NEWTON'S METHOD AND FRACTALS

AARON BURTON

ABSTRACT. In this paper Newton's method is derived, the general speed of convergence of the method is shown to be quadratic, the basins of attraction of Newton's method are described, and finally the method is generalized to the complex plane.

1. SOLVING THE EQUATION $f(x) = 0$

Given a function f, finding the solutions of the equation $f(x) = 0$ is one of the oldest mathematical problems. General methods to find the roots of $f(x) = 0$ when $f(x)$ is a polynomial of degree one or two have been known since 2000 B.C. [3]. For example, to solve for the roots of a quadratic function $ax^2 + bx + c = 0$ we may utilize the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Methods to solve polynomials of degree three and four were discovered in the 16th century by the mathematicians dal Ferro, Tartaglia, Cardano, and Ferrai [3]. Finally, in 1826 it was discovered from work done by Abel, that there is no general method to solve polynomials of degree five or greater [3].

Since it is not possible to solve all equations of the form $f(x) = 0$ exactly, an efficient method of approximating solutions is useful. The algorithm discussed in this paper was discovered by Sir Issac Newton, who formulated the result in 1669. Later improved by Joseph Raphson in 1690, the algorithm is presently known as the Newton-Raphson method, or more commonly Newton's method [3].

Newton's method involves choosing an initial guess x_0, and then, through an iterative process, finding a sequence of numbers $x_0, x_1, x_2, x_3, \cdots$ [1] that converge to a solution. Some functions may have several roots. Later we see that the root which Newton's method converges to depends on the initial guess x_0. The behavior of Newton's method, or the pattern of which initial guesses lead to which zeros, can be interesting even for polynomials. When generalized to the complex plane, Newton's method leads to beautiful pictures.

In this paper, we derive Newton's method, analyze the method's speed of convergence, and explore the basins of attraction[2]. Finally, we extend Newton's method to the complex plane, and through the aid of computer programming view the complex basins of attraction for several polynomials.

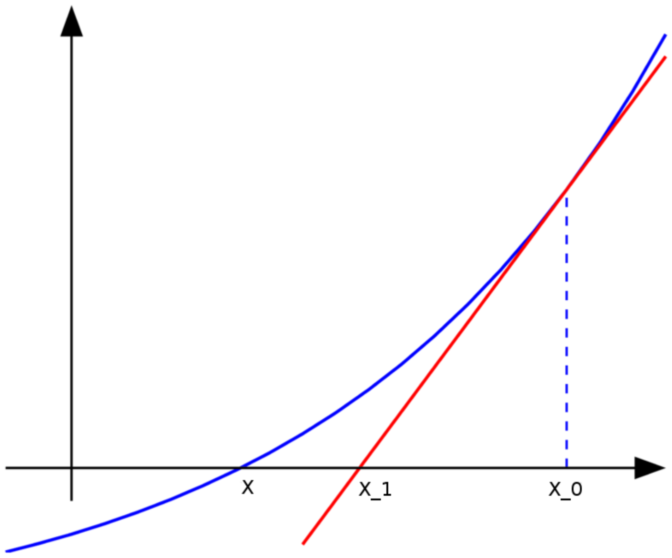

FIGURE 1. The geometry of Newton's method

2. DERIVING NEWTON'S METHOD

As stated in Section 1, Newton's method involves choosing an initial approximation x_0, and then, through an iterative process, finding a sequence of numbers x_0, x_1, x_2, x_3, $\cdots$ that converges to a solution. Recall our goal is to approximate the root of a function $f(x)$, thus once we chose our x_0 we hope to find a point x_1 (related to x_0 in some way) that is a better approximation for the root of the function $f(x)$. Given a single point x_0 there are many ways in which we could proceed to find the point x_1. In this paper we will use the tangent line at x_0. The tangent line provides the best linear approximation to the function $f(x)$ at the point x_0, thus we are implicitly assuming that the tangent line will intersect the x-axis near the desired root. This assumption seems to be valid based on Figure 1. In Section 2 we will discuss how this assumption breaks down under certain circumstances.

Now, suppose $f(x)$ is a differentiable function on an interval $[a, b]$ for which we wish to approximate the root. We begin by making a guess or estimate of the root's location, that is specifying an initial point $(x_0, 0)$. To determine our next estimate $(x_1, 0)$, we draw the tangent line through $(x_0, f(x_0))$. The point at which the tangent line intersects the x axis is $(x_1, 0)$. Using our initial guess x_0, the value of the function $f(x_0)$ and the slope of the tangent line $f'(x_0)$ we can then find the equation of the tangent line at $(x_0, f(x_0))$ using the point-slope formula:

$$y - f(x_0) = f'(x_0)(x - x_0).$$

To solve for the x intercept we set $y = 0$ and rearrange terms to get

$$-f(x_0) = f'(x_0)(x - x_0)$$

[1]Often called the orbit of x_0.

[2]Formally defined in Section 6, a Basin of Attraction is the set of points which converge to a particular root.

$$x - x_0 = \frac{-f(x_0)}{f'(x_0)}$$

and finally

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

The x intercept is our new guess, or estimate, x_1. Thus we have, $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. To find x_2, we begin the whole process over again. However, this time we begin with the point $(x_1, 0)$ and solve for the point $(x_2, 0)$. Repeating this algorithm generates a sequence of x values x_0, x_1, x_2, $\cdots$ by the rule

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \;\; f'(x_n) \neq 0$$

we define as the Newton iteration function $N(x)$. Formally, given a differentiable function f, the Newton function for f is:

$$N(x) = x - \frac{f(x)}{f'(x)}. \tag{1}$$

Then

$$x_1 = N(x_0)$$

$$x_2 = N(x_1) = N(N(x_0)) = N^2(x_0)$$

and, in general,

$$x_n = N^n(x_0),$$

where the notation N^n means N applied n times.

3. Where Newton's method fails

One natural question that arises is whether Newton's method will always converge to a root.

3.1. Initial guess is a critical point of $f(x)$.

Recall from equation (1) that the definition of the Newton iteration function is

$$N(x) = x - \frac{f(x)}{f'(x)}.$$

From this definition we see that $N(x)$ will not exist if $f'(x) = 0$. If we chose an initial point where $f'(x) = 0$, then Newton's method will fail to converge to a root. Similarly if $f'(x_n) = 0$ for some iteration x_n, then Newton's method will also fail to converge to a root. The former case is illustrated for $f(x) = x^3 + 1$ in Figure 2. If we happen to choose our initial guess as $x = 0$, Newton's method fails to converge since the tangent line at $x = 0$ never intersects the x axis.

3.2. No root to find.

Another way in which Newton's method will fail to converge to the root of a function is if there is no root. Consider the graph of $f(x) = x^2 + 1$ in Figure 3. The function $f(x) = x^2 + 1$ never crosses the x axis, and thus there is no possible solution. If we choose an initial guess, it can be proved that Newton's method will chaotically move around the x axis.[3]

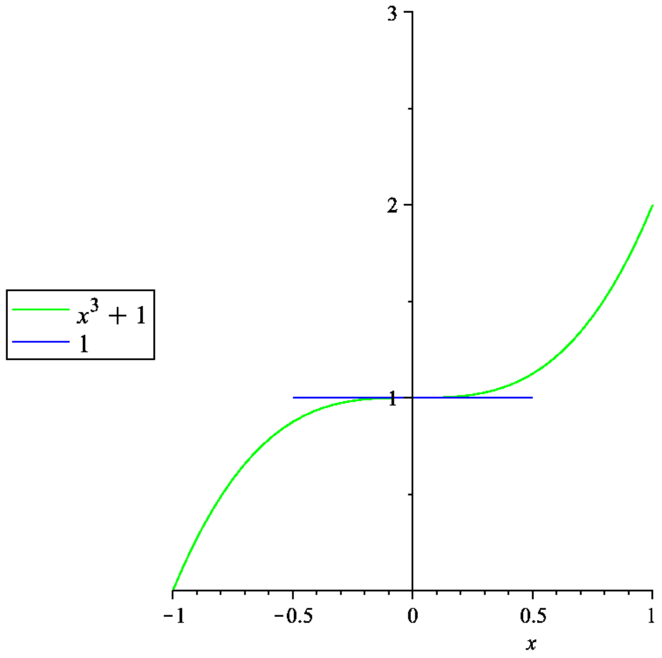

FIGURE 2. $f(x) = x^3 + 1$. The initial guess coincides with a critical point.

