The Emergence of Patterns in Nature and Chaos Theory

Ryan Greenup & James Guerra

September 24, 2020

Contents

1 Report

1.1 Hausdorff Dimension Ryan

1.1.1 Topological Equivalence

Topology is an area of mathematics concerned with ideas of continuity through the study of figures that are preserved under homeomorphic transformations. [15]

Two figures are said to be homeomorphic if there is a continuous bijective mapping between the two shapes [27, 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 THIS IS NOT CORRECT, I MUST SHOW THAT THE DIMENSION IS $\frac{ln(4)}{\ln(3)}$ BUT I'M SIMPLY OUT OF TIME.

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

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

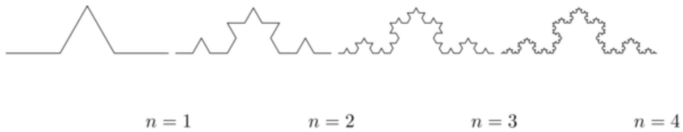


Figure 1: Progression of the Koch Snowflake

}

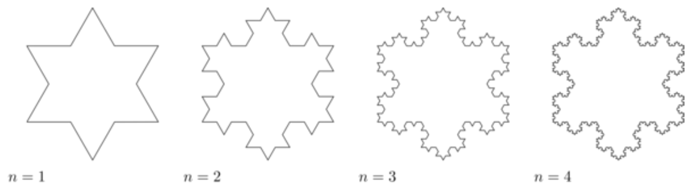


Figure 2: Progression of the Koch Snowflake

In the development of topology

1.1.2 Notes from p. 27 Falconer

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

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

- The use of balls is a simpler but equivalent approach to the theory [12, §2.4] because any set of diamater r can be enclosed in a ball of radius $\frac{r}{2}$ [10, 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. [12, 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.

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:

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

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

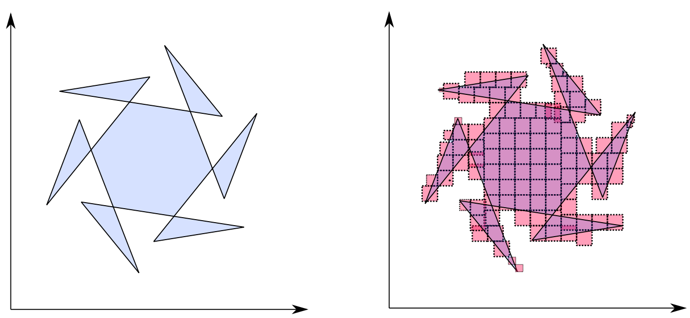


Figure 3: The shape on the left corresponds to $F \subset \mathbb{R}^{\nvDash}$, each identical square box on the right represents a set U_i.

Lower Dimension Hausdorff Measurements

1. 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 1.1.2 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.

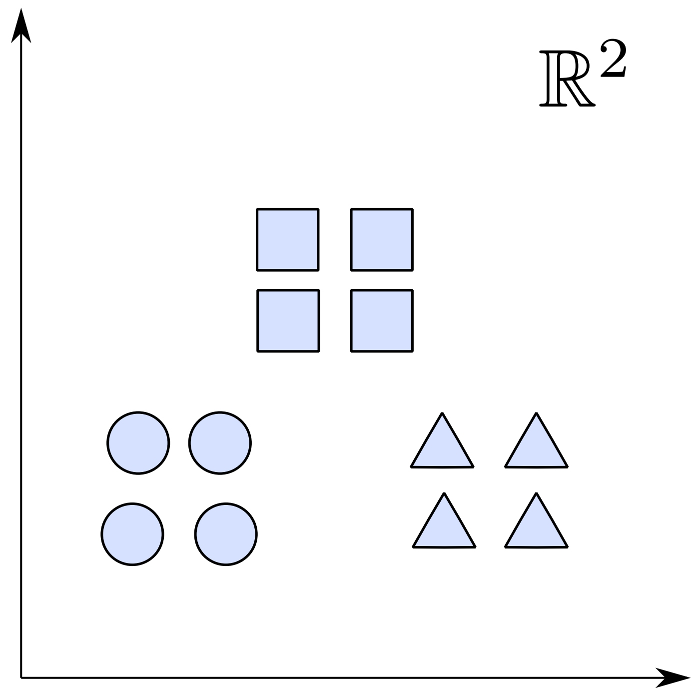


Figure 4: A disconnected subset of $\mathbb{R}^2$, the squares have a diameter of $\sqrt{2}$, the circles 1 and the equilateral triangles 1.

This is consistent with shapes of other shapes as well, consider the koch snowflake introduced in section 1.1.1 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.

2. 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:

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

[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.

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

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

$$\leq \lim_{\delta\to 0}\left(\delta^{t-s}\right)\cdot \lim_{\delta\to 0}\left(\sum_i \left[|U_i|^s\right]\right) \tag{15}$$

$$\leq 0 \tag{16}$$

Thus $\forall t > s$, if s is a finite limit value:

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

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

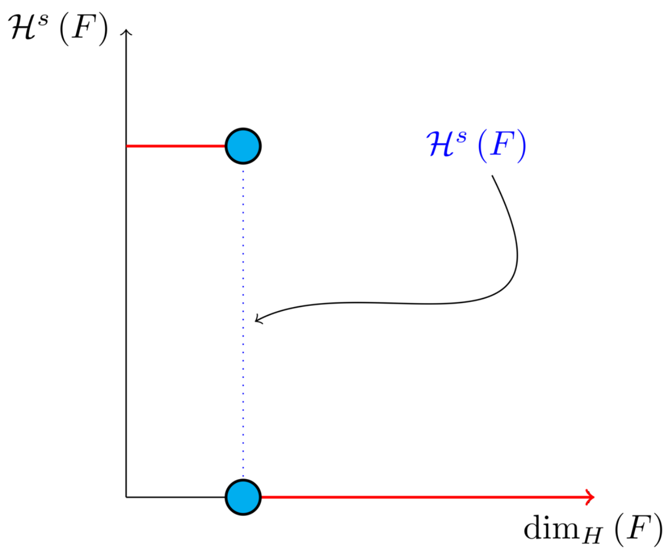


Figure 5: The value of the s-dimensional *Hausdorff Measure* of some subset of *Euclidean space* $F \in \mathbb{R}^n$ is 0 or ∞ when the dimension of F is not equal to s.

