Project 1 Fractals

Problem to solve

The problem presented is to solve the equation: $z^3 - 1 = 0$, since it is a polynomial of third degree there will be 3 distinct answers to the solution. Since the method that I'm going to use to solve it is the newton methods and it needs an initial guess, we can input all the points on a grid and see where they converge. Depending on the solution where they converge, we can color them one way or another resulting in a fractal.

Method used

As mentioned before the method used is going to be newton method. It is an iterative method which uses the derivative at an initial point to calculate the slope of the function, and base on the intersection of that slope with the x axis we can find the next point. Then the process repeat itself using that final point as the new initial point. The exact function is as follows:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

The function f(x) and f'(x) are:

$$f(x) = x^3 - 1$$

$$f'(x) = 3x^2$$

Since depending on the initial value x_0 the method can diverge or converge. And thus, since it is an iterative method, we need to create stopping conditions.

The method only run for a fixed number of iterations, denoted by a variable, the initial value is on 10 iterations, but I also tried 20 iterations as well. This fixed maximum number of iterations is set for the cases of divergence.

In the case that the point converge we can end the iterations before running the maximum iterations. To accomplish this 2 metrics are measure:

$$|x_{i+1} - x_i| < e$$

$$\left|x_{i+1}^3 - 1\right| < e$$

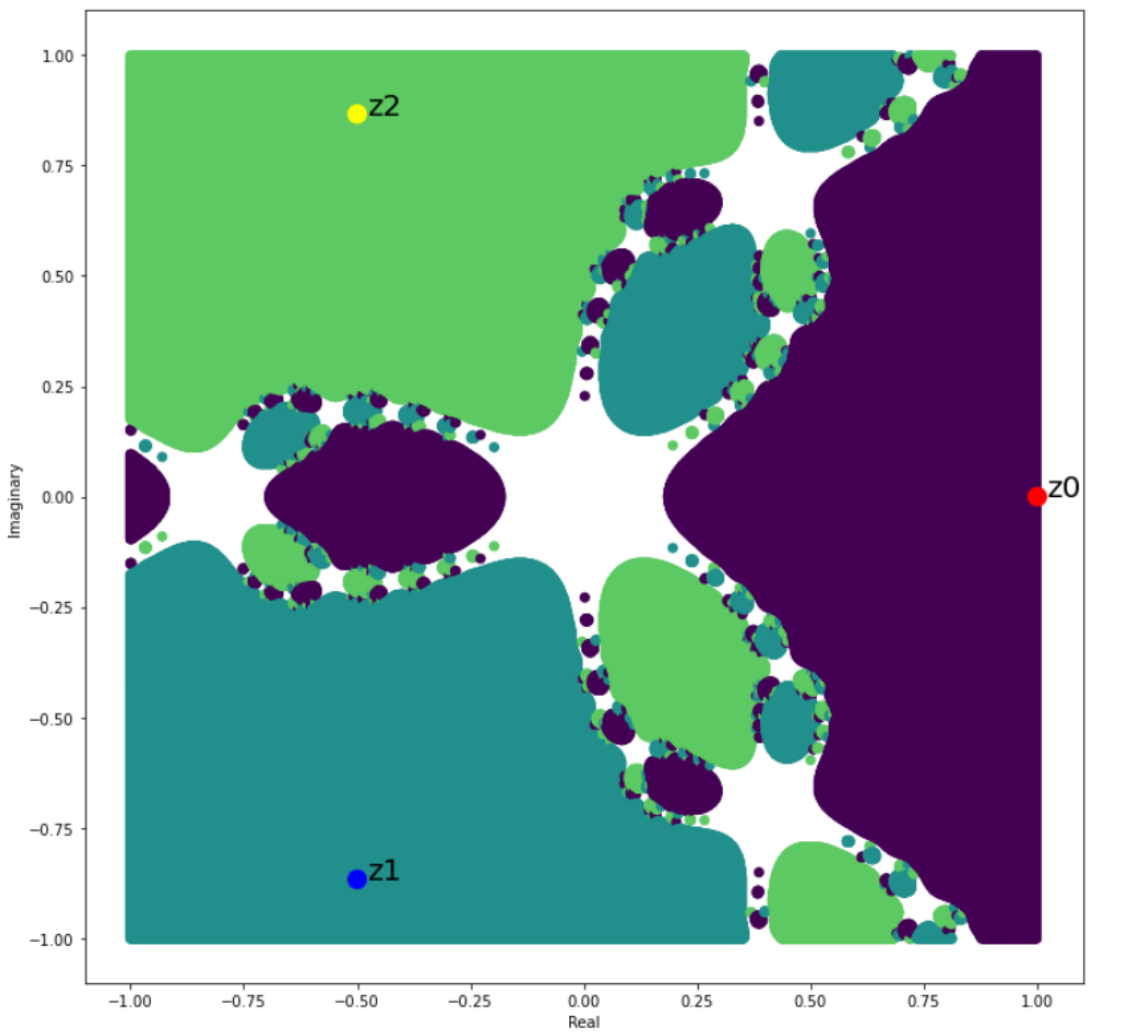
Both equations need to be true to enable the early stopping condition, 'e' is a variable initially set to 10^{-4}, this variable stands for error.

