A Brief Insight into Complex Polynomial Root-finding and Fractals

Name: Shao Yanjun, Number: 19307110036

June 30, 2021

Abstract

This is Daniel's final project of "Numerical Algorithms with Case Studies II".

1 Julia Set, Mandelbrot Set and Newton Fractals [1]

The modern day interest in Julia sets and related mathematics began in the year 1920, which was initiated by French mathematician Gaston Julia. In 1918 he wrote a paper titled *"M´emoire sur l'iteration des functions rationelles"*[2] (A Note on the Iteration of Rational Functions) where he first introduced the modern idea of a Julia set. Personally, I began my research on this topic when I was watching Numberphile videos on Youtube [3]. Comments suggested that the algorithm is not too complicated to implement in Matlab, so I spent a whole afternoon to make it work with animation and relatively high resolution. Hopefully, it works quite well under finite iterations of calculation.

A mapping of the form reading,

$$f : z \to z^2 + c \qquad c \in \mathbb{C}, |c| \leq 2$$

is defined, so that for different values of the parameter c, we can find a conceptually easiest way to define Julia set.

Definition 1.1. *Let c be any complex number. The smallest closed set in the complex plane that contains all fixed points and all periodic points of the map $f_c(z) = z^2 + c$ is called Julia set of the map f_c, denoted by J_c.*

We can also derive a trivial but exciting Lemma for the property of Julia Set. However, I will focus more on the visualization of it instead of paying too much attention on theoretical proof.

Lemma 1.2. *The Julia set J_c of a function $f_c(z) = z^2 + c$ is bounded.*

Proof. Choose $r = \max(|c|, 2 + \epsilon)$ and let $|z| \geq r$,

$$\begin{aligned} |f_c(z)| = |z^2 + c| &\geq |z^2| - |c| \\ &\geq (r-1)|z| \geq (1+\epsilon)|z| \end{aligned}$$

[1] Mahanta, Arun & Sarmah, Hemanta & Paul, Ranu & Choudhury, Gautam. (2016). Julia set and some of its properties. 5. 99-124.

[2] Julia G.," M'emoire Sur I'iteration des fonctions rationelles" Journal de Math. Pure ET Appl. 8(1918) 47-245.Republished in Herve M. (ed), Oeuvers de Gaston Julia: Vol.1, Gauthiers-Villars 1968, 121-333.

[3] *https://www.youtube.com/watch?v=oCkQ7WK7vuY*

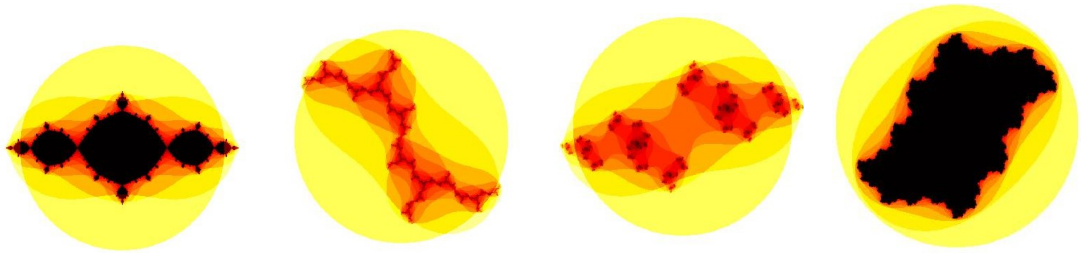
With this, we will deduce $f_c^n(z)$ for all $|z| \geq r$,

$$f_c^n(z) \geq (1+\epsilon)^n |z| \to \infty$$

This will give a bound for $J_c \subset B(0,r)$. □

This Lemma at least gives a sound guarantee that whatever we initialize our iteration on $z \in \mathbb{C}$ and $|z| < r$, a Julia Set can be plot with amazing patterns.