Hausdorff Dimension

1.2 Fractals Generally

- Many Fractals have a non finite dimension
- An exception to this is the Mandelbrot set or dragon curve which are two dimensional

1.3 Generating Self Similar Fractals

Examples

Vicsek Fractal

Sierpinskis Carpet Explained more in the book [4]

Triangle

1. Chaos Game

1.4 Fractal Dimensions Ryan

Three ways to generate

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

1.4.1 Turtle

Matrices can't explain all patterns, Turtles are useful

Dragon Curve

Koch Snowflake

1.4.2 Calculating the Dimension of Julia Set

It converges too slowly The Julia set (discussed in section 1.5) 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 $z \to z^2 + \alpha + i\beta$ and measureing whether or not that value diverges or converges. This process is demonstrated in listing 1.

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

```
#!/bin/julia
function juliaSet(z, num, my_func, boolQ=true)
    count = 1
    # Iterate num times
```

[4] See Ch. 2.7 of [27, Ch. 2.7]

```
    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 1: 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 $z \to z^2 + \alpha + i\beta$

Using Linear Regression

- Avoiding `Abs` is twice as fast
- Column wise is faster in fortran/julia/R slower in C/Python

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

```
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

1.4.3 My Fractal

Graphics

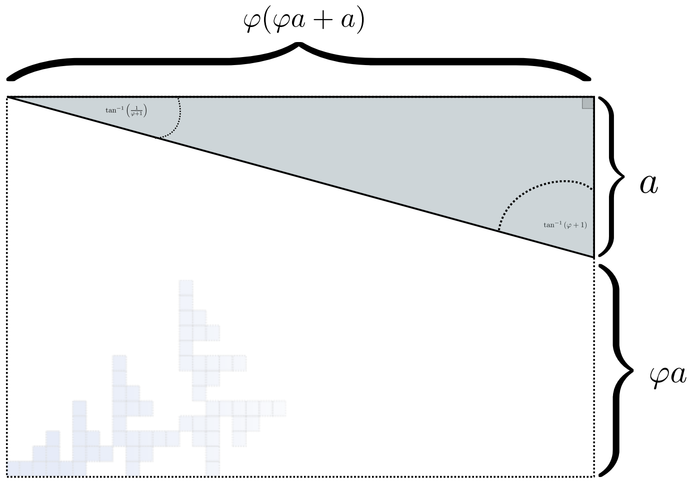


Figure 6: TODO

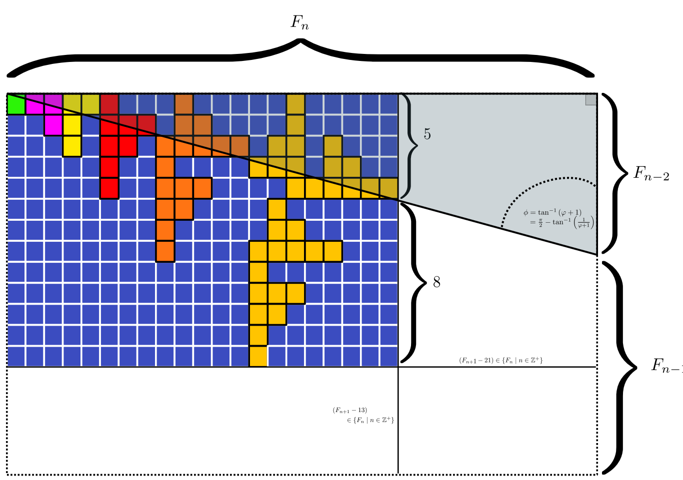


Figure 7: TODO

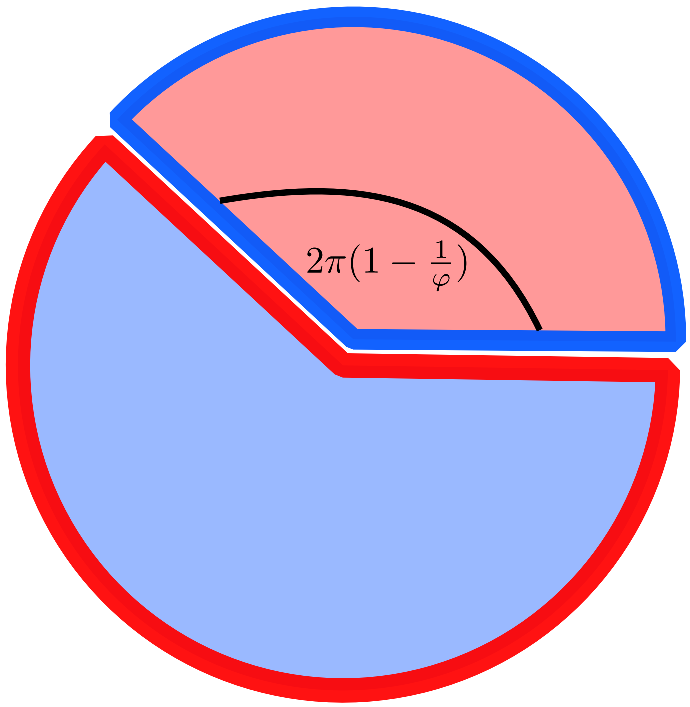


Figure 8: TODO

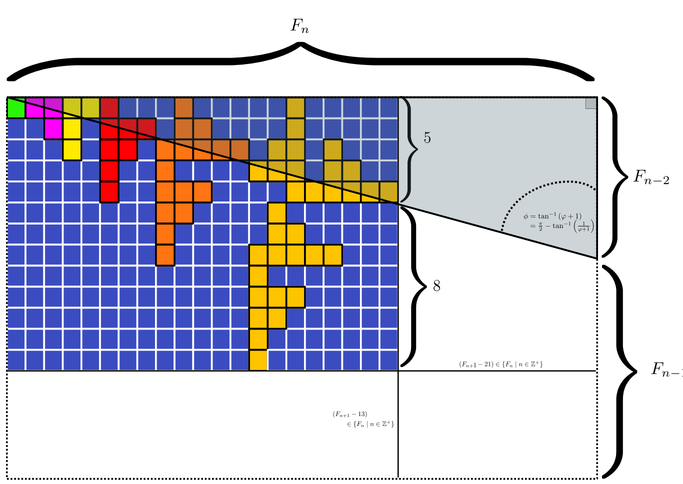


Figure 9: TODO

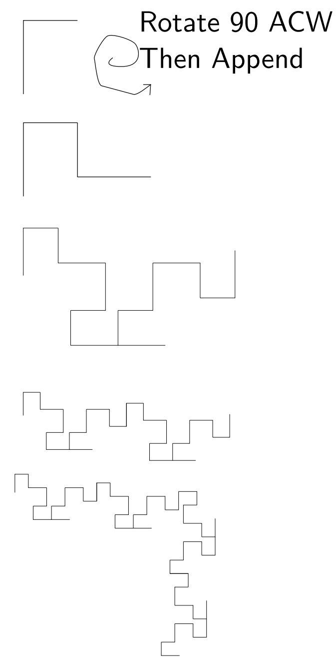


Figure 10: TODO

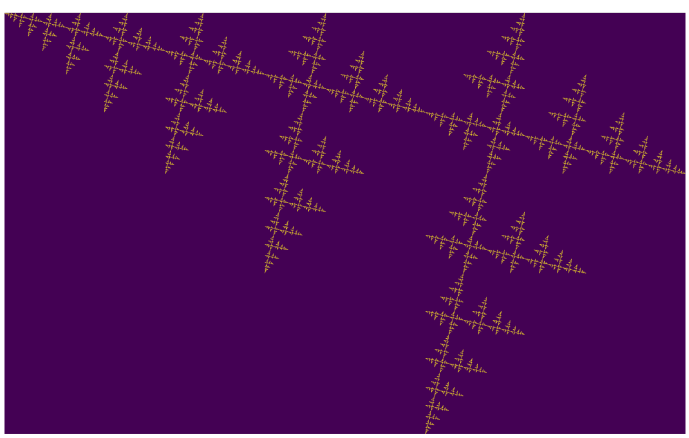
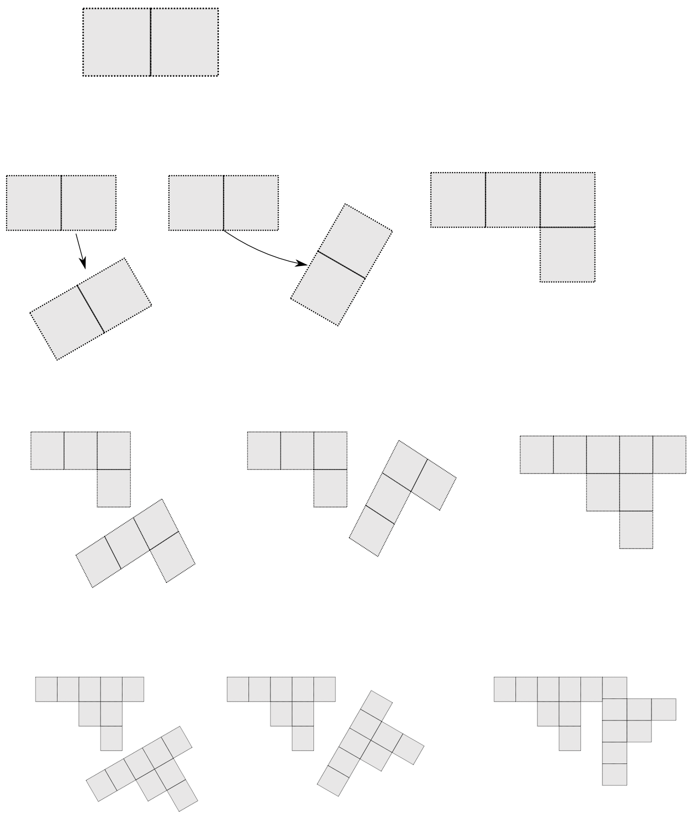


Figure 11: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well



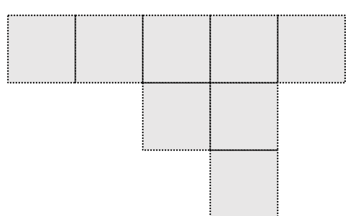


Figure 12: Fractal that emerges by Rotating and appending boxes, this demonstrates the relationship between the Fibonacci numbers and golden ratio very well

