Julia Sets

Tyler Klein Ben Zager Jack O'Shea

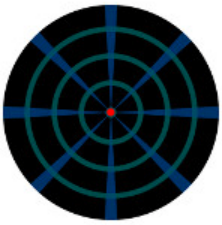
November 8, 2016

Introduction

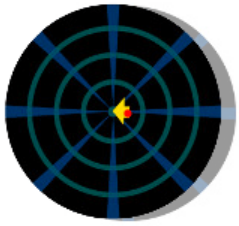
Julia sets are sets of complex numbers that do not converge to a point when iterated upon by a given map. When computed and plotted using various algorithms, these sets often produce very beautiful images, sometimes fractals, in the complex plane. The case most often studied is functions of the form $f(Z) = Z^2 + C$, where Cis any number in the complex plane. In our project, we studied two function groups: functions of the form $f(Z) = Z^n + C$, and variations of Newtons method.

Intuitive Explanation for Fractal Formation

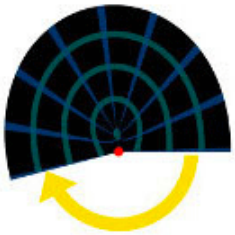
Computer researcher Karl Sims offers an extremely intuitive explanation of why fractals in the complex plane result from these iterative functions. The following explanation and images are adapted from his work at: http://www.karlsims.com/julia.html Instead of using the equation $z = z^2 + c$, we'll transform it by subtracting c from both sides and taking the square root. Now our equation looks like $z = \pm\sqrt{z - c}$ and we can apply this to a set of points in the complex plane to generate the shape of the Julia set. We will use the disk of radius 2 centered at the origin of the complex plane as our set. Note the red dot represents the origin and it is assumed the reals lay along the horizontal axis and the imaginaries on the vertical.



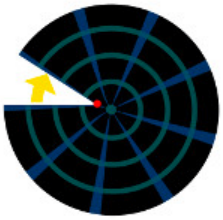
We begin to carry out the transformation $z = \pm\sqrt{z - c}$ by first applying the translation transformation inside the square root function, subtracting the complex number c from each point in the set. Using a c value of $.274 - .008i$ means we will shift each point on the disk .274 units left (along the real axis), and only slightly up (0.008 units along the imaginary axis)

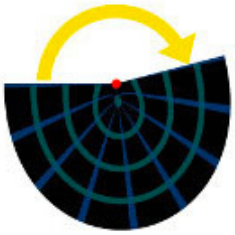


The next step is applying the square root function to the set. The first effect of this transformation is halving the angle between all points and the positive real axis

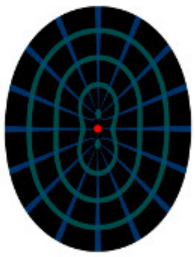


The square root function produces positive and negative results, but the figures above only shows the positive result. The following transformation also occurs and produces the negative half of the set



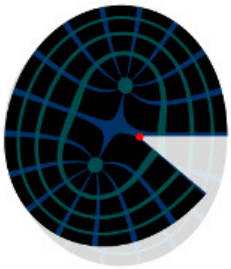


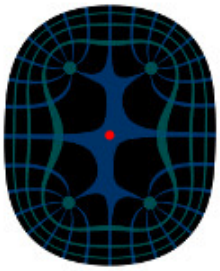
These two resulting sets are merged to form



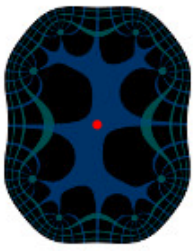
The square root function also halves the distance between any point in the complex plane and the origin, so the final step in this iteration is to shrink the above set by a factor of 2.

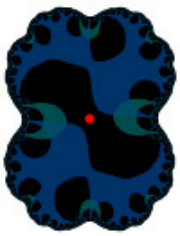
The first iteration is now complete, and we restart the process beginning with the figure above instead of the disk with radius 2

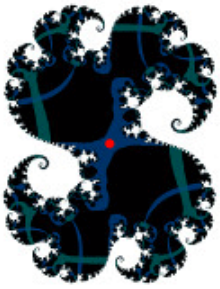




The following figures are the results of 3, 6, 30, and 100 iterations, respectively