Alejandro C. Parra Garcia

If the number of iterations for a given value reaches the maximum set iterations, that initial point will be consider divergent.

Since we are working on the complex plain, we need to work with complex number, that means that all the formulas need to work with them. Since the code is run on python, we don't need to worry much about it, that is because python have in built complex numbers and can handle operations with them.

Using 10 iterations with an e of 10^{-4}, and plotting in a 1,000 x 1,000 grid over the coordinates $-1 \leq Real(x) \leq 1 \; and \; -1 \leq Imaginary(x) \leq 1$ we get the next result:



We can see that there are huge regions in white, meaning that they didn't converge. We can see in the following image that around 83.96% of the initial points converge.

```
print("Total Number of points evaluated: ", df.shape[0])
print("total Number of points that converge: ", df[df.converges].shape[0])
print("% of points that converge: ", df[df.converges].shape[0]/df.shape[0])

df
```

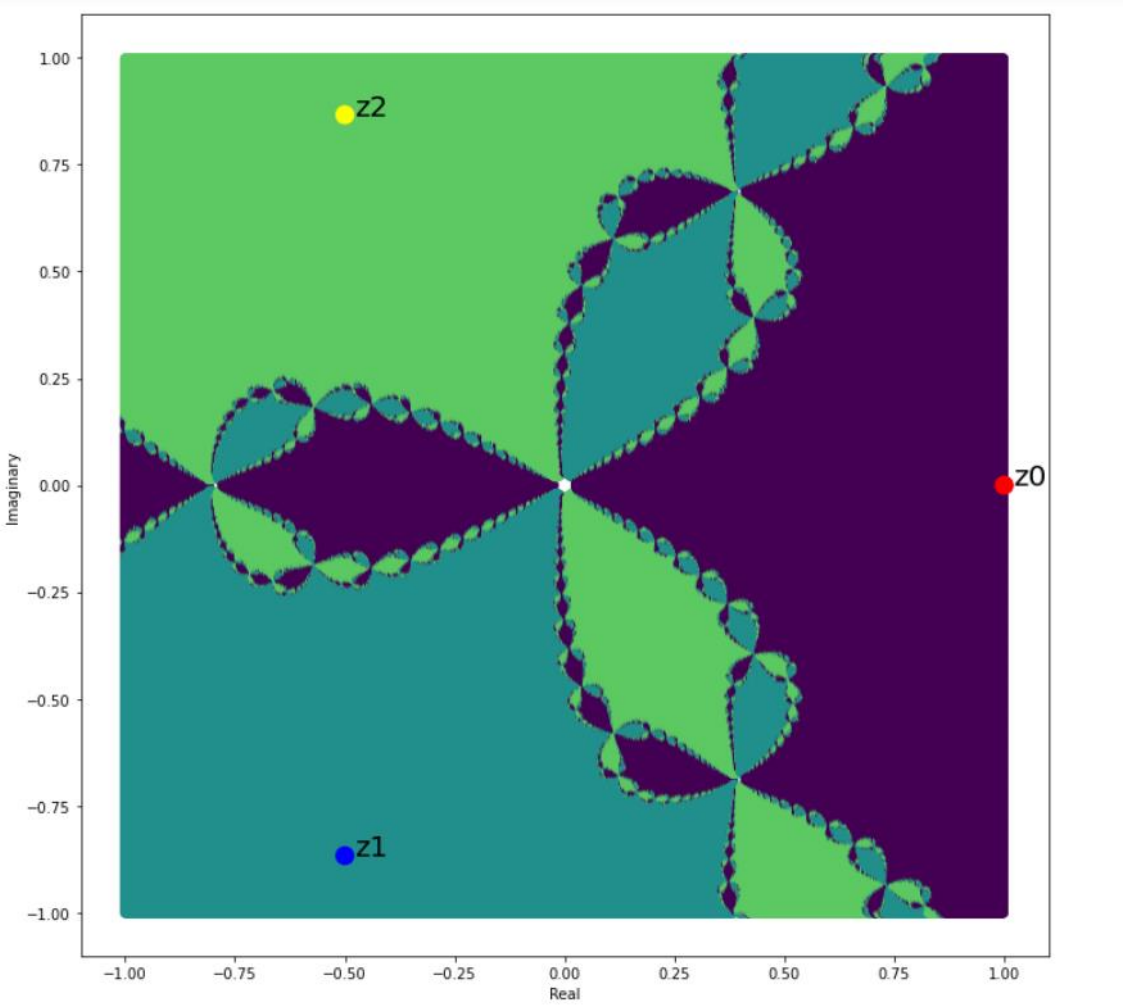
```
Total Number of points evaluated:  1000000
total Number of points that converge:  839664
% of points that converge:  0.839664
```