Discuss Pattern shows Fibonacci Numbers

Angle Relates to Golden Ratio

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

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

```
#+BEGIN_SRC python :exports both :results output graphics file :file ./a.png
#+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")
```

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

Python

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

Dimension of my Fractal

$\log_{\varphi}(2)$

1.5 Julia Sets and Mandelbrot Sets

The julia set is the outline.

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

1.5.1 The math behind it

Like Escaping after 2

1.6 Turing

1.7 Appendix

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

2 Outline

2.1 Introduction — Ryan

This project, at the outset, was very broadly concerned with the use of *Python* for computer algebra. Much to the the reluctance of our supervisor we have however resolved to look at a broad variety of tools (see section 2.8.3), in particular a language we wanted an opportunity to explore was *Julia* [4] [5].

In order to give the project a more focused direction we have decided to look into: [6]

- The Emergence of patterns in Nature
- Chaos Theory & Dynamical Systems
- Fractals

These three topics are very tightly connected and so it is difficult to look at any one in a vacuum, they also almost necessitate the use of software packages due to the fact that these phenomena appear to occur in recursive systems, more over such software needs to perform very well under recursion and iteration (making this a very good focus for this topic generally, and an excuse to work with Julia as well).

2.2 Programming Recursion — Ryan

As an introduction to *Python* generally, we undertook many problem questions which have been omitted from this outline, however, this one in particular offered an interesting insight into the difficulties we may encounter when dealing with recursive systems.

