Fractals – The Beauty of Mathematics

Ina S. Feder

31st March 2024

To most people, mathematics is a frustrating and abstract subject, which is difficult to understand, with pointless, made up problems. Few would describe mathematics as beautiful in the literal sense. However, there is a part of mathematics where images and pattern emerges that I believe most people would find beautiful and almost artistic, namely in fractals. Fractals were first introduced to me by my grandfather, Jens Feder, who passed away in 2019.

1 What is a fractal?

A fractal is a never-ending geometric pattern, it is infinitely complex and self-similar across different scales, with detailed structure at arbitrary small scales. In essence, it is a pattern that repeats itself, regardless of how zoomed in, it appears similar to a whole image [2, 3].

2 The Mandelbrot set

The Mandelbrot set is an example of a fractal. The Mandelbrot set is generated by iterating a simple, discrete function, defined in Equation (1), on points of a complex plane.

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

Where c is a complex number and the initial value of z, is always zero, that is $z_0 = 0$.[1]

[1] To find the Mandelbrot set, z_0 must always equal zero, so $z_1 = c$.

3 Displaying the Mandelbrot set

The Mandelbrot set is defined by the iterative equation: Equation (1), that takes a varying number of iterations to stay bounded or unbounded. The Mandelbrot set is defined by all those points in the complex plane where z_n stay bounded for all values of n. However, by colouring the points in the complex plane dependent on *how many iterations* it takes for the sequence to be deemed to go to infinity (i.e. those points *not* in the Mandelbrot set) produces an infinitely repeating, colourful, beautiful and what seems as infinite complex patterns at all magnifications (see Figure 2 for a peek preview).

Further, using the same iterative equation, but changing the initial variable z_0 across the complex plane (and keep c a constant), the same equation produces a whole new set of fractals, known as the Julia set.

In order to visualize Equation (1), I need to explain the fundamentals of complex numbers.

4 What are complex numbers?

Not all numbers have a square root. If you square a real number, the result will always be a positive number, since two negatives, or two positives, multiplied together always gives a positive number. However, what if you try to square root a negative number [5]?

For instance, how would you find the square root of -1? Since any square number must be positive, the square root of -1 does not exist as a real number. As a result, mathematicians defined i to be the square root of -1 and called any multiple of i to be an imaginary number.[2] Surprisingly, such imaginary numbers work well with algebra, for instance:

$$\begin{aligned} \sqrt{-1} &= i \\ i^2 &= -1 \end{aligned}$$

$$\begin{aligned} \sqrt{-49} &= 7i \\ \sqrt{-8} &= 2i\sqrt{2} \\ (4i)^2 &= -16 \end{aligned}$$

[2]The definition i is actually the 9th character of the Greek alphabet, iota, and in tradition of mathematical variables represented by i rather than i.

A "complex" number is any imaginary number with a real number, in the form $x + y \cdot i$. While real numbers can be represented on a number line, complex numbers are represented in a plane [6].

5 How are complex numbers represented graphically?

All numbers such as reals, integers, rational and irrational numbers can be plotted on a number line. However, a complex number consist of a real number and an multiple of the imaginary number i, we cannot plot both on a single number line. Instead of a number line, mathematicians utilise a plane, the *complex plane*, to represent complex numbers where the horizontal x-axis is the "real axis", and the vertical y-axis is the "imaginary" axis as shown in Figure 1. This is similar to a vector in a plane, and normal vector algebra would apply. For instance, the magnitude (or size, denoted $|z|$) of a complex number is defined by Equation (2), just as it is for a vector in the x-y plane [4].

$$|z| = \sqrt{x^2 + y^2} \tag{2}$$

In Figure 1, the complex number $z = 2 + 3i$ exists in the complex plane, and its magnitude is $\sqrt{2^2 + 3^2} = \sqrt{13}$. It is in the complex plane that the Mandelbrot set exists as it gives interesting results when c in Equation (1) is a complex number.

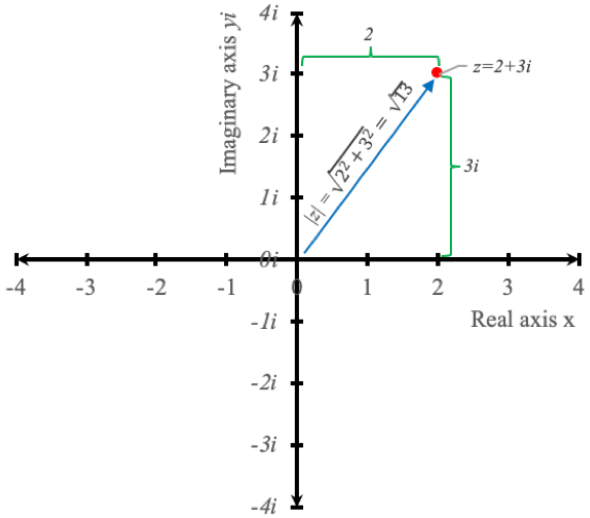

Figure 1: The real x-axis and the imaginary y-axis forms the complex plane.

