return Int(num)
end



# * Make a Picture
"""
Loop over a matrix and apply apply the julia-set function to
the corresponding complex value
"""
function make_picture(width, height, my_func)
    pic_mat = zeros(width, height)
    zoom = 0.3
    for i in 1:size(pic_mat)[1]
        for j in 1:size(pic_mat)[2]
            x = (j-width/2)/(width*zoom)


            y = (i-height/2)/(height*zoom)
            pic_mat[i,j] = juliaSet(x+y*im, 256, my_func)
        end
    end
    return pic_mat
end
```

Listing 15: Produce a series of fractals using julia

```julia
# * Use GR to Save a Bunch of Images
  ## GR is faster than PyPlot
using GR
function save_images(count, res)
    try
        mkdir("/tmp/gifs")
    catch
    end
    j = 1
    for i in (1:count)/(40*2*π)
        j = j + 1


        GR.imshow(make_picture(res, res, z -> z^2 + 0.8*exp(i*im*9/2))) #
        ↪  PyPlot uses interpolation = "None"
        name = string("/tmp/gifs/j", lpad(j, 5, "0"), ".png")
        GR.savefig(name)
    end
end

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

Listing 16: Generate and save the images with GR

```bash
# Use montage multiple times to get recursion for fun
montage (ls *png | sed -n '1p;0~600p') 0a.png
montage (ls *png | sed -n '1p;0~100p') a.png
montage (ls *png | sed -n '1p;0~50p') -geometry 1000x1000  a.png

# Use ImageMagick to Produce a gif (unreliable)
convert -delay 10 *.png 0.gif

# Use FFMpeg to produce a Gif instead
ffmpeg                      \
    -framerate 60           \


    -pattern_type glob      \
    -i '*.png'              \
    -r 15                   \
    out.mov
```

Listing 17: Using `bash`, `ffmpeg` and *ImageMagick* to combine the images and produce an animation.

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 $z \to z^2 + z_0$ where z_0 is unchanging or more clearly as a seqeuence:

$$z_{n+1} = z_n^2 + c \tag{44}$$
$$z_0 = c \tag{45}$$

This strategy is implemented in listing and produces the output shown in figure 31.

Figure 31: Mandelbrot Set produced in *Python* as shown in listing 18

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.[25]

```
function mandelbrot(z, num, my_func)
    count = 1
    # Define z1 as z
    z1 = z
    # Iterate num times
    while count ≤ num
        # check for divergence
        if abs(z1)>2
            return Int(count)
        end
        #iterate z
        z1 = my_func(z1) + z
        count=count+1
    end
        #if z hasn't diverged by the end
    return Int(num)
end

function make_picture(width, height, my_func)
    pic_mat = zeros(width, height)
    for i in 1:size(pic_mat)[1]
        for j in 1:size(pic_mat)[2]
            x = j/width
            y = i/height
            pic_mat[i,j] = mandelbrot(x+y*im, 99, my_func)
        end
    end
```

[25] https://www.dropbox.com/s/jd5qf1pi2h68f2c/mandelbrot-400mpx.fits?dl=0

```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
def mandelbrot(z, num):
    ''' runs the process num amount of times and returns the count of
    divergence'''
    count = 0
    # Define z1 as z
    z1 = z
    # Iterate num times
    while count <= num:
        # check for divergence
        if magnitude(z1) > 2.0:
            #return the step it diverged on
            return count
        #iterate z
        z1 = cAdd(cMult(z1, z1),z)
        count+=1
        #if z hasn't diverged by the end
    return num

import numpy as np


p = 0.25 # horizontal, vertical, pinch (zoom)
res = 200
h = res/2
v = res/2

pic = np.zeros([res, res])
for i in range(pic.shape[0]):
    for j in range(pic.shape[1]):
        x = (j - h)/(p*res)
        y = (i-v)/(p*res)
        z = [x, y]
        col = mandelbrot(z, 100)
        pic[i, j] = col

import matplotlib.pyplot as plt
plt.imshow(pic)
# plt.show()
```

Listing 18: All values of c that lead to a closed *Julia-set*

```
    return pic_mat
end


using FITSIO
function save_picture(filename, matrix)
    f = FITS(filename, "w");
    # data = reshape(1:100, 5, 20)
    # data = pic_mat
    write(f, matrix)  # Write a new image extension with the data

    data = Dict("col1"=>[1., 2., 3.], "col2"=>[1, 2, 3]);
    write(f, data)  # write a new binary table to a new extension

    close(f)
end

# * Save Picture
#---------------------------------------------------------------
my_pic = make_picture(20000, 20000, z -> z^2) 2000^2 is 4 GB
save_picture("/tmp/a.fits", my_pic)
```

Figure 32: Screenshot of Mandelbrot FITS image produced by listing

7 Appendix

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

7.1 Finding Material

```
recoll -c /home/ryan/Dropbox/Books/Textbooks/Mathematics/Chaos_Theory/cha⌋
↪  os_books_recoll &
↪  disown
```

