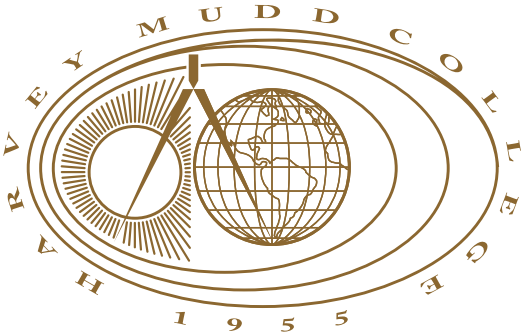
2005

Generalized Julia Sets: An Extension of Cayley's Problem

Owen Lewis
Harvey Mudd College



Generalized Julia sets: An Extension of Cayley's Problem

Owen Lewis

Dr. Jon Jacobsen, Advisor

Dr. Lesley Ward, Reader

May 10, 2005

HARVEY MUDD COLLEGE
Department of Mathematics

Abstract

There are many iterative techniques to find a root or zero of a given function. For any iterative technique, it is often of interest to know which initial seeds lead to which roots. When the iterative technique used is Newton's Method, this is known as Cayley's Problem. In this thesis, I investigate two extensions of Cayley's Problem. In particular, I study generalizations of Newton's Method, in both $\mathbb{C}$ and $\mathbb{R}^2$, and the associated fractal structures that arise from using more sophisticated numerical approximation techniques.

Contents

List of Figures

Acknowledgments

For patiently helping me to design and debug the algorithms used in this thesis, I would like to thank Dr. John Neuberger and Forrest Briggs. Furthermore, I would like to acknowledge Brain Tagiku for allowing me to develop a variation of his *basins.c* routine.

I would like to thank my advisor Dr. Jacobsen for his many hours of guidance over the last 3 years, as well as my 2^{nd} reader Dr. Ward for her valued input and infinite kindness.

Finally I would like to thank my good friend Ian Win for his encouragement and support throughout this process.

Chapter 1

Introduction

1.1 The Origins of the Problem

In many contexts and applications, one is faced with the task of finding a value x such that $f(x) = 0$ for some function f. Despite being a relatively easy problem to state, for certain choices of f, this task may be quite difficult. In the late 17^{th} century, drawing on the newly developed tools of Calculus, Newton proposed utilizing the iterative method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad x_n \in \mathbb{R}$$

to generate a sequence $\{x_n\}_{n=0}^{\infty}$ such that

$$\lim_{n\to\infty} x_n = \xi$$

where $f(\xi) = 0$. Indeed, for x_0 sufficiently close to ξ and sufficient conditions on f, the sequence $\{x_n\}$ converges to the root ξ. This is now known as Newton's Method. In 1879 A. Cayley questioned the behavior of Newton's Method in a different setting. Mimicking Newton's Method, Cayley considered the sequence generated by

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n)} \quad z_n \in \mathbb{C} \tag{1.1}$$

where p is a *complex* polynomial. Specifically, Cayley wished to characterize the global basins of attraction for each root ξ_i of p [2]. More precisely, these basins of attraction, $A(\xi_i) := \{z_0 \in \mathbb{C} : z_n \to \xi_i\}$, are the initial conditions that Newton's Method carries to each root. This task of identifying such

basins for complex polynomials is known as "Cayley's Problem." Having succeeded in characterizing these sets in the case of quadratic polynomials, Cayley remarked that "*the next succeeding case of the cubic equation appears to present considerable difficulty.*" Subsequent work by two French mathematicians would show that Cayley's difficulty was well justified.

Nearly 40 years later, groundbreaking work by G. Julia and P. Fatou shed some light on Cayley's Problem during the course of their respective research on the iteration of rational functions [7], [6]. The ideas they developed, including the notion of the Julia set, and were later extended by Brolin [1] to reveal that Cayley's Problem was not so trivial to answer as to pose.

1.2 The Quadratic Case

Let us first consider Cayley's Problem for the case of the quadratic. Many characteristics that we will wish to explore in more complicated cases will be present. In order to describe the behavior of Newton's Method, a few simple definitions are in order. Consider an arbitrary function $N : \mathbb{C} \to \mathbb{C}$.

Definition 1. *The set $\gamma = \{z_1, \ldots, z_n\}$ is a cycle of order n (or an n-cycle) provided $N(z_i) = z_{i+1}$ and $N(z_n) = z_1$.*

Definition 2. *The multiplier of an n-cycle γ is $\lambda = N'(\gamma)$, where $N'(\gamma) = \frac{d}{dz}N^n(z_i)$ for all $z_i \in \gamma$. We say γ is attracting (repelling) if $|\lambda| < 1$ ($|\lambda| > 1$). If $|\lambda| = 1$, then $\lambda = e^{2\pi i\phi}$. We say γ is rationally indifferent for $\phi \in \mathbb{Q}$ and irrationally indifferent for $\phi \notin \mathbb{Q}$.*

Note. These definitions for cycles include the special case of fixed points of N, corresponding to $n = 1$.

Now, it is possible, through a suitable change of variables to render any quadratic polynomial equivalent to one of the form $p(z) = z^2 + c$ for some $c \in \mathbb{C}$. For simplicity we will explore $p(z) = z^2 - 1$. In this case Newton's method reduces to $N(z) = z - \frac{z^2-1}{2z} = \frac{z^2+1}{2z}$ and the two attractive fixed points of N (roots of p), ξ_1 and ξ_2, are 1 and -1 respectively (There is also a repelling fixed point at ∞). Exactly as one's intuition would suggest,

$$A(1) = \{z \in \mathbb{C} : Re(z) > 0\}$$
$$A(-1) = \{z \in \mathbb{C} : Re(z) < 0\}.$$

Notice that $\partial A(-1) = \partial A(1) = \{z \in \mathbb{C} : Re(z) = 0\}$. Thus the two basins of attraction share a common boundary, namely the imaginary axis. Letting the "Julia set" be

$$J := \partial A(1) = \partial A(-1),$$

then it is clear that J is the imaginary axis. Indeed, consider the Möbius transformation $T(z) = \frac{z-1}{z+1}$. Notice that $T^{-1}(w) = \frac{1+w}{1-w}$. Then $\widehat{N}(w) := T \circ N \circ T^{-1}(w)$ is the map $w \mapsto w^2$ (since N and $\widehat{N}$ are conjugate maps, we can study N by considering the much simpler map $\widehat{N}$). A calculation shows that

$$\begin{aligned} T(1) &= 0 \\ T(-1) &= \infty \\ T(\infty) &= 1 \end{aligned}$$

and

$$\begin{aligned} T(J) = \widehat{J} &= \{z : |z| = 1\} \\ T(A(1)) = \widehat{A}(0) &= \{z : |z| < 1\} \\ T(A(-1)) = \widehat{A}(\infty) &= \{z : |z| < 1\}. \end{aligned}$$

It is now clear that in the w-domain, 1 is a repelling fixed point ($|\widehat{N}'(1)| > 1$), ∞ and 0 are the attracting fixed points and that their basins of attraction are indeed separated by $\widehat{J}$. Furthermore $\widehat{J}$ is completely invariant under $\widehat{N}$ since $\widehat{N}(\widehat{J}) = \widehat{J} = \widehat{N}^{-1}(\widehat{J})$. If $z = e^{2\pi i \frac{k}{2^n}}$ for $k, n \in \mathbb{N}$ then $\widehat{N}^n(z) = 1$. Thus we see that the set of pre-images of the repelling fixed point is dense in the Julia set. Now by construction, $\widehat{J}$ is the set of points for which Newton's Method fails to find a root. However, we can now see that on $\widehat{J}$, the dynamics of $\widehat{N}$ are rather complex. Similarly the set of (repelling) cycles $\{z = e^{2\pi i \frac{k}{2^n - 1}}$ for $k, n \in \mathbb{N}\}$ is dense in $\widehat{J}$ (as is the set of pre-images of this set). This characteristic of dense *inverse* orbits leads to the notion that $\widehat{N}$ is a chaotic map on $\widehat{J}$. It is important to note that this argument can be altered to suite any quadratic polynomial and show that in the case of two distinct roots, the Julia set is the perpendicular bisector of the segment joining the two roots [3].