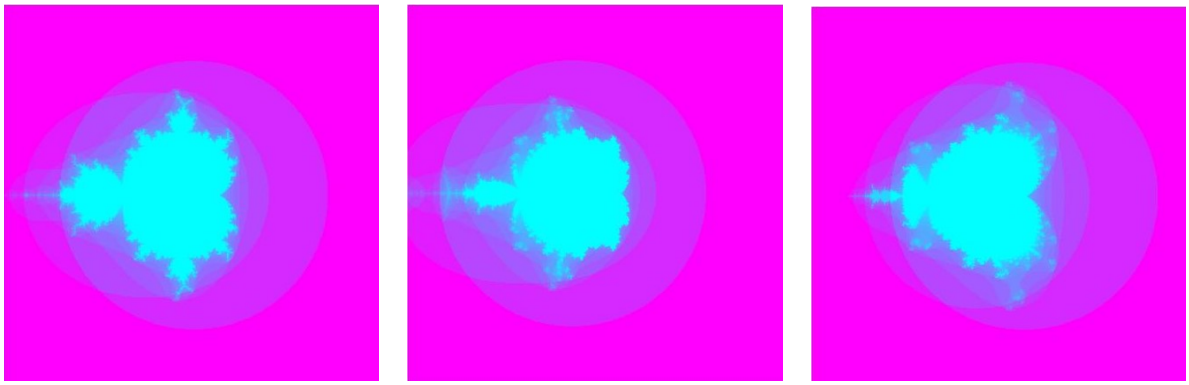
A animated gif is generated in ./fig/Julia.gif directory, and here are some 'one-milion-dollar' glimpses of fixed $c \in \mathbb{C}$. Every pixel in the image represents a complex number $a + bi \in \mathbb{C}$ where $(a, b) \in [-2, 2] \times [-2, 2]$.



(a) $c = -1$ (b) $c = i$ (c) $c = -0.866 + 0.5i$ (d) $c = 0.5i$

Figure 1: Julia Set

Meanwhile, if we fix z and represent every pixel with different $ca + bi \in \mathbb{C}$ where $(a, b) \in [-2, 2] \times [-2, 2]$, another fancy fractal named Mandelbrot Set can be displayed.



(a) z=0 (b) z=0.5 (c) z=-0.5i (Love)

Figure 2: Mandelbrot Set

However, historically, the study of iteration of maps begins with the Newton's method. Cayley and Schröder independently studied Newton's method over the complex plane during the 1870s. In 1879 paper *"The Newton-Fourier Imaginary Problem"* by Sir Arthur Cayley[4], he applied the Newton-Fourier iterative method (also known as the Newton–Raphson method) to find the roots of

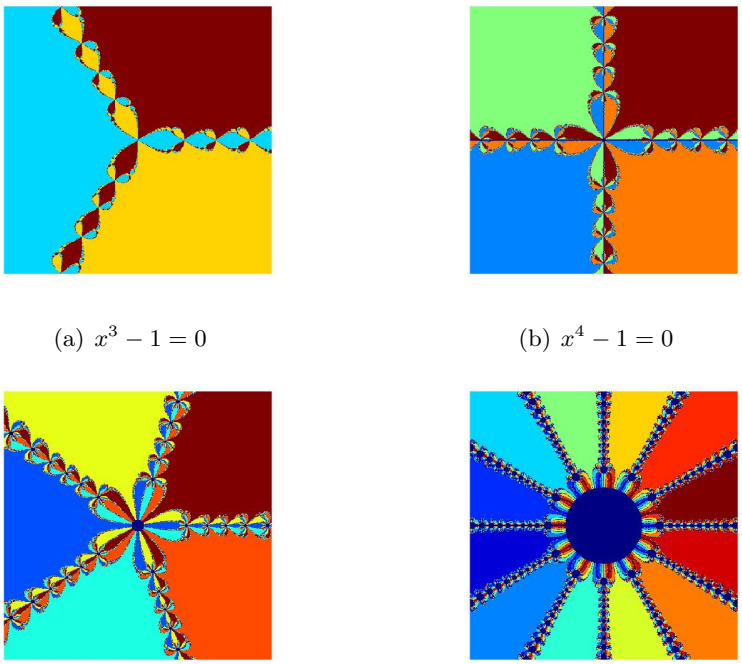
[4]Peitgen H, Jurgens H., and Saupe D., "Chaos and Fractals: New Frontiers of Science" Springer Verlag, 1992.

the equation $f(z) = z^3 + c = 0$. We can also plot out the Newton Fractals of different polynomial equations, which denotes the convergence regions of the initial points $x_0 \in \mathbb{C}$ to different roots on complex Plane.