[5] See section

[6] The amount of independence that our supervisor afforded us to investigate other languages is something that we are both extremely grateful for.

2.2.1 Iteration and Recursion

Consider the series shown in (18)[7] :

$$g\left(k\right)=\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2+\sqrt{3}}}{3}\frac{\sqrt{2+\sqrt{3+\sqrt{4}}}}{4}\cdot\ldots\frac{\sqrt{2+\sqrt{3+\ldots+\sqrt{k}}}}{k} \tag{18}$$

let's modify this for the sake of discussion:

$$h\left(k\right)=\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{3+\sqrt{2}}}{3}\cdot\frac{\sqrt{4+\sqrt{3+\sqrt{2}}}}{4}\cdot\ldots\cdot\frac{\sqrt{k+\sqrt{k-1+\ldots\sqrt{3+\sqrt{2}}}}}{k} \tag{19}$$

The function h can be expressed by the series:

$$h\left(k\right)=\prod_{i=2}^{k}\left(\frac{f_i}{i}\right)\quad:\quad f_i=\sqrt{i+f_{i-1}},\;\; f_1=1$$

Within *Python*, it isn't difficult to express h, the series can be expressed with recursion as shown in listing 2, this is a very natural way to define series and sequences and is consistent with familiar mathematical thought and notation. Individuals more familiar with programming than analysis may find it more comfortable to use an iterator as shown in listing 3.

```python
from sympy import *
def h(k):
    if k > 2:
        return f(k) * f(k-1)
    else:
        return 1

def f(i):
    expr = 0
    if i > 2:
        return sqrt(i + f(i -1))
    else:
        return 1
```

Listing 2: Solving (19) using recursion.

```python
from sympy import *
def h(k):
    k = k + 1 # OBOB
    l = [f(i) for i in range(1,k)]
    return prod(l)

def f(k):
    expr = 0
    for i in range(2, k+2):
```

[7]This problem is taken from Project A (44) of Dr. Hazrat's *Mathematica: A Problem Centred Approach* [16]

```
            expr = sqrt(i + expr, evaluate=False)
        return expr/(k+1)
```
Listing 3: Solving (19) by using a `for` loop.

Any function that can be defined by using iteration, can always be defined via recursion and vice versa [6, 5] (see also [32, 17]),

there is however, evidence to suggest that recursive functions are easier for people to understand [2] and so should be favoured. Although independent research has shown that the specific language chosen can have a bigger effect on how well recursive as opposed to iterative code is understood [31].

The relevant question is "*which method is often more appropriate*?", generally the process for determining which is more appropriate is to the effect of:

1. Write the problem in a way that is easier to write or is more appropriate for demonstration
2. If performance is a concern then consider restructuring in favour of iteration
   - For interpreted languages such ***R*** and *Python*, loops are usually faster, because of the overheads involved in creating functions [32] although there may be exceptions to this and I'm not sure if this would be true for compiled languages such as *Julia*, *Java*, ***C*** etc.

Some Functions are more difficult to express with Recursion in

Attacking a problem recursively isn't always the best approach however. Consider the function $g(k)$ from (18):

$$\begin{aligned} g(k) &= \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{3}}}{3} \frac{\sqrt{2+\sqrt{3+\sqrt{4}}}}{4} \cdot \ldots \frac{\sqrt{2+\sqrt{3+\ldots+\sqrt{k}}}}{k} \\ &= \prod_{i=2}^{k} \left(\frac{f_i}{i}\right) \quad : \quad f_i = \sqrt{i + f_{i+1}} \end{aligned}$$

Observe that the difference between (18) and (19) is that the sequence essentially *looks* forward, not back. To solve using a `for` loop, this distinction is a non-concern because the list can be reversed using a built-in such as `rev`, `reversed` or `reverse` in *Python*, ***R*** and *Julia* respectively, which means the same expression can be implemented.

To implement with recursion however, the series needs to be restructured and this can become a little clumsy, see (20):

$$g(k) = \prod_{i=2}^{k} \left(\frac{f_i}{i}\right) \quad : \quad f_i = \sqrt{(k-i) + f_{k-i-1}} \tag{20}$$

Now the function could be performed recursively in *Python* in a similar way as shown in listing 4, but it's also significantly more confusing because the f function now has k as a parameter and this is only made significantly more complicated by the differing implementations of variable scope across common languages used in Mathematics and Data science such as `bash`, ***R***, *Julia*, *Python*.

If however, the `for` loop approach was implemented, as shown in listing 5, the function would not significantly change, because the `reversed()` function can be used to flip the list around.

What this demonstrates is that taking a different approach to simply describing this function can lead to big differences in the complexity involved in solving this problem.

```
from sympy import *
def h(k):
    if k > 2:
        return f(k, k) * f(k, k-1)
    else:
        return 1

def f(k, i):
    if k > i:
        return 1
    if i > 2:
        return sqrt((k-i) + f(k, k - i -1))
    else:
        return 1
```

Listing 4: Using Recursion to Solve (18)

```
from sympy import *
def h(k):
    k = k + 1 # OBOB
    l = [f(i) for i in range(1,k)]
    return prod(l)

def f(k):
    expr = 0
    for i in reversed(range(2, k+2)):
        expr = sqrt(i + expr, evaluate=False)
    return expr/(k+1)
```

Listing 5: Using Iteration to Solve (18)

2.3 Fibonacci Sequence — Ryan:James

2.3.1 Introduction — Ryan

The *Fibonacci Sequence* and *Golden Ratio* share a deep connection[8] and occur in patterns observed in nature very frequently (see [30, 3, 24, 25, 20, 28]), an example of such an occurence is discussed in section 2.3.4 .

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.

2.3.2 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.

[8] See section

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 6, however using this function will reveal that it is extraordinarily slow, as shown in listing 7, this is because the results of the function are not cached and every time the function is called every value is recalculated[9], 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 [14], 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 8.

```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 6: Defining the *Fibonacci Sequence* (21) using Recursion

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

## 2.245seconds
```

Listing 7: Using the function from listing 6 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)
```

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

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

Listing 8: Caching the results of the function previously defined 7

```
 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 9. Using a compiled language such as *Julia* however would be thousands of times faster still, as demonstrated in listing 10.

