The Emergence of Patterns in Nature and Chaos Theory

Ryan Greenup & James Guerra

October 19, 2020

Contents

1 Report

What follows below are some dummy headings for want of testing the style

1.1 Cauchy Integral Formula

1.1.1 Central Limit Theorem

Fibonacci Sequence

Sequences and Series generally

2 Hausdorff Dimension

2.1 Topological Equivalence

Sources for this section on topology are primarily. [21, p. 106]

Topology is an area of mathematics concerned with ideas of continuity through the study of figures that are preserved under homeomorphic transformations. [9]

Two figures are said to be homeomorphic if there is a continuous bijective mapping between the two shapes [21, p. 105].

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.

Historically the concept of dimension was a difficult problem with a tenuous definition, while 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 however.

Consider the koch fractal in figure 1 (see also figure 2), at each iteration the perimeter is given by $p_n = p_{n-1}\left(\frac{4}{3}\right)$, this means if the shape is scaled by some factor s the the following relationship holds.

The number of edges in the koch fractal is given by:

$$\begin{aligned} N_n &= N_{n-1} \cdot 4 && (1) \\ &= 3 \cdot 4^n && (2) \end{aligned}$$

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:

$$\begin{aligned} p_n &= N_n \times l && (4) \\ &= 3 \cdot 4^n \times \frac{s \cdot l_o}{3^n} && (5) \\ &= 3 \cdot s \cdot l_0 \left(\frac{4}{3}\right)^n && (6) \end{aligned}$$

The koch snowflake, is defined such that there are no edges, every point on the curve is the vertex of an equilateral triangle. Every time the koch curve is iterated, one edge is reduced in length by a scale of 3 and the overall length increases by a factor of 4, this means if the overall shape was scaled by a factor of s the number of segments.

Briggs and Tyree provide a great introduction.

the scale of resolution increases 3 fold

$$\begin{aligned} s \cdot p_n &= (4/3)^n \cdot s \cdot P_0 &(7)\\ &\propto \left(\frac{4}{3}\right)^n &(8)\\ \implies n &= \frac{\ln(4)}{\ln(3)} &(9) \end{aligned}$$

In ordinary geometric shapes this value n will be the dimension of the shape,

See [24, p. 414] for working.

The idea is we start with the similarity dimension [24, p. 413] which should be equal to the hausdorff and box counting for most fractals, but for fractals that aren't so obviously self similar it won't be feasible [15, p. 393] but for the julia set we'll need to expand the concept to box counting, we don't know whether or not the dimension of the julia set is constant across scales so we use linear regression to check, this is more important for things like coastlines.

with respect to that shapes *measure*. For example consider measure similar to mass, a piece of wire when scaled in length, will increase in mass by a factor of that scale, whereas a sheet of material would increase in mass by a factor proportional to the square of that scaling.

In the case of the koch snowflake, the measure of the shape, when scaled, will increase by a factor of

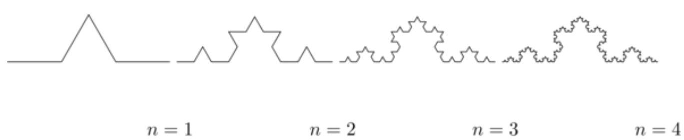


Figure 1: Progression of the Koch Snowflake

}

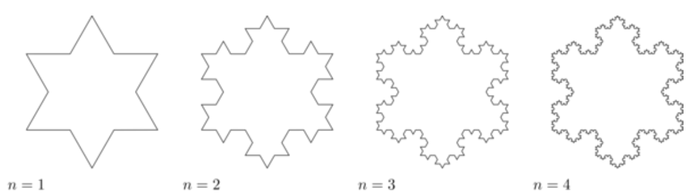


Figure 2: Progression of the Koch Snowflake

In the development of topology

2.2 Hausdorff Dimension

Sources for this section on Hausdorff Dimension are primarily [5, Ch. 2]

2.2.1 Measure

Let F be some arbitrary subset of euclidean space $\mathbb{R}^n$, [1]

Consider a collection of sets,$\{U_i : i \in \mathbb{Z}^+, U \subset \mathbb{R}^n\}$, each of which having a diameter less than δ.

The motivating idea is that if the elements of U can be laid ontop of F then U is said to be a δ -cover of F, more rigorously this could be defined:

$$F \subset \bigcup_{i=1}^{\infty} [U_i] \quad : 0 \leq |U_i| \leq \delta \tag{10}$$

An example of this covering is provided in figure 3, 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 however that the shapes needn't be squares, they could be any arbitrary figure.

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
  - The use of balls is a simpler but equivalent approach to the theory [6, §2.4] because any set of diamater r can be enclosed in a ball of radius $\frac{r}{2}$ [4, p. 166]
- F could be a more abstracted figure like figures 3 or 4 and $\{U_i\}$ a collection of various different lines, shapes or 3d objects.

The Hausdorff measure is concenred with only the diamater of each element of $\{U_i\}$ and considers $\sum_{i=1}^{\infty} [|U_i|^s]$ where the covering of U_i minimizes the summation. [6, p. 27]

$$\mathcal{H}_\delta^s(F) = \inf \left\{ \sum_{i=1}^{\infty} |U_i|^s \quad : \quad \{U_i\} \text{ is a } \delta\text{-cover of } F \right\}, \quad \delta, s > 0 \tag{11}$$

in 2 dimensions, this is equivalent to considering the number of boxes, of diamater $\leq \delta$ that will cover over a shape as shown in figure 3, the delta Haussendorf measure $\mathcal{H}_\delta^s(F)$ will be the area of the boxes when arranged in such a way that minimises the area.

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

As δ is made arbitrarily small H_δ^s will approach some limit, in the case of figures 3 and 4 the value of $\mathcal{H}_\delta^2$ will approach the area of the shape as $\delta \to 0$ and so the s^{th} dimensional Hausendorff measure is given by:

$$\mathcal{H}^s = \lim_{\delta \to 0} \left(\mathcal{H}_\delta^s \right) \tag{12}$$

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

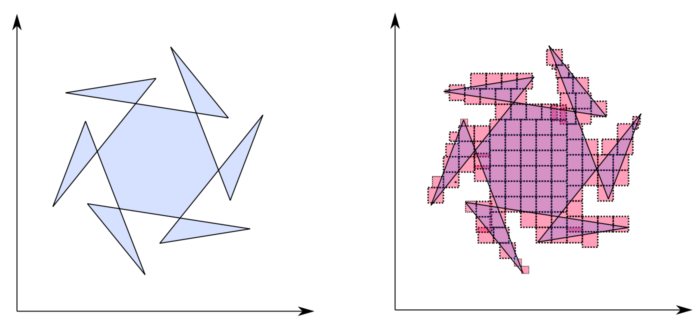


Figure 3: The shape on the left corresponds to $F \subset \mathbb{R}^{\nvDash}$, each identical square box on the right represents a set U_i.

Lower Dimension Hausdorff Measurements

Examples Consider again the example of a 2D shape, the value of $\mathcal{H}^1$ would still be defined by (11), but unlike $\mathcal{H}^2$ in section 2.2.1 the value of $|U_i|^1$ would be considered as opposed to $|U_i|^2$.

As δ is made arbitrarily small the boxes 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.

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 be zero.

This is consistent with shapes of other shapes as well, consider the koch snowflake introduced in section 2.1 and shown in figure 1, the dimension of this shape is greater than 1, and the number of lines necessary to describe that shape is also infinite.