1.3 Polynomials of Higher Order

In the case of the cubic polynomial, we would theoretically like to find a similar conjugation allowing easy classification of the Julia set of N, and

hence easy classification of the basins of attraction that we seek. In the simplest case of $p(z) = z^3 - 1$, one might expect a simple ternary division of the plane, such as:

$$J = \{z \in \mathbb{C} : z = \rho e^{\pi i \frac{2k+1}{3}},\ \rho > 0,\ k \in \mathbb{N}\}.$$

However, in this instance, intuition fails us [11], [14]. As we will discuss, the basins of attraction $A(\xi_i)$ for all 3 roots ξ_i share a common boundary (namely the Julia set J). So clearly the Julia set cannot be as simple as "intuition" suggests. Because J bounds each $A(\xi_i)$, for any point $z \in J$ and any neighborhood U containing z, $U \cap A(\xi_i) \neq \emptyset$ for all roots ξ_i of p. This leads to the term *3 corner point* being used in the case of a cubic p [3]. Numerical investigations into the basins of attraction $A(\xi_i)$ and their boundary J for the cubic polynomial $p(z) = z^3 - 1$ have been carried out and pictures of these sets are well known [3], [14], [12]. Note that the Julia set exhibits a fractal structure (see Figure 1.1).

The authors of [3], [14] and [12] also investigate in depth the one parameter family of polynomials $p_\lambda(z) = z^3 + (\lambda - 1)z - \lambda$, because through the proper change of variables, any cubic polynomial is equivalent to one in this family. In this case, since

$$N'(z) = \frac{p(z)p''(z)}{(p'(z))^2} = 0$$

we see that the roots of $N'(z)$ (i.e., the *critical points* of N) are the zeros of p and p'' (namely 0). Now if N were to have an attractive cycle γ of order $d \geq 2$, then by the work of Fatou, one of the critical points must lie in $A(\gamma)$. Since the zeros of p, ξ_i each clearly lie in their respective $A(\xi_i)$, one need only track the orbit of the "free" critical point 0 to determine the existence of an attracting cycle. All three of the references above provide computer generated images of the set of values λ for which no attracting cycle is present. For these values of λ Newton's Method converges to a root "almost-everywhere." Remarkably, this set appears to exhibit an infinite amount of self-similarity, bearing a resemblance to the Mandelbrot set. Numerous figures of other fractal sets generated by varying λ are also included.

It should be noted, however, that in cases where the polynomial p is of degree $d > 3$, Newton's Method is in some sense a "bad" algorithm [14].

Definition 3. *Let $T : \Sigma \to \Sigma$ be a rational function, and $p(z)$ be a polynomial. We say that $T(z)$ is convergent for p if $T^n(z) \to \xi_i$, where $p(\xi_i) = 0$, for all z in an open, dense subset of Σ*

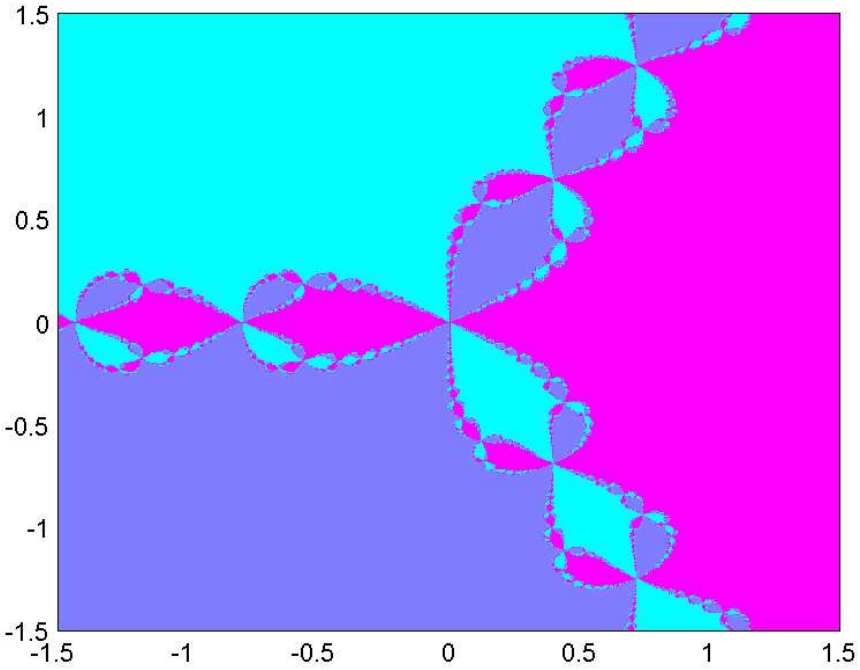


Figure 1.1: Basins of attraction for $p(z) = z^3 - 1$.

Now let T be a rational map from the space of complex polynomials of degree d to the space of rational functions on Σ. T is then called a *purely iterative algorithm*, and we say that T is *generally convergent* if $T(p(z))$ is convergent for p, for all p in an open dense subset of the space of polynomials.

Theorem 1.3.1 (von Haeseler, Peitgen [14]). *No generally convergent, purely iterative algorithm exists for finding roots of complex polynomials of degree $d \geq 4$.*

We note that if the operation of complex conjugation is allowed, then generally convergent algorithms exists for polynomials of any degree.

1.4 Damped and Continuous Newton's Method

If we modify the standard Newton's Method by introducing a parameter δ, we get what is sometimes referred to as the Damped Newton's Method.

$$N(z) = z - \delta \frac{p(z)}{p'(z)}, \quad 0 < \delta < 1. \tag{1.2}$$

The method is in fact consistent for any $\delta \in \mathbb{C}$ such that $|\delta| < 2$. However, we will only be concerned with a small, real parameter δ. The remarkable feature of the damped Newton's Method is that it can realized as an Euler approximation of a differential equation using a step size $\Delta t = \delta$. To see this, let

$$z_{i+1} = z_i - \Delta t \frac{p(z_i)}{p'(z_i)}.$$

Rearranging yields

$$\frac{z_{i+1} - z_i}{\Delta t} = -\frac{p(z_i)}{p'(z_i)}.$$

Finally, letting $\Delta t \to 0$ we have

$$z'(t) = -\frac{p(z)}{p'(z)}. \tag{1.3}$$

Equation (1.3) is known as Continuous Newton's Method, and can be thought of as a type of "gradient-flow" [13]. More generally, gradient-flow for a vector-valued function $f(z)$ is described by the differential equation

$$z'(t) = -\frac{1}{2}\nabla |f|^2.$$

However, it is true that in the case where f is a complex analytic function, $\frac{1}{2}\nabla|f|^2 = f\overline{f}'$. Therefore,

$$z'(t) = -\rho(z)\frac{f(z)}{f'(z)}$$

where $\rho(z) = |f'(z)|^2$. Thus it can be seen that equation (1.3) is a rescaled version of gradient-flow.