6 How does the iteration equation work?

The Equation (1) defines the Mandelbrot set. We can use this equation to present how the results vary over iterations. The initial starting position is always $z_0 = 0$ in the definition of the Mandelbrot set. Below we illustrate the general iteration to arrive at z_1 to z_n.

$$\begin{aligned} z_1 &= z_0^2 + c \\ z_2 &= z_1^2 + c \\ z_3 &= z_2^2 + c \\ z_4 &= z_3^2 + c \\ &\vdots \\ z_n &= z_{n-1}^2 + c \end{aligned}$$

If c were not a complex number, but say the real number, $c = 1$, the values of the first 5 iterations would be:

$$
\begin{aligned}
z_1 = \quad & 0^2 + 1 \quad && = 1 \\
z_2 = \quad & 1^2 + 1 \quad && = 2 \\
z_3 = \quad & 2^2 + 1 \quad && = 5 \\
z_4 = \quad & 5^2 + 1 \quad && = 26 \\
z_5 = \quad & 26^2 + 1 \quad && = 677
\end{aligned}
$$

With this starting value of $c = 1$, as n tends to infinity, so does the result of z_n, thus 1 is not in the Mandelbrot set. An obvious result is that z_n is always 0, and thus in the Mandelbrot set, if we set $c = 0$:

$$
\begin{aligned}
z_1 = \quad & 0 \quad && = 0 \\
z_2 = \quad & 0^2 + 0 \quad && = 0 \\
z_3 = \quad & 0^3 + 0 \quad && = 0
\end{aligned}
$$

However, a more interesting result happens if we set $c = -1$:

$$
\begin{aligned}
z_1 = \quad & -1 \quad && \\
z_2 = \quad & (-1)^2 - 1 \quad && = 0 \\
z_3 = \quad & 0^1 - 1 \quad && = 1 \\
z_4 = \quad & 1^2 - 1 \quad && = 0
\end{aligned}
$$

With this initial value of c, the pattern loops and repeats itself over and over, creating a cycle which produces the results 0 and -1, this is in the Mandelbrot set as the value of z_n does not tend to infinity, but stays bounded.

However, we start getting very interesting results by setting $c = -1.8$:

$$
\begin{aligned}
z_1 = \quad & -1.8 \quad && \\
z_2 = \quad & (-1.8)^2 - 1.8 \quad && = 1.44 \\
z_3 = \quad & (1.44)^2 - 1.8 \quad && = 0.2736 \\
z_4 = \quad & (0.2736)^2 - 1.8 \quad && = -1.72514304 \\
z_5 = \quad & (-1.72514304)^2 - 1.8 \quad && = 1.176118508
\end{aligned}
$$

This shows that as n increases, the results of z_n display "chaotic"[3] behaviour, that is, the z_n value cannot easily be predicted without computing all the iterations to n and calculating z_n.

That is, using the iteration of Equation (1), the series of numbers either tend to a fixed point (which could be infinity, zero or a constant), go in a cycle, or have chaotic behaviour. The Python code 1 in the Appendix produces the first z_n values for a given c and n.

Remember that in Equation (1) c can be a complex number. That is, c can be the sum of a real number, x, and a multiple y of i, i.e. of the form $c = x + y \cdot i$. As we have explored some examples of the iterations of Equation (1) on the real axis, let's explore some results when c is on the imaginary axis, that is, where $x = 0$ and $y \neq 0$. For instance, $c = i$, how does the Mandelbrot Equation (1) evolve?

$$\begin{aligned}
z_1 &= i \\
z_2 &= i^2 + i & &= -1 + i \\
z_3 &= (-1+i)^2 + i & &= -i \\
z_4 &= (-i)^2 + i & &= -1 + i \\
z_5 &= (-1+i)^2 + i & &= -i
\end{aligned}$$

This, as in the example above using $c = -1$, which goes in an endless loop of cycles of 2 numbers, thus i is in the Mandelbrot set. If $c = -i$:

$$\begin{aligned}
z_1 &= -i \\
z_2 &= (-i)^2 - i & &= -1 - i \\
z_3 &= (-1-i)^2 - i & &= i \\
z_4 &= i^2 - 1 & &= -1 - i \\
z_5 &= (-1-i)^2 - i & &= i
\end{aligned}$$

This pattern also goes in pattern cycles of 2 and z_n does not escape to infinity, so $c = -i$ is in the Mandelbrot set. For instance, if $c = 2i$:

$$\begin{aligned}
z_1 &= 2i \\
z_2 &= (2i)^2 + 2i & &= -4 + 2i \\
z_3 &= (-4+2i)^2 + 2i & &= 12 - 14i \\
z_4 &= (12-14i)^2 + 2i & &= -52 - 334i
\end{aligned}$$

[3]Chaotic behaviour is when a sequence produces apparently random and unpredictable results, in a system of deterministic equations.

