Fractal dimension of the Julia set

Project report

Portik Attila, BKF0JP

October, 2020

Contents

1 Introduction

In the theory of complex dynamics, the Julia set is the set of unstable states of the dynamic system or in other words, the Julia set consists of points of phase space such that an arbitrarily small perturbation can cause drastic changes in the time evolution of the points. The time evolution of the dynamical system can describe by a function which maps a point of the phase space back into the phase space in every time step. Based on the above we can say, the behaviour of the time evolution function on the Julia set is "chaotic". An interesting property of the Julia set is the self-similarity, which means the Julia set is exactly or approximately similar to a part of itself. The self-similarity is a typical property of fractals so we can look at the Julia set like a fractal and measure the dimension of it.

Motivation

A simple quantum information protocol, wherein each step one carries out postselection on part of the qubits resulting in an effective nonlinear time evolution for the remaining qubits. This kind of iterated protocols can describe as a complex dynamic system. Where the time evolution function can be derived from the transformations on the qubits. By the study of the properties of the function, we get information about the dynamical properties of the protocol.

2 Description of the problem and theoretical background

Our protocol is built up from a three-qubit gate (G_{CNOT}), measurement, postelection, and a general quantum gate (U_p), each of these can be represented with an operator on the Hilbert-space of states of qubits. The first gate is a CNOT (Controlled NOT) gate, which acts on three qubits and entangles this. The measurement and postselection follows the CNOT gate. We measure two qubits and keep the third qubit if we observe the first qubits in the expected state. In the last step, we use a "tunable" quantum gate on the remaining qubit. The next figure shows one step of the protocol.

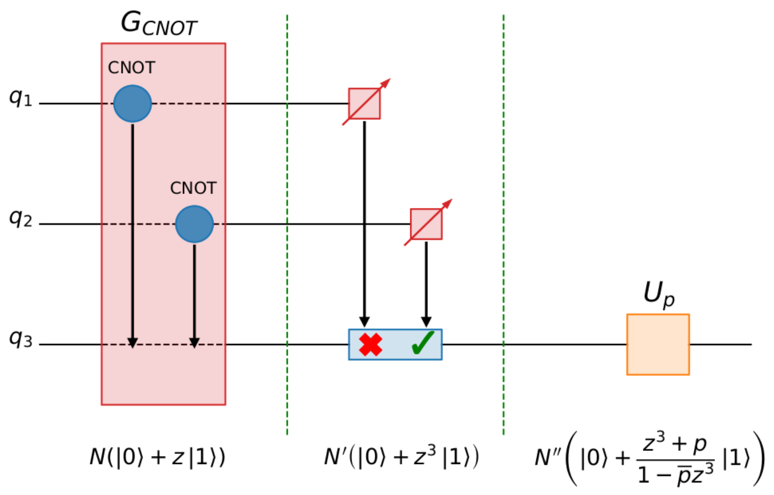

G_{CNOT}
CNOT
CNOT
q_1
q_2
q_3
U_p
$N(|0\rangle + z|1\rangle)$
$N'(|0\rangle + z^3|1\rangle)$
$N''\left(|0\rangle + \frac{z^3+p}{1-\bar{p}z^3}|1\rangle\right)$

Figure 1: Quantum circuit of the protocol. The figure shows the gates which make up the protocol. Under each part of the circuit are the state of the qubit.

In Rieamann representation the state of a qubit can be given by a single complex variable

$$N \cdot (|0\rangle + z\,|1\rangle) \quad z \in \hat{\mathcal{C}}\,, \tag{2.1}$$

where $N = 1/\sqrt{1+z^2}$. The transformation of the state can describe with a complex function

$$\hat{T}\left(N \cdot (|0\rangle + z\,|1\rangle)\right) = N' \cdot (|0\rangle + z'\,|1\rangle) \Rightarrow z' = f_p(z) = \frac{p + z^3}{1 - \bar{p}z^3}\,. \tag{2.2}$$

In this approach the protocol can be described as a complex dynamic system. For the complex dynamics the phase space can be the Riemann sphere $\left(\hat{\mathcal{C}}\right)$, the states of the system are represented by a complex number, and the time evolution function map the Riemann sphere onto itself: $f : \hat{\mathcal{C}} \to \hat{\mathcal{C}}$. In the case of discrete-time, the evolution of a state $z_0 \in \hat{\mathcal{C}}$ can be described by a difference equation

$$z_{n+1} = f(z_n) \;\Rightarrow\; z_1 = f(z_0)\,. \tag{2.3}$$

The state after many time steps can be determined by the iteration of the function

$$z_n = \underbrace{(f \circ f \circ \cdots \circ f)}_{n \text{ times}} = f^{\circ n}(z_0)\,. \tag{2.4}$$

2.1 Calculation of the Julia set

The Julia set of a function contains the unstable points of the iterated function system. More formally we can say that the Julia set is the closure of the set of repelling periodic points. In this project, I use two algorithms to calculate the Julia set of the function. The two algorithms are two different approaches to the problem and use different properties of the Julia set.