You may notice that the function used to produce these graphical representations of the Julia Set for $f(z)$ is actually the inverse of f. This process certainly differs from the typical algorithm used to generate the figures, but the end result is the same. Let **J** denote the Julia Set of our function, C denote the complex plane, and D denote the disk of radius 2 in the complex plane. Typically

we generate **J** by applying the map $f: C \to C$ such that $f(z) = \lim_{z\to\infty} z_k$ where $z_k = z_{k-1}^2 + c$ to some subset of C that we guess contains **J**, (lets call this C). We begin mapping each $z \in C'$ and if $f(z) \in D$ we say $z \in \mathbf{J}$. The domain and codomain of this algorithm is denoted C because as we test more points of C, we can use our results to eliminate other points of C without explicit testing them. As our algorithm progresses, the domain tends towards **J**. With the inverse function, however, we can properly state the domain and codomain from the start: $f^{-1} : D \to J$ such that $f^{-1}(z) = \lim_{k\to\infty} z_k$ where $z_k = \sqrt{z_{k-1} - c}$. The image of $f^{-1}(D)$ is clearly **J** because for any $z \in D$, $f(f^{-1}(z)) = z$ and $z \in D$ = the disk of radius 2 centered at the origin, which is exactly how we define an element of a Julia Set. What is this explanation useful for? It can help us predict the shape of some Julia Sets before creating them, and have a better understanding of why certain c value result in fractals with certain symmetries. When $c = 0$, all points diverge except for the origin. The process above offers another method for understanding why this is true. Take the disk of radius 2 and simply rotate it in place, then shrink it around the origin. Repeating this process infinite times yields the Julia Set, a disk with an infinitely small radius, that only contains the origin. When the imaginary component of c is 0, there will be no vertical translation of D before the rotation, so the Julia set will be symmetric about the horizontal or real axis.

Simple Polynomials

For polynomials of the form $f(Z) = Z^n + C$, the function was applied repeatedly to a grid of points in the complex plane centered at the origin. Before each iteration, each point was checked to see if it was outside of the escape radius. The escape radius is a radius beyond which points cannot possibly converge to a root. This is due to the fact that beyond the radius, each iteration takes the point further away from the origin, without bound. As a result, we know that all points outside of this radius wont converge. The iteration at which points cross the radius is stored and used for plotting. The escape radius for these functions can be derived quite easily, as shown below[1]. Let $f(z) = z^n + C$

$$
\begin{aligned}
|f(z)| &= |z^n + C| \\
\frac{|f(z)|}{|z|} &= \frac{|z^n + C|}{|z|} \\
|z^n + C| &\geq |z^n| - |C| && \text{via triangle theorem} \\
\frac{|f(z)|}{|z|} &\geq \frac{|z^n| - |C|}{|z|}
\end{aligned}
$$

[1] `http://mrob.com/pub/muency/escaperadius.html`

$$\frac{|f(z)|}{|z|} \geq \frac{|z|^n - |C|}{|z|} \quad \text{via Proof 1}$$

$$\frac{|f(z)|}{|z|} \geq |z|^{n-1} - \frac{|C|}{|z|}$$

$$|z|^{n-1} - \frac{|C|}{|z|} \geq |z|^{n-1} - 1 \quad \text{when} \quad |z| \geq |C|$$

$$|z|^{n-1} - 1 \geq 1 \quad \text{when} \quad |z|^{n-1} \geq |2|$$

$$\therefore \frac{|f(z)|}{|z|} \geq 1 \quad \text{when} \quad |z| \geq \sqrt[n-1]{2} \geq |C|$$

As a result, the point grows rapidly when $|z| \geq \sqrt[n-1]{2}$, so we know all points that lead to an iteration that has a magnitude above $\sqrt[n-1]{2}$, the sequence diverges.

Iteration Grid Code

Our code works by first creating an equally spaced grid of points using the `meshgrid` command in Matlab. We declare a center point, the origin, the distance and the step size, and use meshgrid to create two matrices that can be added together to create a grid.

```
x=linspace(xcen-d,xcen+d,points);
y=linspace(ycen-d,ycen+d,points);
[x,y]=meshgrid(x,y); %creates 2 matrices
z = x+1i.*y;
```

The escape radius is then calculated, and the for loop is executed.

```
escapeRadius = nthroot(2,powerN-1);
for iter=0:iters
        escaped = abs(z)>escapeRadius; % particles that
            "escaped"
        z(escaped) = NaN; % prevent further iterations
        kGrid(escaped) = iter;
        z=func(z); % apply mapping
end
```

Inside the for loop, the function first checks if any points have escaped, and if so, it denotes the iteration after which the point escaped. Then, the points are set to NaN, so the program ignores them for future calculations. All of the points outside the escape radius automatically have a value of 0, since they are escaped before any iteration takes place. The function then applies the mapping for the next iteration, then repeats. The program then returns `kGrid`, which is the grid of the

iterations after which each point escape (if it didn't escape, it's value is 0). These values will be refered to as the 'K-values' from now on.