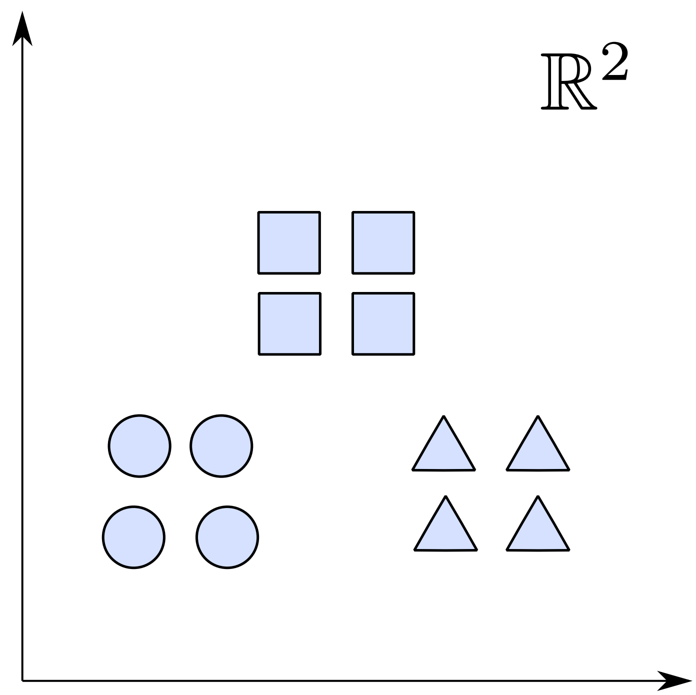


Figure 4: A disconnected subset of $\mathbb{R}^2$, the squares have a diameter of $\sqrt{2}$, the circles 1 and the equilateral triangles 1.

Formally If the dimension of F is less than s, the Hausdorff Measure will be given by: [2]

$$\dim(F) < s \implies \mathcal{H}^s(F) = \infty \tag{13}$$

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: [3]

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

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:

$$\begin{aligned} \lim_{\delta \to 0} \left(\sum_i \left[|U_i|^t \right] \right) &\leq \lim_{\delta} \left(\delta^{t-s} \sum_i \left[|U_i|^s \right] \right) && (14) \\ &\leq \lim_{\delta \to 0} \left(\delta^{t-s} \right) \cdot \lim_{\delta \to 0} \left(\sum_i \left[|U_i|^s \right] \right) && (15) \\ &\leq 0 && (16) \end{aligned}$$

and so we have the following relationship:

$$\mathcal{H}^s(F) \in \mathbb{R} \implies \mathcal{H}^t(F) = 0 \quad \forall t > s \tag{17}$$

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

[2] I haven't been able to find a proof for this, I wonder if I could prove it by just applying the definition?

[3] Could fractal dimensions be complex? Maybe there could be a proof to show that the dimension is necessarily complex.

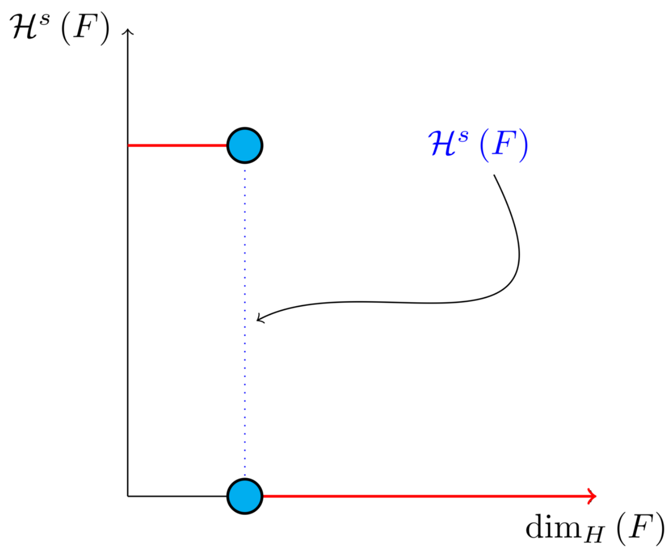


Figure 5: The value of the s-dimensional *Hausdorff Measure* of some subset of *Euclidean space* $F \in \mathbb{R}^n$ is 0 or ∞ when the dimension of F is not equal to s.

2.2.2 Hausdorff Dimension

The value s at which $\mathcal{H}^s$ changes from ∞ to 0, shown in figure 5 and (17) is the definition of the *Hausdorff Measure*, it is a generalisation of the idea of dimension that is typically understood with respect to ordinary shapes and 3D figures.

2.2.3 Research

I feel very inclided to read these notes [4]

3 Box Counting

Sources for this section are primarily:

- Falconer [6, Ch. 3.1]
- Strogatz Non Linear Dynamics [24, Ch. 11.4]
- There are many different notions of dimension, atleast 10 [4, Ch. 4.3]
  - *Hausdorff Dimension* is the oldest and most important with respect to the dimensions of fractals [6, p. 27]

[4] Local Copy

- The Box dimension has only boxes of fixed sizes, unlike the *Hausdorff Dimensions* where in the boxes may be arbitrary sizes less than δ but it is significantly hader to calculate numerically [24, §11.4]

- The problem with the *Hausdorff Dimension*, discussed in §2, is that it is quite involved to solve, many shapes such as the koch snowflake haven't even h

  - Upper and lower bounds for the *Hausdorff Measurement/* haven't even been solved for many fractals, including the Koch Snowflake (shown in figure 2). [25]

The box counting method is widely used because it is relatively easy to calculate [6, p. 41] and in many cases is equal to the *Hausdorff Dimension* [17, p. 11] (see generally [16]).

- TODO:
  - While quite simply this is the number of boxes that scale, we need to more rigorously define it, much like Ch. 3.1 of Falconer [5]
  - We also need to contrast this with the similarity dimension discussed in p. 413 of Strogatz [24]
  - A contrast should be drawn between this and Hausdorff, the most obvious difference being that the boxes are of a fixed size, unlike *Hausdorff* see p. 418 of Strogatz [24]

4 Fractals Generally

While there is no formal definition for the term fractal at this stage, we may decsribe it through the following properties:

- Can be, but not subject to being self-similar [5]. On the contrary fractals can also be shapes like coastlines (which are not self-simialr)

- The dimension of the fractal is the same at every scale.

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

[5] A self-similar shape is one that replicates its shape at every scale.

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.

5 Generating Self Similar Fractals

5.0.1 Examples

Vicsek Fractal The Vicsek Fracatl is self similar, thus we can use it to test our box counting method [6]. The Vicsek Fractal involves a pattern of iterating boxes:

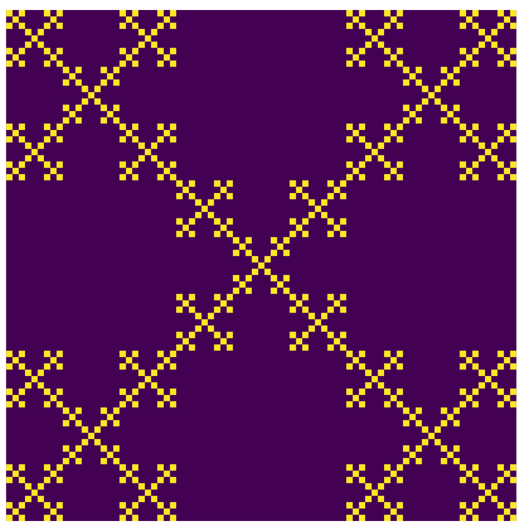


Figure 6: TODO

The above program demonsrates the construction of the Vicsek Fractal and its self-similarity dimension. To do this, we define a recursive function that begins with a 3x3

[6]Since the Vicsek fractal is self similar, we know that the dimension will be constant, as opposed to a dimension that slightly varies, hence there is no need for linear regression which would be necessary to measure someting like a coastline.