3.3. **Periodic cycle.** A third way in which Newton's method will fail to converge is if the initial guess or an iteration coincides with a cycle. For example, consider $f(x) = x^3 - 2x + 2$ and the initial guess of $x_0 = 1$ as shown in Figure 4. With $x_0 = 1$ we see that

$$x_1 = N(x_0) = 1 - \frac{1^3 - 2(1) + 2}{3(1)^2 - 2} = 1 - 1 = 0,$$

and then

$$x_2 = N(x_1) = 0 - \frac{0^3 - 2(0) + 2}{3(0)^2 - 2} = 0 - (-1) = 1.$$

This example is of a cycle with period two, but cycles of other orders may exist as well.[4]

Often the problems just described can be avoided by choosing our initial point wisely and by looking at the derivatives of the function to be approximated. Usually it is helpful to graph the function $f(x)$ if possible before using Newton's method.

[3]For further details, refer to Devaney Chapter 13.2 example four.

[4]See Devaney Chapter 3.3 for an in depth explanation of cycles.

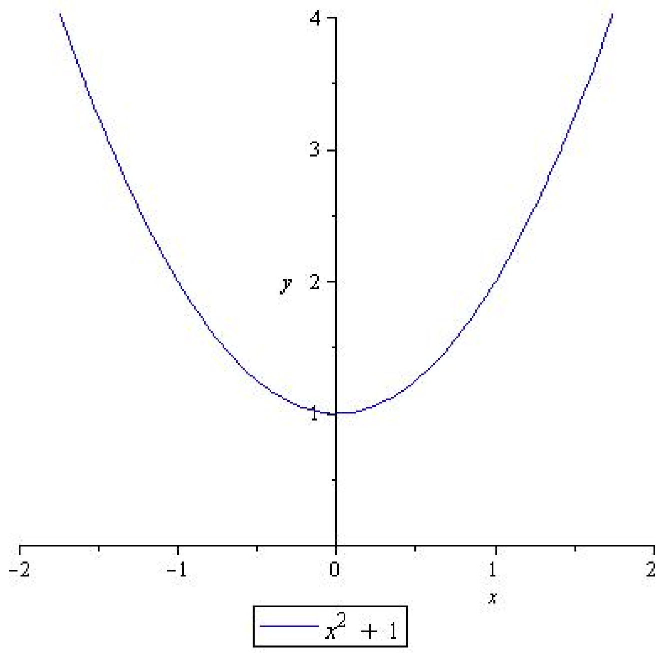

FIGURE 3. $f(x) = x^2 + 1$. Nonexistent root.

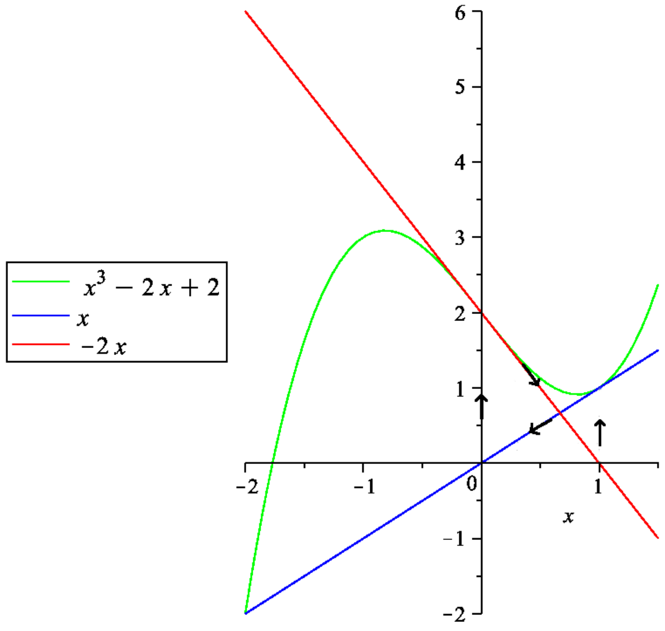

FIGURE 4. $f(x) = x^3 - 2x + 2$. A cycle of period 2.

4. Convergence

A natural extension of Section 3 is the question of convergence. When exactly can we be sure Newton's method will converge to a root? First a few background definitions and a lemma.

Definition 4.1. *A root r of the equation $f(x) = 0$ has* ***multiplicity*** *k if $f(r) = 0$, $f'(r) = 0, \cdots, f^{[k-1]}(r) = 0$ but $f^{[k]}(r) \neq 0$. Here $f^{[k]}(r)$ is the k^{th} derivative of f.*

For example, 0 is a root of multiplicity 2 for $f(x) = x^2 + x^3$ and of multiplicity 1 for $f(x) = x + x^3$.

Definition 4.2. *A point x_0 is a* ***fixed point*** *of a function $f(x)$ if and only if $f(x_0) = x_0$. Moreover, the point x_0 is called an* ***attracting fixed point*** *if $|f'(x_0)| < 1$.*

For our purposes it suffices for the reader to note that if a root is an attracting fixed point of the function $N(x)$, then Newton's method will converge to that point. For more information about fixed and attracting fixed points refer Appendix A.

Lemma 4.1. *If r is a root of multiplicity k for a function $f(x)$, then $f(x)$ may be written in the form*

$$f(x) = (x - r)^k G(x), \text{ where } G(r) \neq 0$$

Proof. Consider the Taylor expansion of a function $f(x)$ centered about the root r.

$$f(x) = f(r) + f'(r)(x - r) + \frac{f''(r)}{2!}(x - r)^2 + \frac{f'''(r)}{3!}(x - r)^3 + \cdots$$

Now suppose that the root r has a multiplicity k. From the definition of multiplicity we have,

$$\begin{aligned} f(x) &= 0 + 0(x - r) + \frac{0}{2!}(x - r)^2 + \cdots + \frac{f^{\{k\}}(r)}{k!}(x - r)^k + HOT \\ &= \frac{f^{\{k\}}(r)}{k!}(x - r)^k + HOT \end{aligned}$$

where HOT stands for higher order terms. From each higher order term we may factor out $(x - r)^k$, so we have

$$f(x) = (x - r)^k G(x)$$

where $G(x) = \frac{f^{\{k\}}(r)}{k!} + HOT$ and $G(x)$ is a function that has no root at r, that is $G(r) \neq 0$. If f is a polynomial, then the multiplicity of any root is always finite. □

4.1. Newton's Fixed Point Theorem.

Now we are ready to prove Newton's method does in fact converge to the roots of a given $f(x)$.

Newton's Fixed Point Theorem 4.2. *Suppose f is a function and N is its associated Newton Iteration function. Then r is a root of f of multiplicity $k > 0$ if and only if r is a fixed point of N. Moreover, such a fixed point is always attracting.*

Proof. Suppose that $f(r) = 0$ but $f'(r) \neq 0$, that is, the root r has multiplicity 1.[5] Then from the definition $N(x) = x - \frac{f(x)}{f'(x)}$, we have $N(r) = r$. Thus, r is a fixed point of N. Conversely, if $N(r) = r$ we must also have $f(r) = 0$.

[5]Commonly known as a simple root.

To see that r is an attracting fixed point, we use the quotient rule to compute

$$N'(x) = \frac{f(x)f''(x)}{(f'(x))^2}. \tag{2}$$

Again assuming $f(r) = 0$ and $f'(r) \neq 0$, we see that $N'(r) = 0$. Since $N'(r) < 1$, r is an attracting fixed point by definition. This proves the theorem subject to assumption that $f'(r) \neq 0$.