In this pattern, the value of z_n will continue to increase as n increases, and thus $c = 2i$ is not in the Mandelbrot set. The code 2 in the Appendix is python code for testing different multiples of i, and the resulting z_n. [1]

7 The Mandelbrot set, visualised

In order to create beautiful visualisation of the Mandelbrot set and its surroundings, a point c on the complex plane is chosen (and $z_0 = 0$) and Equation (1) is iterated a finite number of times (maximum iterations). If during the iteration the magnitude of z_n (that is, $|z_n|$ of Equation (2)) at any point exceeds the maximum number of iteration provided, z_n is deemed to tend to infinity and the point c is given a colour dependent on the number of iterations required to deem z_n as tending to infinity. If this condition is not met, z_n is deemed to be bounded and part of the Mandelbrot set and usually coloured black.

For example, iterating Equation (1) using $c = -1.9 + i$, and allowing for a maximum of 15 iterations, that is, $n_{max} = 15$, we find that it only takes 3 iterations before $|z_3| = 21.1 > 15$, thus we would give the point $(-1.9, i)$ on the complex plane a colour (say blue) signifying it only took 3 iterations before this value of c for Equation (1) tended to infinity. We then repeat the same process for different values of c. Note that we *deem* that when $|z_n|$ is greater than the maximum iterations provided, $|z_n|$ would go to infinity, and if this is not the case, we *deem* that $|z_n|$ is bounded. This is done as it would be too time consuming to calculate $|z_\infty|$ to determine if it is infinite or if it is bounded.[4] Further, different details can emerge dependent on the allowed maximum number of iterations.

For values of c where z_n displays chaotic behvaiour, the number of iterations before $|z_n|$ can be deemed to infinity or remain bounded can seem random. One might assume that the colour result would be "random", but if you look at the images produced, it cannot simply be a random series of iterations for such an intricate pattern to occur.

To summarise, in order to visualise the Mandelbrot set and its surroundings, you iterate over all values of c of the Mandelbrot set and its suroundings and colour code the coordinate c depending on the number of iterations it took to deem z_n to be bounded (black) or deemed to be unbounded (tends to infinity).

[4]Although for some points it can be proven that $|z_\infty|$ is bounded or not using algebra, many points cannot due to the chaotic behaviour for Equation (1) of many interesting values for c.

8 Implementation of the Mandelbrot set

In the Appendix, I wrote a small Python code 3 that can be used to generate an image of the Mandelbrot set. The function `draw_mandelbrot_set` draws the Mandelbrot set, given a set of input parameters. The first two arguments of this function define the real axis boundary, that is, `x_min, x_max` and the imaginary boundaries, `y_min, y_max`. The size (number of points) on the real axis is defined by the 5th argument, `width`. The resolution of the resulting image can be set by the `max_iterations` that sets the maximum n iterations to compute before determining that the point c is in the Mandelbrot set as $|z_n| <$`max_iterations` or deemed to go to infinity as $|z_n| >$`max_iterations`.

The Mandelbrot set exists on the complex plane bounded by approximately -2.01 and 0.7 ($x \in [-2.01, 0.7]$) on the real axis and -1.14i and 1.14i ($y \in [-1.14, 1.14]$) on the imaginary axis. Thus, to generate the Mandelbrot set with those boundaries and a width of 5,000 points (pixels) and maximum iterations of 90, the following call using code 3 is used:

$$\texttt{draw\_mandelbrot\_set}(-2.01, .7, -1.14, 1.14, 5000, 90)$$

Running the code will draw the Mandelbrot set and is shown in Figure 2. By using different input values and zooming into (making the boundaries smaller) interesting parts on the boundary of the Mandelbrot set, beautiful images and fascinating patterns can be found.