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

# n_i+1 = 3n_i ==> n = 3^n
function selfRep(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 = selfRep(B, width)
    end
    return B
end


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

#------------------------------------------------------------
#-- Similarity Dimension ------------------------------------
#------------------------------------------------------------
# Each time it iterates there are 5 more
# 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

mat2  = selfRep(fill(1, 1, 1), 1000)
l2    = sum(mat2)
size2 = size(mat2)[1]
mat1  = selfRep(fill(1, 1, 1), 500)

l1    = sum(mat1)
size1 = size(mat1)[1]

log(l2/l1)/log(size2/size1)
# https://en.wikipedia.org/wiki/Vicsek_fractal#Construction
log(5)/log(3)


  ##  julia> log(l2/l1)/log(size2/size1)
  ##  1.4649735207179269
```

matrix, where the four corner squares and middle square are set to 1 and the rest is set to 0. The function repeats until it reaches some arbitrary set width. Each time the function iterates, 5 more squares are created, increasing by a factor of 3. We can use this information to calculate the dimension of the Vicsek fractal. Using the box counting method, we get:

$$5 = 3^D$$
$$D \ln 3 = \ln 5$$
$$D = \frac{\ln 5}{\ln 3}$$

Sierpinskis Carpet Explained more in the book [7]

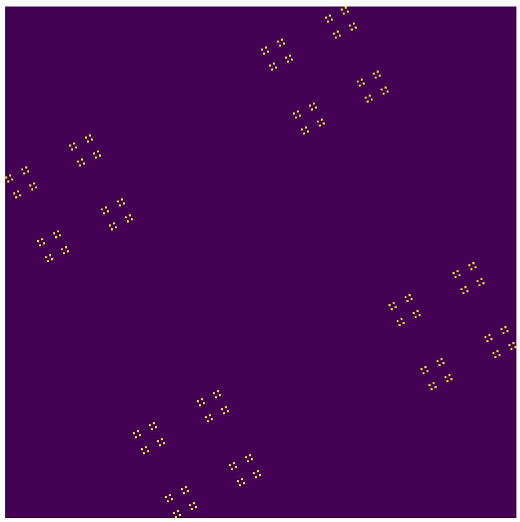


Figure 8: Cantor Dust

Triangle Producing the triangle was more difficult

Chaos Game This would be more accurate than pascals because there would be know **bias** and the model would be more accurate :

[7]See Ch. 2.7 of [21, Ch. 2.7]

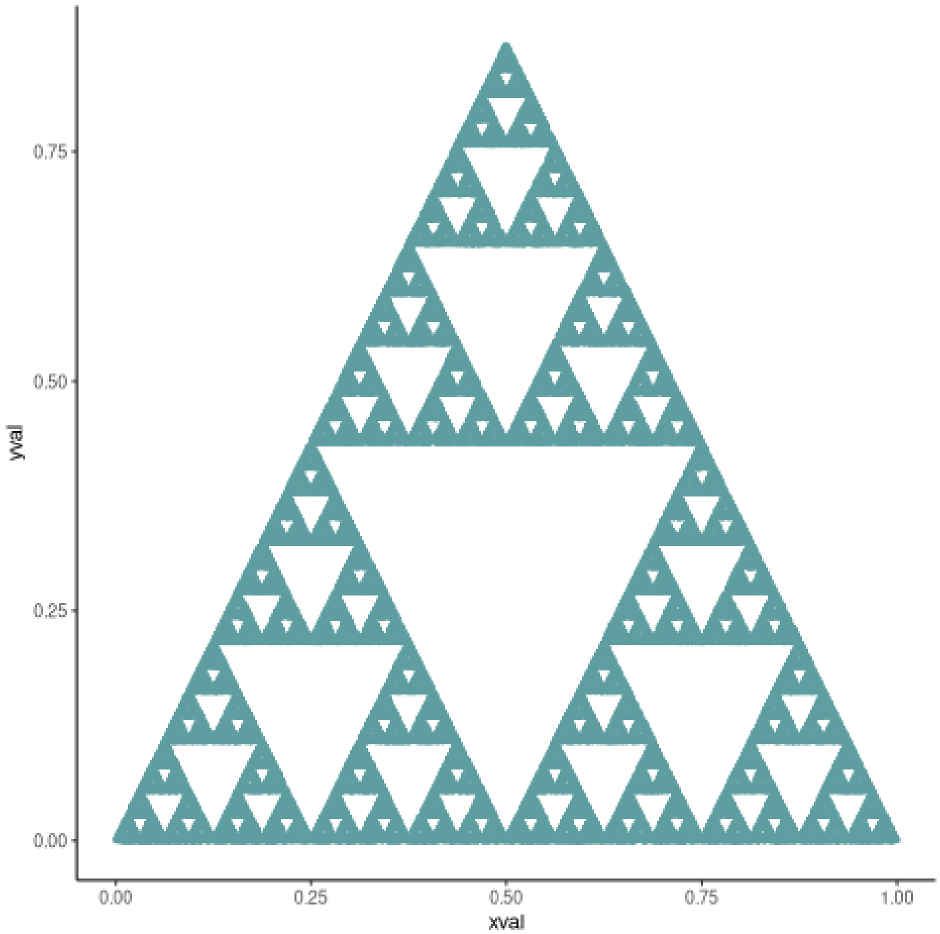
By modifying listing we can get patterns like the cantor dust and sierpinskis carpet shown in figures and .

```r
if (require("pacman")) {
    library(pacman)
  }else{
    install.packages("pacman")
    library(pacman)
  }
  pacman::p_load(tidyverse)


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
    }
}
```



Pascals Triange

1. 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 10. Alternatively, the n^{th} row and k^{th} column can be written in combinatorics form, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

2. 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.

   In figure 2, 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

```julia
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(3000)


l2    = sum(mat2)
size2 = size(mat2)[1]
mat1 = pascal(2000)
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)
```

Listing 2: Julia code demonstrating Sierpinksi's triangle

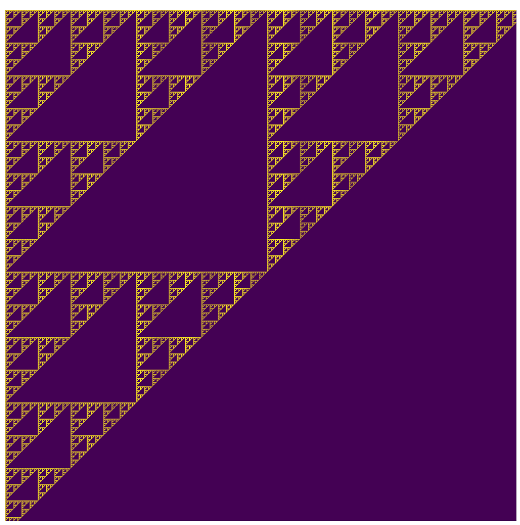


Figure 9: TODO

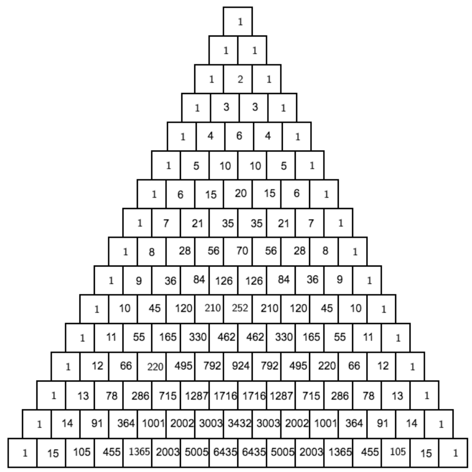
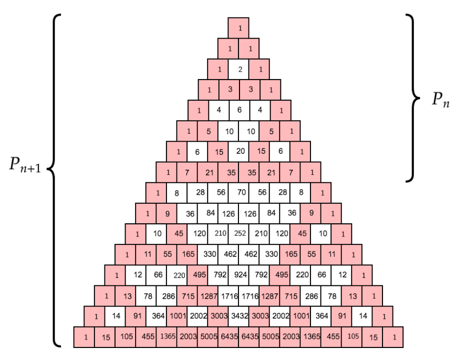


Figure 10: Pascal's triangle



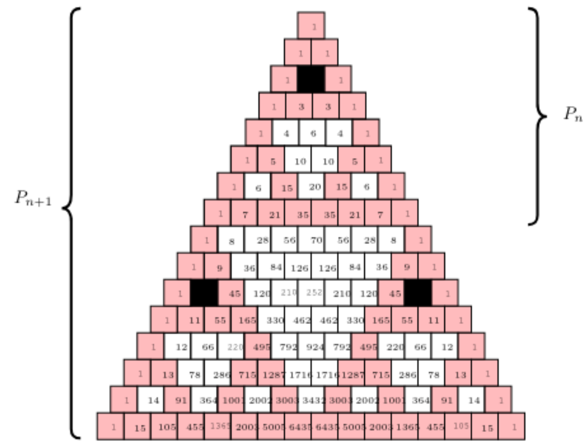


Figure 11: The black squares represent one example of a position on Pascal's triangle that are equivalent modulo 2

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 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 11. 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:

$$\binom{n}{k} = \prod_{i=0}^{m} \binom{n_i}{k_i} \pmod{p} \tag{18}$$

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 [8]. This uses the convention that $\binom{n}{k} = 0$ if $k < n$

[8] Radix refers to a numerical system which uses some number of digits. Since we are working in modulo 2

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

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

Using Lucas' theorem, we can prove that the above statments are equivalent.

We can rewrite r and c in base 2 notation as follows:

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

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

for Pascal's triangle, we are only concerned with the numbers 0 or 1, i.e. a radix 2 or a binary numeric system.

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

Thus, $\binom{r}{c} = \binom{2^n+r}{c} = \binom{2^n+r}{2^n+c} \quad (\text{mod}2)$, which concludes the proof

3. Comment on the dimension lining up Using the box-counting method, we can evaluate the dimension of Sierpinski's triangle.

4. FIx the value