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

```
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 10: 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 .

2.3.3 Exponential Generating Functions

Motivation RYAN

Consider the *Fibonacci Sequence* from (21):

$$\begin{aligned} a_n &= a_{n-1} + a_{n-2} \\ \iff a_{n+2} &= a_{n+1} + a_n \end{aligned} \tag{22}$$

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.

Consider using the following generating function, (the derivative of the generating function as in (24) and (25) is provided in section 2.3.3)

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \cdot \frac{x^n}{n!} \right] \tag{23}$$

$$\implies f'(x) = \sum_{n=0}^{\infty} \left[a_{n+1} \cdot \frac{x^n}{n!} \right] \tag{24}$$

$$\implies f''(x) = \sum_{n=0}^{\infty} \left[a_{n+2} \cdot \frac{x^n}{n!} \right] \tag{25}$$

So the recursive relation from (22) 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 (23):

$$f''(x) = f'(x) + f(x) \tag{26}$$

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

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

By equating this to the power series:

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

Now given that:

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

We can conclude that:

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

By applying the initial conditions:

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

And so finally we have the solution to the *Fibonacci Sequence* 22:

$$\begin{aligned} a_n &= \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] \\ &= \frac{\varphi^n - \psi^n}{\sqrt{5}} \\ &= \frac{\varphi^n - \psi^n}{\varphi - \psi} \end{aligned} \tag{27}$$

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 to the backward: [21]

$$f(x) = \sum_{n=0}^{\infty}\left[a_n\frac{x^n}{n!}\right] \tag{28}$$

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

$$\implies f'(x) = \sum_{n=0}^{\infty}\left[\frac{x^n}{n!}a_{n+1}\right] \tag{29}$$

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

$$f^{(k)}(x) = \sum_{n=0}^{\infty}\frac{a_{n+k}\cdot x^n}{n!} \quad \text{for } k \geq 0 \tag{30}$$

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 (23) shifts the sequence across, then every derivative thereafter does so as well, because the first derivative has been shown to express this property (29), all derivates will.

Homogeneous Proof

RYAN:JAMES

An equation of the form:

$$\sum_{i=0}^{n}\left[c_i \cdot f^{(i)}(x)\right] = 0 \tag{31}$$

is said to be a homogenous linear ODE: [36, 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, [36, 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 (32)). 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 [22]
- $\exists i, j \in \mathbb{Z}^+ \cap [0, k]$
  - These is often referred to as repeated roots [22, 37] with a multiplicity corresponding to the number of repetitions of that root [26, §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 2.3.3 .
2. Proof Consider the linear recurrence relation (32):

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

This implies:

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

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

$$\sum_{i=0}^{k}c_i\sum_{n=0}^{\infty}\frac{x^n}{n!}a_n = 0 \tag{34}$$

By implementing the exponential generating function as shown in (23), this provides:

$$\sum_{i=0}^{k}\left[c_i f^{(i)}(x)\right] \tag{35}$$

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 [36, Ch. 4] :

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

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):

$$
\begin{aligned}
\sum_{i=0}^{k}\left[k_i e^{m_i x}\right] &= \sum_{i=0}^{k}\left[k_i \sum_{n=0}^{\infty}\frac{(m_i x)^n}{n!}\right] \\
&= \sum_{i=0}^{k}\sum_{n=0}^{\infty} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty}\sum_{i=0}^{k} k_i m_i^n \frac{x^n}{n!} \\
&= \sum_{n=0}^{\infty}\left[\frac{x^n}{n!}\sum_{i=0}^{k}\left[k_i m_i^n\right]\right], \quad \exists k_i \in \mathbb{C},\ \ \forall i \in \mathbb{Z}^+ \cap [1,k]
\end{aligned} \tag{36}
$$

Recall the definition of the generating function from (23), by relating this to (36):

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

$\square$

This can be verified by the fibonacci sequence as shown in section 2.3.3 , 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:

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

Repeated Roots of Characteristic Equation **Ryan**

1. Example Consider the following recurrence relation:

$$
\begin{aligned}
a_n - 10a_{n+1} + 25a_{n+2} = 0 \\
\implies \sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right] - 10\sum_{n=0}^{\infty}\left[\frac{x^n}{n!}+\right] + 25\sum_{n=0}^{\infty}\left[a_{n+2}\frac{x^n}{n!}\right] = 0
\end{aligned} \tag{37}
$$

By applying the definition of the exponential generating function at (23) :

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

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

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

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 [37, §4.3].

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

(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 (TODO Prove this in section):

$$
x^k e^x = \sum_{n=0}^{\infty} \left[\frac{x^n}{(n-k)!} \right], \quad \exists k \in \mathbb{Z}^+ \tag{40}
$$

by applying identity (40) to equation (39)

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

Given the defenition of the exponential generating function from (23)

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

$\square$

2. 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:

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

By substituting for the value of the generating function (from (23)):

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

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

$$\begin{aligned}
&\sum_{i=0}^{k}\left[c_i m^i\right] = 0 \wedge m = B, \quad \exists! B \in \mathbb{C} \\
\implies & f(x) = \sum_{i=0}^{k}\left[x^i A_i e^{mx}\right], \quad \exists A \in \mathbb{C}^{+}, \ \forall i \in [1,k] \cap \mathbb{N} && (42) \\
& && (43)
\end{aligned}$$

If we assume that (see section 1):

$$k \in \mathbb{Z} \implies x^k e^x = \sum_{n=0}^{\infty}\left[\frac{x^n}{(n-k)!}\right] \tag{44}$$

By applying this to (42) :

$$\begin{aligned}
f(x) &= \sum_{i=0}^{k}\left[A_i \sum_{n=0}^{\infty}\left[\frac{(xm)^n}{(n-i)!}\right]\right] \\
&= \sum_{n=0}^{\infty}\left[\sum_{i=0}^{k}\left[\frac{x^n}{n!}\frac{n!}{(n-i)}A_i m^n\right]\right] && (45) \\
&= \sum_{n=0}^{\infty}\left[\frac{x^n}{n!}\sum_{i=0}^{k}\left[\frac{n!}{(n-i)}A_i m^n\right]\right] && (46)
\end{aligned}$$

Recall the generating function that was used to get 41:

$$\begin{aligned}
& f(x) = \sum_{n=0}^{\infty}\left[a_n \frac{x^n}{n!}\right] \\
\implies & a_n = \sum_{i=0}^{k}\left[A_i \frac{n!}{(n-i)!} m^n\right] \\
& \quad = \sum_{i=0}^{k}\left[m^n A_i \prod_{0}^{k}\left[n-(i-1)\right]\right] && (47)
\end{aligned}$$

$\because \;\; i \leq k$

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

$\square$

General Proof In sections 2.3.3 and 2.3.3 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, 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

1. Power Series Combination JAMES In this section a proof for identity 44 is provided.

   (a) Motivation

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

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

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

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

   We conjecture thatIf we continue this on, we get:

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

(b) Proof by Induction To verify, let's prove this by induction.

i. Base Test $k = 0$

$$LHS = x^0 e^x = e^x$$
$$RHS = \sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$$

Therefore LHS = RHS, so $k = 0$ is true

ii. Bridge Assume $x^k e^x = \sum_{n=k}^{\infty} \frac{x^n}{(n-k)!}$
Using this assumption, prove for the next element $k+1$

We need $x^{k+1} e^x = \sum_{n=k+1}^{\infty} \frac{x^n}{(n-(k+1))!}$

$$\begin{aligned}
\text{LHS} &= x^{k+1} e^x \\
&= x \cdot x^k e^x \\
&= x \cdot \sum_{n=k}^{\infty} \frac{x^n}{(n-k)!} \quad \text{(by assumption)} \\
&= \sum_{n=k}^{\infty} \frac{x^{n+1}}{(n-k)!} \\
&= \sum_{n=k+1}^{\infty} \frac{x^n}{(n-1-k)!} \quad \text{(re-indexing } n\text{)} \\
&= \sum_{n=k+1}^{\infty} \frac{x^n}{(n-(k+1))!} \\
&= RHS
\end{aligned}$$

So by mathematical induction $f(x) = x^k e^x = \sum_{n=k}^{\infty} \frac{x^n}{(n-k)!}$ for $k \geq 0$
Moving on, by applying identity (40) to equation (39)

2.3.4 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 2.3.4 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*.

Fibonacci Sequence in Nature — Ryan

The distribution of sunflower seeds is an example of the *Fibonacci Sequence* occuring in a pattern observed in nature (see Figure 15).

Imagine that the process a sunflower follows when placing seeds is as follows: [10]

1. Place a seed
2. Move some small unit away from the origin
3. Rotate some constant angle θ (or) 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 11,[11] which combined with *ImageMagick* (see e.g. 21), produces output as shown in figure 13 and 14.

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.

---

[10] This process is simply conjecture, other than seeing a very nice example at *MathIsFun.com* [25], 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.

[11] 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.

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 [25]:

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

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 14, 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 2.3.4 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 13 has a ratio of rotation of $\frac{1}{\pi}$, the spirals look similar to one direction of the spirals occuring in figure 14, it is not clear if there is any significance to this similarity.