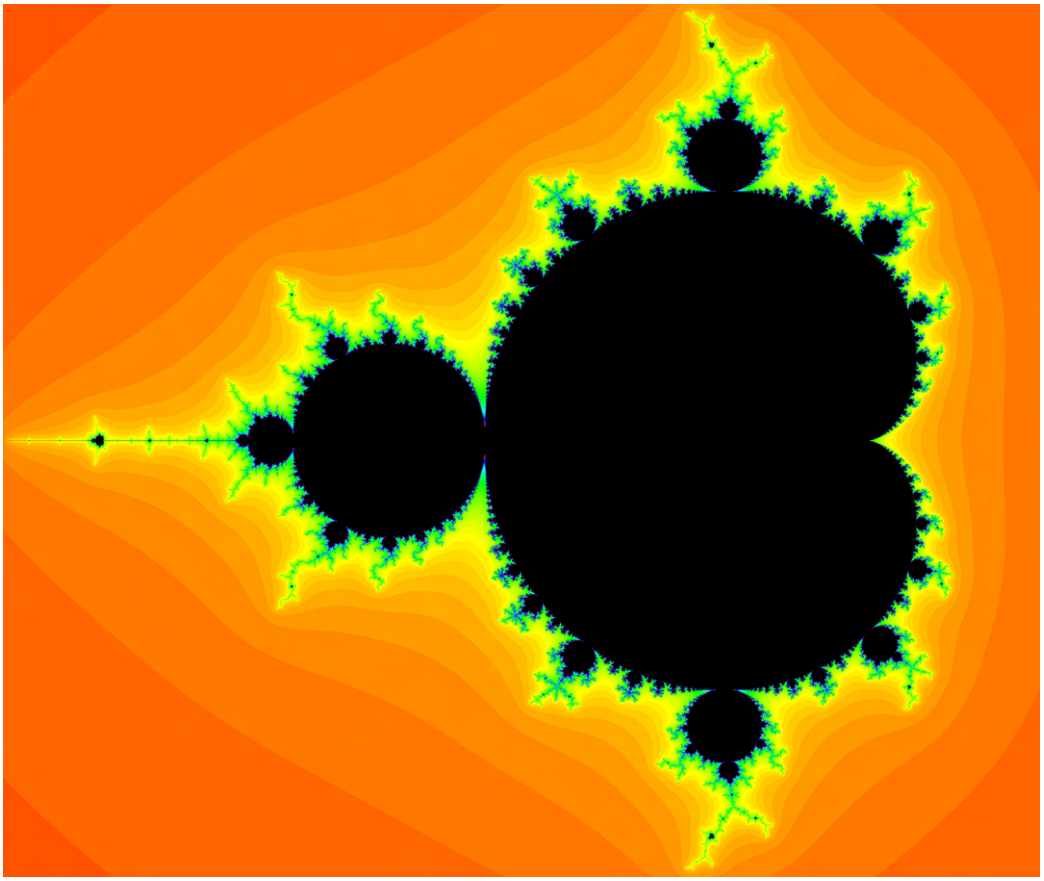

Figure 2: The Mandelbrot Set. $x \in [-2.01, 0.7]$, $y \in [-1.14, 1.14]$, width $= 5000$, max iterations $= 90$.

9 How does the colouring work?

The code 3 uses an "HSV" colouring system to colour the Mandelbrot set. HSV stands for Hue, Saturation, Value. It is traversed linearly through this, and the three linear paths it follows are hue of colour 0 to colour 1, and from saturation 0 to saturation 1. So if you look in the code on line 9,

```
color = 255 * array(colorsys.hsv-to-rgb(n/255.0, 1.0, 0.5))
```

if you edit the first number, 225, you change the hue, and if you edit the second number, 1.0, you change the saturation, and if you edit the last one, 0.5, you change the colour.

10 A journey exploring the Mandelbrot set

An interesting point to use as the centre for magnifiying the Mandelbrot set is centered at $x = -0.7436438891276436$, $y = 0.13182590455455057$. To start the journey through the Mandelbrot set, I start with a distance from the centre in the x (real) and y (imaginary) axis of $r = 8.1041015625$. I then keep the width at 1,500 pixels and then start reducing the distance r by a factor of 10 repeatedly, that is, we do a 100 times magnification each time of the image. The code to create this journey is proviced in code 4.

The Figures 3, 4 and 5 follows this journey of magnifying the original image by a factor all the way to 10^{24}. Along this journey are some mesmerising images and patterns. My favourite one is in the bottom left of Figure 4 where we can see a whole new Mandelbrot set, thus illustrating the self-similar structure of this fractal. Note that in the two first images I set `max_iterations` to 80, while in the others I used 2000. When using 80 for the higher magnifications, one would often create too much black as the code incorrectly assumes the point is in the Mandelbrot set when it is not.

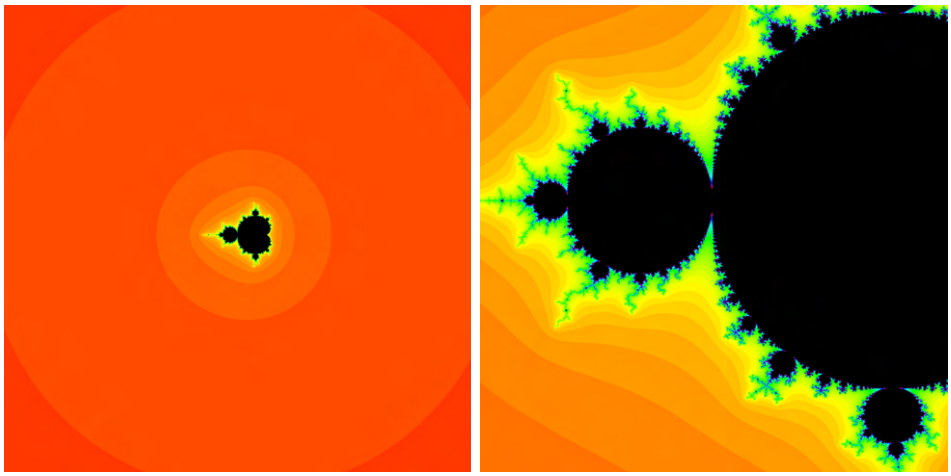

Figure 3: Left: The starting Mandelbrot set. Right: The left image has been magnified by a factor of 10^2 at its centre. The `max_iterations` used here is 80.