If $f'(r) = 0$, then suppose that the root has multiplicity $k > 1$ so that the $(k-1)$th derivative of f vanishes at r but the kth does not. Thus we may write

$$f(x) = (x-r)^k G(x)$$

where G is a function that satisfies $G(r) \neq 0$.[6] Then we have

$$f'(x) = k(x-r)^{k-1}G(x) + (x-r)^k G'(x)$$

$$f''(x) = k(k-1)(x-r)^{k-2}G(x) + 2k(x-r)^{k-1}G'(x) + (x-r)^k G''(x).$$

Therefore, after some cancellation, we have

$$N(x) = x - \frac{(x-r)G(x)}{kG(x) + (x-r)G'(x)}.$$

Hence $N(r) = r$, which shows that the roots of f correspond to fixed points of N when r is a root of multiplicity $k > 1$. Finally we compute

$$N'(x) = \frac{k(k-1)G(x)^2 + 2k(x-r)G(x)G'(x) + (x-r)^2G(x)G''(x)}{k^2G(x)^2 + 2k(x-r)G(x)G'(x) + (x-r)^2G'(x)^2}$$

(note $(x-r)^{2k-2}$ has been factored out of both the numerator and denominator). Now $G(r) \neq 0$, so

$$N'(r) = \frac{k-1}{k} < 1.$$

Thus, r is an attracting fixed point for N. □

To reiterate, Newton's Fixed Point Theorem tells us that the fixed points of the function $N(x)$ are the roots of $f(x)$. Furthermore, because the roots of $f(x)$ are attracting fixed points for $N(x)$, as we iterate $N(x)$ the resulting sequence of points, $x_0, x_1 = N(x_0), x_2 = N(x_1), \cdots$, will converge to the root of $f(x)$.

5. An example of Newton's method

As an example of Newton's method, we approximate the solution to

$$f(x) = x^3 - x - 1 = 0.$$

By examining Figure 5 it is clear that there is a exactly one root between 1 and 2. The Newton function for $f(x)$ is:

$$N(x) = x - \frac{x^3 - x - 1}{3x^2 - 1} = \frac{2x^3 + 1}{3x^2 - 1}.$$

Starting with the initial guess x_0=1, the results of the iteration of the Newton function are:

[6] In the case of $f'(r) = 0$, r is not in the domain of the corresponding function $N(x)$; at r there exists a removeable discontinuity. When we rewrite f using Lemma 4.1 we have removed the discontinuity at r. This subtle difference between the two functions $f(x)$ is of little consequence but is mentioned here for the sake of completeness.

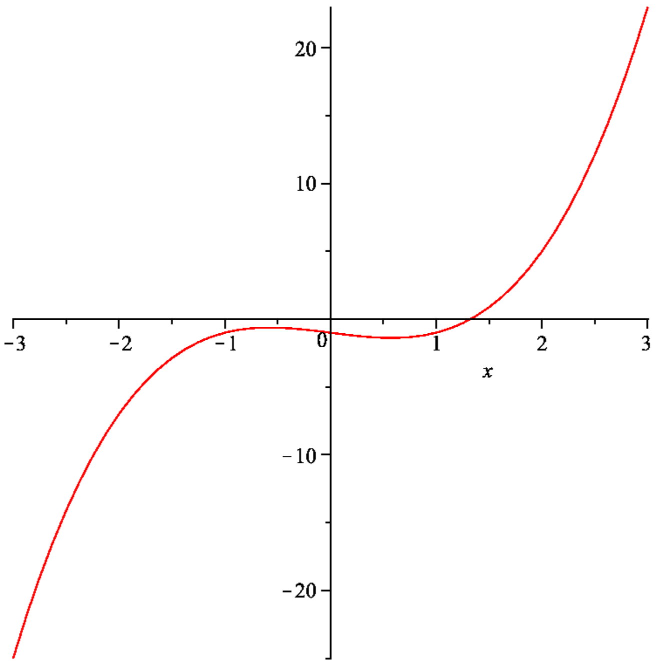

FIGURE 5. Graph of $x^3 - x - 1$

$$\begin{aligned}
x_0 &= 1\\
x_1 &= \mathbf{1}.5\\
x_2 &= \mathbf{1}.34\cdots\\
x_3 &= \mathbf{1.3}252\cdots\\
x_4 &= \mathbf{1.32}47181\cdots\\
x_5 &= \mathbf{1.32471}7957244789\cdots\\
x_6 &= \mathbf{1.3247179572447}4602596091\cdots\\
x_7 &= \mathbf{1.32471795724474602596091}\cdots
\end{aligned}$$

Since x_6 and x_7 agree with each other to the 23rd decimal place, we know that we have found an estimate to 23 decimal places of accuracy[7]. Interestingly, the level of accuracy approximately doubled with each iteration: x_2 was correct to 1 decimal place, x_3 to 2 places, x_4 to 5 places, x_5 to 13 places, and x_6 to at least 23 places. This approximate doubling is characteristic of quadratic convergence.

5.1. **Quadratic convergence.** In general, if a sequence p_n converges to p with $p_n \neq p$, and if there are positive constants λ and α such that

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda$$

then p_n converges to p on the order of α. That is if α equals 2, the sequence converges quadratically. [4]

Before we prove that Newton's method generally converges quadratically, we need the following theorem.

[7]Accuracy here is defined as the number of zeros to the right of the decimal place of $x_{n+1} - x_n$.

Taylor's Theorem with Remainder 5.1. *Let x and x_0 be real numbers, and let f be $k+1$ times continuously differentiable on the interval between x and x_0. Then there exists a number c between x and x_0 such that*

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x-x_0)^2}{2!} + \cdots$$
$$+ f^{(k)}(x_0)\frac{(x-x_0)^k}{k!} + f^{(k+1)}(c)\frac{(x-x_0)^{k+1}}{k!}.$$

[4]

From Theorem 5.1 and from the definition of convergence we have the following lemma,

Lemma 5.2. *If $N'(r) = 0$, then Newton's method will converge quadratically.*

Proof. If $N'(r) = 0$ then from Theorem 5.1 we have,

$$N(x) = N(r) + N'(r)(x - r) + N''(c)\frac{(x-r)^2}{2!},$$

where c is in between x and r. Simplifying and substituting x_n for x we have,

$$N(x_n) - N(r) = N''(c)\frac{(x_n - r)^2}{2},$$

and if we take the limit as $n \to \infty$ we see that,[8]

$$\lim_{n\to\infty} \frac{|x_{n+1} - r|}{|x_n - r|^2} = \frac{|N''(c)|}{2}.$$

Comparing this to the definition of convergence, we see that $\alpha = 2$ and $\lambda = \frac{|N''(c)|}{2}$, thus we conclude that Newton's method converges quadratically if $N'(r) = 0$. Recall from Section 4.1 $N'(r) = 0$ precisely when r is a root of multiplicity 1. □

Generally Newton's method converges quadratically, however, when $N'(r) \neq 0$ the method will converge only linearly as shown by Lemma 5.3.

Lemma 5.3. *If $N'(r) \neq 0$, then Newton's method will converge linearly.*

Proof. If the root of $f(x)$ is not a simple root, recall from Section 4.1 $N'(r) \neq 0$. From Theorem 5.1 we have

$$N(x) = N(r) + N'(c)(x - r).$$

If we rearrange terms, substitute, and take the the limit as $n \to \infty$ we see that

$$\lim_{n\to\infty} \frac{|x_{n+1} - r|}{|x_n - r|} = |N'(c)|.$$

Here $\alpha = 1$ and $\lambda = |N'(c)|$, thus Newton's method converges linearly for non-simple roots. □

Linear convergence of roots of multiplicity greater than 1 is quite unfortunate; a linearly converging algorithm is not as useful as a quadratically converging algorithm.

[8]Recall from Section 1 that $x_{n+1} = N(x_n)$.

5.2. **Modified Newton's method.** If Newton's method converges linearly for a particular function $f(x)$, the next theorem provides a simple way to modify the Newton iteration function of $f(x)$ to make the algorithm converge quadratically.