```
 = 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 11: Simulation of the distribution of sunflowers as described in section 2.3.4

Figure 13: Simulated Distribution of Sunflower seeds as described in section 2.3.4 and listing 11

Figure 14: Optimisation of simulated distribution of Sunflower seeds occurs for $\theta = 2\varphi\pi$ as described in section 2.3.4 and listing 11

Figure 15: Distribution of the seeds of a sunflower (see [7] licenced under CC)

2.4 Persian Recursion — Ryan

This section contains an example of how a simple process can lead to the development of complex patterns when exposed to feedback and iteration.

The *Persian Recursion* is a simple procedure developed by Anne Burns in the 90s [9] that produces fantastic patterns when provided with a relatively simple function.

The procedure is begins with an empty or zero square matrix with sides $2^n + 1,\ \exists n \in \mathbb{Z}^+$ and some value given to the edges:

1. Decide on some four variable function with a finite domain and range of size m, for the example shown at listing 12 and in figure 17 the function $f(w, x, y, z) = (w + x + y + z) \mod m$ was chosen.
2. Assign this value to the centre row and centre column of the matrix
3. Repeat this for each newly enclosed subsmatrix.

This is illustrated in figure 16.

This can be implemented computationally by defining a function that:

- Takes the index of four corners enclosing a square sub-matrix of some matrix as input,
- proceeds only if that square is some positive real value.
- colours the centre column and row corresponding to a function of those four values
- then calls itself on the corners of the four new sub-matrices enclosed by the coloured row and column

Figure 16: Diagram of the Persian Recursion, implemented with *Python* in listing 12

This is demonstrated in listing 12 with python and produces the output shown in figures 17, various interesting examples are provided in the appendix at section 2.8.1 .

By mapping the values to colours, patterns emerge, this emergence of complex patterns from simple rules is a well known and general phenomena that occurs in nature [11, 19].

Many patterns that occur in nature can be explained by relatively simple rules that are exposed to feedback and iteration [27, p. 16], this is a central theme of Alan Turing's *The Chemical Basis For Morphogenesis* [35] which we hope to look in the course of this research.

```
%matplotlib inline
# m is colours
# n is number of folds
# Z is number for border
# cx is a function to transform the variables
def main(m, n, z, cx):
    import numpy as np
    import matplotlib.pyplot as plt

    # Make the Empty Matrix
    mat = np.empty([2**n+1, 2**n+1])
```

```python
    main.mat = mat

    # Fill the Borders
    mat[:,0] = mat[:,-1] = mat[0,:] = mat[-1,:] = z

    # Colour the Grid
    colorgrid(0, mat.shape[0]-1, 0, mat.shape[0]-1, m)

    # Plot the Matrix
    plt.matshow(mat)

# Define Helper Functions
def colorgrid(l, r, t, b, m):
    # print(l, r, t, b)
    if (l < r -1):
        ## define the centre column and row
        mc = int((l+r)/2); mr = int((t+b)/2)

        ## Assign the colour
        main.mat[(t+1):b,mc] = cx(l, r, t, b, m)
        main.mat[mr,(l+1):r] = cx(l, r, t, b, m)

        ## Now Recall this function on the four new squares
               #l r   t   b
        colorgrid(l, mc, t, mr, m) # NW
        colorgrid(mc, r, t, mr, m) # NE
        colorgrid(l, mc, mr, b, m) # SW
        colorgrid(mc, r, mr, b, m) # SE

def cx(l, r, t, b, m):
    new_col = (main.mat[t,l] + main.mat[t,r] + main.mat[b,l] + main.mat[b,r]) % m
    return new_col.astype(int)

main(5,6, 1, cx)
```