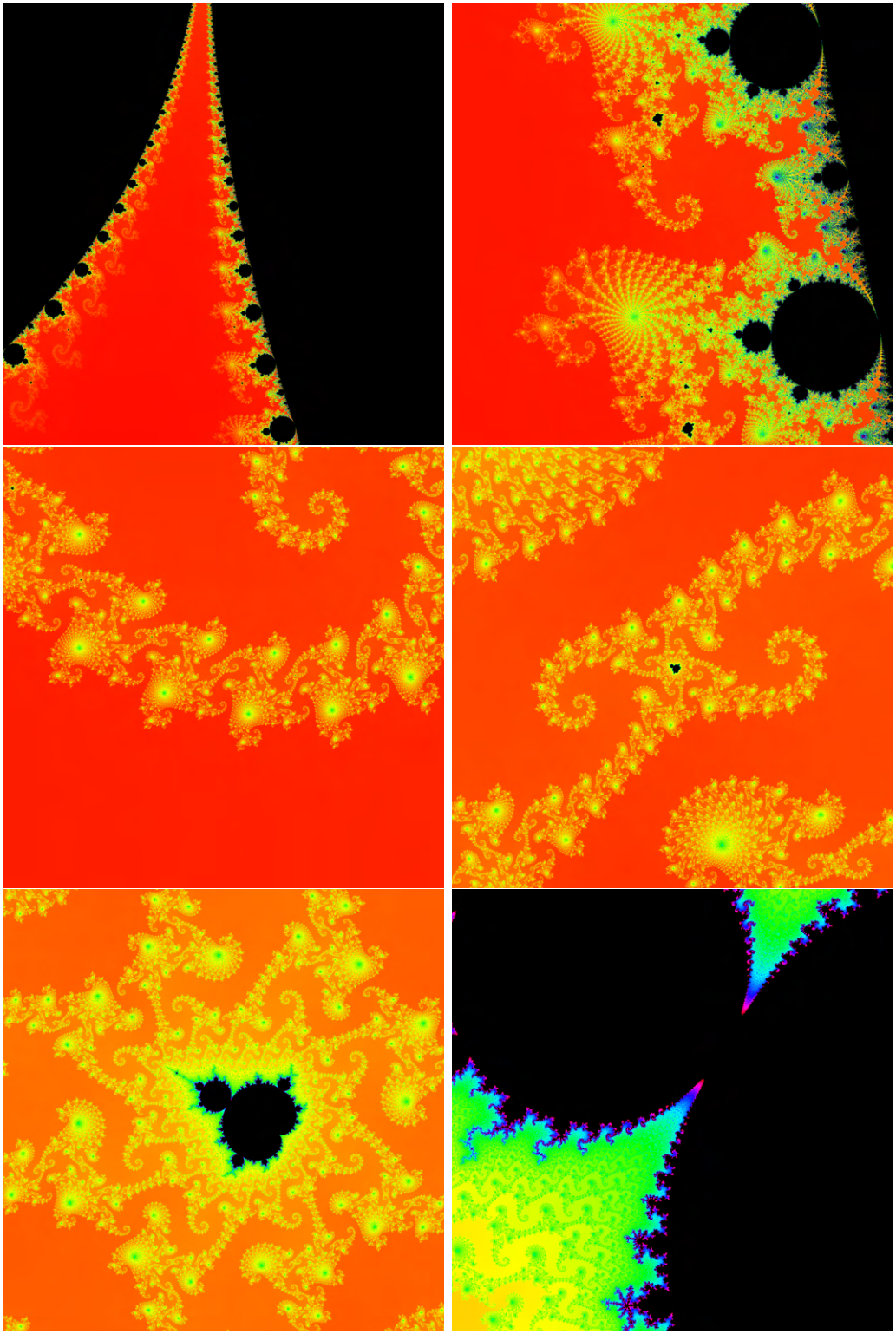

Figure 4: Magnification of 10^4 to 10^{16}, using `max_iterations` = 2000, moving from top left to bottom right.