The iteration rule simply reads,

$$f : x_{k+1} \leftarrow x_k - \frac{P(x_k)}{P'(x_k)}$$

Surprisingly, the convergence regions are not distinctly separated on the complex plane. There are crosses, twists and intersections between the convergence boundaries of each root.



(a) $x^3 - 1 = 0$ (b) $x^4 - 1 = 0$

(c) $x^5 - 1 = 0$ (d) $x^{12} - 1 = 0$

Figure 3: Newton Fractals

Intuitively, we can guess the degree of a polynomial determines the number of regions, which is trivially true because the degree of a polynomial determines the number of its roots.

2 Newton-Raphson Method and Jenkins-Traub Method [5]

Obviously, we may encounter problems with Newton-Raphson method in root-finding if the initial points are ill-chosen and fall in some unexpected regions (i.e the center blue circle of Figure 3(d)). Some global criterion for Newton-Raphson method calls into question. Otherwise, a new algorithm should be designed.

[5] https://en.wikipedia.org/wiki/Jenkins–Traub_algorithm

In 1968, Jenkins and Traub published a paper named *"A Three-Stage Variable-Shift Iteration for Polynomial Zeros and Its Relation to Generalized Rayleigh Iteration"*[6] and introduced a new three-stage process for calculating the zeros of a polynomial with complex coefficients. I will generally introduce this algorithm and give some insights into the convergence properties. Also, what will the Fractal of this algorithm look like? It's also an interesting question to think about.

First, we will clarify some basic definitions,

Definition 2.1. *Consider polynomial* $P(z) = \prod_{j=1}^{k}(z - a_j)^{m_j} \in \mathbb{C}[x]$*, and its derivative will be* $P'(z) = \sum_{j=1}^{k}(m_j P_j(z))$*, where* $P_j = P(z)/(z - a_j)$*.*

Definition 2.2. *Define a serie of auxiliary polynomials* $\{H^{(\lambda)}(z)|\lambda \in \mathbb{N}\}$*, which satisfies*

$$H^{(0)}(z) = P'(z)$$

and

$$H^{(\lambda+1)}(z) = \frac{1}{z - s_\lambda}[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}H^{(\lambda)}(z)]$$

And now we can show the general algorithm of Jenkins-Traub

Algorithm 1 Jenkins-Traub

procedure PHASE 1: NO-SHIFT PROCESS
  $H^{(0)}(z) \leftarrow P'(z)$
  for $\lambda = 1, 2, \cdots, M - 1$ **do** ▷ M=5 is empirically good.
    $H^{(\lambda+1)}(z) \leftarrow \frac{1}{z}[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(0)}{P(0)}H^{(\lambda)}(z)]$
procedure PHASE 2: FIXED-SHIFT PROCESS
  Find a fixed random s, *s.t.* $|s| \leq \min_{1 \leq i \leq n}(|a_i|)$.
  And this will converge to the closest root $|s - a_i| \leq |s - a_j| \forall 1 \leq j \leq n$.
  for $\lambda = M, M + 1, \cdots, L - 1$ **do** ▷ L is a sufficiently large number.
    $H^{(\lambda+1)}(z) \leftarrow \frac{1}{z - s}[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s)}{P(s)}H^{(\lambda)}(z)]$
procedure PHASE 3: VARIABLE-SHIFT PROCESS
  repeat
    $H^{(\lambda+1)}(z) \leftarrow \frac{1}{z - s_\lambda}[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}H^{(\lambda)}(z)]$
    $s_{\lambda+1} \leftarrow s_\lambda - \frac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)}$
  until $|P(s_{\lambda+1})| \leq \epsilon$
return $s_{\lambda+1}$