	complex_number	real_part	imaginary_part	converges	converge_to	color
0	-1.000-1.000j	-1.000	-1.000	True	-0.500000-0.866025j	#21908c
1	-1.000-0.998j	-1.000	-0.998	True	-0.500000-0.866025j	#21908c
2	-1.000-0.996j	-1.000	-0.996	True	-0.500000-0.866025j	#21908c
3	-1.000-0.994j	-1.000	-0.994	True	-0.500000-0.866025j	#21908c
4	-1.000-0.992j	-1.000	-0.992	True	-0.500000-0.866025j	#21908c
...	...	...	...	...	...	...
999995	0.998+0.990j	0.998	0.990	True	1.000000+0.000000j	#440154
999996	0.998+0.992j	0.998	0.992	True	1.000000+0.000000j	#440154
999997	0.998+0.994j	0.998	0.994	True	1.000000+0.000000j	#440154
999998	0.998+0.996j	0.998	0.996	True	1.000000+0.000000j	#440154
999999	0.998+0.998j	0.998	0.998	True	1.000000+0.000000j	#440154

1000000 rows × 6 columns

The rest of the points that didn't converge can be due to the fact that the point indeed is divergent or that the convergence for that initial point is slow at it didn't arrive at the error in the number of maximum iterations set, that is 10.

We can increase the number of points converging if we increase the number of iterations. If we change that number to 20, we get the next results:




```
print("Total Number of points evaluated: ", df.shape[0])
print("total Number of points that converge: ", df[df.converges].shape[0])
print("% of points that converge: ", df[df.converges].shape[0]/df.shape[0])
```

```
Total Number of points evaluated:  1000000
total Number of points that converge:  990373
% of points that converge:  0.990373
```

In this case we can see that the white regions got reduce substantially to basically the point near the 0,0 and some smaller other regions. In this case we can see that more than 99% of the initial points converge to a result.

Convergence and complexity

Newton method is quadratically convergence following:

$$e_{i+1} \approx Me_i^2$$

We can solve for M to check the constant, for that we need the first derivative of the function which we already know and the second derivative:

$$f'(x) = 3x^2$$

$$f''(x) = 6x$$

And the formula is as follow, the method quadratically converge to r, with the constant M if $f'(x) \neq 0$

$$M = \left| \frac{f''(r)}{2f'(r)} \right|$$

Then we can substitute the functions and we get the next:

$$M = \left| \frac{6r}{2 * 3r^2} \right| =$$

Since we have 3 distinct solutions: $1 + 0i, -0.5 - \frac{\sqrt{3}}{2}i, -0.5 + \frac{\sqrt{3}}{2}i$ lets check it for the 3