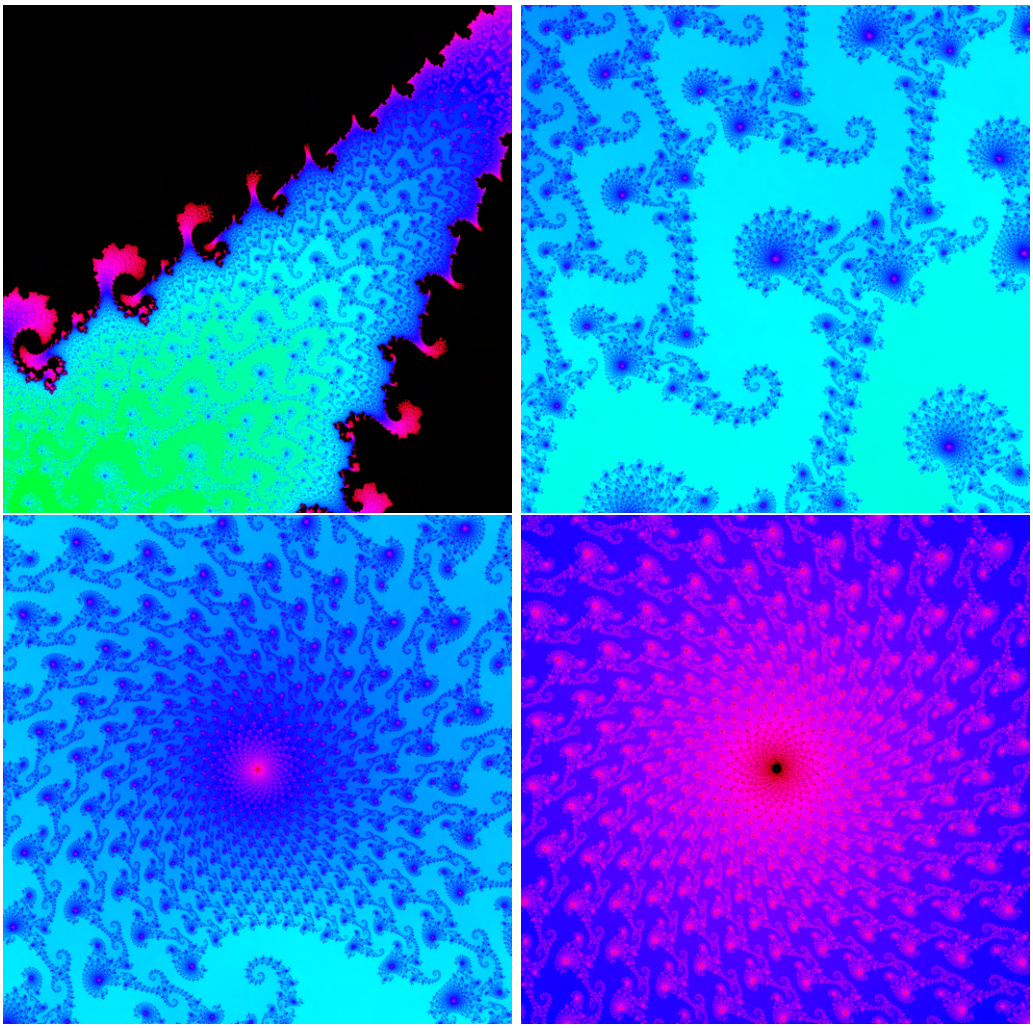

Figure 5: Magnification of 10^{16} to 10^{24}, using `max_iterations` = 2000, moving from top left, to bottom right.

11 Other areas to explore

It is interesting and fun to play with the Mandelbrot code to generate other images, which can be done in multiple ways, such as varying the power of z_{n-1}. For instance, on line 21 of code 3, changing the equation to `z = z ** 4 + c`. This variation is interesting as it creates similar but what seems to be repeated interconnected Mandelbrot sets. For other powers which are less than 1, no image is produced. However, whether the resulting images produced when the power of $z_{n-1} \neq 2$ are fractals would need further investigation. The resulting image of modifying the equation to $z_n = z_n^4 + c$ is shown in Figure 6.

Another way to experiment with code 3 is to input the code 5, which creates the Julia set instead of the Mandelbrot set. A Julia set is remarkably also generated from Equation (1), but the subtle change is that the value of z_0 changes and is mapped across the complex plane, while c remains a constant. This simple small change in the implementation creates a whole new set of beautiful, fascinating fractals! As one changes the value of c, the set expands and creates a different pattern. An example of the Julia set is produced using this code (i.e. when $c = -0.4+0.6i$) and is shown in Figure 7.

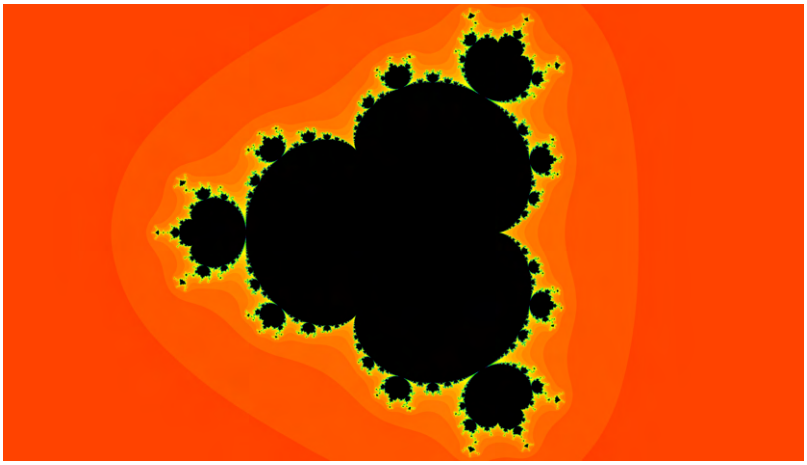

Figure 6: This is the Mandelbrot set when z_{n-1} is raised to the power of 4. It uses the calling function `draw_mandelbrot_set(-2.01,.7,-1.14,1.14,5000,90)`.