```
julia> log(l2/l1)/log(size2/size1)
2.082583161459976

julia> # https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle
       log(3)/log(2)
1.5849625007211563
```

6 Fractal Dimensions

See generally [24, Ch. 11] Three ways to generate

1. Chaos Game
2. Iteration Like Matrices and Turtles
3. Testing if each region Belongs
   (a) Like Julia Set

6.1 Turtle

Matrices can't explain all patterns, Turtles are useful

```
using Shapefile
using Luxor
using Pkg


#-------------------------------------------------------------
#-- Dragon Curve --------------------------------------
#-------------------------------------------------------------


function snowflake(length, level,  , s)
    scale(s)
    if level == 0
        Forward( , 100)
        Turn( , -90)
        Rotate(90)
#        Rectangle( , length, length)
        return
    end
    length = length/9
    snowflake(length, level-1,  )
    Turn( , -60)
    snowflake(length, level-1,  )
    Turn( , 2*60)
    snowflake(length, level-1,  )
    Turn( , -180/3)
    snowflake(length, level-1,  )
end
@png begin
      = Turtle()
    Pencolor( , 1.0, 0.4, 0.2)
    Penup( )
    Turn( ,180)
    Forward( , 200)
```

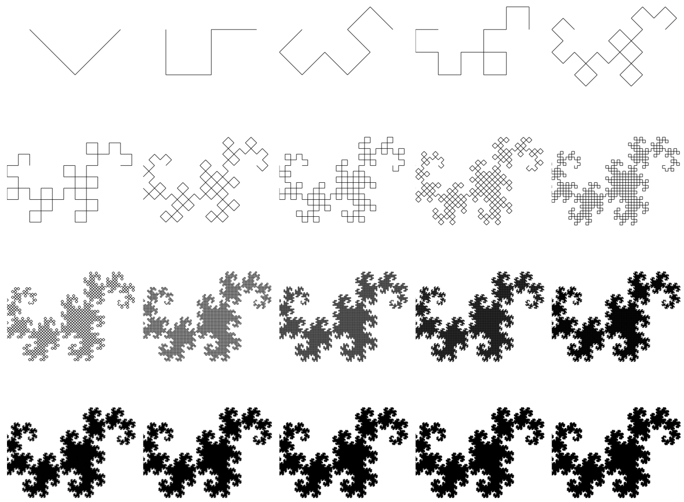


Figure 12: TODO

Figure 13: TODO

6.1.1 Dragon Curve

6.1.2 Koch Snowflake

6.2 Calculating the Dimension of Julia Set

It converges too slowly The Julia set (discussed in section 7) can be solved by ...

explain the code a little bit here

as shown in listing

A value on the complex plane can be associated with the julia set by iterating that value against a function of the form $z \rightarrow z^2 + \alpha + i\beta$ and measureing whether or not that value diverges or converges. This process is demonstrated in listing 4.

By associating each value on the complex plane with an element of a matrix an image of this pattern may be produced, see for example figure RABBIT

So I run the code shown in listing **??** which calls a file `./Julia-Set-Dimensions-functions.jl` which is shown in listing **??** which returs the values shown in table 1.

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

###############################################################
#### Investigate Plot ########################################
```

```
#!/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 4: 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 $z \to z^2 + \alpha + i\beta$

```julia
############################################################

f(z) = z^2 -1

test_mat = make_picture(800,800, z -> z^2 + 0.37-0.2*im)
test_mat = make_picture(800,800, z -> z^2 + -0.123+0.745*im)
test_mat = make_picture(800,800, f)
GR.imshow(test_mat) # PyPlot uses interpolation = "None"


test_mat = outline(test_mat)
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")

## Return the perimeter
sum(test_mat)



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
#





using CSV

@time data=scaleAndMeasure(9000, 10000 , 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

# import Gadfly
#
# iris = dataset("datasets", "iris")
# p = Gadfly.plot(iris, x=:SepalLength, y=:SepalWidth, Geom.point);
# img = SVG("iris_plot.svg")
# draw(img, p)


# The trailing `;` supresses output, equivalently:



## Other Fractals to look at for this maybe?
  # GR.imshow(test_mat) # PyPlot uses interpolation = "None"
  # GR.imshow(make_picture(500, 500, z -> z^2 + 0.37-0.2*im)) # PyPlot uses interpolation = "None"
  # GR.imshow(make_picture(500, 500, z -> z^2 + 0.38-0.2*im)) # PyPlot uses interpolation = "None"
  # GR.imshow(make_picture(500, 500, z -> z^2 + 0.39-0.2*im)) # PyPlot uses interpolation = "None"
```

```
using GR
using DataFrames
using Gadfly
using GLM
```

```julia
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 ########################################
############################################################
# TODO this should be inside a function

"""
# 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
```

```julia
    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)

    data = DataFrame(scale = scale, mass = mass)
    return data
end
```

This returns the Values:

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

Table 1: TODO

6.2.1 Using Linear Regression

- Avoiding `Abs` is twice as fast
- Column wise is faster in fortran/julia/R slower in C/Python We have no evidence to show that the dimension will be stable, this is good for coastlines and stuff.

  to do that we use linear regression.

Performance

- Switching from `abs()` to sqaured help
- Taking advantage of multi core processing in loops
- pmap was chosen because it scales better for expensive jobs.

  Comparison