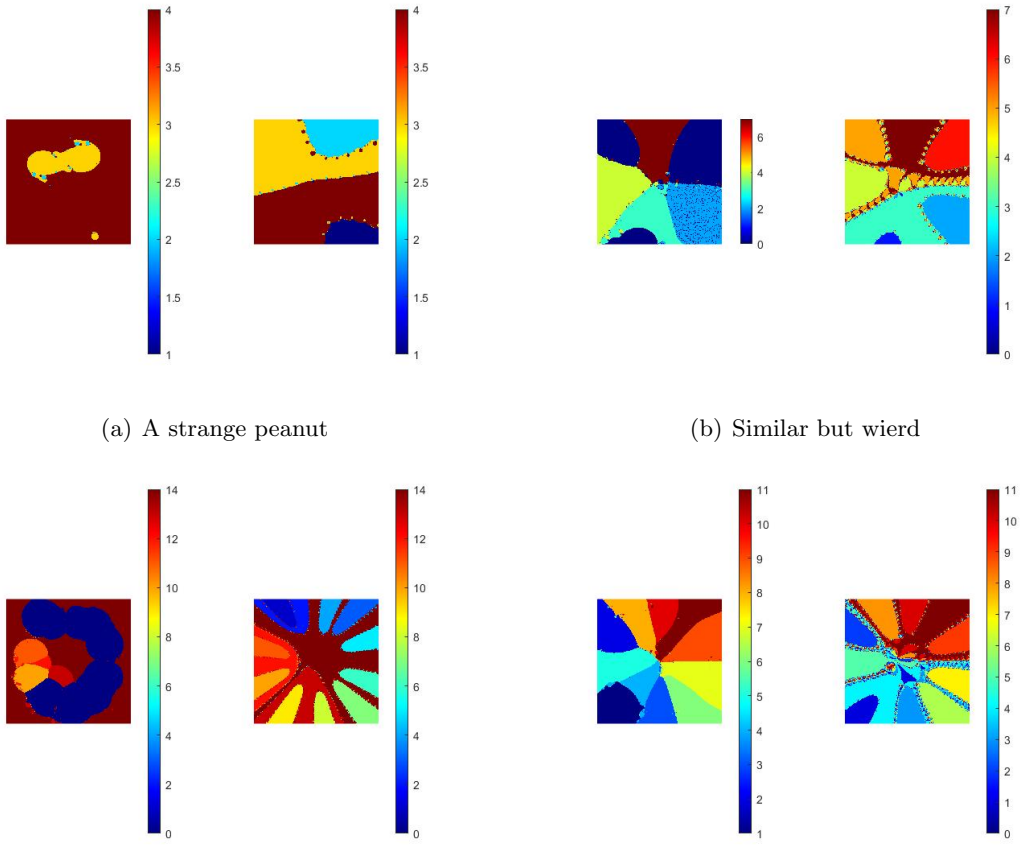
And $s_{\lambda+1}$ will be an arbitrary complex root dependent on the choice of our fixed shift s. Continue to factorize $(z - s_{\lambda+1})$ from $P(z)$ to get $P_{k-1}(z) = P_k(z)/(z - s_{\lambda+1})$. Repeat Jenkins-Traub until $deg(P) = 1$ so that all roots are found.

[6] M. A. Jenkins and J. F. Traub, A three-stage variable-shift iteration for polynomial zeros in relation to generalized Rayleigh iteration, Numer. Math. 14 (1970), 252-263.

Algorithm 2 Root-Finding

1: **repeat**
2: $root_i \leftarrow$Jenkins-Traub$(P_i(z))$
3: $P_{i-1}(z) \leftarrow P_i(z)/(z - root_i)$
4: **until** $deg(P(z)) = 1$
5: **return** *roots*

Before diving deep into convergence analysis, we may play with our new algorithm and plot out some random Fractals.



(a) A strange peanut

(b) Similar but wierd

(c) A baby sitter

(d) Master's painting

Figure 4: Newton Fractals

Most Fractals of Jenkins-Traub method look ridiculous and make no sense at all, or rather we may simply regard it as a random painting instead of Fractal.

To give an overall view of the algorithm performance, we first give some Lemma and Definitions.