Backward iteration

The iteration of the inverse function is a well-known method to determine the Julia set. The base idea behind this algorithm is the following: The Julia set contains the repelling points of the function, thus we can not find this through iteration of function. If we start to back iterate the function, what is to say we start to iterate inverse of the function then repelling points become attractive points. The repelling periodic points of the function are the attractive periodic points of the inverse function. So if we found the attractive points and their basin of attraction for the inverse function then we found the Julia set of the direct function. But it is not this simple, because the studied function is a fraction of third order polynomials, so it has three inverse functions. Each three inverse can be calculated, the difference between their value at the same point is in the argument of the results. The problem is the following: during the iteration in each step the number of the points triples, which means if we start with one point, after 15 steps we have $\sum_{i=0}^{15} 3^i = 21523360$ points, therefore we can not make more steps, but to find points from every area of the Julia set need to do. The solution is the random choice between the inverses in each step. In this way, we use every combination of inverses, but in every step we have just one new point. The next property of the Julia set guarantee, that with this method we can find every point of the Julia set: If $z_0 \in J(f)$, then the set off all iterated pre-images $\{z : f^{\circ n} = z_0 \text{ for some } n \geq 0\}$ is everywhere dense in $J(f)$. ([5]) That means, if we start with any $z_0 \in J(f)$ and compute all pre-images $f(z_1) = z_0$, then compute all pre-images $f(z_2) = z_1$, and so on, thus eventually coming arbitrarily close to every point of $J(f)$.

I made my own implementation of this method in python. The script first calculates the repelling fixed points of the function, because these are in the Julia set, thus are good options as starting points. The number of these can be 2 or 4 depending on the parameter of the function: $s \in [2,4]$. The fixed points can calculate as the roots of the equation:

$$\overline{p}z^4 + z^3 - z + p = 0\,. \tag{2.5}$$

I solve this by computing the eigenvalue of the companion matrix of the polynomial

$$\begin{pmatrix} 0 & 0 & 0 & -\frac{p}{\overline{p}} \\ 1 & 0 & 0 & \frac{1}{\overline{p}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{\overline{p}} \end{pmatrix} \mathbf{u} = \lambda \mathbf{u} \tag{2.6}$$

/ It is easy with the functions of NumPy. Then I determined the multiplier, which is the derivative of the function in the case of fixed points. If its multipliers absolute value is higher than 1, then it is repelling. After that start to iterate the inverse functions.

$$f(z) = \frac{p+z^3}{1-\overline{p}z^3} \Rightarrow \left(f^{-1}\right)^3 = \frac{z-1}{\overline{p}z-1} = r \cdot \exp(i\theta)$$
$$f_1^{-1} = \sqrt[3]{r}\exp\left(i\frac{\theta}{3}\right) \quad f_2^{-1} = \sqrt[3]{r}\exp\left(i\frac{\theta}{3}\right)\exp\left(\frac{2\pi}{3i}\right) \quad f_3^{-1} = \sqrt[3]{r}\exp\left(i\frac{\theta}{3}\right)\exp\left(\frac{4\pi}{3}i\right) \tag{2.7}$$

Every step chooses randomly between the pre-images. Practically choose between the cube-roots. Is a different choice for every branch of back-iteration. After n steps, we get s series with length n, which are the points of the Julia set.

The advantage of this method is that does not need too many resources. The disadvantage we can't analyze a separate part of the plane.

Forward iteration

The base of the second algorithm is another property of the Julia set. The attractive periodic points of the function and their basin of attraction are in the Fatou set, which is the complement of the Julia set. That means if we check the convergence of the points of the complex plane we can compute the elements of the Fatou and the Julia set. I made a program that implements this method. First, selects a range from the complex plane, and discretizations this with a given resolution. The points of the plane are stored as an array. Then starts to iterate the function in every point. After m iterations, searches the periodicity of the points during $m/2$ iterations, thus determines the non-convergence points. To use this algorithm we need more resources because must work with a large array, but we can study just one part of the plane, thus the resolution becomes higher. It is not a problem to look to just a part of the Julia set because the whole set and a part of it have the same box-counting dimension, as a result of the self-similarity of the Julia set.

2.2 Estimating the box-counting dimension

The measure of the fractal dimension is the Hausdorff dimension, we can say it is the measure of the roughness of fractal. The Minkowski dimension (or box-counting dimension) is another way to determining the fractal dimension of a set. It is similar to the Hausdorff dimension but much easier to calculate. Luckily the two measures of the fractal dimension are the same in the case of the Julia set. We know the Hausdorff dimension of the Julia set of our function is generally between 1 and 2, but it can exist some extreme case when it is higher.

The method which I use for estimating is based on an interesting concept of the dimension of the objects. Imagine a cube in 3D. We cut it up to eight equal parts. The small cube's side length will be half of the side length of the original cube, but the volume of the new cubes will be one-eighth of the volume of the original cube. So it means when we scale the side length with a factor then the volume is scale by that factor raised to the third power. It is a general idea, and it is easy to see that in 2D the power of the scale is 2, and in 1D it is 1.

$$L' = sL \Rightarrow V' = s^D V \,, \tag{2.8}$$

where D is the dimension. The assumption is that the relationship is true for objects with fraction dimensions too, namely for the fractals. I used this idea to compute the box-counting dimension of the fractals. The method for calculating the fractal dimension is relatively easy. To compute the dimension for a fractal we put a grid on the plane of the fractal and count how many boxes are required to cover the set. After that, we make the grid finer, which is equivalent to scaling up the fractal, and count again the number of required boxes. We continue this process and watching the changing of the number of boxes.