Plotting Code

To plot the K-values, we created a function that takes the grid of K-values, and generates an image using `imagesc`. This command generates an image from a matrix of values using scaled colors. Since our grid has only 1 number, we use a colormap to ensure we have interesting color schemes. The script includes some auto-naming of plots, saving capabilities, and other small features. Here are some images generated using the two functions in combination.

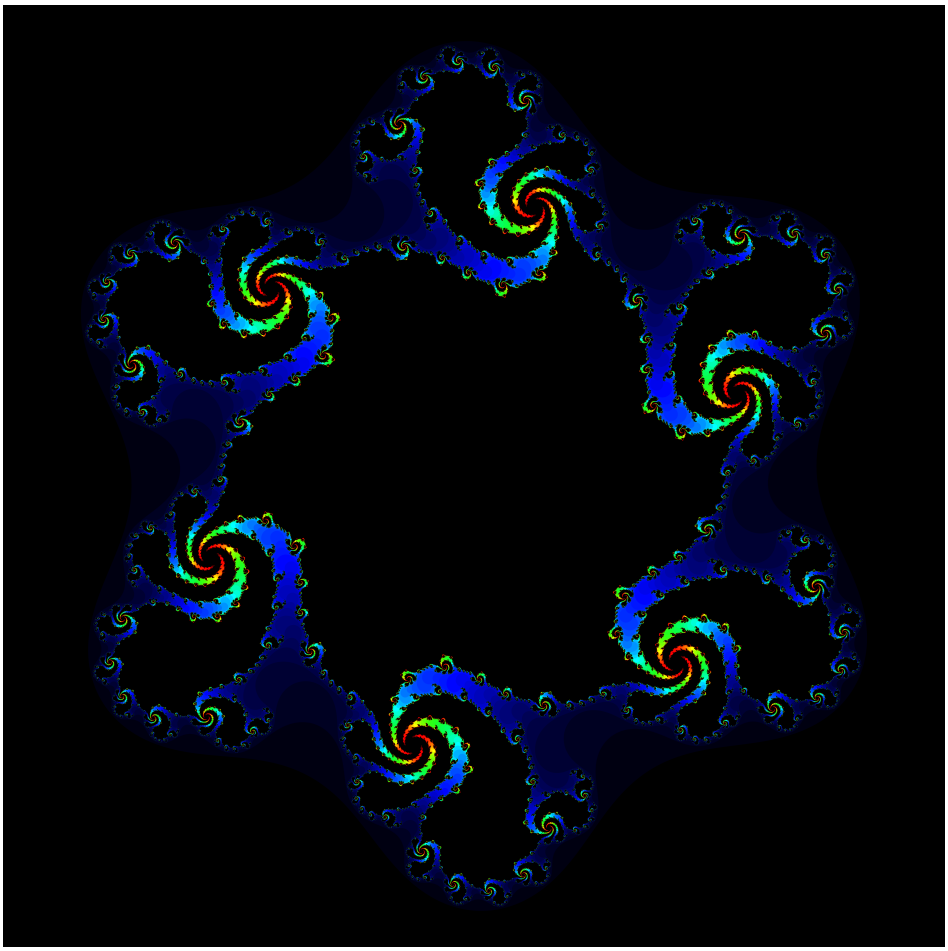


Figure 3: Fractal for C=.8i

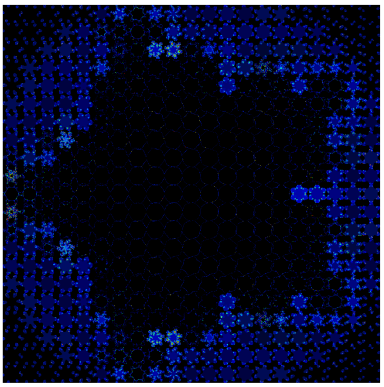


Figure 4: Sweep of C values

Newtons Method for Generating Julia Sets

Newtons method is a root finding technique that is a special case of fixed point iteration, in the form of $x_{k+1} = x_k - \frac{f(x)}{f'(x)}$, with an initial approximation, $x0$. Newtons method can converge quadratically with an initial guess that is close enough, as long as the derivative at the root is not zero. By applying Newtons method to functions of the form $f(z) = z^n + c$, on a set of points in the complex plane, we can generate a special case of Julia sets called Newton fractals. As we will see, the complicated dynamics of Newton fractals provides some insight on the convergence of Newtons method, and how it can be affected by initial guesses and roots of higher multiplicity. There are two main methods for creating Newton fractals. The first method maps a starting point to the root where it converges. The other method maps a starting point to the number of iterations required for convergence, up to some given tolerance. We will start by discussing the first method.