Modified Newton's method 5.4. *Suppose that there is a function f that has a root r of multiplicity $k > 1$, that is Newton's method converges linearly to the root. If we multiply the second term of the Newton iteration function by k, Newton's method will converge quadratically to the root. The modified Newton Iteration function is thus,*

$$N_k(x) = x - \frac{kf(x)}{f'(x)}.$$

Proof: Recall from Section 4.1 if r is a root of multiplicity $k > 1$ then $N(x)$ may be written as,

$$N(x) = x - \frac{(x-r)G(x)}{kG(x) + (x-r)G'(x)}.$$

Multiplying the second term of $N(x)$ by k we have,

$$N_k(x) = x - \frac{k(x-r)G(x)}{kG(x) + (x-r)G'(x)}.$$

Using the quotient rule we compute $N_k'(x)$ to be,[9]

$$N_k'(x) = 1 - \frac{(kG + (x-r)G')(kG + k(x-r)G') - (k(x-r)G)(kG' + G' + (x-r)G'')}{(kG + (x-r)G')^2}.$$

Finally we simplify, substitute $x = r$, and see,[10]

$$N_k'(r) = 1 - \frac{k^2(G(r))^2}{k^2(G(r))^2} = 1 - 1 = 0.$$

Thus, we see that $N'(r) = 0$ and Newton's method will converge quadratically as proved by Lemma 5.2.

Unfortunately, it is often difficult to find the multiplicity of a root, thus the modified Newton's method is more of a theoretical tool, rather then a practical algorithm.

6. Basins of attraction

So far we have mainly concerned ourselves with the question of convergence for individual starting points. Now we expand our previous work to encompass functions with multiple roots. Specifically, we are concerned with finding the initial guesses that will converge to a certain root.

Consider the function $f(x) = x^3 - 2x$. The Newton function of $f(x)$ is $N(x) = x - \frac{x^3-2x}{3x^2-2}$. If we choose $x_0 = 1$ as our starting guess and iterate the function we get the following sequence of points:

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 2 \\ x_2 &= 1.6 \\ x_3 &= 1.442253521 \\ x_4 &= 1.415010637 \end{aligned}$$

[9]Here G, G', and G'' denote $G(x)$, $G'(x)$, and $G''(x)$ respectively.

[10]$G(r) \neq 0$ by Lemma 4.1.

$$x_5 = 1.414214235$$
$$x_6 = 1.414213562$$
$$x_7 = 1.414213563$$

The series is converging. Now take the same function $f(x)$ but start with $x_0 = .7$. This results in the following sequence of points:

$$x_0 = .7$$
$$x_1 = -1.294339623$$
$$x_2 = -1.433222702$$
$$x_3 = -1.414585178$$
$$x_4 = -1.414213709$$
$$x_5 = -1.414213563$$
$$x_6 = -1.414213563\cdots$$
$$x_7 = -1.414213563\cdots$$

The series is converging to a different root. Even though we chose initial points relatively close together, their orbits converged to completely different roots. This leads us to hypothesize that our initial guess determines the root to which Newton's method will converge. To formalize this hypothesize first we need a definition.

Definition 6.1. *If r is a root of $f(x)$, the* ***basin of attraction*** *of r, is the set of all numbers x_0 such that Newton's method starting at x_0 converges to r. In symbols,*

$$B(r) = \{x_0 | x_n = N^n(x_0) \textit{ converges to } r\}.$$

[3]

To illustrate the basins of attraction, let us find the basins of attraction for the function $f(x) = x^3 - x$. Solving for the roots of $f(x)$ we find that $f(x)$ has roots at -1, 0, and 1. First, let us confirm that these points are indeed attracting fixed points. Recall from the definition of a fixed point, that a point r is a fixed point of $N(x)$ if $N(r) = r$. Furthermore, from Newton's Fixed Point theorem, fixed points of $N(x)$ are always attracting fixed points. Computing the Newton function of $f(x)$ we see, $N(x) = x - \frac{x^3-x}{3x^2-1}$. Substituting -1, 0, and 1 for x we see that the roots are indeed attracting fixed points of $N(x)$.

To find the basins of attraction we first look at the graph of $f(x)$ in Figure 6. From the graph we see that if $x_0 \geq 1$, x_n will converge to 1. That is, $[1, \infty) \subset B(1)$. Furthermore, if x_0 is between the critical point $\frac{1}{\sqrt{3}}$ and 1 the first iteration x_1 will be greater then 1, so x_n will also converge to 1. So we have $(\frac{1}{\sqrt{3}}, \infty) \subset B(1)$. Finally if $x_0 = \frac{1}{\sqrt{3}}$, Newton's method fails because $\frac{1}{\sqrt{3}}$ is a critical point of $f(x)$. The interval $(\frac{1}{\sqrt{3}}, \infty)$ is the largest open interval about 1; we call this the **local basin of attraction** or the largest basin of attraction for the point $x = 1$. In this particular case, the local basin of attraction is symmetric; $(-\infty, \frac{-1}{\sqrt{3}}) \subset B(-1)$ because of symmetry of the function. We see this by a similar argument as just presented for $x_0 \leq -1$ and then for x_0 between the other critical point $\frac{-1}{\sqrt{3}}$ and -1.

Finally we consider the last root of $f(x)$ at $x = 0$. By carefully looking at the graph in Figure 6 or by iterating points near 0, we notice that the points seem to oscillate around 0: if $x_0 > 0$, then $N(x_0) < 0$. For example, if $x_0 = .3$ we get the following sequence:

$$x_0 = .3$$
$$x_1 = -0.0739726027$$

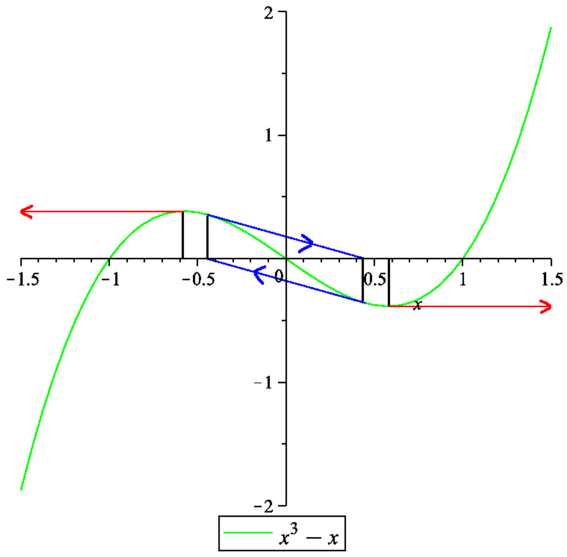

FIGURE 6. Graph of $f = x^3 - x$

$$x_2 = 0.00082305938$$
$$x_3 = -.0000000011151.$$

This oscillation from positive to negative values suggests that we should look for a cycle of period two in $N(x)$. A cycle of period two is a point x such that $N^2(x) = x$. Notice that $f(x)$ is an odd function. That is $f(-x) = -f(x)$. Since $f(x)$ is odd so is $N(x)$. This symmetry greatly simplifies finding periodic points of $N(x)$. Since $N^2(x) = N(N(x))$ we have:

$$N(N(x)) = N(-x) = x.$$

That is we need to solve where $N(x) = -x$. So we have,

$$-x = \frac{2x^3}{3x^2 - 1}, \;\; 5x^3 - x = 0, \;\; x = 0, \pm\frac{1}{\sqrt{5}}.$$

We know 0 is not a periodic point because it is a fixed point. So we conclude that $\pm\frac{1}{\sqrt{5}}$ are our periodic points of period two. We have thus found that the local basin of attraction for 0 is $(\frac{-1}{\sqrt{5}}, \frac{1}{\sqrt{5}})$. To summarize, the local basin of attraction for the critical point -1 is $(-\infty, \frac{-1}{\sqrt{3}})$. The local basin of attraction for the critical point 0 is $(\frac{-1}{\sqrt{5}}, \frac{1}{\sqrt{5}})$. And finally the local basin of attraction for the critical point 1 is $(\frac{1}{\sqrt{3}}, \infty)$.