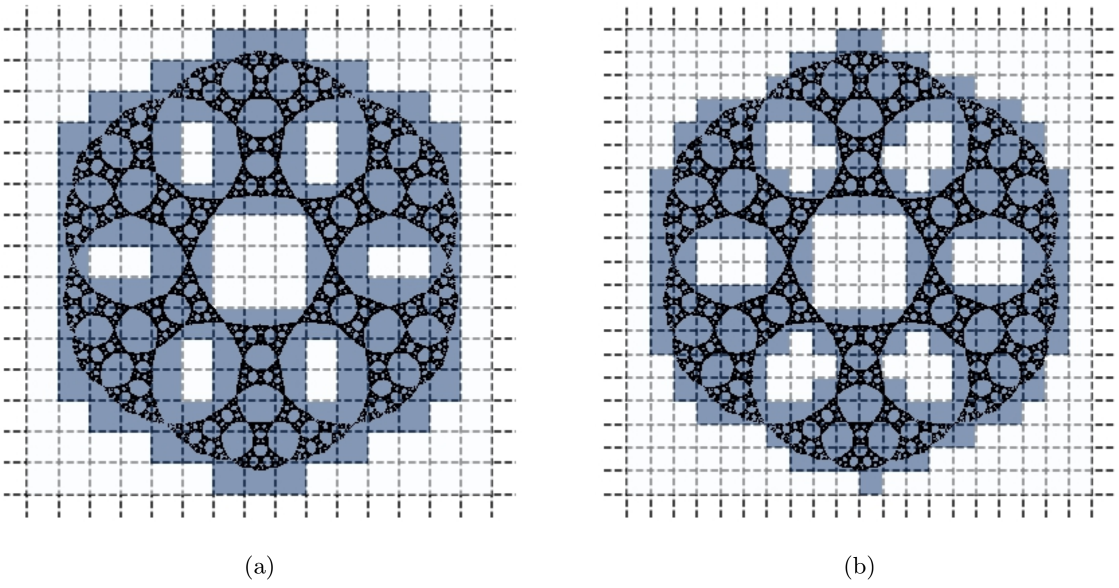


Figure 2: Illustration of the box-counting. The black object is the Julia set, and the colored squares are the boxes that need to cover it.

Based on the above the relationship between the number of boxes and the scaling factor is exponential

$$N(s) = cs^d \,, \tag{2.9}$$

where the d is the dimension. We can express the value of the dimension. First take the logarithm of both sides

$$\log(N) = \log\left(cs^d\right) = \log(c) + d\log(s) \,. \tag{2.10}$$

Then the d is

$$d = \frac{\log(N)}{\log(s)} - \frac{1}{\log(s)} \,. \tag{2.11}$$

Because scaling up the object is equal to scaling down the grid, we can use the scale factor (s') of the grid, that is the inverse of the s. Whit this the d is the following:

$$\log(N) = \log\left(c/s^d\right) = \log(c) - d\log(s) \Rightarrow d = -\frac{(N)}{\log(s)} - \frac{1}{\log(s)} \,. \tag{2.12}$$

In the numerical calculation, I determine the d with linear regression to thelog$(N) - \log(s)$ data pairs.

I made a program to convert the points of the Julia set to an array and calculate the number of boxes in case of different scales. The program stores a range from the complex plane as a square array. The indexes of an element indicate the value of the associated point

$$(k,l) \rightarrow z = x + yi = k \cdot \frac{x_{max} - x_{min}}{n} + l \cdot \frac{y_{max} - y_{min}}{n} i \,, \tag{2.13}$$

where $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ is the range, n is the shape of the array. The value of the elements of the array is 1 or 0 depending on the shape of the Julia set. If the associated point is in the Julia set then the element is 1, else it is 0. The program chose values of the $\log(s)$ between two bounders -which are input parameters- uniformly. I use a modified histogram algorithm, this calculates the bin size from the scale factor and the shape of the array. Then calculates the number of the filled bins.

Julia set	Borders of boxes	Boxes	Number of boxes
$\begin{pmatrix} 0&1&1&0&0&0&0&0 \\ 0&0&0&1&0&0&1&0 \\ 1&1&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&0&1&0&1 \\ 0&1&0&0&0&0&1&0 \\ 0&0&0&0&1&0&1&0 \end{pmatrix}$	$\left(\begin{array}{\|cc\|cc\|cc\|cc\|} \hline 0&1&1&0&0&0&0&0 \\ 0&0&0&1&0&0&1&0 \\ \hline 1&1&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0 \\ \hline 1&0&0&1&0&0&1&0 \\ 0&1&0&0&0&1&0&1 \\ \hline 0&1&0&0&0&0&1&0 \\ 0&0&0&0&1&0&1&0 \\ \hline \end{array}\right)$	$\begin{pmatrix} 1&1&0&1 \\ 1&0&1&0 \\ 1&1&1&1 \\ 1&0&1&1 \end{pmatrix}$	12

Table 1: The illustration of the counting of the numbers of the boxes.