Returning our attention to (1.3), for z_0 such that $p'(z_0) \neq 0$, standard theory of differential equations guarantees a solution $z(t)$ to the initial value problem

$$z'(t) = -\frac{p(z)}{p'(z)}, \quad z(0) = z_0 \tag{1.4}$$

on some maximal time interval (ω_-, ω_+). Therefore, the singular set $S = \{z \in \mathbb{C} : p'(z) = 0\}$ presents a problem when analyzing the behavior of solutions to (1.4). In [9] the author utilizes the initial value problem

$$p(z)'(t) = -p(z(t)), \quad z(0) = z_0. \tag{1.5}$$

This is a slightly more general formulation of Continuous Newton's Method, since a brief calculation shows trajectories $z(t)$ that satisfy (1.4) also satisfy (1.5). However using (1.5) allows the author to "flow" right through the singular set S by concatenating trajectories that accumulate on S. A few key results summarize the behavior of (1.5).

Theorem 1.4.1. *If $z(t)$ satisfies (1.5) for all $t > 0$, then*

$$\lim_{t\to\infty} z(t) = \xi_i$$

exists, and $p(\xi_i) = 0$.

Theorem 1.4.2. *If $c \in \mathbb{C}$, then $c = z(t)$ for some $t \in \mathbb{R}$ and some $z(t)$ satisfying (1.5).*

Theorem 1.4.3. *If M is the set of $c \in \mathbb{C}$ such that c belongs to no trajectory that accumulates on S, then every component M_i of M contains just one root ξ_i of p. Furthermore, if $z(t) \in M_i$ for some t, then*

$$\lim_{t\to\infty} z(t) = \xi_i.$$

Thus, we see that continuous Newton's Method does indeed find roots of a polynomial p. To show this, assume $z(t)$ solves (1.5). Rearranging terms yields

$$\frac{p(z)'(t)}{p(z(t))} = -1,$$

and integrating with respect to t yields

$$\log p(z(t)) = -t + \log p(z(0)).$$

Finally, exponentiation shows that

$$p(z(t)) = p(z_0)e^{-t},$$

where $z_0 = z(0)$. In the case where $p(z) = z^3 - 1$, we can explicitly solve for $z = \sqrt[3]{(z_0^3 - 1)e^{-t} + 1}$. Thus we see that solutions decay exponentially to a root of unity (i.e., a root of p). What's more, we see that in the case of $p(z) = z^3 - 1$, the continuous Newton's Method reduces to

$$z'(t) = -\frac{z^3 - 1}{3z^2}.$$

It is clear then that $S = \{0\}$ and a phase portrait shows that the trajectories that accumulate on S are exactly the rays that intuition mistakenly suggested as the Julia set for the conventional Newton's Method for this polynomial. This behavior can also be seen by computing pictures of the Julia set J for damped Newton's Method and then allowing δ to shrink toward zero [12].

1.5 New Directions

In recent years, research into Newton's Method has moved into a more general setting. Since one may simply view a complex polynomial as a mapping from $\mathbb{R}^2$ to $\mathbb{R}^2$, surely there exists a higher dimensional analogue to (1.1). Specifically, if one wants to find zeros of a function $G : \mathbb{R}^n \to \mathbb{R}^n$, then Newton's Method takes the form

$$x_{n+1} = N(x_n) = x_n - \delta[DG(x_n)]^{-1}G(x_n), \tag{1.6}$$

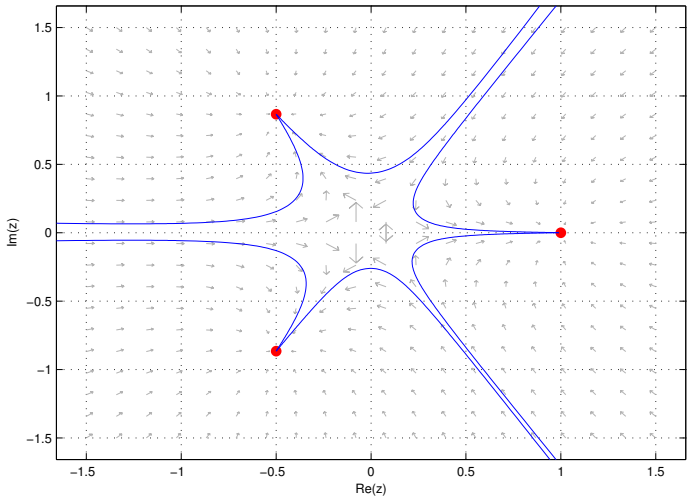


Figure 1.2: Vector Field and trajectories for Continuous Newton's Method when $p(z) = z^3 - 1$.

where $DG(x)$ denotes the Jacobian matrix of the map G. Note that this is still an Euler step for the differential equation

$$\begin{cases} x'(t) = -[DG(x(t))]^{-1}G(x(t)), \\ x(0) = x_0. \end{cases} \tag{1.7}$$

Equations (1.6) and (1.7) are only defined on $D = \mathbb{R}^n \setminus S_G$, where $S_G := \{x \in \mathbb{R}^n : detDG(x) = 0\}$ is the singular set of G. In [10], the authors provide an in-depth case study of (1.6) and (1.7), utilizing throughout an example G obtained through a 2 point finite difference scheme for the nonlinear boundary value problem

$$\begin{cases} u_{ss} + \lambda f(u) = 0, \quad 0 < s < 1, \\ u(0) = u(1) = 0. \end{cases}$$

Key results show that for x_0 in certain domains,

$$\frac{d}{dt}G(x(t)) = -G(x(t)).$$

Therefore, $G(x(t)) = e^{-t}G(x_0)$. From this we immediately see that if $x(t)$ is defined for all $t > t_0$, $G(x(t)) \to 0$. However, in the example investigated in [10], there also exist domains C such that for $x_0 \in C$, and the corresponding solution x to (1.7), we have $x(t) \in S$ for $t < \infty$. That is to say that the singular set absorbs trajectories in finite time. Numerical simulation of (1.6) indicates a Julia-like set of fractal nature associated with the system. However, the structure of this set is not as well understood as the classical Julia sets. The authors conduct numerous computer experiments investigating it's nature. Following in the spirit of [10], sets of this type associated with approximation of (1.7) will be one of the major topics of this thesis.