```julia
function tme()
    start = time()
    data = scaleAndMeasure(900, 1000, 9)
    length = time() - start
    print(length, "\n")
    return length
end
times = [tme() for i in 1:10 ]
```

Function	Mean Time
pmap	2.2825

6.3 My Fractal

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:

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

6.3.1 Graphics

6.3.2 Discuss Pattern shows Fibonacci Numbers

Angle Relates to Golden Ratio

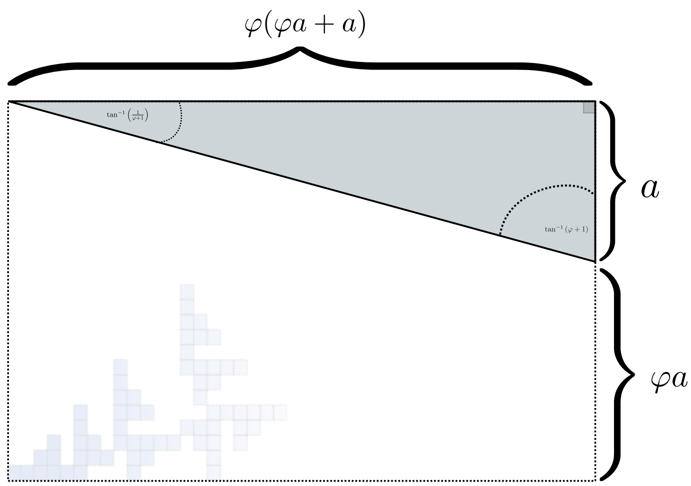


Figure 14: TODO

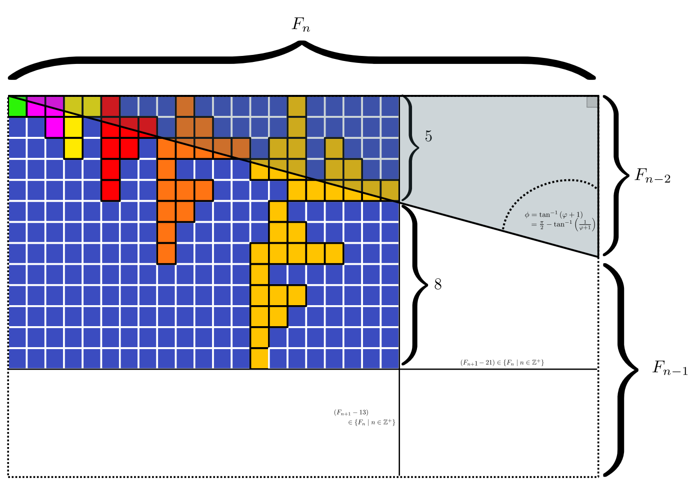


Figure 15: TODO

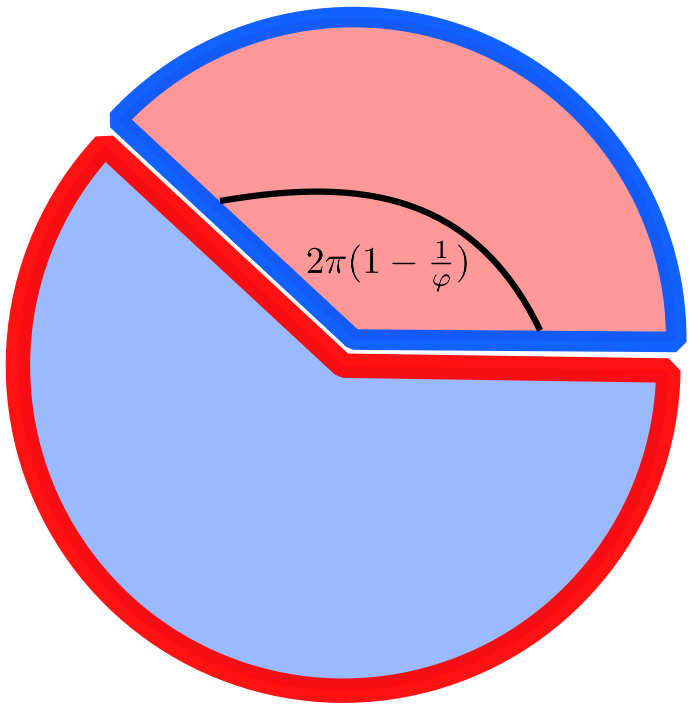


Figure 16: TODO

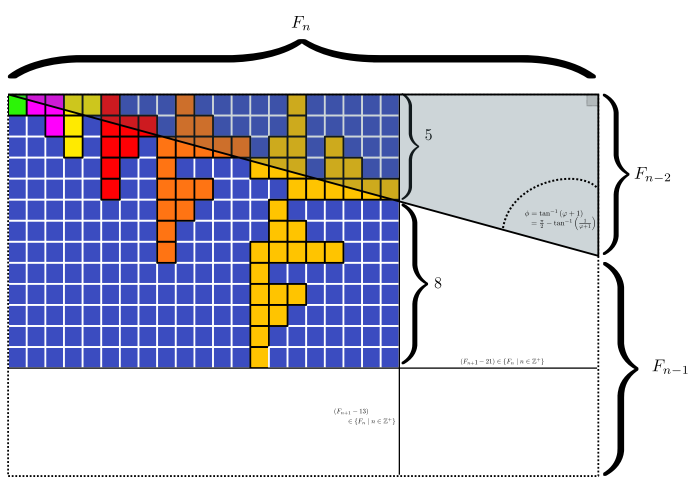


Figure 17: TODO

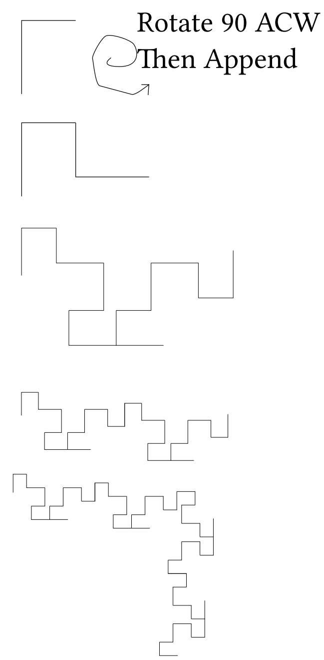


Figure 18: TODO

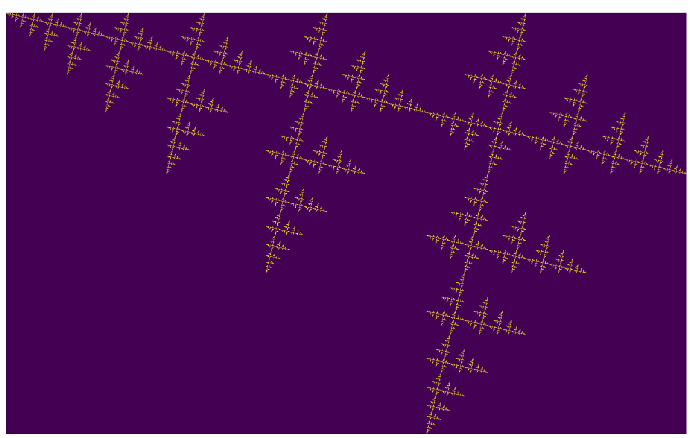
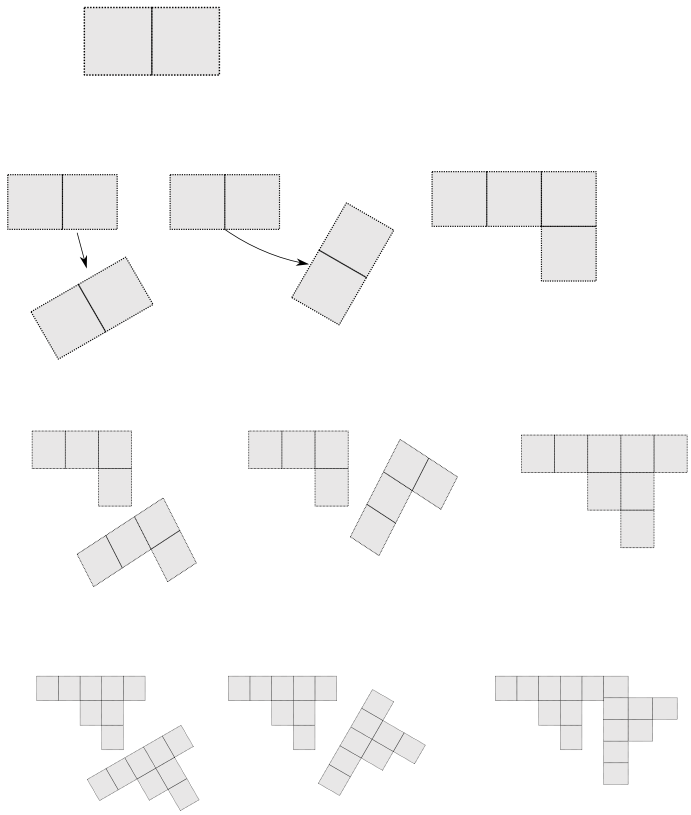


Figure 19: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well



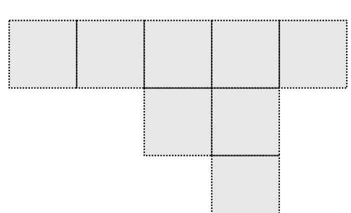


Figure 20: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well.

6.3.3 Prove Fibonacci using Monotone Convergence Theorem

Consider the series:

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

Such that:

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

Show that the Series is Monotone

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

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

Show that the Series is Bounded

Find the Limit

$$\begin{aligned} G &= \frac{F_n + F_{n+1}}{F_{n+1}} \\ &= 1 + \frac{F_{n-1}}{F_n} \\ \text{Recall that } F_n &> 0 \forall n \\ &= 1 + \frac{1}{|G|} \\ \implies 0 &= G^2 - G + 1; \quad G > 0 \\ \implies G = \varphi &= \frac{\sqrt{5} - 1}{2} \quad \square \end{aligned}$$

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:

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

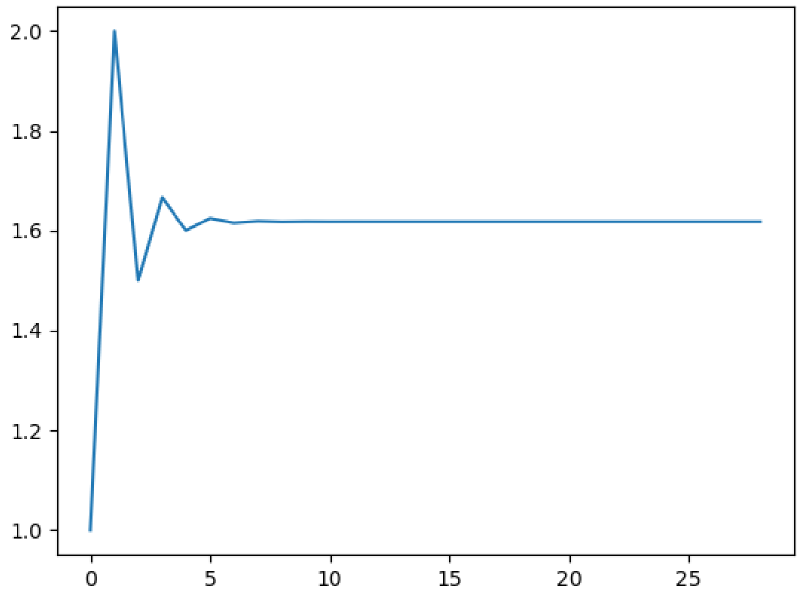
or how 0 is the additive identity and it shows why generating functions are so useful. Observe also that

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

Python