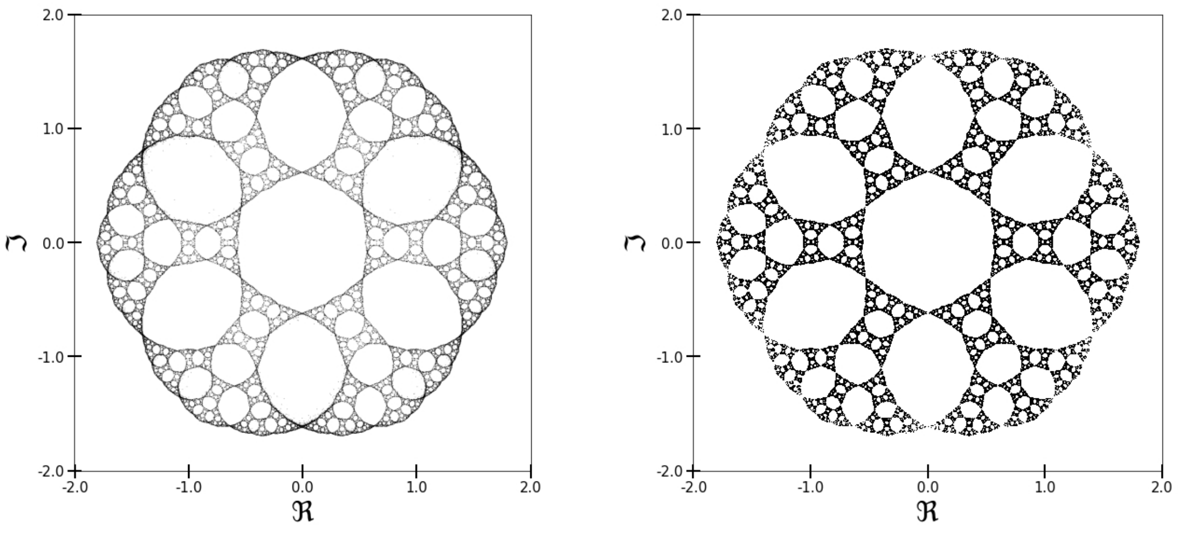
The last step is the linear regression on data points, and the absolute value of the slope of the line will be the dimension of the fractal. The scripts that I use for numerical calculation is being attached to the report. I used the NumPy, Scipy, and Matplotlib package of python.

3 Presentation of results

The main results of my project is a program that calculates the Julia set of the function and estimates its box-counting dimension. With this program, I study the shape of the Julia set and the box-counting dimension depends on the p parameter. In contrast with the quadratic case, the dynamics of the third-order rational function were not examined.

3.1 Self-similarity

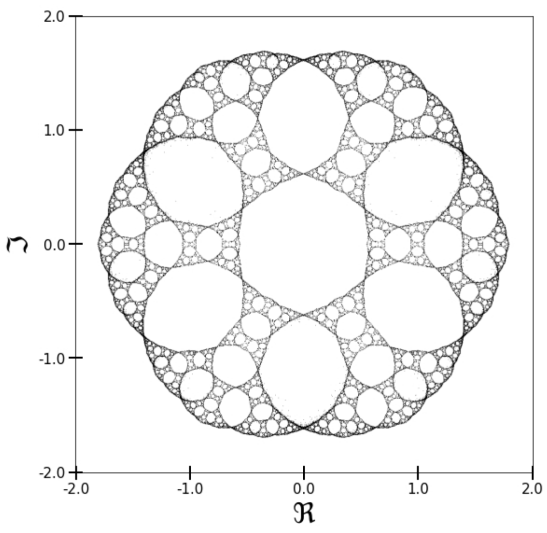
First I look to the Julia set of function with the parameter $p = i$. This is a nice example because of the structure of the Julia set. We can check the self-similarity of the Julia set, calculate the set and estimate the dimensions of different parts of it.



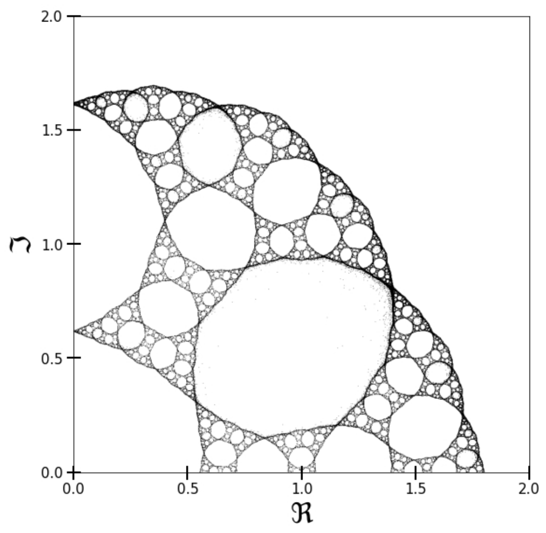
(a) Frontward iteration (b) Backward iteration

Figure 3: The Julia set of the function $f_i = \dfrac{z^3 + i}{1 + iz^3}$.