Chapter 2

Theory of Rational Mappings

2.1 Rational Complex Functions

To understand the existing work associated with Cayley's Problem, we must first introduce some terminology. Also, it will be helpful to consider the "point" at infinity, therefore we will be treating maps of the extended complex plane (sometimes known as the Riemann Sphere). It should be noted that the extended complex plane forms a metric space under the *spherical norm* d_s, where

$$d_s(z,w) = \frac{2|z-w|}{\sqrt{(|z|^2+1)(|w|^2+1)}},\ d_s(z,\infty) = \frac{2}{\sqrt{|z|^2+1}} \quad z,w \in \mathbb{C}.$$

Definition 4. *Let $\Sigma = \mathbb{C} \cup \{\infty\}$ be the extended complex plane. A function $R : \Sigma \to \Sigma$ is said to be a rational function if $R(z) = \frac{P(z)}{Q(z)}$ where P and Q are complex polynomials.*

The degree, d of a rational function, is the maximum of the degrees of P and Q. For the polynomials such as Cayley considered, Newton's Method reduces to a rational mapping on Σ, allowing the use of the ideas of Brolin, Fatou, and Julia to solve Cayley's Problem. However, it will be necessary to make some definitions before proceeding.

Definition 5. *A complex number $z \in \Sigma$ is a critical point of R provided $R'(z) = 0$. The set of all critical points will be denoted by S.*

The behavior of cycles, fixed points and critical points will be essential to characterizing the behavior of iterates of R on a global scale. However one additional construct will also be required.

Definition 6. *A rational mapping $R : \Sigma \to \Sigma$ is said to be* normal *at a point $z \in \Sigma$ if for some neighborhood U of z, the sequence $\{R^n|_U\}$ is equicontinuous with respect to the spherical norm d_s.*

Remark. This is a simpler characterization of normality than Montel originally formulated, however for our purposes it will suffice [14].

We are now ready to formally define the Julia set which was of such interest in Chapter 1. In honor of the pioneers of this field, the set of points where R is not normal is known as the *Julia set* of R and $\Sigma \setminus J$ is known as the *Fatou set*.

Definition 7. 1. $J := \{z \in \Sigma : R \text{ is not normal at } z\}$.

2. $F := \Sigma \setminus J$.

The behavior of R on the Julia set is remarkably complex, and it turns out that J is crucial to understanding the dynamics of Newton's Method for complex polynomials. Some basic properties of the Julia set follow, for verification of these properties, see [1],[14].

Theorem 2.1.1. 1. *J is closed.*

2. *$J \neq \emptyset$.*

3. *$R(J) = J = R^{-1}(J)$.*

4. *J is perfect (i.e., J is dense in itself).*

5. *If J contains some open neighborhood U, then $J = \Sigma$.*

6. *If P_R is the set of repelling periodic points, then $P_R \subset J$. Furthermore, $\overline{P_R} = J$.*

This final result plays an important role in determining the dynamics of R on J. This density of repelling cycles leads to the type of chaotic behavior of R on the Julia set that was mentioned earlier [14].

Theorem 2.1.2. *If $\alpha(z_0) := \{z \in \Sigma : R^n(z) = z_0, n \in \mathbb{N}\}$ is the set of preimages of z_0, and $z_0 \in J$, then $\overline{\alpha(z_0)} = J$.* [1]

Thus we see that the behavior witnessed in the example of Chapter 1 is characteristic of Newton's method for all complex polynomials.

[1] Clearly in this case $\alpha(z_0) \subset J$.

2.2 Applications to Cayley's Problem

Since R is normal at any attractive fixed point or cycle [1], it is clear J does not contain these points. In fact, if γ is an any attractive cycle, and $A(\gamma) := \{z \in \Sigma : \gamma \textit{ is the set of limit points of } R^n(z)\}$ is the basin of attraction for γ, then the following result holds [14].

Theorem 2.2.1. *$A(\gamma) \subset F$ and $\partial A(\gamma) = J$.*

Thus we see that the invariant Julia set necessarily divides the basins of attraction for the various cycles of R. Furthermore, let us define $A^*(\gamma) := \bigcup A_i$ (where A_i is the maximal connected component of $A(\gamma)$ that contains z_i for each $z_i \in \gamma$) to be the *immediate* basin of attraction of γ.

Theorem 2.2.2. *For each attractive cycle γ, $A^*(\gamma) \cap S \neq \emptyset$. In particular, $A^*(\gamma)$ contains at least one critical point of R. [1]*

This means that with knowledge of the Julia set and the orbits of the critical points, large components of each basin of attraction can be identified. The difficulty arises in determining which points lay in the Julia set. None of the results above offer an obvious way to quickly calculate J. The truth of the matter is that this is inherently difficult, due to the fact that J will often be a fractal. Under quite general conditions, J consists of an infinite number of Jordan curves and has a well defined tangent line nowhere [1].

Chapter 3

The Julia set and Capacity Dimension

3.1 Damped Newton's Method

We return now to the Damped Newton's Method (1.2), where $p(z) = z^3 - 1$. Again, we wish to explore the idea that this is a simple approximation to the differential equation (1.3). As we have already discussed, in the case of (1.3), there does not exist a Julia set of fractal dimension bounding the basins of attraction for the system. However, in the case of (1.2), we know that a Julia set of fractal dimension exists for all $\delta > 0$. By visualizing the basins of attraction for various values of δ, it becomes clear that the step size of the approximation method affects the Julia set somehow (Fig. 3.1). Whether the structure of the fractal is somehow altered, or merely scaled is not immediately apparent. In order to investigate this more thoroughly, we will wish to examine the fractal dimension of the Julia set as our quantity of interest. More than just a curiosity, the fractal dimension of the Julia set has significance in this context of approximating a differential equation.

Recall from Theorem 2.2.1 that the Julia set necessarily bounds all basins of attraction for the system. Thus, in any neighborhood of J we will find subsets of *each and every* basin of attraction. This means that trajectories corresponding to arbitrarily close initial conditions will have dissimilar long term behavior (they will each converge to distinct roots). Consequently, our approximation algorithm is in a sense unstable at points of J. It is for this reason that we are interested in the dimension of the Julia set: it gives us a notion of the size of the set where our approximation algorithm is "bad".

3.2 Capacity Dimension

Unfortunately, the very definition of fractal dimension has not been standardized. There exist several various notions of fractal dimension, each useful in certain circumstances. The Hausdorff and Minkowski-Bouligand dimensions are both highly applicable to sets of fractal nature, however due to their abstract definitions, are of limited use in our numerical simulations.[1] Therefore, we will be computing the *capacity dimension* of the Julia set. The notion of capacity dimension stems from the assumption that we should observe a scaling relation of the type

$$N_\epsilon \propto \frac{1}{\epsilon^d},$$

where d is the capacity dimension and N_ϵ is the number of neighborhoods of radius ϵ needed to cover the set in question. In particular,

$$\log N_\epsilon \propto d \log \frac{1}{\epsilon} \quad \text{or} \quad d \propto \frac{\log N_\epsilon}{\log \epsilon^{-1}}.$$

Thus, we can approximate d by choosing several values of ϵ, finding N_ϵ, and calculating the slope of the best linear fit to the data set $\left\{ \left(\log N_\epsilon, \log \frac{1}{\epsilon}\right) \right\}$. A full derivation of the capacity dimension can be found in [8]. We not that the capacity dimension does not always agree with more rigorous constructions such as Hausdorff dimension. The data structures used in our numerical calculations are well suited to the computation of the capacity dimension, thus it will be our quantity of interest. Figure 3.2 shows our calculated capacity dimension d of the Julia set as a function of the step size δ of our solution algorithm (1.2).