For $1 + 0i$:

$$M = \left| \frac{6(1 + 0i)}{2 * 3(1 + 0i)^2} \right| = 1 + 0i$$

$$M = \left| \frac{6\left(-0.5 - \frac{\sqrt{3}}{2}i\right)}{2 * 3\left(-0.5 - \frac{\sqrt{3}}{2}i\right)^2} \right| = -0.5 + \frac{\sqrt{3}}{2}i$$

$$M = \left| \frac{6\left(-0.5 + \frac{\sqrt{3}}{2}i\right)}{2 * 3\left(-0.5 + \frac{\sqrt{3}}{2}i\right)^2} \right| = -0.5 - \frac{\sqrt{3}}{2}i$$

For the different solutions this are the constant M for the evolution of the error.

We also need to take into consideration that in this case we are not solving the equation once, we are doing it on a grid of 1,000 x 1,000 points, that means that 1 million points are evaluated and in turn newton method is run 1 million times.

The number of operations of the newton method depends on the max iteration, for our case 10 or 20. Since we have early stopped conditions the method will stop before 10 or 20 iterations depending on the error threshold that we set.

So, we can set a maximum complexity base on the number of iterations: $\max\_iter * \left(\frac{f(x)}{f'(x)}\right)$
This is the theoretical maximum limit, but in reality, it stops earlier. In our case 99% of the times, it ended before 20 iterations and 83.96% of the cases ended before 10 iterations.

We now as well that it runs N^2 based on the grid we define. So, in our case we define a 1,000 x 1,000 grid so the newton method will run 1 million times. We can use this to set a <u>maximum</u> operation that the hole code will need to operate.

$$N^2 * \left(\max\_iter * \left(\frac{f(x)}{f'(x)} \right) \right)$$

Of course, there are other operations that I haven't consider on this calculus, such as saving the results in order to plot it later, or assigning colors to the solutions, setting the exact initial values for the newton method base on the grid. These operations have not been considered, since it won't make a significant change and will only complicate the results.

We can check it by changing the maximum number of iterations and the dimensions of the grid and see how the time evolves. Bellow we can see the time in seconds it took the hole program to run depending on the max_iter parameter.

```
Iterations:  1 --- 3.4717490673065186 seconds ---
Iterations:  2 --- 4.992631196975708 seconds ---
Iterations:  3 --- 5.677894592285156 seconds ---
Iterations:  4 --- 7.153274059295654 seconds ---
Iterations:  5 --- 8.438056468963623 seconds ---
Iterations:  6 --- 10.512917518615723 seconds ---
Iterations:  7 --- 11.24073338508606 seconds ---
Iterations:  8 --- 11.833204746246338 seconds ---
Iterations:  9 --- 11.851273775100708 seconds ---
Iterations:  10 --- 12.471421718597412 seconds ---
Iterations:  12 --- 13.076879262924194 seconds ---
Iterations:  14 --- 13.103393793106079 seconds ---
Iterations:  16 --- 13.058542966842651 seconds ---
Iterations:  18 --- 12.693711996078491 seconds ---
Iterations:  20 --- 12.519815921783447 seconds ---
Iterations:  25 --- 12.871939897537231 seconds ---
Iterations:  30 --- 12.970703601837158 seconds ---
```

We can see that it starts following a linear pattern, which is what we expected. Since the parameter max_iter only affects linearly at the complexity, but it starts to flatten at around 10 iterations, and it is completely still at around 12-13 iteration. This is expected, because of the

early stopping condition, when a point converges to a solution, at some iteration his error is smaller than the one set on the parameter e this makes the method stops since it has found a solution. Since most points converge in less iteration than the one set it makes the program stop earlier, which I turn makes it faster.

In the case when we modify the Dimensions of the grid, that is how many points we are going to use, we get the next results:

```
Dimensions:  100 --- 0.1293315887451172 seconds ---
Dimensions:  200 --- 0.4208803176879883 seconds ---
Dimensions:  300 --- 0.9025528430938721 seconds ---
Dimensions:  400 --- 1.6710004806518555 seconds ---
Dimensions:  500 --- 3.27073335647583 seconds ---
Dimensions:  600 --- 4.717536687850952 seconds ---
Dimensions:  700 --- 5.754428148269653 seconds ---
Dimensions:  800 --- 7.895517110824585 seconds ---
Dimensions:  900 --- 9.600504159927368 seconds ---
Dimensions:  1000 --- 11.339685201644897 seconds ---
Dimensions:  1500 --- 25.926889419555664 seconds ---
Dimensions:  2000 --- 47.16484570503235 seconds ---
```

As expected, the evolution is exponential, since the Dimensions (parameter N) affects quadratically to the number of operations. In this case we plot every point meaning that the complexity evolves at a quadratic rate.