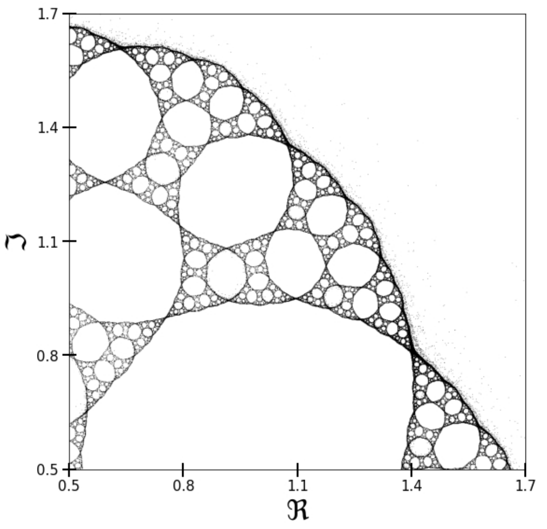
Fort the study of self-similarity I use the forward iteration algorithm because that can look just a part of the Julia set with the same resolution of the plane. I used the following range of the complex plane: $[-2, 2] \times [-2, 2]$, $[0, 2] \times [0, 2]$, $[0.5, 1.7] \times [0.5, 1.7]$, $[0.8, 1.5] \times [0.8, 1.5]$. The corresponding box-counting dimensions are: 1.508, 1.52, 1.517, 1.514. The calculation is in the 'self-similarity1.py' notebook. It contains the details of the calculation of the Julia set and the regression. The box dimension of the parts is close to each other according to the theory. We know the box dimension of the part must be equal, based on it can conclude to the accuracy of the calculation. We can see the first decimals of values are the same, but the second decimals are different.



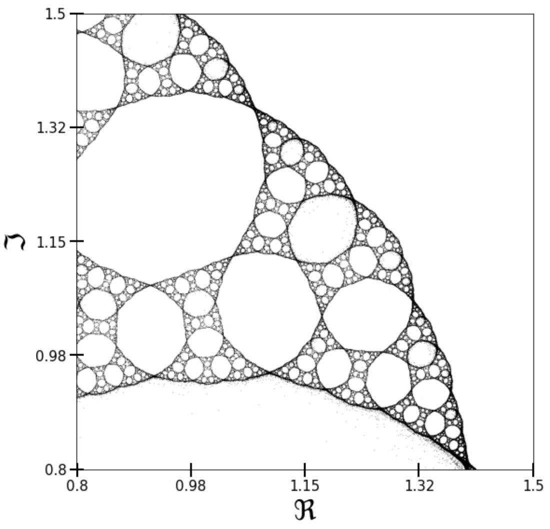
(a) $d = 1.508$



(b) $d = 1.52$



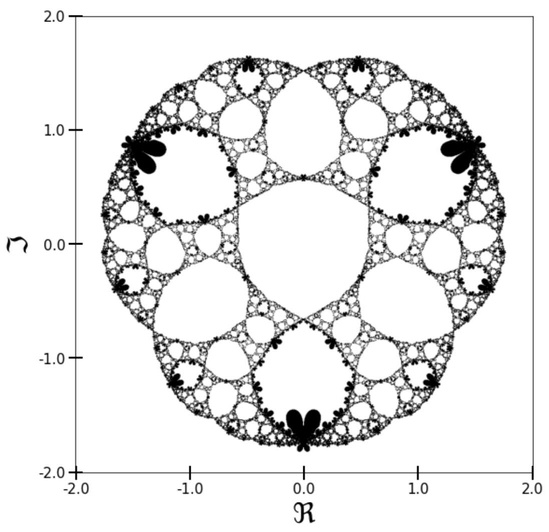
(c) $d = 1.517$



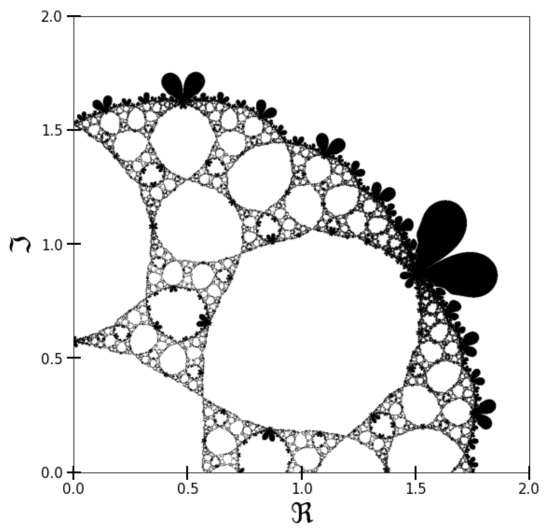
(d) $d = 1.514$

Figure 4: The sections of the Julia set of the f_i.