At this point it is important to state that during benchmark testing, our calculated capacity dimensions only agreed with true capacity dimension within an error of 0.08. Also, we know that values of d corresponding to $\delta < 0.2$ are unreliable due to computational limitations. For these small values of δ, we find an abundance of trajectories that do not reach an root within the maximum number of iterations specified by our implementation. These trajectories give the appearance of a fourth, fictitious basin of attraction and thus a larger Julia set. This cap was implemented in order to keep computation times more tractable. Keeping this in mind, Figure 3.2 exhibits some rather unexpected behavior. From a cursory examination

[1] See [8] or [5] for a more extensive discussion of fractal dimension

of Figure 3.1, one might expect that the Julia set J is merely scaled by δ, and that the fractal dimension remains unaffected. Our calculations of d indicate that this is not, in fact, true. The data is not consistent with a constant function. Indeed, it appears that the fractal dimension is a decreasing function of δ. Again, this seems completely counter-intuitive. One might expect the fractal dimension to decrease as $\delta \to 0$ and (1.2) becomes a better approximation of (1.3). Furthermore, based on knowledge of the behavior of (1.3), one might expect that

$$\lim_{\delta \to 0} d = 1.$$

This does not appear to be the case at all. This, combined with the large increase in d for values of $\delta < 0.5$ suggests that the dimension may not depend continuously on the step size of the approximation. Furthermore, a cursory examination of Damped Newton's Method for other 3^{rd} degree polynomials has produced remarkably similar data, with a characteristic drop in fractal dimension for values of $\delta \geq 0.5$.

3.3 A More Sophisticated Approximation

All results up to this point suggest that discretization of (1.3) with an Euler approximation inherently creates a set of fractal nature, upon which the system exhibits nontrivial dynamics. However, keeping in mind that Newton's Method can be derived from a continuous dynamical system, the question immediately arises: Does this behavior persist under more sophisticated numerical integration techniques? Specifically, we wish to investigate the behavior of (1.3) when a 4^{th} order Runge-Kutta method is employed to approximate trajectories. [2] In particular, the Runge-Kutta method employs a similar time step to that of the Euler approximation, which will also be referred to as δ, and we wish to explore how the basins of attraction vary with respect to δ.

Figure 3.3 shows the 3 basins of attraction under the Runge-Kutta approximation for several step sizes δ. Immediately, we see that these are not the same Julia sets found when using an Euler approximation. If anything, these pictures indicate a higher level of complexity than we find using an Euler approximation. The large connected component of each basin appears to be smaller, and the radial, braided structures are quite intricate.

[2] For a derivation of the Runge-Kutta method employed here, see [4].

Indeed, Figure 3.3 indicates that the Julia set we see with an Euler approximation of (1.3) is less complex (in that it has a lower fractal dimension) than the Julia set associated with a Runge-Kutta approximation, but only for larger values of δ. If we interpret this to mean that the Runge-Kutta algorithm is unstable at a "larger" set of points, then these findings stand in contrast to the fact that Runge-Kutta is a better approximation scheme than Euler.

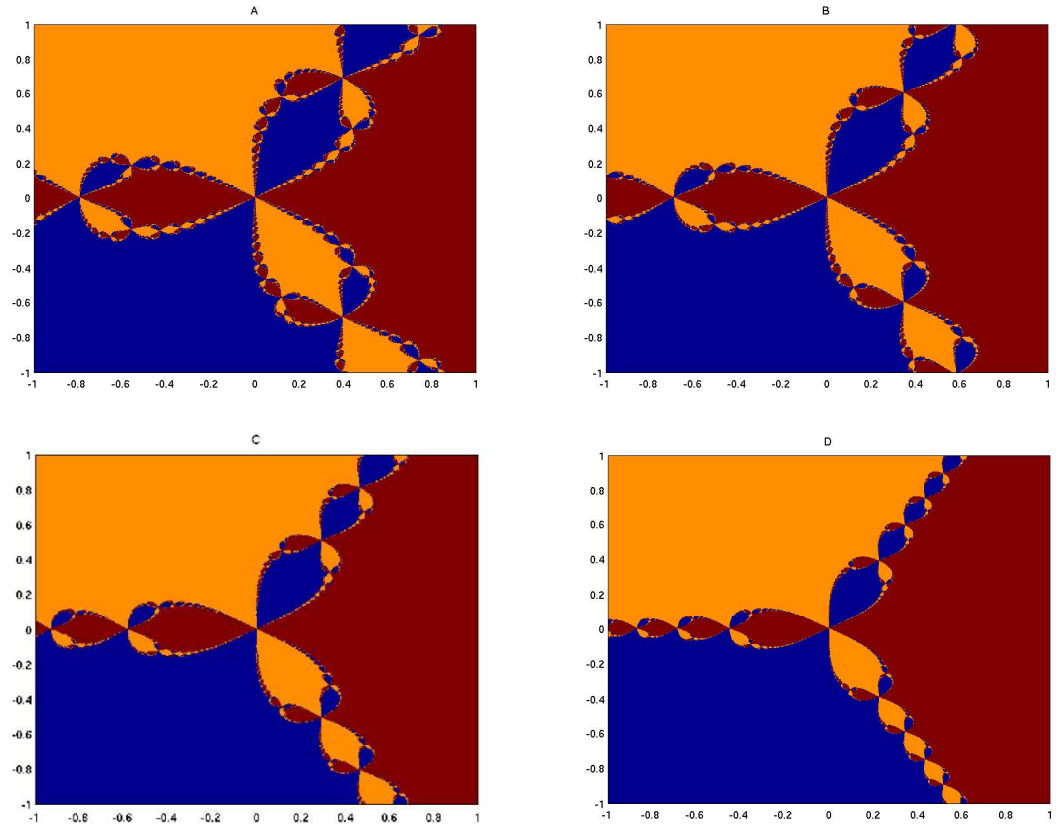


Figure 3.1: Damped Newton's Method for the polynomial $p(z) = z^3 - 1$. The basins of attraction of the 3 roots of unity are shown for (A) $\delta = 1$, (B) $\delta = \frac{3}{4}$, (C) $\delta = \frac{1}{2}$, and (D) $\delta = \frac{1}{4}$.

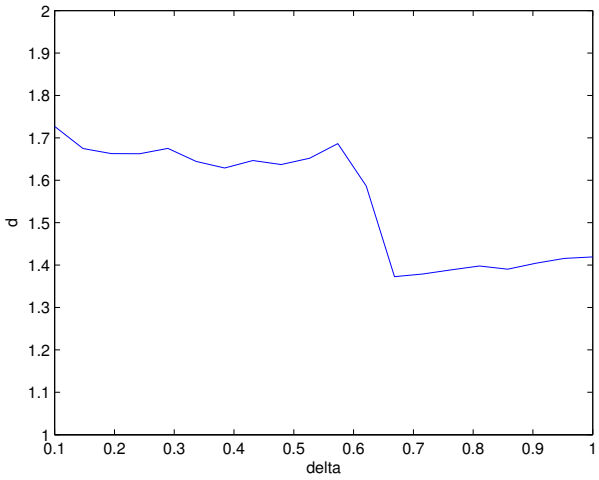


Figure 3.2: Dimension of the Julia set for (1.2) as a function of δ when $p(z) = z^3 - 1$.