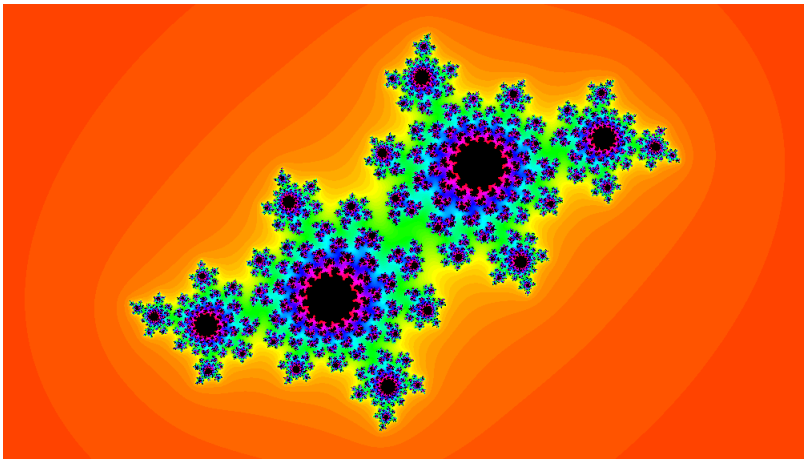

Figure 7: This is the Julia set, called by the function `draw_mandelbrot_set(-2.01,2,-1.14,1.14,1000,90)` modified by code 5.

12 Reflections on the Mandelbrot set

What is it that makes the Mandelbrot set so special? Such a simple code, and only one equation, and infinite complexity - that's so cool!

One way of thinking about the Mandelbrot set is to compare it to the universe and nature. Although these and the Mandelbrot set are not equivalents, it is interesting to compare them if you assume that they are. The

universe has infinite complexity, and is insanely large and full of beautiful patterns from galaxies and stars to crystals and lattice structures. My grandfather[5] once said that you could fit all the fundamental known laws of physics equations on a single piece of paper. This is similar to Mandelbrot's equation – a simple equation, producing all of these complex, beautiful results.

This might also gives us an understanding of why it might be so hard to discover the laws of nature. If you saw the images presented here, could you derive Equation (1)? All this infinite complexity in our surroundings, and likely just a few equations that explain the entire universe!

Another way of looking at it is how the Mandelbrot set puts the universe's size into perspective. If the universe is encapsulated in the Mandelbrot set, where the area of the Mandelbrot set is equal to $4.4 \times 10^{26} m^2$, which is the length of the universe ($4.4 \times 10^{26} m$), you can look at the value of $1m^2$ in the Mandelbrot set to see the difference as shown in Figure 8. Even at this magnification, we see there continues to be intricate patterns that could be dived into even further [6]

[5]Jens Feder was a professor in physics at the University of Oslo, Norway.

[6]Note that we are magnifying the area of a part of the souroundings of the Mandelbrot set by the length of the universe in metres. The more relevant comparison would be area with area, however, when running the code with such magnificaton introduces numerical noise that as a result does not generate a clear image.

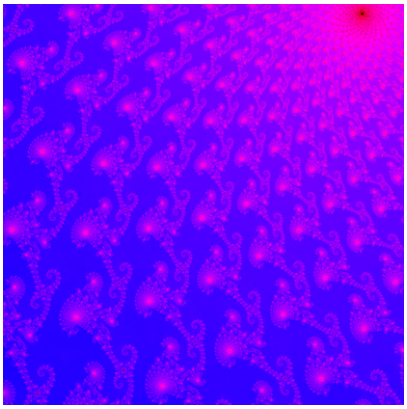

Figure 8: The magnification centered at the point in Section 10 by the ratio of 1m to the length of the known univirse.

13 Conclusion

I hope you would agree with me that there are parts of mathematics that creates literally beautiful and even artisitc images and patterns, as I have shown in this essay.

I think what makes the Mandelbrot set so special and beautiful is the fact that something as banally simple as Equation (1) creates infinite complexities. Zooming into the suroundings of the Mandelbrot set, as I did in Section 10, the pattern would infinitely expand to give new, self-similar complex patterns. It creates this artistic variation depending on the number of iterations, and shows that art encapsulates math. It is also a very interesting way to show how maths does not need to be incredibly complicated to be fun, and to have beauty in it. It's often assumed you need to have studied maths to a high level to understand it's beauty and complexity, but the Mandelbrot set disproves this and highlights that with just some simple maths and computer code, beauty emerges.