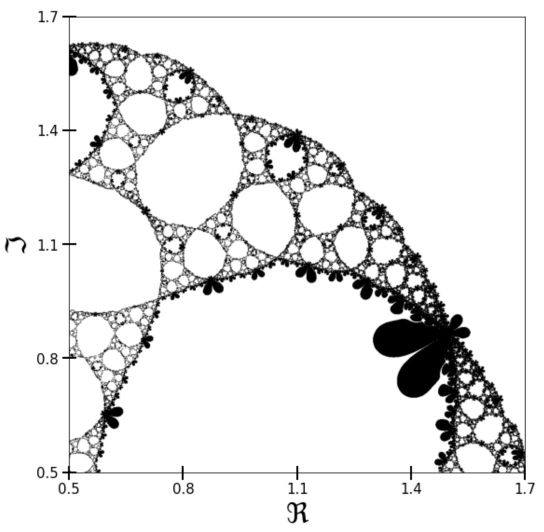
I made the same calculation in case of parameter $p = \frac{\sqrt{3}}{2}i$. The ranges are the same. Then the box-counting dimensions: 1.621, 1.646, 1.618, 1.621. We found the same in this case too.



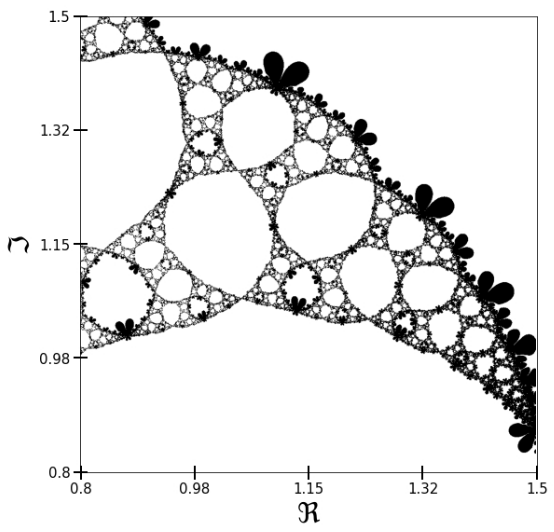
(a) $d = 1.621$



(b) $d = 1.646$



(c) $d = 1.618$

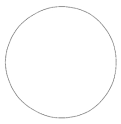
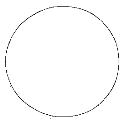
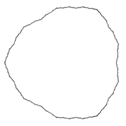
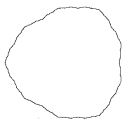
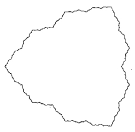
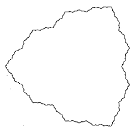
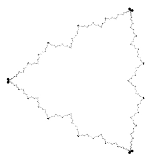
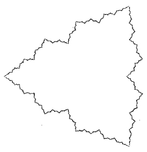
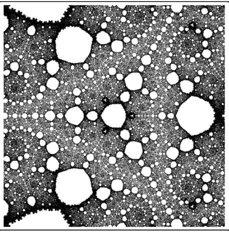
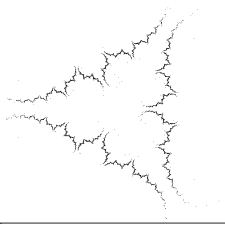
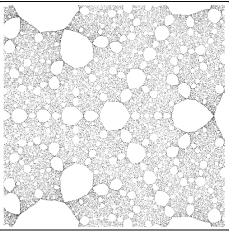
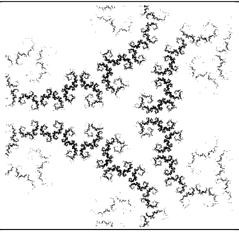
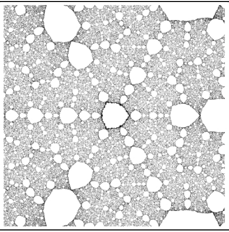
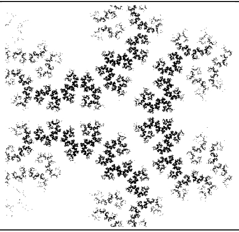


(d) $d = 1.621$

Figure 5: The sections of the Julia set of the $f_{\sqrt{3}i/2}$.

3.2 Fractal dimensions

In the next part of the report, I present different Julia sets and its box-counting dimension. I analyze the real and the imaginary axis of parameter space, which means I chose real and pure complex numbers, the reason was that along the real ax every type of dynamics appears. In the case of the imaginary parameters the dynamics of the functions are more simple in general. I start with the real parameters, present some typical examples in the next table. In the case of some parameters the forward iterations algorithm does not work, because to find the periodic points need too many iterations and too high resolution. In these cases, the estimation obviously is not correct.

p	Forward iteration	dimension	Backward iteration	dimension
0.01		1.057		0.991
0.1		1.059		1.069
0.2		1.16		1.121
0.3		1.195		1.175
0.4		1.831		1.432
0.5		1.831		1.575
0.6		1.811		1.503