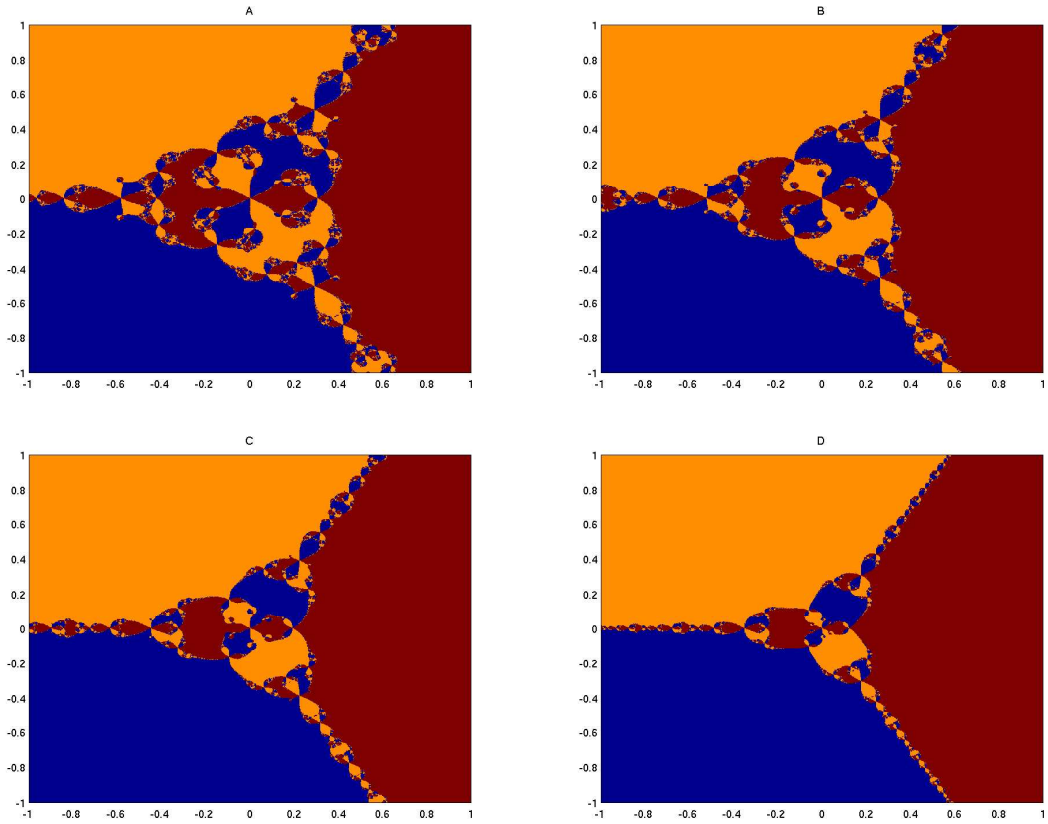


Figure 3.3: Runge-Kutta approximation of (1.3) for $p(z) = z^3 - 1$. The basins of attraction of the 3 roots of unity are shown for (A) $\delta = 1$, (B) $\delta = \frac{3}{4}$, (C) $\delta = \frac{1}{2}$, and (D) $\delta = \frac{1}{4}$.

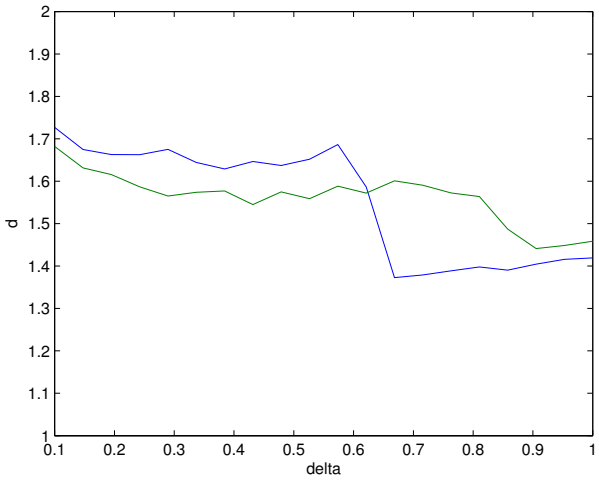


Figure 3.4: Dimension of the Julia set as a function of δ for both approximations of (1.3) when $p(z) = z^3 - 1$. The Euler approximation is shown in blue, while the Runge-Kutta approximation is shown in green. Both curves contain 20 equally spaced data points.

Chapter 4

Newton's Method in $\mathbb{R}^n$

4.1 Developing an Example

In many contexts, one may be concerned with finding the zeros of some f, where f is *not* a complex analytic function. Engineers are often concerned with finding zeros of functions in $\mathbb{R}^n$ for rather large values of n. As we have discussed, in this case Newton's Method generalizes to equation (1.6), and the corresponding continuous method (1.7). In order to investigate the behavior of (1.7) it is necessary for us to choose an example G. The general system permits too wide a range of behavior for the analysis to be tractable. For the time being, we will consider the example G studied in [10]. We begin by posing the one-dimensional nonlinear boundary value problem

$$\begin{cases} u_{ss} + \lambda f(u) = 0, \quad 0 < s < 1, \\ u(0) = u(1) = 0, \end{cases} \tag{4.1}$$

where λ is a real parameter. It may be helpful to think of (4.1) as describing the steady state of a reaction-diffusion system on the unit interval. In order to approximate a solution u to (4.1), we discretize the interval $(0, 1)$ using a uniform mesh of n interior points

$$\begin{aligned} s_i =& \frac{i}{n+1}, \; i = 1, \ldots, n \\ x_i =& u(s_i), \; \Delta s = \frac{1}{n+1}. \end{aligned} \tag{4.2}$$

Using a centered difference method, we approximate (4.1) by the system of n equations,

$$(n+1)^2(x_{i+1} - 2x_i + x_{i-1}) + \lambda f(x_i) = 0, \quad i = 1, \ldots, n.$$

Collecting these equations and multiplying through by a constant yields

$$G(x) = Mx - \mu F(x) = 0,$$

where

$$\mu = -\frac{\lambda}{(n+1)^2}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

$$M = \begin{bmatrix} 2 & -1 & 0 & \dots \\ -1 & 2 & -1 & \dots \\ 0 & -1 & 2 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

and

$$F(x) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

Thus, a root ξ of G represents an approximate solution to (4.1). In [10], the authors investigate two nonlinearities, $f(s) = s - s^2$ and $f(s) = s - s^3$, however we will restrict ourselves to the former case for the time being. The reason for this choice is that in this case the bifurcation behavior of solutions to (4.1) is well documented, and consequently the roots of G can be predicted somewhat. Furthermore, for our experimentation, we will be utilizing a 2 point interior mesh, and thus using Newton's Method in $\mathbb{R}^2$ to find roots of

$$G(x, y) = \begin{bmatrix} 2x - \mu(x - x^2) - y \\ 2y - \mu(y - y^2) - x \end{bmatrix}. \tag{4.3}$$

4.2 An Euler Approximation

Now that we have a function G for case study we wish to numerically integrate (1.7) in order to study the behavior of solutions. The simplest algorithm for doing this is an Euler approximation. The resulting iterative method is of course (1.6), Newton's Method for $\mathbb{R}^n$, where δ is the time step of the approximation to (1.7).[1] Recall that we first encountered the Julia set as the set of points for which Newton's Method fails to find a zero. However, without the aid of complex analysis, it is much more difficult

[1] It should be noted that this is the technique used in [10].

to characterize such points. In particular, the mappings are not necessarily complex rational maps, and the the theory of Julia and Fatou need not apply.