Lemma 2.3. *Denote* $H^{(\lambda)}(z) = \sum_{j=1}^{k} c_j^{(\lambda)} P_j(z)$, *and* $c^{(\lambda+1)} = \dfrac{m_j}{\prod_{t=0}^{\lambda}(s_t - a_j)}$

Proof. Use the iteration form,

$$\begin{aligned}
H^{(\lambda+1)}(z) &= \frac{1}{z-s_\lambda}[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}H^{(\lambda)}(z)] \\
&= \frac{P(z)}{z-s_\lambda}[\sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{z-a_j} - \sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{s_\lambda-a_j}] \\
&= \sum_{j=1}^{k}\frac{c_j^{(\lambda)}P(z)}{(z-a_j)(s_\lambda-a_j)} \\
&= \sum_{j=1}^{k}\frac{c_j^{(\lambda)}P_j(z)}{s_\lambda-a_j} = \sum_{j=1}^{k}c_j^{(\lambda+1)}P_j(z)
\end{aligned}$$

So we have,

$$c^{(\lambda+1)} = \frac{c_j^{(\lambda)}}{s_\lambda-a_j} = \cdots = \frac{m_j}{\prod_{t=0}^{\lambda}(s_t-a_j)}$$

□

Definition 2.4. *We denote* $a_1 = \arg\min_j(|a_j - s|)$, $r_j^{(\lambda)} = \dfrac{s_\lambda - a_1}{s_\lambda - a_j}$, $d_j^{(\lambda)} = \dfrac{c_j^{(\lambda)}}{c_1^{(\lambda)}}$, $T_\lambda = |\dfrac{s_{\lambda+1}-a_1}{s_\lambda-a_1}|$.

And finally here gives a simplified Theorem of convergence,

Theorem 2.5. *Given that s is chosen correctly in the algorithm and L is sufficiently large, if $D_L = \sum_{j=2}^{n}|d^{(L)}| < \dfrac{1}{3}$, then we will have $s_\lambda \to a_1$ when $\lambda \to \infty$*

Proof. Since $A = \prod_{k=0}^{\lambda}T_k = \dfrac{|s_\lambda - a_1|}{|s_0 - a_1|}$, if $T_\lambda < 1$, $\forall\lambda \geq L$, we will have $|s_\lambda - a_1| \to 0$. Next we will prove $T_\lambda < 1$ using given conditions.

$$\begin{aligned}
\frac{s_{\lambda+1}-a_1}{s_\lambda-a_1} &= \frac{s_\lambda - \dfrac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)} - a_1}{s_\lambda - a_1} && (1)\\
&= 1 - \frac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)(s_\lambda-a_1)} && (2)\\
&= 1 - \frac{P(s_\lambda)\sum_{i=1}^{n}c_j^{(\lambda)}/(s_\lambda-a_j)}{P(s_\lambda)\sum_{i=1}^{n}c_j^{(\lambda)}(s_\lambda-a_1)/(s_\lambda-a_j)^2} && (3)\\
&= 1 - \frac{1+\sum_{j=2}^{n}d_j^{(\lambda)}r_j^{(\lambda)}}{1+\sum_{j=2}^{n}d_j^{(\lambda)}[r_j^{(\lambda)}]^2} && (4)\\
&= \frac{\sum_{j=2}^{n}d_j^{(\lambda)}[r_j^{(\lambda)}]^2 - \sum_{j=2}^{n}d_j^{(\lambda)}r_j^{(\lambda)}}{1+\sum_{j=2}^{n}d_j^{(\lambda)}[r_j^{(\lambda)}]^2} && (5)
\end{aligned}$$

Since $|r^{(L)}| = |\frac{s_\lambda - a_1}{s_\lambda - a_j}| < 1$

$$T_\lambda < \frac{|\sum_{j=2}^{n} d_j^{(\lambda)}| + |\sum_{j=2}^{n} d_j^{(\lambda)}|}{1 + |\sum_{j=2}^{n} d_j^{(\lambda)}|}$$
$$= \frac{2D_\lambda}{1 + D_\lambda} < 1$$

□

Plot out several comparison of convergence between Newton-Raphson method and Jenkins-Traub method, we can't observe as what Jenkins and Traub asserts that there method outperforms traditional Newton-Raphson method. However, they grant a convergence of any solution with polynomial degree up to infinity.