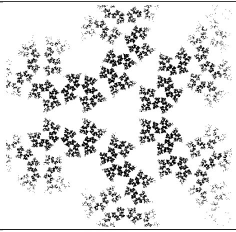
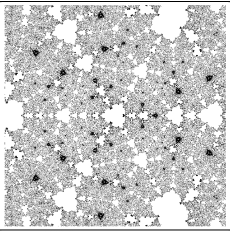
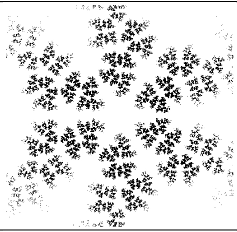
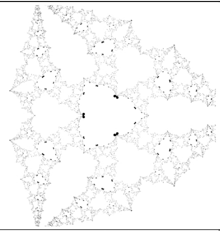
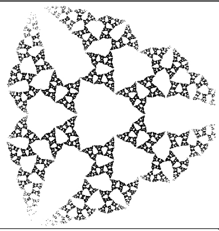
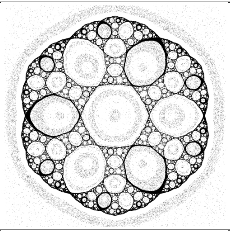
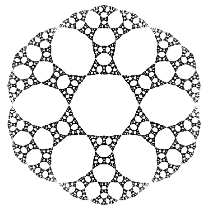
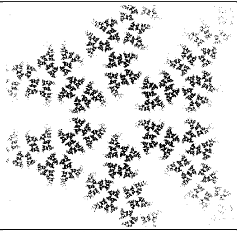
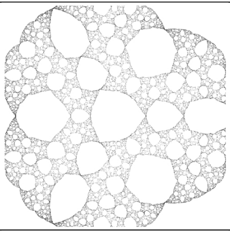
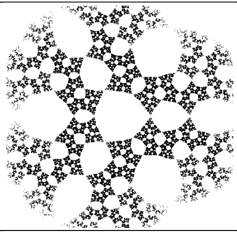
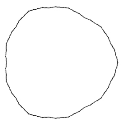
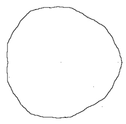
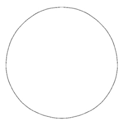
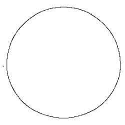
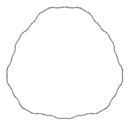
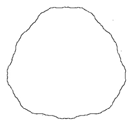
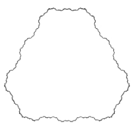
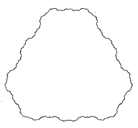
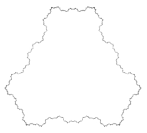
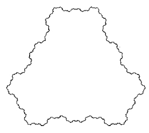
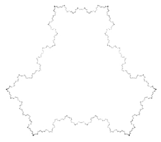
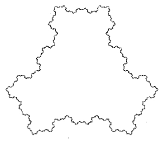
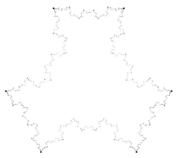
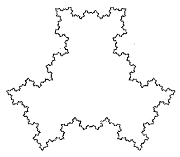
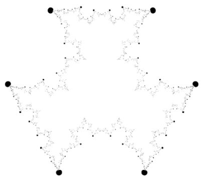
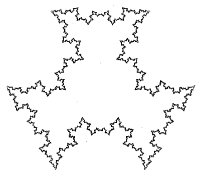
p	Forward iteration	dimension	Backward iteration	dimension
0.7		1.911		1.533
0.8		1.858		1.540
0.9		1.609		1.509
1		1.974		1.615
1.2		1.911		1.641
1.5		1.667		1.629
15		1.132		1.016

Table 2: The table contains the estimated values of the box-counting dimension. Where the picture is missing or the value of the dimension is about 2, there the algorithm did not find the Julia set.

The case of the pure imaginary parameters are more simple, the Julia set is similar for the all parameter.

p	Forward iteration	dimension	Backward iteration	dimension
0.01i		1.062		0.989
0.1i		1.057		1.064
0.2i		1.122		1.106
0.3i		1.194		1.118
0.4i		1.198		1.1256
0.5i		1.245		1.39
0.6i		1.323		1.51

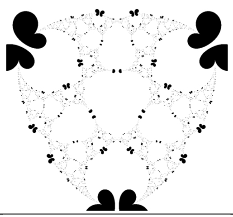
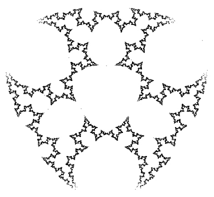
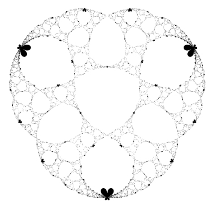
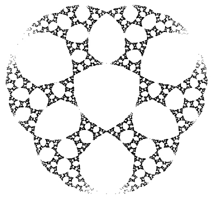
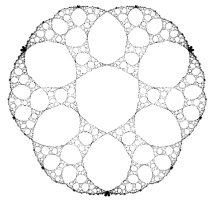
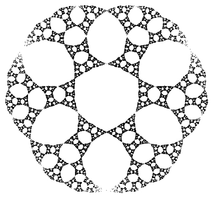
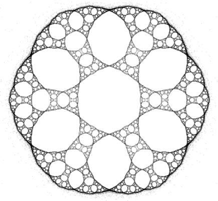
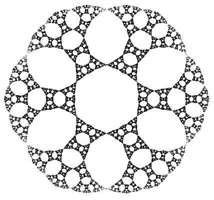
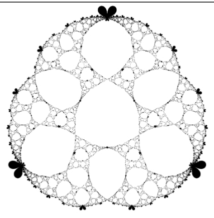
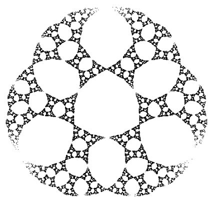
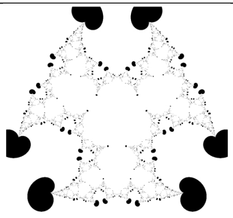
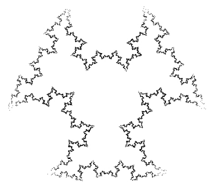
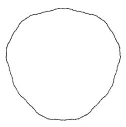
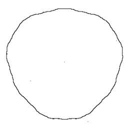
p	Forward iteration	dimension	Backward iteration	dimension
0.7i		1.409		1.477
0.8i		1.491		1.522
0.9i		1.534		1.578
1i		1.55		1.529
1.2i		1.572		1.46
1.5i		1.39		1.118
15i		0.9		1.016

Table 3: The table contains the estimated values of the box-counting dimension of Julia sets of pure imaginary parameters.