```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")
```



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

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

6.3.5 Dimension of my Fractal

$\log_{\varphi}(2)$

6.3.6 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
```

```julia
    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))
```

7 Julia Sets and Mandelbrot Sets

The julia set is the outline.

The mandelbrot has to do with whether or not it's connected.

7.1 The math behind it

7.1.1 Like Escaping after 2

I cannot figure this out, I need more time, look around Ch. 12 of falconer [5]

8 Fibonacci Sequence

8.1 Introduction

The *Fibonacci Sequence* and *Golden Ratio* share a deep connection[9] and occur in patterns observed in nature very frequently (see [23, 1, 18, 19, 12, 22]), an example of such an occurence is discussed in section 8.4.1.

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.

8.2 Computational Approach

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:

$$F_n = F_{n-1} + F_{n-2} \tag{19}$$

This type of recursive relation can be expressed in *Python* by using recursion, as shown in listing 5, however using this function will reveal that it is extraordinarily slow, as shown in listing 6, this is because the results of the function are not cached and every time the function is called every value is recalculated[10], 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 [8], 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 7.

[9] See section

[10] Dr. Hazrat mentions something similar in his book with respect to *Mathematica*® [10, Ch. 13]

```python
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 5: Defining the *Fibonacci Sequence* (19) using Recursion

```python
start = time.time()
rec_fib(35)

print(str(round(time.time() - start, 3)) + "seconds")


## 2.245seconds
```

Listing 6: Using the function from listing 5 is quite slow.

```python
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 7: Caching the results of the function previously defined 6

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

## 8.3923e-05seconds
```

Restructuring the problem to use iteration will allow for even greater performance as demonstrated by finding F_{10^6} in listing 8. Using a compiled language such as *Julia* however would be thousands of times faster still, as demonstrated in listing 9.

```
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 8: Using Iteration to Solve the Fibonacci Sequence

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

Listing 9: Using Julia with an iterative approach to solve the 1 millionth fibonacci number

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

8.3 Exponential Generating Functions

8.3.1 Motivation

Consider the *Fibonacci Sequence* from (19):

$$\begin{aligned} & a_n = a_{n-1} + a_{n-2} \\ \iff & a_{n+2} = a_{n+1} + a_n \end{aligned} \tag{20}$$

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

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

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

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

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

8.3.2 Example

Consider using the following generating function, (the derivative of the generating function as in (22) and (23) is provided in section 8.3.3)

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \cdot \frac{x^n}{n!} \right] \tag{21}$$

$$\implies f'(x) = \sum_{n=0}^{\infty} \left[a_{n+1} \cdot \frac{x^n}{n!} \right] \tag{22}$$

$$\implies f''(x) = \sum_{n=0}^{\infty} \left[a_{n+2} \cdot \frac{x^n}{n!} \right] \tag{23}$$

So the recursive relation from (20) could be expressed :

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

And hence by applying (21):

$$f'' \left(x \right) = f' \left(x \right) + f \left(x \right) \tag{24}$$

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

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

By equating this to the power series:

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

Now given that:

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

We can conclude that:

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

By applying the initial conditions:

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

And so finally we have the solution to the *Fibonacci Sequence* 20:

$$
\begin{aligned}
a_n &= \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] \\
&= \frac{\varphi^n - \psi^n}{\sqrt{5}} \\
&= \frac{\varphi^n - \psi^n}{\varphi - \psi} \qquad (25)
\end{aligned}
$$

where:

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

8.3.3 Derivative of the Exponential Generating Function

Base Differentiating the exponential generating function has the effect of shifting the sequence to the backward: [13]

$$
\begin{aligned}
f(x) &= \sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right] \qquad (26) \\
f'(x)) &= \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right]\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}x}\left(a_0 \frac{x^0}{0!} + a_1 \frac{x^1}{1!} + a_2 \frac{x^2}{2!} + a_3 \frac{x^3}{3!} + \ldots \frac{x^k}{k!}\right) \\
&= \sum_{n=0}^{\infty}\left[\frac{\mathrm{d}}{\mathrm{d}x}\left(a_n \frac{x^n}{n!}\right)\right] \\
&= \sum_{n=0}^{\infty}\left[\frac{a_n}{(n-1)!} x^{n-1}\right] \\
\implies f'(x) &= \sum_{n=0}^{\infty}\left[\frac{x^n}{n!} a_{n+1}\right] \qquad (27)
\end{aligned}
$$