Why Newton method?

Newton method is faster than the alternative bisection or FPI, since it has a quadratic convergence instead of a liner one of bisection of FPI.

Bisection

In our case, we are trying to create a fractal using newton method, colorizing the initial guess base on the root that finds. In the case of Bisection this wouldn't be possible since this method do not require an initial guess but in turn require an initial interval, meaning that we could not apply this method to draw the fractal.

Another important fact is that Bisection cannot find complex solutions of the equation. That means that, in our case of $z^3 - 1 = 0$ It will only find the solution 1.

FPI

In this case we indeed need an initial guess instead of an interval, which means that we could use it in our grid. Also, it can word with complex numbers which means it cand find the 3 solutions of our equation.

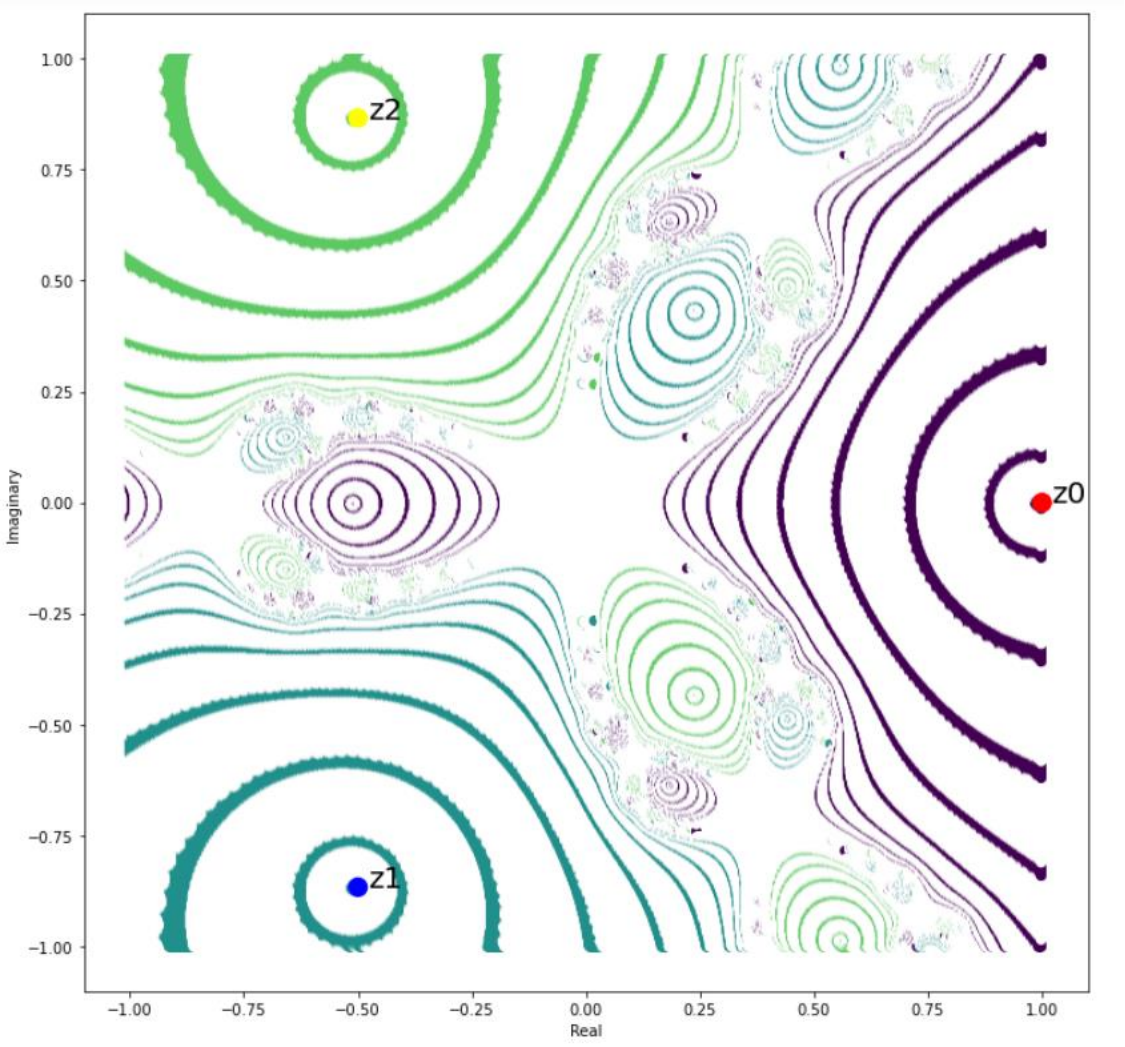
In fact, since Newton method can be thought of as an FPI, that means that we could use FPI to draw a graph, but I'm unsure how the resulting image will look like. But newton method is faster then FPI, so it is preferred over FPI. Another problem with FPI is that we need to think about how to right the equation as a FP problem, since there are multiple different ways.