Listing 12: Implementation of the persian recursion scheme in *Python*

Figure 17: Output produced by listing 12 with 6 folds

2.5 Julia Sets — Ryan

2.5.1 Introduction

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

2.5.2 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.

2.5.3 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[12] and define the operations as described in listing 13.[13]

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 [14], this is implemented in listing 14 and the corresponding output shown in 18.

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

[12]See figure for the obligatory *XKCD* Comic
[13]This technique was adapted from Chapter 7 of *Math adventures with Python* [13]
[14]these cascading values are much like brightness in Astronomy

```
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

```
%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)
```

```python
        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

Figure 18: 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 19 and 20.

Figure 19: Circle of Convergence for $f_0 : z \to z^2 - 1$

Figure 20: Circle of Convergence for $f_{\frac{1}{4}+\frac{i}{2}} : z \to z^2 + \frac{1}{4} + \frac{i}{2}$

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[15].

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. [16]

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 21 and a corresponding animation is available online[17]].

```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
```

[15] This approach was inspired by an animation on the *Julia Set* Wikipedia article [18]

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

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

```julia
        # 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*)
        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

```
# 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.

Figure 21: Various fracals corresponding to $f_{0.8e^{\pi i \tau}}$

2.6 MandelBrot — Ryan

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

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

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

$$z_{n+1} = z_n^2 + c \tag{48}$$
$$z_0 = c \tag{49}$$

This strategy is implemented in listing and produces the output shown in figure 22.

```
%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*

Figure 22: Mandelbrot Set produced in *Python* as shown in listing 2.6

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.[18]

```
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)
```

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

```
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)
    # 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 23: Screenshot of Mandelbrot FITS image produced by listing

2.7 Relevant Sources

To guide research for our topic *Chaos and Fractals* by Pietgen, Jurgens and Saupe [27] will act as a map of the topic broadly, other than the sources referenced already, we anticipate referring to the following textbooks that we access to throughout the project:

- *Integration of Fuzzy Logic and Chaos Theory* [23]
- *Advances in Chaos Theory and Intelligent Control* [1]
- *NonLinear Dynamics and Chaos* [33]
- *The NonLinear Universe* [29]
- *Chaos and Fractals* [27]

- *Turbulent Mirror* [8]
- *Fractal Geometry* [12]
- *Math Adventures with Python* [13]
- *The Topology of Chaos* [15]
- *Chaotic Dynamics* [34]

Ron Knott's website appears also to have a lot of material related to patterns, the *Fibonacci Sequence* and the *Golden Ratio*, we intend to have a good look through that material as well. [28]

2.8 Appendix

```python
from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
init_printing()
init_printing(use_latex='mathjax', latex_mode='equation')


import pyperclip
def lx(expr):
    pyperclip.copy(latex(expr))
    print(expr)

import numpy as np
import matplotlib as plt

import time

def timeit(k):
    start = time.time()
    k
    print(str(round(time.time() - start, 9)) + "seconds")
```

Listing 19: Preamble for *Python* Environment

2.8.1 Persian Recursian Examples

```python
%config InlineBackend.figure_format = 'svg'
main(5, 9, 1, cx)
```

Listing 20: Enhance listing 12 to create 9 folds

```
%config InlineBackend.figure_format = 'svg'
def cx(l, r, t, b, m):
    new_col = (main.mat[t,l] + main.mat[t,r] + main.mat[b,l] + main.mat[b,r]-7)
        % m
    return new_col.astype(int)
main(8, 8, 1, cx)
```

Listing 21: Modify the Function to use $f(w, x, y, z) = (w + x + y + z - 7) \mod 8$

Figure 24: Output produced by listing 21 using $f(w, x, y, z) = (w + x + y + z - 7) \mod 8$

```
%config InlineBackend.figure_format = 'svg'
import numpy as np
def cx(l, r, t, b, m):
    new_col = (main.mat[t,l] + main.mat[t,r]*m + main.mat[b,l]*(m) +
        main.mat[b,r]*(m))**1 % m + 1
    return new_col.astype(int)
main(8, 8, 1, cx)
```

Listing 22: Modify the function to use $f(w, x, y, z) = (w + 8x + 8y + 8z) \mod 8 + 1$

Figure 25: Output produced by listing 22 using $f(w, x, y, z) = (w + 8x + 8y + 8z) \mod 8 + 1$

2.8.2 Figures

Figure 26: XKCD 2028: Complex Numbers

2.8.3 Why Julia

The reason we resolved to make time to investigate *Julia* is because we see it as a very important tool for mathematics in the future, in particular because:

- It is a new modern language, designed primarily with mathematics in mind
  - First class support for UTF8 symbols
  - Full Support to call ***R*** and *Python*.
- Performance wise it is best in class and only rivalled by compiled languages such as *Fortran Rust* and ***C***
  - *Just in Time Compiling* allows for a very useable *REPL* making Julia significantly more appealing than compiled languages
  - The syntax of Julia is very similar to *Python* and ***R***
- The `DifferentialEquations.jl` library is one of the best performing libraries available.