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

$$
f^{(k)}(x) = \sum_{n=0}^{\infty} \frac{a_{n+k} \cdot x^n}{n!} \quad \text{for } k \geq 0 \qquad (28)
$$

Assume that. $f^{(k)}(x) = \sum_{n=0}^{\infty} \frac{a_{n+k} \cdot x^n}{n!}$

Using this assumption, prove for the next element $k+1$
We need $f^{(k+1)}(x) = \sum_{n=0}^{\infty} \frac{a_{n+k+1} \cdot x^n}{n!}$

$$\begin{aligned}
\text{LHS} &= f^{(k+1)}(x) \\
&= \frac{\mathrm{d}}{\mathrm{d}x}\left(f^{(k)}(x)\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{n=0}^{\infty} \frac{a_{n+k} \cdot x^n}{n!}\right) && \text{by assumption} \\
&= \sum_{n=0}^{\infty} \frac{a_{n+k} \cdot n \cdot x^{n-1}}{n!} \\
&= \sum_{n=1}^{\infty} \frac{a_{n+k} \cdot x^{n-1}}{(n-1)!} \\
&= \sum_{n=0}^{\infty} \frac{a_{n+k+1} \cdot x^n}{n!} \\
&= \text{RHS}
\end{aligned}$$

Thus, if the derivative of the series shown in (21) shifts the sequence across, then every derivative thereafter does so as well, because the first derivative has been shown to express this property (27), all derivates will.

8.3.4 Homogeneous Proof

An equation of the form:

$$\sum_{i=0}^{n}\left[c_i \cdot f^{(i)}(x)\right] = 0 \tag{29}$$

is said to be a homogenous linear ODE: [26, 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, [26, 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 (30)). 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 [14]
- $\exists i, j \in \mathbb{Z}^+ \cap [0, k]$
  - These is often referred to as repeated roots [14, 27] with a multiplicity corresponding to the number of repetitions of that root [20, §3.2]

Unique Roots of Characteristic Equation

Example An example of a recurrence relation with all unique roots is the fibonacci sequence, as described in section 8.3.2.

Proof Consider the linear recurrence relation (30):

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

This implies:

$$\sum_{n=0}^{\infty} \left[\sum_{i=0}^{k} \left[\frac{x^n}{n!} c_i a_n\right]\right] = 0 \tag{30}$$

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

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

By implementing the exponential generating function as shown in (21), this provides:

$$\sum_{i=0}^{k} \left[c_i f^{(i)}(x)\right] \tag{33}$$

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 [26, 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):

$$\begin{aligned}
\sum_{i=0}^{k} [k_i e^{m_i x}] &= \sum_{i=0}^{k} \left[k_i \sum_{n=0}^{\infty} \frac{(m_i x)^n}{n!} \right] \\
&= \sum_{i=0}^{k} \sum_{n=0}^{\infty} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty} \sum_{i=0}^{k} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty} \left[\frac{x^n}{n!} \sum_{i=0}^{k} [k_i m_i^n] \right], \quad \exists k_i \in \mathbb{C}, \ \ \forall i \in \mathbb{Z}^+ \cap [1, k]
\end{aligned} \tag{34}$$

Recall the definition of the generating function from (21), by relating this to (34):

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

$\square$

This can be verified by the fibonacci sequence as shown in section 8.3.2, 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

Example Consider the following recurrence relation:

$$a_n - 10a_{n+1} + 25a_{n+2} = 0 \tag{35}$$
$$\implies \sum_{n=0}^{\infty}\left[a_n\frac{x^n}{n!}\right] - 10\sum_{n=0}^{\infty}\left[\frac{x^n}{n!}+\right] + 25\sum_{n=0}^{\infty}\left[a_{n+2}\frac{x^n}{n!}\right] = 0$$

By applying the definition of the exponential generating function at (21) :

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

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

$$\begin{aligned} m^2 - 10m + 25 &= 0 \\ (m-5)^2 &= 0 \\ m &= 5 \end{aligned} \tag{36}$$

Herein lies a complexity, in order to solve this, the solution produced from (36) can be used with the *Reduction of Order* technique to produce a solution that will be of the form [27, §4.3].

$$f(x) = c_1 e^{5x} + c_2 x e^{5x} \tag{37}$$

(37) 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 (TODO Prove this in section):

$$x^k e^x = \sum_{n=0}^{\infty}\left[\frac{x^n}{(n-k)!}\right], \quad \exists k \in \mathbb{Z}^+ \tag{38}$$

by applying identity (38) to equation (37)

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

Given the defenition of the exponential generating function from (21)

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right] \\ \iff a_n &= c_{15}^n + c_2 n_5^n \end{aligned}$$

□

Proof In order to prove the the solution for a k^{th} order recurrence relation with k repeated

Consider a recurrence relation of the form:

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

By substituting for the value of the generating function (from (21)):

$$\sum_{i=0}^{k}\left[c_i f^{(k)}(x)\right] \tag{39}$$

Assume that (39) corresponds to a chareceteristic polynomial with only 1 root of multiplicity k, the solution would hence be of the form:

$$\sum_{i=0}^{k}\left[c_i m^i\right] = 0 \land m = B, \;\; \exists! B \in \mathbb{C}$$

$$\implies f(x) = \sum_{i=0}^{k}\left[x^i A_i e^{mx}\right], \quad \exists A \in \mathbb{C}^+, \;\; \forall i \in [1, k] \cap \mathbb{N} \tag{40}$$

$$\tag{41}$$

If we assume that (see section 8.3.4):

$$k \in \mathbb{Z} \implies x^k e^x = \sum_{n=0}^{\infty}\left[\frac{x^n}{(n-k)!}\right] \tag{42}$$

By applying this to (40) :

$$
\begin{aligned}
f(x) &= \sum_{i=0}^{k}\left[A_i \sum_{n=0}^{\infty}\left[\frac{(xm)^n}{(n-i)!}\right]\right] \\
&= \sum_{n=0}^{\infty}\left[\sum_{i=0}^{k}\left[\frac{x^n}{n!}\frac{n!}{(n-i)}A_i m^n\right]\right] && (43) \\
&= \sum_{n=0}^{\infty}\left[\frac{x^n}{n!}\sum_{i=0}^{k}\left[\frac{n!}{(n-i)}A_i m^n\right]\right] && (44)
\end{aligned}
$$

Recall the generating function that was used to get 39:

$$
\begin{aligned}
f(x) &= \sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right] \\
\implies a_n &= \sum_{i=0}^{k}\left[A_i \frac{n!}{(n-i)!} m^n\right] \\
&= \sum_{i=0}^{k}\left[m^n A_i \prod_{0}^{k}\left[n-(i-1)\right]\right] && (45)
\end{aligned}
$$

$\because \quad i \leq k$

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

$\square$

General Proof In sections 8.3.4 and 8.3.4 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, when the charecteristic polynomial has either complex roots or 1 repeated root. Generally 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

Power Series Combination In this section a proof for identity 42 is provided.

1. 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:

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

8.4 Fibonacci Sequence and the Golden Ratio

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

$$\begin{aligned}
F_n &= \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\varphi^n - \psi^n}{\sqrt{5}} \\
\iff \frac{F_{n+1}}{F_n} &= \frac{\varphi^{n+1} - \psi^{n+1}}{\varphi^n - \psi^n} \\
\iff \lim_{n\to\infty}\left[\frac{F_{n+1}}{F_n}\right] &= \lim_{n\to\infty}\left[\frac{\varphi^{n+1} - \psi^{n+1}}{\varphi^n - \psi^n}\right] \\
&= \frac{\varphi^{n+1} - \lim_{n\to\infty}\left[\psi^{n+1}\right]}{\varphi^n - \lim_{n\to\infty}\left[\psi^n\right]} \\
\text{because } \mid \psi \mid < 0 \; n \to \infty \implies \psi^n \to 0: & \\
&= \frac{\varphi^{n+1} - 0}{\varphi^n - 0} \\
&= \varphi
\end{aligned}$$

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 8.4.1 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*.