Mapping by root

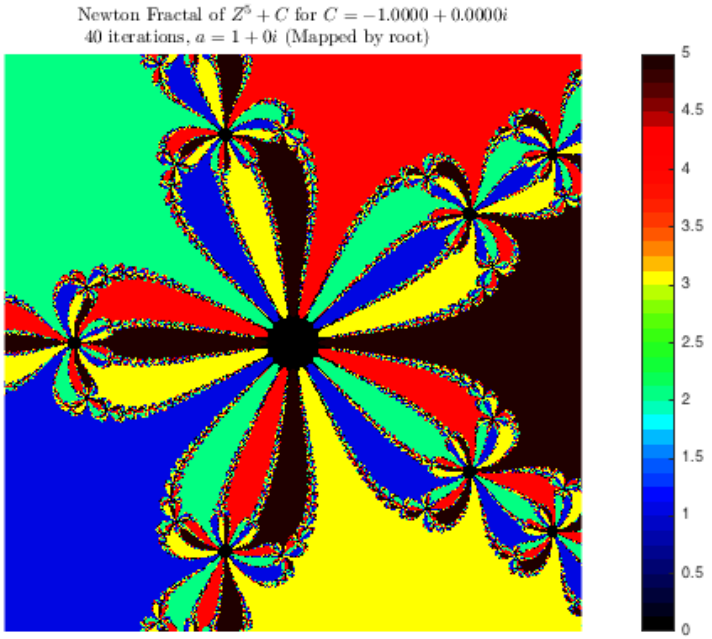


Figure 5

To map by root, we applied Newtons method for a given complex polynomial function to a set of points in the complex plane. After a set number of iterations, we find the root where each point converges, by checking if the distance from the point to the root is less than some given tolerance. After each point is labeled with a root, those roots are then mapped to a color, to form an image like the one seen above. Each colored region depicts a basins of attraction, which is a set of points which all converge to the same root. The boundaries of these regions are what define the fractal, and those points are part of the Julia set. (more to come)

Mapping by iteration

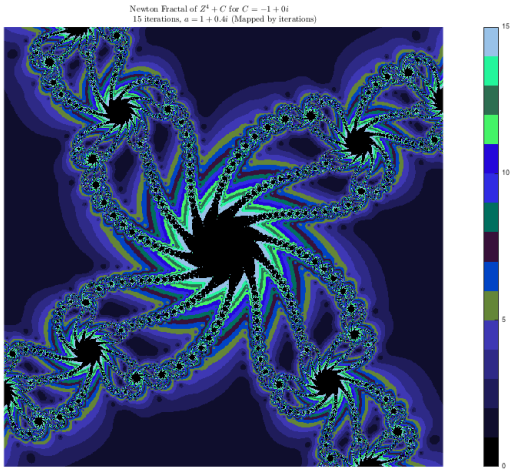


Figure 6

To map by iteration, we proceed with Newtons method. After each iteration, we check if the distance moved by each point is less than a given tolerance, implying that it has converged. If so, that point is labeled with the current iteration count, and removed from the set for all following iterations. (more to come)

Appendix

Proof 1: Powers of Complex Numbers

$$\text{Let } z = a + bi \qquad |z| = r$$

Since z is complex, it can be written as $z = r(\cos\theta + i\sin\theta)$ for some θ

$$e^{ix} = \cos x + i\sin x$$

$$(e^{ix})^n = (\cos x + i\sin x)^n = e^{nix} = \cos nx + i\sin nx$$

$$z^n = r^n(\cos\theta + i\sin\theta)^n$$

$$z^n = r^n(\cos n\theta + i\sin n\theta)$$

$$|z^n| = |r^n(\cos n\theta + i\sin n\theta)|$$

$$= r^n|\cos n\theta + i\sin n\theta|$$

$$= r^n \qquad \text{via Pythagorean Theorem}$$

$$\therefore \quad |z^n| = |z|^n$$