Extra

At some point in the creation of the code I encounter a bug that resulted in these graphs. It was just a bug with the colors, nothing mayor, but since the resulting plot was curios, I decided to share it.

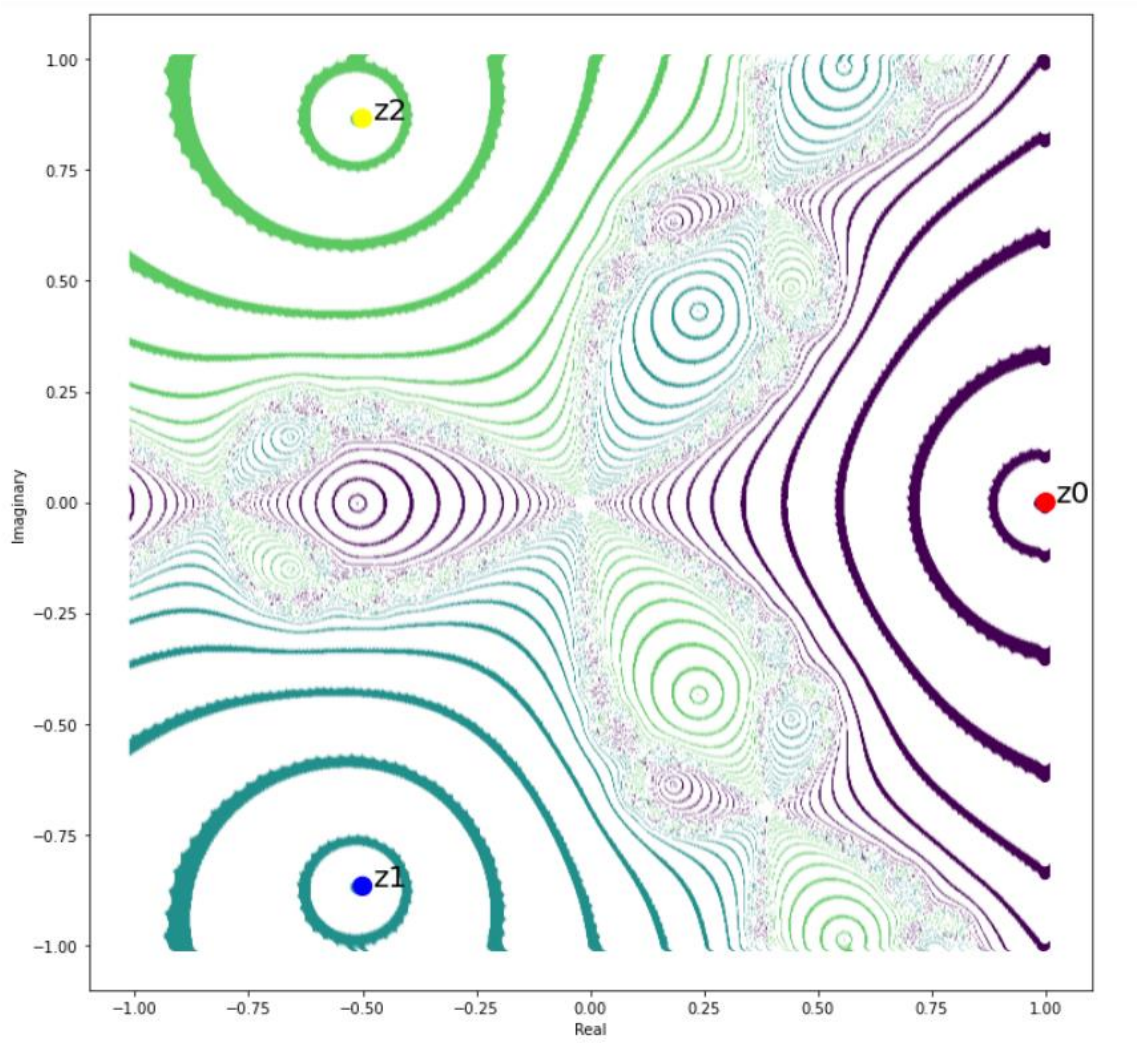
n=10

error=1e-2



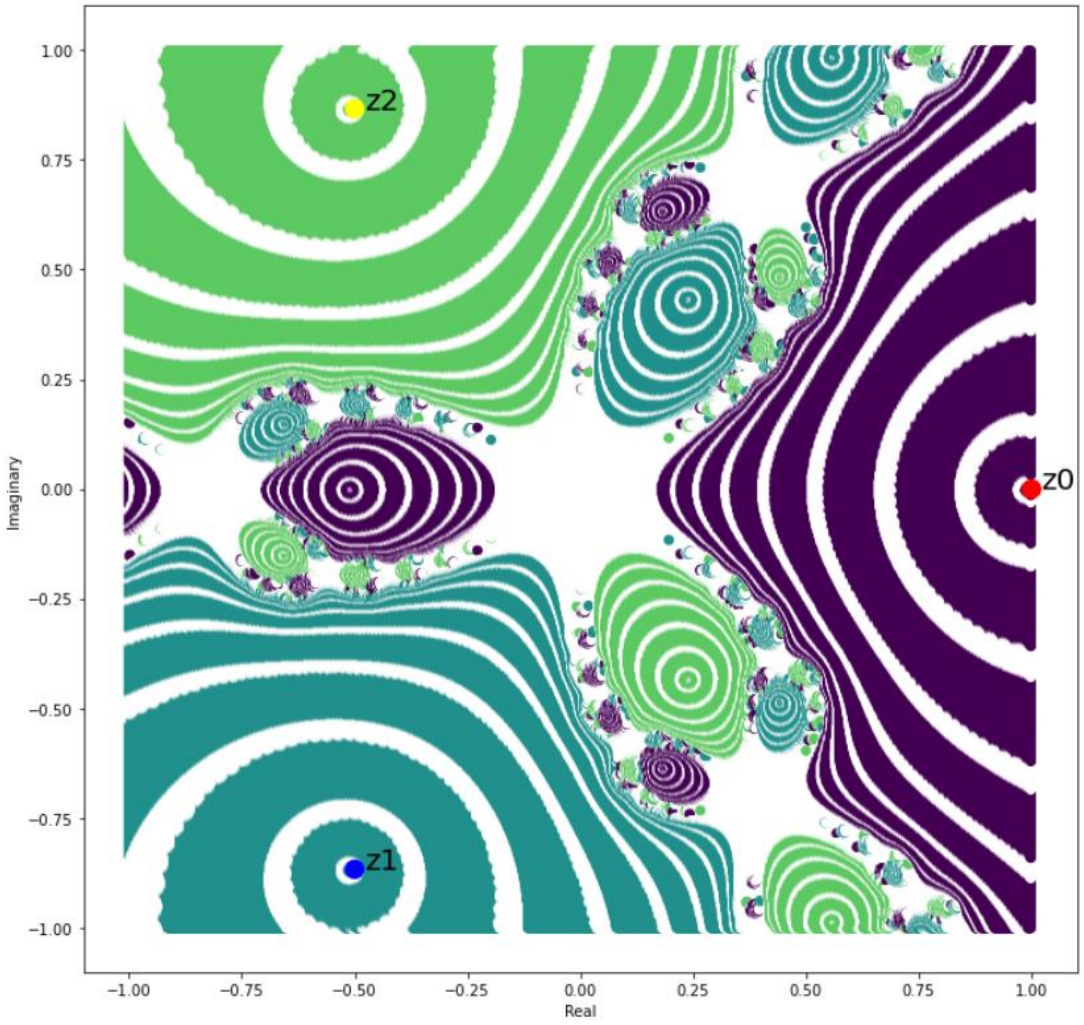
n=20

error=1e-2