The calculations are in the 'imag.ipynb' and 'real.ipynb' python notebooks.

3.3 The parameter space

I studied the structure of the parameter space from the viewpoint of the box-counting dimension. I select a range from the parameter plane and discrediting it with a given resolution, then calculating the dimension for every parameter. The result is presented in the following picture:

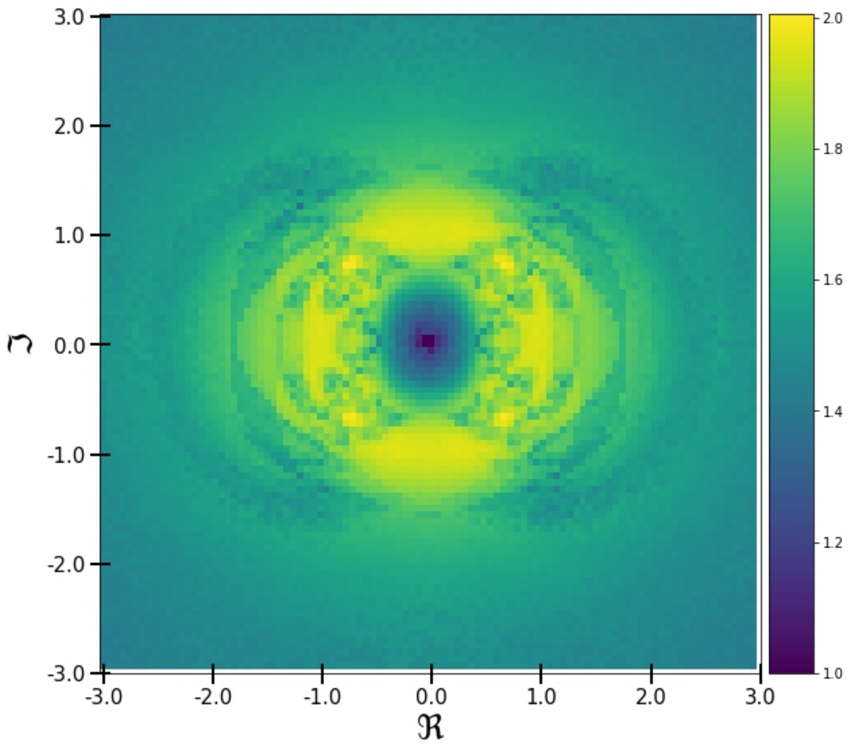


Figure 6: The structure of the parameter space. The figure is colored based on the value of the box-counting dimension.

4 Discussion

In the project, I managed to determine the Julia set, show the self-similarity of it, and estimate its box-counting dimension. The algorithm which is used is not enough good to determine the dimension accurately, but I succeeded the show the appearance of fractal behavior. In the previous part, you can see the changing of the fractal dimension depend on the parameter. For small parameters, the Julia set is a smooth closed curve and its dimension is about 1, like in the case of big parameters too. Whit the increase of the parameter the set becomes more roughly and its dimension is increasing too. The maximum value of the dimension is about 1.6 (I found higher values but these appear because of the inaccuracy of the algorithm). In the case of pure imaginary parameters the Julia set is in the center range of the complex plane. For some real parameters, the Julia set to spread out, theoretically, in some cases, the Julia set is the whole plane.

Most errors are caused by numerical precision. Lots of information are lost because of the resolution of the complex plane, and the maximal number of iteration. On my computer, I used 1000 $\times$ 1000 arrays and a maximum of 2000 iterations. Basically, the box-counting dimension is not an accurate measure of the dimension, at many times it is enough, but exists cases where not. Exists more accurate but more complex algorithms and methods to determine the dimension. The Hausdorff-dimension of the Julia set can be calculated by evaluation of the derivative of the function in periodic points, but the calculate of the periodic points is a hard problem. [4]

Conclusion

I showed quantitative of the fractal behavior of the Julia set for iterated third order functions system: the self-similarity properties and the fractional dimension.

References

[1] Jiaxin Wu, Win Jim, Shuo Mi, and Jimbo Tang, An effective method to compute the box-counting dimension based on themathematical definition and intervals (2020)

[2] `https://en.wikipedia.org/wiki/Julia_set`

[3] `https://en.wikipedia.org/wiki/Minkowski-Bouligand_dimension`

[4] Bi Feng, and Li Chuan-Feng, Fractals in Quantum Information Process (2013)

[5] John Milnor, Dynamics in one complex variable