Notice that we have not discussed the intervals $(\frac{-1}{\sqrt{3}}, \frac{-1}{\sqrt{5}})$ and $(\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{3}})$. In these intervals Newton's method has radically different behavior. According to our previous analysis, if $x_0 = \frac{1}{\sqrt{3}}$ then $N(x_0)$ does not exist because the tangent line at x_0 has a slope of zero. If x_0 is some number slightly less then $\frac{1}{\sqrt{3}}$ then the tangent line intersects the x-axis at some large negative number. That is, $N(x_0)$ is a large negative number and thus x_0 is in $B(-1)$. If we continue decreasing x_0 by some small

i	b_i	$b_i - b_{i-1}$	$(b_i - b_{i-1})/(b_{i+1} - b_i)$
0	.577350		
1	.465601	.11749	7.26
2	.4502020	.015399	6.18
3	.4477096	.0024924	6.03
4	.4472962	.0004134	6.01
5	.44722736	.00006884	6.00
6	.44721589	.00001147	6.00
7	.44721398	.00000191	6.00
$\vdots$	$\vdots$	$\vdots$	$\vdots$
∞	.447213595		

TABLE 1. Lengths of intervals and ratios of lengths of successive intervals.

amount we see that x_0 stays in $B(-1)$ until $N(x_0) = \frac{-1}{\sqrt{3}}$. At this critical point the slope of the tangent line is zero, and $N(N(x_0))$ does not exist. Thus, we have found a small interval that is contained in the basin of -1. We can approximate the interval by solving the equations $N(x) = \frac{-1}{\sqrt{3}}$ and $N(x) = \frac{1}{\sqrt{3}}$ for x. Doing so, we find the interval is approximately $(0.465601, 0.577350)$. By symmetry, we know the symmetric interval $(-0.577350, -0.465601)$ is contained in $B(1)$. Now we continue decreasing x_0 an arbitrarily small amount below .465601 such that $x_1 = N(x_0)$ is greater than $\frac{-1}{\sqrt{3}}$. The tangent line at x_1 intersects the x-axis at a large positive number, that is $N(x_1)$is a large positive number and $N(x_1) \subset B(1)$. As we continue to decrease x_0 below .465601, x_1 becomes even greater than $\frac{-1}{\sqrt{3}}$ and $x_2 = N(x_1)$ decreases towards $\frac{1}{\sqrt{3}}$. When $x_2 = N(x_1) = \frac{1}{\sqrt{3}}$ then the slope of the tangent line is zero again and $N(N(x_0))$ does not exist. Approximating $N(x) = -0.465601$ for x we find $x_0 \approx .450202$. Thus the interval $(.450202, .465601) \subset B(1)$. In general, we find a sequence of numbers $b_0 = \frac{1}{\sqrt{3}} > b_1 \approx 0.465601 > b_2 \approx 0.450202 > b_3 > \cdots$ such that

$$(b_i, b_{i-1}) \subset B(-1) \text{ when } i \text{ is odd,}$$

and

$$(b_i, b_{i-1}) \subset B(1) \text{ when } i \text{ is even.}$$

The numbers b_i are determined by successively solving equations $N(b_i) = b_{i-1}$. The values of the first few b_i's are given in Table 1, along with the lengths of the intervals (b_i, b_{i-1}) and the ratios of lengths of the successive intervals. Each $B(-1)$ and $B(1)$ consists of infinitely many intervals, whose lengths decrease approximately geometrically. An arbitrarily small movement of x_0 to the left of $\frac{1}{\sqrt{5}}$ causes convergence to shift between 1 and -1 infinitely often.

7. Newton's method in the Complex Plane

7.1. **Newton's method on the complex plane.** If needed, please refer to Appendix B for a quick review of the complex numbers. Newton's method directly generalizes to the complex plane. Let $N(z) = z - \frac{f(z)}{f'(z)}$, and z_0 be a complex number, then the iterations of $N^n(z_0)$ will in general converge quadratically to a zero of $f(z)$ [3]. Consider the complex polynomial $f(z) = z^2 + 1$. Recall from Figure 3 in Section 3 that the corresponding real function $f(x) = x^2 + 1$ has no real roots. Unlike $f(x) = x^2 + 1$ the corresponding complex function does have a solution; in fact $f(z) = z^2 + 1$ has two solutions at $z = i$ and $z = -i$. If we choose z_0 on the real axis (that is $y = 0$) then the iterates of $N(z)$ behave exactly as they do for $f(x) = x^2 + 1$, that is they behave chaotically. However if we chose z_0 off the real axis, Newton's method converges.

$$\begin{array}{ll}
z_0 = 1 + .5i & z_0 = .5 - i \\
z_1 = .1 + .4500i & z_1 = .0500 - .9000i \\
z_2 = -.1853 + 1.2838i & z_2 = -.0058 - 1.0038i \\
z_3 = -.0376 - 1.0234i & z_3 = -i \\
z_4 = -.0009 + .9996i & \\
z_5 = i &
\end{array}$$

Now we consider the basins of attraction for complex polynomials.

7.2. **Complex Basins of attraction.** The basins of attraction for complex Newton's method was first considered by Arthur Cayley. In 1879 he published the following theorem:

Cayley's Theorem 7.2.1. *Let the complex quadratic polynomial $f(z) = az^2+bz+c$ have zeros α and β in the complex plane. Let L be the perpendicular bisector of the line segment from α to β. Then, when Newton's method is applied to $f(z)$, the half-planes into which L divides the complex plane are exactly $B(\alpha)$ and $B(\beta)$, the basins of attraction to α and β* [3].

Cayley's Theorem completely describes the basins of attraction for complex Newton's method applied to quadratic complex polynomials. Starting at a point z_0 complex Newton's method converges to α exactly when $|z_0 - \alpha| < |z_0 - \beta|$. However, if z_0 lies on the perpendicular bisector L, complex Newton's method will not converge and behave chaotically.

Once again consider the complex quadratic $f(z) = z^2 + 1$. Recall that the two roots of the function are $z = \pm i$. Thus, we can see that the perpendicular bisector of the two roots will be $z = 0$ or the real axis. Applying Cayley's Theorem, we know points above the real axis will converge to the root i and points below the real axis will converge to the root $-i$. Since the real axis is the perpendicular bisector, any initial value chosen on the real axis will not converge. Refer to Figure 7 for a visual of the basins of attraction for the complex quadratic $z^2 + 1$.

Cayley also considered complex cubics, but was unable to find an obvious division for the basins of attraction. It was only later in the early 20th century that the mathematics Fatou and Julia began to understand the nature of complex cubic polynomials. Further still, beginning in the 1980s mathematicians were able to finally create pictures of the basins of attraction of complex cubic functions [3]. In the next section we will outline a method for viewing the complex basins of attraction of complex cubic polynomials.

7.3. **Programming the basins.** In order to view the basins of attraction for complex polynomials of degree two or greater we make use of a computer. There are several algorithms that can be used to display the basins of attraction for complex Newton's method. The method that is used in this paper to find the basins of attraction for a given complex function $f(z)$ is as follows;

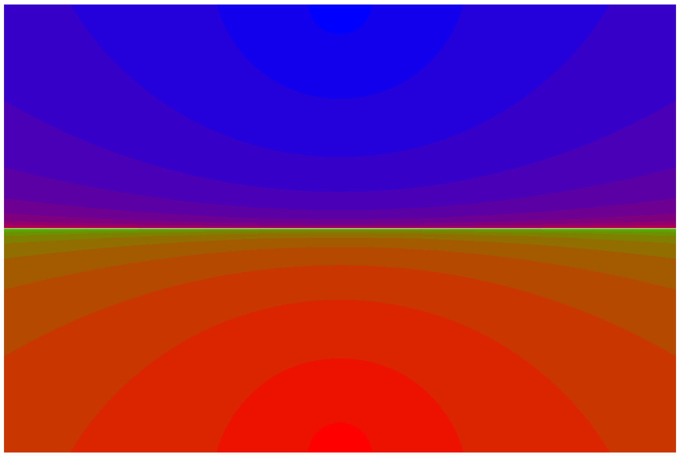

FIGURE 7. The basins of attraction for $z^2 + 1$.

(1) Compute $f'(z)$ and $N(z)$.
(2) Compute the roots of $f(z)$ via factoring or numerical approximation on f.
(3) Pick an initial point and calculate the distance between the point and the roots of f. If the distance is less then some small ϵ color the point the root color.
(4) If not, iterate until the distance between the iterate and the roots of f is less then some small value ϵ. Color the original point the appropriate root color.
(5) Repeat for all points within view.

This is a generalized version of the algorithm that is used in this paper. For specifics, the C++ code is located in Appendix C. If you are interested in learning to program in C++, Steve Heller's excellent introductory book is listed in the reference section of this paper.