Appendix: Python code

I used Python to implement the figures in this essay. Python is a free programming language that can be donwloaded from `https://www.python.org/`. I used Python version 3.11.5, but I believe it would run on most verison of Python. I used the pillow package and the numpy package which can be installed by typing `pip install pillow` and `pip install numpy` on the command prompt after installing Python.

```python
def sequence(c,n):
    #equation = Z(n) = Z(n-1)^2 + c
    print("Z(0) = 0")
    z = 0
    for i in range(1,n+1):
        z = z*z + c
        print(f"z({i}) = {z}")

c = float(input("Value of c: "))
n = int(input("Number of loops (value n): "))
sequence(c,n)
```

Python code 1: Iteration of Equation (1) using real values of c only.

```python
def sequence(c,n):
    #equation = Z(n) = Z(n-1) + c
    print("Z(0) = 0")
    z_original = 0
    for i in range(1,n+1):
        z = z*z+ c*1j # in python, the imaginary number i is \
        ↪ denoted by the letter jj (as i is often used for \
        ↪ counting)
        print(f"z({i}) = {z}")

c = float(input("Value of c, as a multiple of imaginary number
i: "))
n = int(input("Number of loops (value n): "))
sequence(c,n)
```

Python code 2: Iteration series with imaginary numbers

```python
# Python code for Mandelbrot Fractal
# Import necessary libraries
from PIL import Image
from numpy import array
import colorsys
from datetime import datetime
def rgb_conv(n):
    ''' Color the pixel dependent on how many iterations n it \
    ↪ took for the mandelbrot iteration to '''
    color = 255 * array(colorsys.hsv_to_rgb(n/255.0, 1.0, 0.5))
    return tuple(color.astype(int))
```

```
11 def mandelbrot(x: float, i:float, max_iterations: int)->tuple:
12     '''Perform the mandelbrot iteration. If the absolute \
+     ↪ value of z > max_iterations, the z is assumed to go to \
+     ↪ infininte. We then color that point (pixel) dependent \
+     ↪ on how many iterations it took to get the absolute \
+     ↪ value of z > max_iterations
13     x : the x-coordinate on the real axis
14     i : the y-coordinate on the imaginary axis
15     max_iterations : the number of iterations to perform to \
+     ↪ check for |z|> max_iterations, if not then color it \
+     ↪ black (i.e. black are points in the Mandelbrot set).'''
16     c = x+1j*i
17     z = 0
18     for i in range(1, max_iterations):
19         if abs(z) > max_iterations:
20             return rgb_conv(i)
21         z = z ** 2 + c
22     return (0, 0, 0)

23 def draw_mandelbrot_set(x_min:float, x_max:float, \
+ ↪ y_min:float, y_max:float, width:int, \
+ ↪ max_iterations,file_name='mandelbrot.png'):
24     '''Draw the result of the mandelbrot iteration beteen the \
+     ↪ coordinates with a width and height'''

25     # We calculate the height from the width so that the step \
+     ↪ size is esentially the same along the real and \
+     ↪ imaginary axis so that the set does not look streatched.
26     x_step = (x_max-x_min)/width
27     height = int((y_max-y_min)/x_step)
28     y_step = (y_max-y_min)/height

29     # Allocate image size
30     img = Image.new('RGB', (width, height))
31     pixels = img.load()

32     for x in range(width):
33         # displaying the progress as percentage
34         if round(x/width * 100, 2) % 1 == 0:
35             print("%.1f %%" % (x / width * 100.0))
36         for y in range(height):
37             pixels[x, y] = mandelbrot(x_min+x*x_step,y_min+ \
+             ↪ y*y_step,max_iterations)
```

```
38      img.show()
39      img.save(file_name)

40  #draw the result at different interesting areas of the \
+   ↪ complex plane
41  draw_mandelbrot_set(-2.0,1,-1,1,1000,255)
```

Python code 3: Code to color the complex plane depending on the resulting number of iterations to determine if the point is withing the Mandelbrot set (black) or is expected to escape to infinity.

```
1  x= -0.7436438891276436
2  y = 0.13182590455455057
3  R = 8.1041015625
4  for i in range(0,14):
5      scale = 10**i
6      r=R / scale
7      file = f"mandelbrot_x{scale}.png"
8      draw_mandelbrot_set(x-r,x+r,y-r,y+r,1500,2000,file)
```

Python code 4: Create the plots for a journey through the Mandelbrot set using the centre $(x, y \cdot i)$ and a width of about 8 and magnify 100 times each step to a final magnification of 10^{26}. Use max interations of 80 instead of 2,000 for the smaller values of i.

```
1  z = c
2  c = -0.4 + 0.6j # Using different start points gives different \
+  ↪ Julia sets.
```

Python code 5: Julia set can easily be generated using code 3 by inserting the lines above between line 17 and 18 in code 3.

References

[1] Robert Devaney. What is the mandelbrot set? URL `https://plus.maths.org/content/what-mandelbrot-set`.

[2] Jens Feder. *FRACTALS*. Plenum-Press, 1988.

[3] Fractal Foundation. What are fractals. URL `https://fractalfoundation.org/resources/what-are-fractals/`.

[4] House of Math. The complex plane. URL https://www.houseofmath.com/encyclopedia/numbers-and-quantities/numbers/complex-numbers/introduction/what-does-the-complex-plane-mean.

[5] Jed Skilling. The undeserved mystery of complex numbers. URL https://tomrocksmaths.com/2023/08/10/teddy-rocks-maths-essay-competition-2023-student-winner/.

[6] Wikipedia. Complex numbers - wikipedia, March 2024. URL https://en.wikipedia.org/wiki/Complex_number#Further_reading.