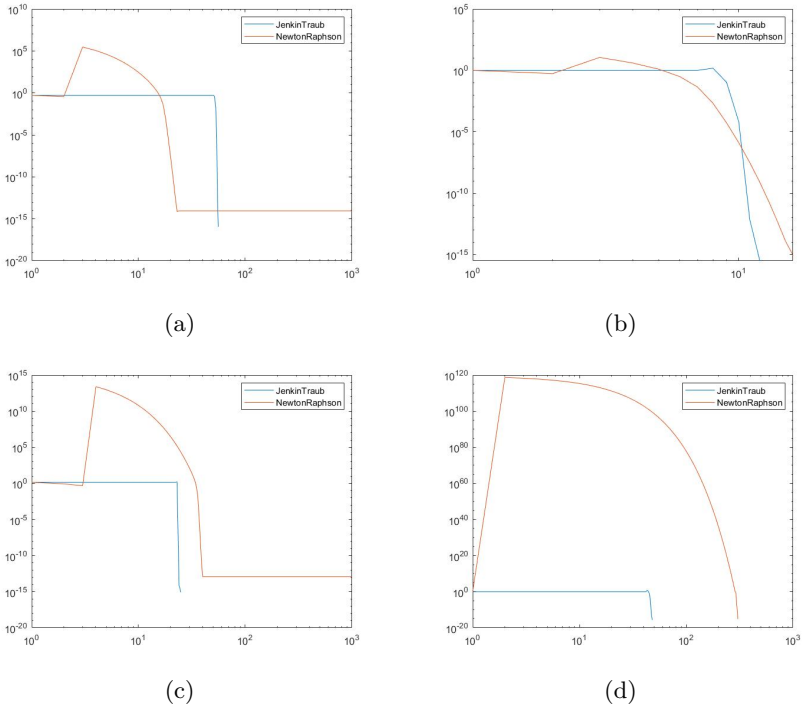


Figure 5: Convergence

This is just a glimpse of some convergence analysis. Thanks to the strong capability of Matlab to conduct complex calculations with really large numbers, we can generally observe the convergence of Newton-Raphson-method (and sometimes it outperforms Jenkins-Traub method). However, if we conduct experiment on C, which is incapable of calculations with large float numbers, Newton-Raphson method will give an error instead of finding a root.

It is worth-noted that the mathematical software Mathematica also apply this Jenkins-Traub method for calculating the zeros of a polynomial with complex coefficients. [7] This method, to some extent, is the state-of-art.

[7] Wankere R. Mekwi, Iterative Methods for Roots of Polynomials, Exeter College, University of Oxford, 2001.

3 Further Discussions and the Companion Matrix

The above polynomial roots problem can be alternatively turned into the eigenvalues problem of the companion matrix. A polynomial equation $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$ has the following plug-in to a shifted-QR algorithm on the following matrix,

$$\begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -\frac{a_0}{a_n} & -\frac{a_1}{a_n} & \cdots & -\frac{a_{n-2}}{a_n} & -\frac{a_{n-1}}{a_n} \end{pmatrix}$$

Dekker and Traub[8] came up with a shifted QR algorithm for calculating the eigenvalues of the Hermitan matrices. The shifts may be viewed as Newton-Raphson iteration on a sequence of rational functions converging to a first degree polynomial.

Also, all stages of the Jenkins–Traub complex algorithm may be represented as a linear algebra problem of determining the eigenvalues of the above companion matrix. That is an explanation from the higher perspective, but we can easily deduce why the shift s appears in the construction of $H_k(z)$. The three variants of no shift, constant shift and generalized Rayleigh shift directly matches with the three stages of the algorithm.

[8]Dekker, T. J. and Traub, J. F. (1971), The shifted QR algorithm for Hermitian matrices, Lin. Algebra Appl., 4(2), 137–154.