8. Program Results

Using the program described in Section 7.3 the following pictures can be obtained. In Figure 8 $f(z) = z^3 - z$ is pictured. The roots of $f(z) = z^3 - z$ are $z = 0, \pm 1$. Figure 9 is a picture of one of the bulbs of the Figure 8. The colors represent the various roots. In Figures 8 and 9 the points which converge to the root at $z = 0$ are colored blue, while the roots at $z = -1$ and $z = 1$ are colored red and green respectively. The different color shades represent the amount of iterations needed for that particular point to converge.

Figure 10 shows the basins of attraction for $f(z) = z^3 - 1$. The roots of $f(z) = z^3 - 1$ are $z = 1$, $z = -\frac{1}{2} + \frac{\sqrt{3}}{2}$, and $z = -\frac{1}{2} - \frac{\sqrt{3}}{2}$. Notice that the color shading is hardly noticeable in Figure 10. However, if we magnify the middle of the picture (as in Figure 11) we are able to see that color shading is indeed present. This is because all the points within view converge very quickly to their respective roots.

Figure 12 shows the basins of attraction for $f(z) = z^4 - 1$. The roots of $f(z) = z^4 - 1$ are $z = 1$, $z = -1$, $z = i$, and $z = -i$ and are colored green, red, blue, and teal respectively.

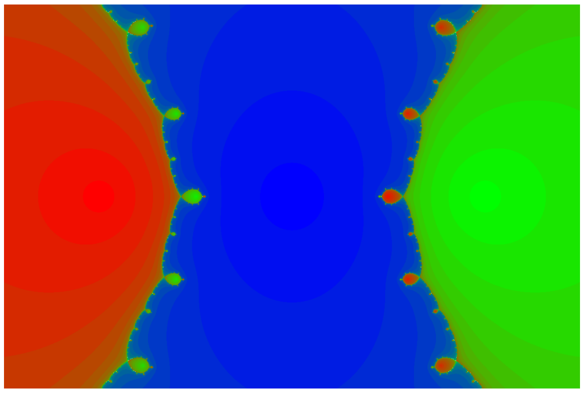

FIGURE 8. The basins of attraction for $z^3 - z$.

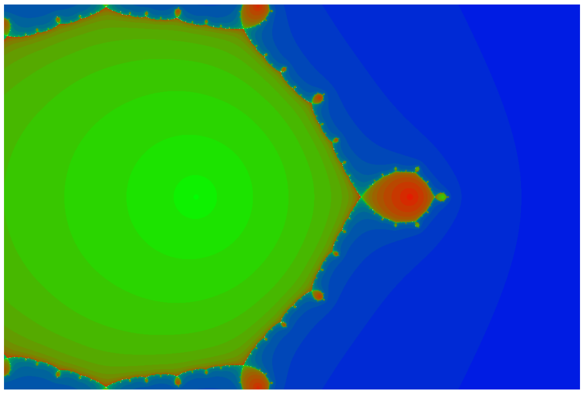

FIGURE 9. Magnification of one of the bulbs seen in Figure 8

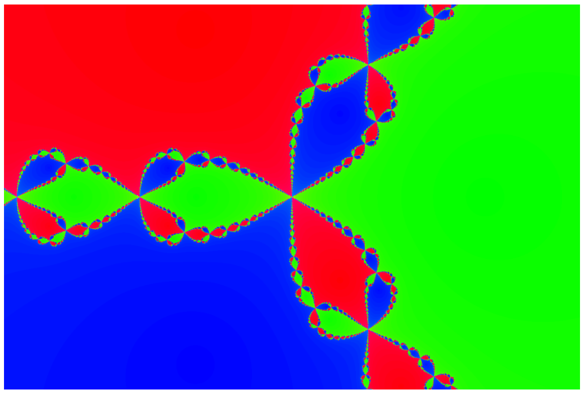

FIGURE 10. The basins of attraction for $z^3 - 1$.

9. Conclusion

In this paper we restricted ourselves to only a few complex polynomials that had roots found via factoring. An interested reader could expand on the C++ program

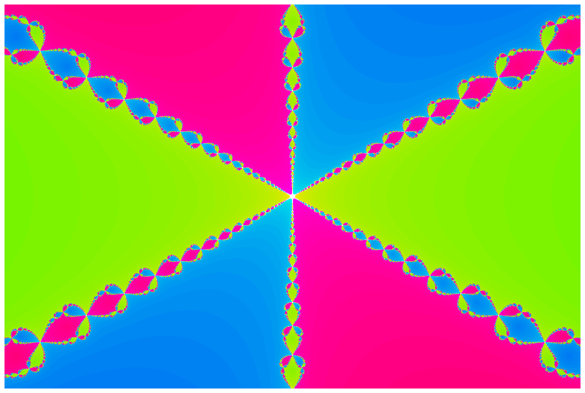

FIGURE 11. Magnification of the middle of $f(z) = z^3 - 1$.

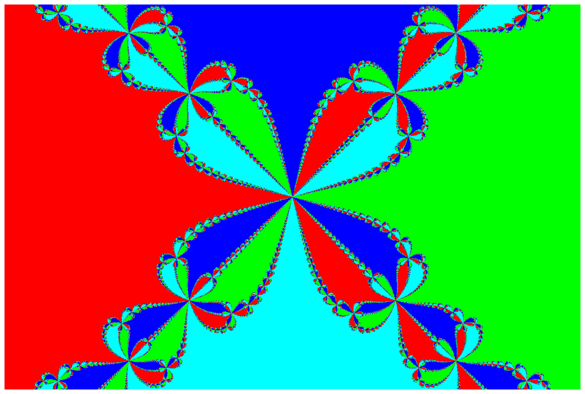

FIGURE 12. The basins of attraction for $f(z) = z^4 - 1$.

found in Appendix C and view the basins of attraction for any complex function they wish. Additional pictures of Newton's method in the complex plane can be found in Peitgen and Richter [1986, Chapter 6] and Peitgen [1984]. Becker and Dörfler [1989, Chapter 4] have a number of do-it-yourself computer experiments involving Newton's method and other ways of generating fractals. Devaney [1] contains a discussion relating Newton's method and chaotic dynamical systems both in the real and complex plane.

REFERENCES

[1] Robert L. Devaney. *A First Course in Chaotic Dynamical Systems*. Westview Press. 1992.
[2] Steve Heller. *Introduction to C++*. Academic Press. 1997.
[3] Philip D. Straffin JR. *UMAP Modules: Tools for Teaching 1991. Newtons Method and Fractal Patterns.* I NEED TO GET PUBLISHER FROM LIBRARY
[4] Timothy Sauer. *Numerical Analysis*. Pearson. 2006.

Appendix A. Fixed and Attracting Fixed points

Recall from Definition 4.2 that a point x_0 is a *fixed point* of a function $f(x)$ if and only if $f(x_0) = x_0$. Moreover, the point x_0 is called an *attracting fixed point* if $|f'(x_0)| < 1$.

To illustrate more clearly the implications of fixed and attracting fixed points consider the following. Suppose we wish to find the fixed points of the function $f(x) = x^3$. To do so, we use the definition of a fixed point and solve for where $x^3 = x$. Doing so yields $x = 0$ and $x = \pm 1$. These are the fixed points of the function $f(x) = x^3$. We can verify this by computing the following:

$$f(0) = 0^3 = 0,\ f(-1) = -1^3 = -1,\ f(1) = 1^3 = 1.$$

These points are called the fixed points for $f(x) = x^3$ because as we preform function iteration at these points, the orbit or sequence of points generated stays constant.

By computing the derivative and substituting in the fixed points we see that

$$f'(x) = 3x^2$$
$$f'(0) = 0$$
$$f'(1) = 3$$
$$f'(-1) = 3.$$

Now since $|f'(0)| < 1$ we know by definition that 0 is an attracting fixed point for $f(x) = x^3$. This implies that if we pick a point near enough to 0, say 0.5, the function should converge to 0 if we iterate.

$$f(.5) = (.5)^3 = .125$$
$$f(.125) = 0.001953125$$
$$f(.001953125) = 0.000000007$$

Newton's method is merely a form of fixed point iteration. Newton's method is designed so that the Newton iteration function $N(x)$ has attracting fixed points at the roots of the function $f(x)$. That is, as we iterate $N(x)$, the sequence of points will converge to the roots of $f(x)$.