As we mentioned earlier, in the case of (1.7) there exist trajectories that reach the singular set S in finite time. Thus the authors of [10] propose the following candidates for *generalized Julia* sets, or *Julia-like* sets of (1.6):

- $J_1 = \overline{\{x \in \mathbb{R}^2 : \ N^k(x) \in S, k \geq 0\}}$,
- $J_2 = \overline{\{x \in \mathbb{R}^2 : \ ||N^k(x)|| \to \infty, \text{ as } k \to \infty\}}$.

While it is postulated that $J_2 = J_1$, most of the analysis in [10] deals with J_1. Numerical investigation into the set J_1 has shown a remarkably complex structure. For the examples considered here and in [10] it turns out that while J_1 exhibits self-similarity and other fractal characteristics, it also contains components that are smooth one-manifolds. Thus, J_1 is in actuality a partial fractal.

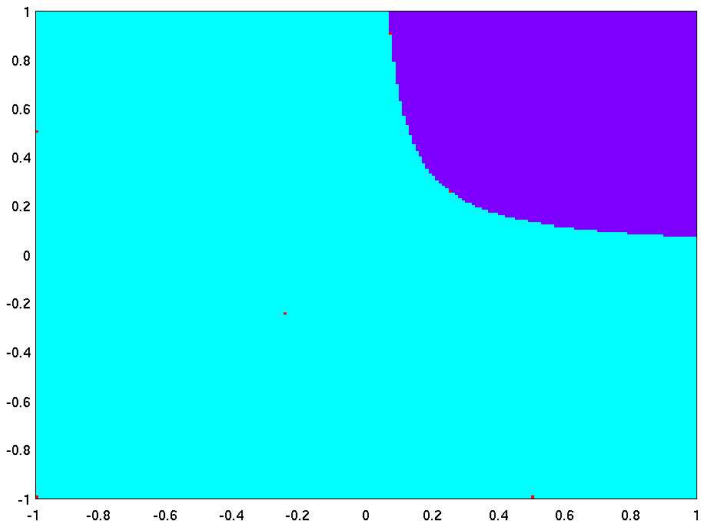


Figure 4.1: Damped Newton's Method for (4.3): $\mu = 2$, $\delta = 1.5$.

In another deviation from the Newton's method for the complex plane, it is no longer the case that the Julia set bounds the basins of all attractive cycles. Figure 4.1 shows the basins of attraction when $\mu = 2$ and an Euler step size of $\delta = 1.5$ is used. For this value of μ, $G(x)$ has two distinct roots, $\xi_0 = (0, 0)$ corresponding to the trivial solution of (4.1), and

$\xi_1 = (x_1, x_1)$, $x_1 > 0$, corresponding to the positive solution. The cyan region of the plane represents $A(\xi_0)$, while the blue region represents $A(\xi_1)$. The faint red regions are a portion of J_1. The remarkable fractal structure that was present in the setting of the complex plane appears to have been lost. However, we have reason to believe that this is not the case, and that higher resolution images in the spirit of Figure 4.1 will reveal more of J_1. In Figure 4.2 the two basins $A(\xi_0)$ and $A(\xi_1)$ have each been divided into four regions, corresponding to the direction with which the sequence of iterates $\{N^n(x)\}$ approaches ξ_i. The interfaces between these 8 regions show a large amount of complexity and suggest a set of fractal nature. Indeed, this structure appears nearly identical to the set J_1 investigated in [10]. If related, this would prove to be a most interesting feature of the Julia-like set, however at this time a claim to that effect cannot be substantiated.

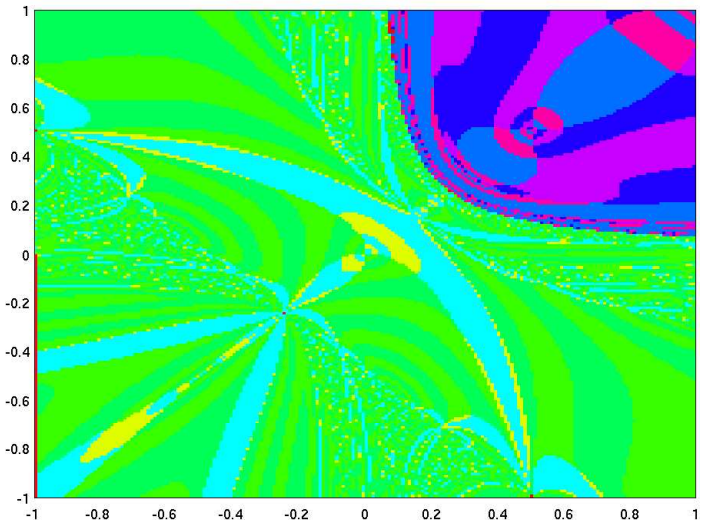


Figure 4.2: Decomposition of the basins of attraction for $\mu = 2$, $\delta = 1.5$.

4.3 Runge-Kutta Approximation

As in Chapter 3, we are interested in examining the behavior of (1.7) under approximation with a Runge-Kutta scheme. Figure 4.3 show the results of our investigation into the basins of attraction for (1.7) using a Runge-Kutta method. $A(\xi_0)$ and $A(\xi_1)$ are shown in green and red respectively.

Again, we see evidence of a Julia-like set with fractal properties. Figures 4.4 and 4.5 show the self-similarity of the interface between $A(\xi_0)$ and $A(\xi_1)$. Interestingly, these figures indicate behavior similar to that of Newton's Method for complex polynomials. It appears that the Julia-like set is again bounding the various basins of attraction. However, altering the step size δ seems to have a more profound impact on the structure of the basins $A(\xi_i)$ than we saw in the setting of the complex plane. Variation of δ appears to alter the fractal set's topology. Exactly how is not clear. Unfortunately, due to prohibitively large computational demands, were only able to conduct a cursory investigation into the capacity dimension this Julia-like set. For the basins of attraction shown in Figure 4.3, we have calculated capacity dimensions of (A)1.86, (B)1.88, (C)1.80, and (D)1.80. This is in fact a smaller variation than witnessed for either approximation method in Chapter 3 and suggests that the capacity dimension may indeed be constant with respect to δ. Once again we find completely unexpected behavior. Throughout this project we have learned that visually gauging the complexity of these basin boundaries is extremely difficult.