7.2 Font Lock

```
;; match:
;;; \scite:key\s
(add-to-list 'font-lock-extra-managed-props 'display)
(font-lock-add-keywords nil
 '((" \\(cite:[a-z0-9A-Z]\+\\)" 1 '(face nil display "□"))))


;; match
;;; [[cite:key][p. num]]
(add-to-list 'font-lock-extra-managed-props 'display)
(font-lock-add-keywords nil
```

```
'((" \\(\\[\\[cite:[a-z0-9A-Z]\+\\]\\[\.\*\\]\\]\\)" 1 '(face nil
↪  display "□"))))
```

7.3 Resources Used for the Hausdorff Dimension

Research for §2.3.2 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* [6, Ch. 6]
- Falconer, K. J., *Fractal geometry: mathematical foundations and applications* [8, Ch. 2]
- Gouyet, J., *Physics and fractal structures* [15, §1.3]
- Vicsek, T., *Fractal Growth Phenomena* [48, Ch. 4]
  - See also p 14 specifically
- Tél, T., Gruiz, M., & Kulacsy, K., *Chaotic dynamics: an introduction based on classical mechanics* [46, §2.1.2]
- Peitgen, H., Jürgens, H., & Saupe, D., *Chaos and fractals: new frontiers of science* [38, §4.3]

References

[1] *Astropy*. URL: https://www.astropy.org/ (visited on 10/20/2020) (cit. on p. 21).

[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. 7).

[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. 46).

[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. URL: https://epubs.siam.org/doi/10.1137/141000671 (visited on 08/28/2020) (cit. on pp. 13, 61).

[5] Simon Brass. *CC Search*. 2006, September 5. URL: https://search.creativecommons.org/ (visited on 08/28/2020) (cit. on p. 62).

[6] 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. 4, 8, 72).

[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 p. 9).

[8] 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, 6, 8, 9, 11, 12, 72).

[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. 12, 63).

[10] 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. 64).

[11] *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. 5).

[12] *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. 46).

[13] 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. 6).

[14] Ralph Gomory. "Benoît Mandelbrot (1924–2010)". In: *Nature* 468.7322 (7322 Nov. 2010), pp. 378–378. ISSN: 1476-4687. DOI: 10.1038/468378a. URL: https://www.nature.com/articles/468378a (visited on 10/19/2020) (cit. on pp. 3, 5).

[15] 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, 72).

[16] 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).

[17] 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 (cit. on p. 46).

[18] jan wassenaar. *Cantor Dust*. Jan. 7, 2005. URL: http://www.2dcurves.com/fractal/fractald.html (visited on 10/20/2020) (cit. on p. 14).

[19] 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. URL: https://doi.org/10.1023/A:1021790111404 (visited on 10/19/2020) (cit. on p. 4).

[20] *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. 65).

[21] *JuliaAstro · JuliaAstro*. URL: https://juliaastro.github.io/dev/index.html (visited on 10/20/2020) (cit. on p. 21).

[22] *JuliaGraphics/Luxor.Jl*. JuliaGraphics, Oct. 19, 2020. URL: https://github.com/JuliaGraphics/Luxor.jl (visited on 10/20/2020) (cit. on p. 21).

[23] *JuliaImages/Images.Jl*. JuliaImages, Oct. 13, 2020. URL: https://github.com/JuliaImages/Images.jl (visited on 10/20/2020) (cit. on p. 21).

[24] 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. 46).

[25] 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).

[26] 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. 52).

[27] 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. 54).

[28] 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).

[29] *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).

[30] ImageMagick Studio LLC. *ImageMagick*. URL: https://imagemagick.org/ (visited on 10/20/2020) (cit. on p. 21).

[31] 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. pmid: 17837158. URL: https://science.sciencemag.org/content/156/3775/636 (visited on 10/19/2020) (cit. on p. 5).

[32] 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).

[33] 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. 5).

[34] Mark Pollicott. *Fractals and Dimension Theory*. 2005-May. URL: https://warwick.ac.uk/fac/sci/maths/people/staff/mark_pollicott/p3 (cit. on p. 12).

[35] 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. 46).

[36] *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. 46, 61).

[37] 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. 54).

[38] 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. 6, 24, 63, 72).

[39] *Pillow — Pillow (PIL Fork) 8.0.0 Documentation*. URL: https://pillow.readthedocs.io/en/stable/ (visited on 10/20/2020) (cit. on p. 21).

[40] 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).

[41] 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. 46).

[42] 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. 4, 6, 7, 9).

[43] Shelly Allen. *Fibonacci in Nature*. URL: https://fibonacci.com/nature-golden-ratio/ (visited on 08/28/2020) (cit. on p. 46).

[44] 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. 7).

[45] 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. URL: http://link.springer.com/10.1007/s00283-013-9428-y (visited on 10/20/2020) (cit. on p. 24).

[46] 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. 4, 72).

[47] 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. 24).

[48] 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. 4, 14, 29, 72).

[49] *Welcome to Python.Org*. URL: https://www.python.org/ (visited on 10/20/2020) (cit. on p. 13).

[50] 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. URL: https://www.nature.com/articles/s41598-017-08924-9 (visited on 10/19/2020) (cit. on p. 4).

[51] 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. URL: https://doi.org/10.1007/s11769-002-0022-z (visited on 10/19/2020) (cit. on p. 4).

[52] Dennis G Zill and Michael R Cullen. *Differential Equations*. 7th ed. Brooks/Cole, 2009 (cit. on pp. 53, 55).

[53] 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. 54, 56).