For more details on fixed and attracting fixed points refer to Devaney chapters 3-5 listed in the reference section of this paper.

Appendix B. Review of Complex Numbers

A complex number z has the form $z = x + iy$, where x and y are real numbers and i is a symbol having the property that $i^2 = -1$. Thus a complex number is made of two parts; the real part x and the imaginary part iy. We represent the complex number $z = x+iy$ as the point (x, y) on the complex plane. The horizontal axis or x axis is the real axis; it corresponds to the same x axis as used in ordinary geometry. However, the vertical y axis is unlike the ordinary y axis used in ordinary geometry. We call the vertical axis the imaginary axis.

Addition and subtraction are done component wise, that is if $z = x + iy$ and $w = u + iv$, then

$$z + w = (x + u) + i(y + v),$$

and

$$z - w = (x - u) + i(y - v).$$

The distance between any two complex numbers is the same as the distance between two points in the real plane. So if we want to find the distance between z and w we have,

$$|z-w| = \sqrt{(x-u)^2+(y-v)^2}.$$

Multiplication between two complex numbers is done using the distributive laws and the property that $i^2=-1$. For example,

$$zw = (x+iy)(u+iv) = xu + i(xv+yu) + i^2(yv) = (xu-yv)+i(xv+yu).$$

To divide complex numbers, we use the method of rationalizing the denominator:

$$\frac{z}{w} = \frac{x+iy}{u+iv} \times \frac{u-iv}{u-iv} = \frac{(xu+yv)+i(yu-xv)}{u^2+v^2} = \frac{xu+yv}{u^2+v^2} + i\Big(\frac{yu-xv}{u^2+v^2}\Big).$$

Derivatives of functions of a complex variable are computationally equivalent to derivatives of functions of real variables. For example, if $f(z)=z^4+z^3-z$, then $f'(z)=4z^3+3z^2-1$.

Appendix C. C++ Code

C.1. **Main.cpp.**

```
// Main.cpp
// Aaron Burton
// Whitman College
// May 2009

#include <iostream>
#include <Magick++.h>
#include <cstdlib>
#include <string>
#include <iomanip>
#include "CoordPlane.h"
#include "Complex.h"
#define NUM_CELLS 5
#define HORIZONTALFIX 8
#define VERTICALFIX 3
#define NOTCOLORED 0
#define COLORED 1
using namespace std;
using namespace Magick;

string itos(int);
```

```
void OverLayGrid(Image&,const CoordPlane&);
void ZoomIn(Image&,CoordPlane&);
void DrawFractal(Image&,CoordPlane&);

int main(){
  CoordPlane Plane;
  Image image;

  image=Image(Geometry(Plane.HorizontalResolution(),
       Plane.VerticalResolution()),"white"); // Creates a blank image.
  DrawFractal(image,Plane); // Draws the basins.
  OverLayGrid(image,Plane); // Overlays zoom-in grid.
  Plane.print(); // Prints coordinates
  image.display(); // Displays image
  ZoomIn(image,Plane); // Zoom-in/save loop

}

// Most important function. Change this to draw the basins
// of attraction for different functions.

void DrawFractal(Image& Img,CoordPlane& CP){

// Also color can be added here.
  int i,j,k;
  Complex Z,N;

  for (i=0;i<CP.HorizontalResolution();i++){
    for (j=0;j<CP.VerticalResolution();j++){

      Z=Complex(CP.h2x(i),CP.v2y(j)); //

      // Check First 25 iterates of N
      for (k=1;k<=25;k++){

Z=(2*Z*Z*Z)/((3*Z*Z)-1); // f(x)=z^3-z; N(z) for f(x).

// Check distance between interates of N(z) and the roots.
if ((Z.RealPart()+1)*(Z.RealPart()+1)+(Z.ImagPart()*Z.ImagPart())
    <= .0001){
  Img.pixelColor(i,j,ColorRGB(1.0,0,0)); // root=-1
  break;
}
if (((Z.RealPart()-1)*(Z.RealPart()-1))+(Z.ImagPart()*Z.ImagPart())
    <= .0001){
  Img.pixelColor(i,j,ColorRGB(0,1.0,0)); // root=1
  break;
}
if ((Z.RealPart()*Z.RealPart())+(Z.ImagPart()*Z.ImagPart()) <= .0001){
  Img.pixelColor(i,j,ColorRGB(0,0,1.0)); // root=0
  break;
}
// Colors the point black if it does not converge.
if (k==25) Img.pixelColor(i,j,ColorRGB(0,0,0));
      }
    }
  }
}

void OverLayGrid(Image& Img,const CoordPlane& CP){
  int i,j;
  double xinterval,yinterval;

  xinterval=(CP.HorizontalResolution()-1)/NUM_CELLS;
  yinterval=(CP.VerticalResolution()-1)/NUM_CELLS;

  for (i=1;i<=NUM_CELLS-1;i++){
    Img.draw(DrawableLine(xinterval*i,0,xinterval*i,CP.VerticalResolution()));
    Img.draw(DrawableLine(0,yinterval*i,CP.HorizontalResolution(),
  yinterval*i));
  }
  for (i=1;i<=NUM_CELLS*2;i=i+2){
    for (j=1;j<=NUM_CELLS*2;j=j+2){
      Img.draw(DrawableText(xinterval*j/2-HORIZONTALFIX,
    yinterval*i/2+VERTICALFIX,
    itos((i/2*NUM_CELLS+j/2)+1)));
    }
  }
}

string itos(int x) {
  string result;
  char num[100];

  sprintf(num,"%3i",x);
  result = num;

  return result;
}

string ftos(double x) {
   string result;
   char num[100];

   sprintf(num,"%3.9f",x);
   result = num;
   return result;
}

void ZoomIn(Image& Img,CoordPlane& CP){
  int GetView(Image&,CoordPlane&);
  int CellNum,i,j;
  double x_min_new,x_max_new,y_min_new,y_max_new;
  double xinterval,yinterval;

  while(true){

    CellNum=GetView(Img,CP);

    xinterval=(CP.xUpperBound()-CP.xLowerBound())/NUM_CELLS;
    yinterval=(CP.yUpperBound()-CP.yLowerBound())/NUM_CELLS;
    i=(CellNum-1)/NUM_CELLS+1;
    j=(CellNum-1)%NUM_CELLS+1;

    x_min_new=CP.xLowerBound()+(j-1)*xinterval;
    x_max_new=CP.xLowerBound()+j*xinterval;
    y_min_new=CP.yUpperBound()-(i*yinterval);
    y_max_new=CP.yUpperBound()-((i-1)*yinterval);

    CP=CoordPlane(CP.HorizontalResolution(),x_min_new,x_max_new,y_min_new,
  y_max_new) ;

    setprecision(16);
    Img=Image(Geometry(CP.HorizontalResolution(),
       CP.VerticalResolution()),"white");
    DrawFractal(Img,CP);
    CP.print();
    OverLayGrid(Img,CP);
    Img.display();
  }
}

int GetView(Image& Img2,CoordPlane& CP2){
  int Cell=0;
  string c;
  while (Cell<1 || Cell>NUM_CELLS*NUM_CELLS){
    cout << endl << "Enter 'q' to quit or 's' to save previous fractal"
 << endl;
    cout <<"Enter a cell number to zoom-in on: ";
    // saves fractal as 1280x853
    if (cin.peek()=='s' || cin.peek()=='S'){
      CP2=CoordPlane(1280,CP2.xLowerBound(),CP2.xUpperBound(),
     CP2.yLowerBound(),CP2.yUpperBound());
```

```
      Img2=Image(Geometry(CP2.HorizontalResolution(),
       CP2.VerticalResolution()),"white");
      DrawFractal(Img2,CP2);
      c="Aaron Burton [" + ftos(CP2.xLowerBound()) + "," +
ftos(CP2.xUpperBound()) + "] X [" + ftos(CP2.yLowerBound()) +
"," + ftos(CP2.yUpperBound()) + "]";
      Img2.draw(DrawableText(CP2.HorizontalResolution()-450,
      CP2.VerticalResolution()-15,c));
      Img2.write("fractal.png"); // File Name.
      cout << "Saved fractal.png";
      exit(0);
    }
    // quits on q
    if (cin.peek()=='q' || cin.peek()=='Q'){
      exit(0);
    }
    else {
      // get cell # to zoom-in on.
      cin >> Cell;
      if (cin.peek()=='\n'){
c=cin.get();
      }
    }
  }
  return Cell;
}
```