Other Packages Other packages that are on our radar for want of investigation are listed below, in practice it is unlikely that time will permit us to investigate many packages or libraries

- Programming Languages and CAS
  - Julia
    - SymEngine.jl
    - Symata.jl
    - SymPy.jl
  - Maxima
    - Being the oldest there is probably a lot too learn
  - Julia
  - Reduce
  - Xcas/Gias
  - Python
    - Numpy
    - Sympy
- Visualisation
  - Makie
  - Plotly
  - GNUPlot

References

[1] Ahmad Taher Azar and Sundarapandian Vaidyanathan, eds. *Advances in Chaos Theory and Intelligent Control*. 1st ed. 2016. Studies in Fuzziness and Soft Computing 337. Cham: Springer International Publishing : Imprint: Springer, 2016. 1 p. ISBN: 978-3-319-30340-6. DOI: 10.1007/978-3-319-30340-6 (cit. on p. 41).

[2] A. C Benander, B. A Benander, and Janche Sang. "An Empirical Analysis of Debugging Performance Differences between Iterative and Recursive Constructs". In: *Journal of Systems and Software* 54.1 (Sept. 30, 2000), pp. 17–28. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(00)00023-6. URL: http://www.sciencedirect.com/science/article/pii/S0164121200000236 (visited on 08/24/2020) (cit. on p. 17).

[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. 18).

[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. 15, 31).

[5] Corrado Böhm. "Reducing Recursion to Iteration by Algebraic Extension: Extended Abstract". In: *ESOP 86*. Ed. by Bernard Robinet and Reinhard Wilhelm. Red. by G. Goos et al. Vol. 213. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 111–118. ISBN: 978-3-540-16442-5 978-3-540-39782-3. DOI: 10.1007/3-540-16442-1_8. URL: http://link.springer.com/10.1007/3-540-16442-1_8 (visited on 08/24/2020) (cit. on p. 17).

[6] Corrado Böhm. "Reducing Recursion to Iteration by Means of Pairs and N-Tuples". In: *Foundations of Logic and Functional Programming*. Ed. by Mauro Boscarol, Luigia Carlucci Aiello, and Giorgio Levi. Red. by G. Goos et al. Vol. 306. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 58–66. ISBN: 978-3-540-19129-2 978-3-540-39126-5. DOI: 10.1007/3-540-19129-1_3. URL: http://link.springer.com/10.1007/3-540-19129-1_3 (visited on 08/24/2020) (cit. on p. 17).

[7] Simon Brass. *CC Search*. 2006, September 5. URL: https://search.creativecommons.org/ (visited on 08/28/2020) (cit. on p. 33).

[8] John Briggs and F. David Peat. *Turbulent Mirror: An Illustrated Guide to Chaos Theory and the Science of Wholeness*. 1st ed. New York: Harper & Row, 1989. 222 pp. ISBN: 978-0-06-016061-6 (cit. on p. 42).

[9] Anne M. Burns. ""Persian" Recursion". In: *Mathematics Magazine* 70.3 (1997), pp. 196–199. ISSN: 0025-570X. DOI: 10.2307/2691259. JSTOR: 2691259 (cit. on p. 33).

[10] 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. 4).

[11] *Emergence How Stupid Things Become Smart Together*. Nov. 16, 2017. URL: https://www.youtube.com/watch?v=16W7c0mb-rE (visited on 08/25/2020) (cit. on p. 33).

[12] 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, 42).

[13] 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 pp. 35, 42).

[14] *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. 19).

[15] 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 pp. 2, 42).

[16] Roozbeh Hazrat. *Mathematicaő: 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 pp. 16, 19).

[17] *Iteration vs. Recursion - CS 61A Wiki*. Dec. 19, 2016. URL: https://www.ocf.berkeley.edu/~shidi/cs61a/wiki/Iteration_vs._recursion (visited on 08/24/2020) (cit. on p. 17).

[18] *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. 37).

[19] Sophia Kivelson and Steven A. Kivelson. "Defining Emergence in Physics". In: *npj Quantum Materials* 1.1 (1 Nov. 25, 2016), pp. 1–2. ISSN: 2397-4648. DOI: 10.1038/npjquantmats.2016.24. URL: https://www.nature.com/articles/npjquantmats201624 (visited on 08/25/2020) (cit. on p. 33).

[20] 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. 18).

[21] 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. 23).

[22] 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. 25).

[23] 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. 41).

[24] Nikoletta Minarova. "The Fibonacci Sequence: Natures 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. 18).

[25] *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. 18, 31, 32).

[26] 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. 25).

[27] 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, 7, 33, 41).

[28] 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 pp. 18, 42).

[29] Alwyn Scott. *The Nonlinear Universe: Chaos, Emergence, Life*. 1st ed. The Frontiers Collection. Berlin ; New York: Springer, 2007. 364 pp. ISBN: 978-3-540-34152-9 (cit. on p. 41).

[30] Shelly Allen. *Fibonacci in Nature*. URL: https://fibonacci.com/nature-golden-ratio/ (visited on 08/28/2020) (cit. on p. 18).

[31] A.P. Sinha and I. Vessey. "Cognitive Fit: An Empirical Study of Recursion and Iteration". In: *IEEE Transactions on Software Engineering* 18.5 (May 1992). Choose the Right Language for the Right Job, pp. 368–379. ISSN: 00985589. DOI: 10.1109/32.135770. URL: http://ieeexplore.ieee.org/document/135770/ (visited on 08/24/2020) (cit. on p. 17).

[32] S Smolarski. *Math 60 – Notes A3: Recursion vs. Iteration*. Feb. 9, 2000. URL: http://math.scu.edu/~dsmolars/ma60/notesa3.html (visited on 08/24/2020) (cit. on p. 17).

[33] 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. 41).

[34] 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 p. 42).

[35] Alan Turing. "The Chemical Basis of Morphogenesis". In: *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* 237.641 (Aug. 14, 1952), pp. 37–72. ISSN: 2054-0280. DOI: 10.1098/rstb.1952.0012. URL: https://royalsocietypublishing.org/doi/10.1098/rstb.1952.0012 (visited on 08/25/2020) (cit. on p. 33).

[36] Dennis G Zill and Michael R Cullen. *Differential Equations*. 7th ed. Brooks/Cole, 2009 (cit. on pp. 24, 25).

[37] 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. 25, 27).