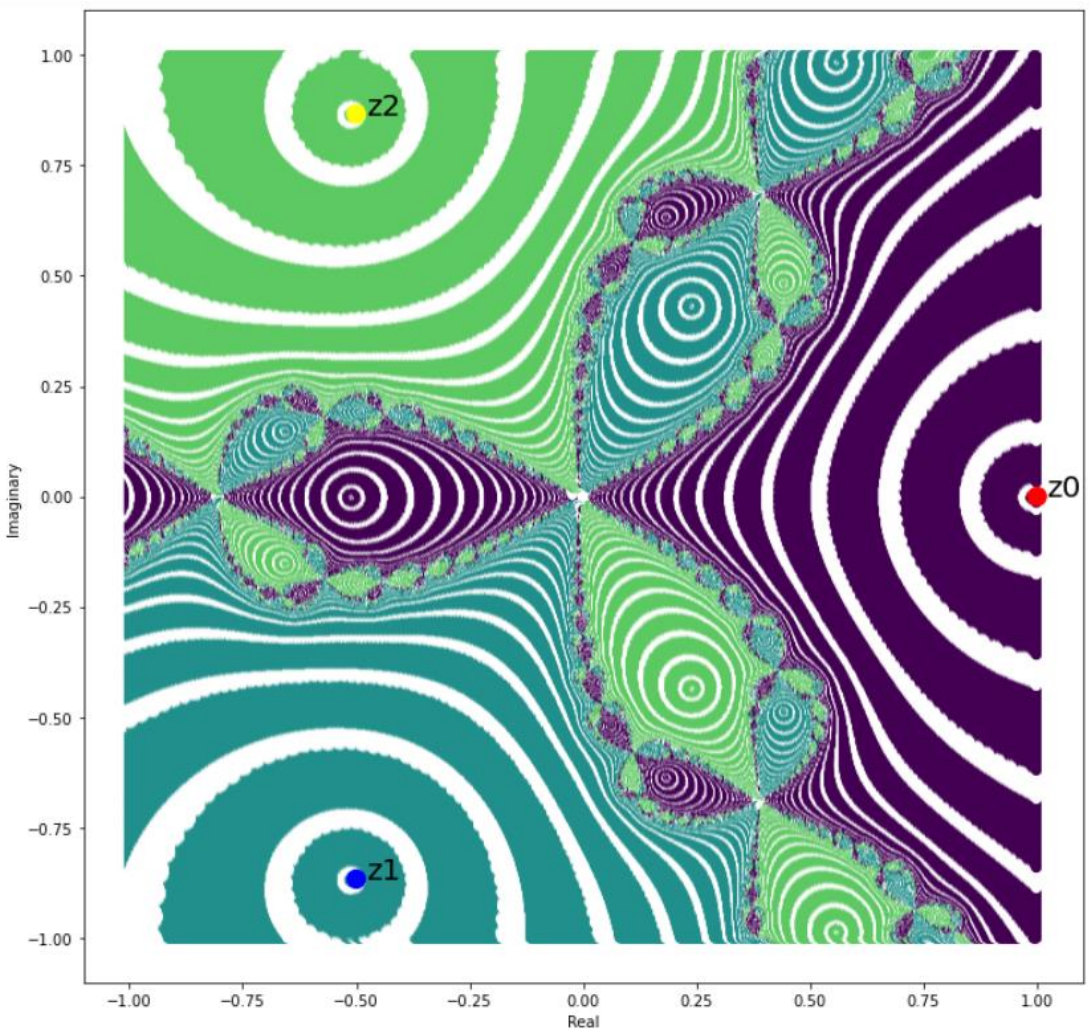
n=10

error=1e-3



n=20

error=1e-3


z2
z1
z0
Imaginary
Real
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00