C.2. **Complex.h.**

```
// Complex.h: Header file for the Complex class.
// Aaron Burton
// Whitman College
// May 2009

#ifndef COMPLEX_H
#define COMPLEX_H
#include <iostream>
using namespace std;

// A simple complex number class
class Complex {
  // friend functions and overloads
  friend double abs(const Complex&);
  friend Complex operator+(const Complex&,const Complex&);
  friend Complex operator-(const Complex&,const Complex&);
  friend Complex operator*(const Complex&,const Complex&);
  friend Complex operator/(const Complex&,const Complex&);
  friend Complex operator-(const Complex&); //negation
  friend Complex operator!(const Complex&); //conjugation
  friend ostream& operator<<(ostream&,const Complex&);
  public:
    // constructor, first arg sets real part, second arg sets imag part,
    // defaults are set to 0
    Complex(double=0,double=0);
    // returns the real part of the complex number
    double RealPart() const;
    // sets the real part of the complex number to the double arg
    void SetRealPart(double);
    // returns the imaginary part of the complex number
    double ImagPart() const;
    // sets the imaginary part of the complex number to the double arg
    void SetImagPart(double);

  private:
    // variables for holding the real and imaginary parts
    double _real, _imag;
};
#endif
```

C.3. **Complex.cpp.**

```
// Complex.cpp: The Complex class needed for Main.cpp.
// Aaron Burton
// Whitman College
// May 2009

#include <iostream>
#include <cassert>
#include "CoordPlane.h"
using namespace std;

CoordPlane::CoordPlane(){

  _h_max=800;
  _x_min=-1.5;
  _x_max=1.5;
  _y_min=-1;
  _y_max=1;
  _SquareAspect(); // sets _v_max
}

CoordPlane::CoordPlane(const int hmax, const double xmin,
       const double xmax, const double ymin,
       const double ymax){
  _h_max=hmax;
  _x_min=xmin;
  _x_max=xmax;
  _y_min=ymin;
  _y_max=ymax;
  _SquareAspect(); // sets _v_max
  assert(_x_min <= _x_max);
  assert(_y_min <= _y_max);
  assert(_h_max>100);
}

double CoordPlane::h2x(const int h) const{
  double Slope;

  Slope=(_x_max-_x_min)/(_h_max-1);
  return Slope*h+_x_min;
}

double CoordPlane::v2y(const int v) const{
  double Slope;

  Slope=(_y_max-_y_min)/(-1*(_v_max-1));
  return Slope*v+_y_max;
}

int CoordPlane::x2h(const double x) const{
  double Slope;

  Slope=(_x_max-_x_min)/(_h_max-1);
  return int((x-_x_min)/Slope);
}

int CoordPlane::y2v(const double y) const{
  double Slope;
  Slope=(_y_min-_y_max)/(-1*(_v_max-1));
  return int((y-_y_min)/Slope);
}

int CoordPlane::HorizontalResolution() const{
  return _h_max;
}

int CoordPlane::VerticalResolution() const{
  return _v_max;
}

double CoordPlane::xLowerBound() const{
  return _x_min;
}
```

```
double CoordPlane::yLowerBound() const{
  return _y_min;
}
```

C.4. **CoordPlane.h.**

```
// CoordPlane.h: Header file for the Coordplane class.
// Aaron Burton
// Whitman College
// May 2009

#ifndef COORDPLANE_H
#define COORDPLANE_H

class CoordPlane {
  public:
    // Default constructor, sets reasonable default values.
    CoordPlane();
    // Sets horizontal pixel resolution and corresponding x,y-coord plane,
    // h_max, x_min, x_max, y_min, y_max
    // v_max will be set by the private member function _SquareAspect()
    CoordPlane(const int, const double, const double, const double,
const double);
    // converts horizontal pixel coord, to x value
    double h2x(const int) const;
    // converts vertical pixel coord, to y value
    double v2y(const int) const;
    // converts x value, to horizontal pixel coord
    int x2h(const double) const;
    // converts y value, to vertical pixel coord
    int y2v(const double) const;
    // Returns the current horizontal resolution.
    int HorizontalResolution() const;
    // Returns the current vertical resolution.
    int VerticalResolution() const;
    // Returns the current lower bound of the x range
    double xLowerBound() const;
    // Returns the current lower bound of the y range
    double yLowerBound() const;
    // Returns the current upper bound of the x range
    double xUpperBound() const;
    // Returns the current upper bound of the y range
    double yUpperBound() const;

    // Prints a synopsis of the current class data to the screen.
    void print() const;

  private:
    int _h_max, _v_max;
    double _x_min, _x_max, _y_min, _y_max;

    // Sets vertical resolution to correct aspect ratio
    void _SquareAspect();
};

#endif
```

C.5. **CoordPlane.cpp.**

```
// CoordPlane.cpp
// Aaron Burton
// Whitman College
// May 2009

#include <iostream>
#include <cassert>
#include "CoordPlane.h"
using namespace std;

CoordPlane::CoordPlane(){

  _h_max=800;
  _x_min=-1.5;
  _x_max=1.5;
  _y_min=-1;
  _y_max=1;
  _SquareAspect(); // sets _v_max
}

CoordPlane::CoordPlane(const int hmax, const double xmin,
       const double xmax, const double ymin,
       const double ymax){
  _h_max=hmax;
  _x_min=xmin;
  _x_max=xmax;
  _y_min=ymin;
  _y_max=ymax;
  _SquareAspect(); // sets _v_max
  assert(_x_min <= _x_max);
  assert(_y_min <= _y_max);
  assert(_h_max>100);
}

double CoordPlane::h2x(const int h) const{
  double Slope;

  Slope=(_x_max-_x_min)/(_h_max-1);
  return Slope*h+_x_min;
}

double CoordPlane::v2y(const int v) const{
  double Slope;

  Slope=(_y_max-_y_min)/(-1*(_v_max-1));
  return Slope*v+_y_max;
}

int CoordPlane::x2h(const double x) const{
  double Slope;

  Slope=(_x_max-_x_min)/(_h_max-1);
  return int((x-_x_min)/Slope);
}

int CoordPlane::y2v(const double y) const{
  double Slope;
  Slope=(_y_min-_y_max)/(-1*(_v_max-1));
  return int((y-_y_min)/Slope);
}

int CoordPlane::HorizontalResolution() const{
  return _h_max;
}

int CoordPlane::VerticalResolution() const{
  return _v_max;
}

double CoordPlane::xLowerBound() const{
  return _x_min;
}

double CoordPlane::yLowerBound() const{
  return _y_min;
}

double CoordPlane::xUpperBound() const{
  return _x_max;
}

double CoordPlane::yUpperBound() const{
  return _y_max;
}

void CoordPlane::print() const{
  cout << "Horizontal Resolution= " << _h_max << endl
       << "Vertical Resolution= " << _v_max << endl
       << "X Min = " << _x_min << endl
       << "X Max= " << _x_max << endl
       << "Y Min= " << _y_min << endl
       << "Y Max= " << _y_max << endl;
}

void CoordPlane::_SquareAspect(){
  _v_max=int((_h_max/(_x_max-_x_min))*(_y_max-_y_min));
}
```

C.6. **The Makefile.**

```
\\ Makefile used to compile main.cpp
\\ Aaron Burton
\\ Whitman College
\\ May 2009

Fractal: CoordPlane.o Complex.o main.cpp
g++ CoordPlane.o Complex.o main.cpp \
‘Magick++-config --cppflags --cxxflags --ldflags --libs‘

CoordPlane.o: CoordPlane.cpp CoordPlane.h
g++ -c CoordPlane.cpp

Complex.o: Complex.cpp Complex.h
g++ -c Complex.cpp

clean:
rm -f *.o a.out
```