8.4.1 Fibonacci Sequence in Nature (This may be Removed)

The distribution of sunflower seeds is an example of the *Fibonacci Sequence* occuring in a pattern observed in nature (see Figure 23).

Imagine that the process a sunflower follows when placing seeds is as follows: [11]

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).

[11] This process is simply conjecture, other than seeing a very nice example at *MathIsFun.com* [19], 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.

4. Repeat this process until a seed hits some outer boundary.

This process can be simulated in Julia [2] as shown in listing 10,[12] which combined with *ImageMagick* (see e.g. 27), produces output as shown in figure 21 and 22.

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 [19]:

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

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 22, 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 8.4 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 21 has a ratio of rotation of $\frac{1}{\pi}$, the spirals look similar to one direction of the spirals occuring in figure 22, it is not clear if there is any significance to this similarity.

Figure 21: Simulated Distribution of Sunflower seeds as described in section 8.4.1 and listing 10

Figure 22: Optimisation of simulated distribution of Sunflower seeds occurs for $\theta = 2\varphi\pi$ as described in section 8.4.1 and listing 10

Figure 23: Distribution of the seeds of a sunflower (see [3] licenced under CC)

[12] 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.

```
 = 1.61803398875
 =  ^-1
 = 0.61803398875
function sfSeeds(ratio)
 = Turtle()
    for   in [(ratio*2* )*i for i in 1:3000]
        gsave()
        scale(0.05)
        rotate( )
#        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( )
end 600 600
```

Listing 10: Simulation of the distribution of sunflowers as described in section 8.4.1

9 Julia Sets

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.

9.1 Introduction

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

9.2 Motivation

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

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

Although this seems trivial this can be generalised.

Consider:

- The complex plane for $|z| \leq 1$
- Some function $f_c(z) = z^2 + c, \quad c \leq 1 \in \mathbb{C}$ 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.

9.3 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[13] and define the operations as described in listing 11.[14]

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 [15], this is implemented in listing 12 and the corresponding output shown in 24.

with respect to listing 12:

- 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 24: 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 25 and 26.

Figure 25: Circle of Convergence for $f_0 : z \to z^2 - 1$

Figure 26: Circle of Convergence for $f_{\frac{1}{4}+\frac{i}{2}} : z \to z^2 + \frac{1}{4} + \frac{i}{2}$

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

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

[15] 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 11: Defining Complex Operations with vectors

```
%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 12: Circle of Convergence of z under recursion

To investigate this further consider the more general function $f_{0.8e^{\pi i \tau}} : z \to z^2 + 0.8e^{\pi i \tau}, \quad \tau \in \mathbb{R}$, many fractals can be generated using this set by varying the value of τ[16].

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 13. 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 14 where 1200 pictures at a 2.25 MP resolution were produced. [17]

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 15, the collage is shown in figure 27 and a corresponding animation is available online[18]].

Figure 27: Various fracals corresponding to $f_{0.8e^{\pi i \tau}}$

10 MandelBrot

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

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

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

$$z_{n+1} = z_n^2 + c \tag{46}$$

$$z_0 = c \tag{47}$$

This strategy is implemented in listing and produces the output shown in figure 28.

Figure 28: Mandelbrot Set produced in *Python* as shown in listing 16

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

[16]This approach was inspired by an animation on the *Julia Set* Wikipedia article [11]

[17]On my system this took about 30 minutes.

[18]https://dl.dropboxusercontent.com/s/rbu25urfg8sbwfu/out.gif?dl=0

```
# * 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 13: 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* )
        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 14: 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 15: Using bash, `ffmpeg` and *ImageMagick* to combine the images and produce an animation.

```
%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
```

in little time (about 10 minutes on my system), this is demonstrated in listing and the corresponding FITS image is available-online.[19]

```
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
    return pic_mat
end


using FITSIO
function save_picture(filename, matrix)
    f = FITS(filename, "w");
    # data = reshape(1:100, 5, 20)
```

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

```
    # 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 29: Screenshot of Mandelbrot FITS image produced by listing

11 Appendix

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

11.1 Finding Material

```
recoll -c /home/ryan/Dropbox/Books/Textbooks/Mathematics/Chaos_Theory/c⌟
↪   haos_books_recoll &
↪   disown
```

11.2 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 " "))))
```

11.3 Section attribution

```
Section Attribution


1. Report
2. Hausdorff Dimension                                        :Ryan:
3. Box Counting                                               :Ryan:
4. Fractals Generally                                        :James:
5. Generating Self Similar Fractals                           :Ryan:
6. Fractal Dimensions                                         :Ryan:
7. Julia Sets and Mandelbrot Sets                             :Ryan:
8. Fibonacci Sequence                                   :Ryan:James:
.. 1. Introduction                                            :Ryan:
.. 2. Computational Approach                                  :Ryan:
.. 3. Exponential Generating Functions
..... 1. Motivation                                           :Ryan:
..... 2. Example                                              :Ryan:
..... 3. Derivative of the Exponential Generating Function
..... 4. Homogeneous Proof                              :Ryan:James:
.. 4. Fibonacci Sequence and the Golden Ratio                 :Ryan:
9. Julia Sets                                                 :Ryan:
10. MandelBrot                                                :Ryan:
11. Appendix
12. Bibliography


How I made this:

1. add `#+OPTIONS: tags:t' to the header
2. Collapse all headlines revealing
only the section detail that is desired
3. `C-c C-e C-v t U' 4. This will export
visible to a text buffer, the TOC will have the tags of the visible
headings so just delete what isn't desired
```

References

[1] 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. 42).

[2] 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 p. 58).

[3] Simon Brass. *CC Search*. 2006, September 5. URL: https://search.creativecommons.org/ (visited on 08/28/2020) (cit. on p. 58).

[4] 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. 5, 9).

[5] 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, 10, 42).

[6] 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. 5, 9, 10).

[7] 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. 61).

[8] *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. 42).

[9] 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. 3).

[10] 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. 42).

[11] *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. 64).

[12] 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. 42).

[13] 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. 49).

[14] 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. 51).

[15] 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. 4).

[16] *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. 10).

[17] Mark Pollicott. *Fractals and Dimension Theory*. 2005-May. URL: https://warwick.ac.uk/fac/sci/maths/people/staff/mark_pollicott/p3 (cit. on p. 10).

[18] 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. 42).

[19] *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. 42, 57, 58).

[20] 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. 51).

[21] 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. 2, 3, 13).

[22] 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. 42).

[23] Shelly Allen. *Fibonacci in Nature*. URL: https://fibonacci.com/nature-golden-ratio/ (visited on 08/28/2020) (cit. on p. 42).

[24] 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 pp. 4, 9, 10, 20).

[25] Zhi Wei Zhu, Zuo Ling Zhou, and Bao Guo Jia. "On the Lower Bound of the HausdorffMeasure of the Koch Curve". In: *Acta Mathematica Sinica* 19.4 (Oct. 1, 2003), pp. 715–728. ISSN: 1439-7617. DOI: 10.1007/s10114-003-0310-2. URL: https://doi.org/10.1007/s10114-003-0310-2 (visited on 09/18/2020) (cit. on p. 10).

[26] Dennis G Zill and Michael R Cullen. *Differential Equations*. 7th ed. Brooks/Cole, 2009 (cit. on pp. 50, 52).

[27] 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. 51, 53).