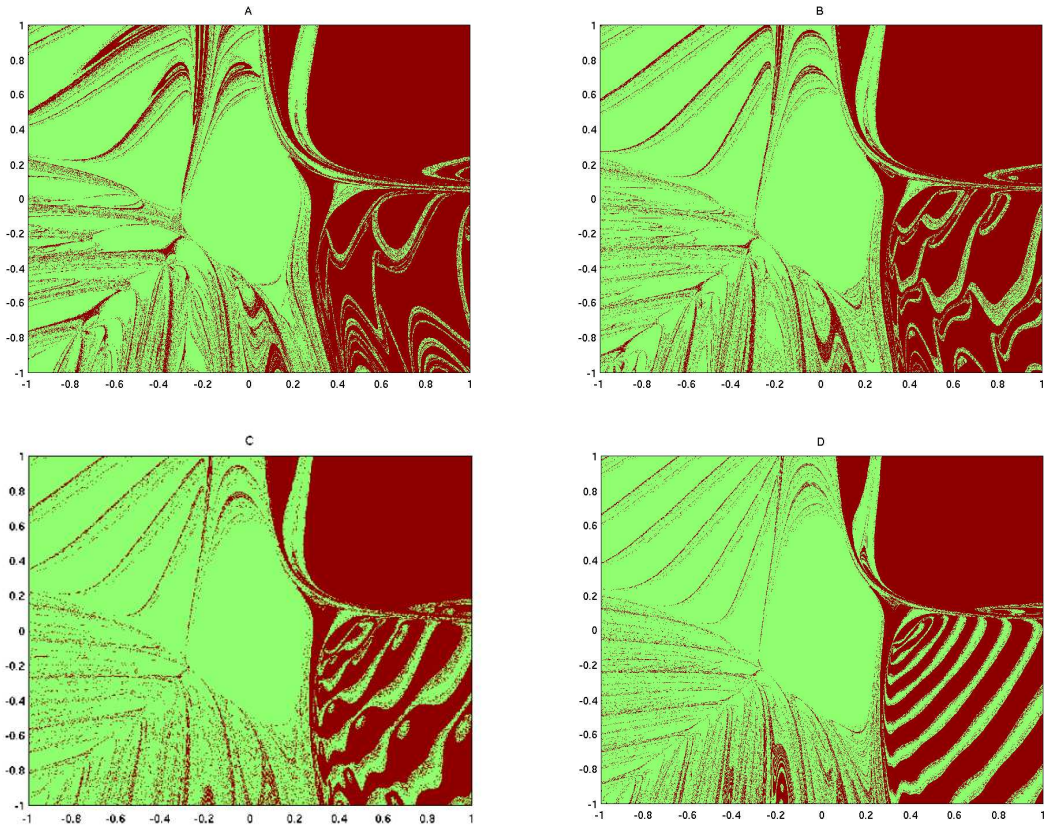


Figure 4.3: Runge-Kutta approximation of (1.7) for (4.3): $\mu = 2$. The basins of attraction of the each fixed point are shown for (A) $\delta = 1$, (B) $\delta = \frac{3}{4}$, (C) $\delta = \frac{1}{2}$, and (D) $\delta = \frac{1}{4}$.

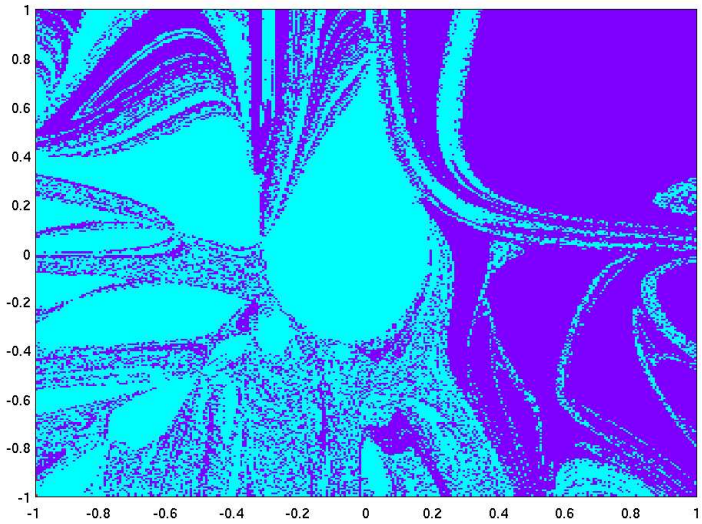


Figure 4.4: Runge-Kutta approximation of (1.7) for (4.3): $\mu = 2$, $\delta = 1.5$.

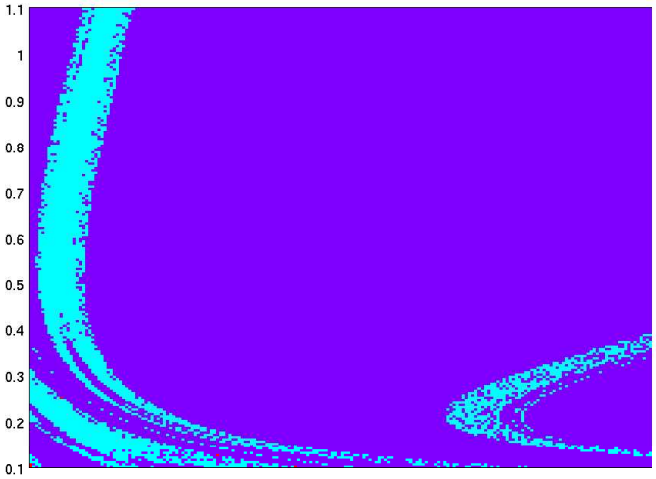


Figure 4.5: Enlarged view of the basins of attraction for $\mu = 2$, $\delta = 1.5$.

Chapter 5

Conclusion

Our investigation into generalizations of Cayley's Problem has uncovered a wealth of unexpected behavior. Perhaps most startling, is the evidence that with respect to the capacity dimension of the associated Julia set, a 4^{th} order Runge-Kutta scheme is actually a less stable algorithm for approximating (1.3) than a simple Euler scheme (in some cases). This seems completely counter-intuitive in light of the fact that the Runge-Kutta scheme can be shown to have a much higher rate of convergence. Furthermore, in the case of (1.7), Runge-Kutta approximation yields basin boundaries with an apparently fractal structure, while an Euler approximation does not. We are currently unable to suitably explain either of these phenomenon, and are of the opinion that this should be a major goal of any future research in this area. Also, it would be interesting to investigate, in more depth, the basin boundaries seen in Figure 4.3. The notion of capacity dimension does not seem to capture the variation with respect to δ that these pictures indicate. Perhaps there is some better way to quantify this.

Finally, we feel our experimentation to this point has suggested a profound question. Motivated by what we have seen in our work with Continuous Newton's Method, one might ask if discretization of (1.7) will always result in a generalized Julia set. The existence of a Julia-like set for these two methods of numerical integration by no means implies that this will always be the case. However, it is entirely possible that Julia-like sets exist for a large family of approximation techniques. Such a finding would open a wide range of research possibilities.

Chapter 6

Appendix

6.1 EULERbasin

The majority of the numerical simulation used in this thesis was done using Matlab®. The standard routine used to do this was *EULERbasin* or a variant there-of. *EULERbasin* creates an array corresponding to a grid of points imposed on $\mathbb{R}^2$, as well as two vectors defining the axes of the grid. The entries of the array are filled with values corresponding to the zero of the given function that Newton's Method carries each point to.

```
%% Eulerbasin
%% Owen Lewis
%% Takes in a symbolic function of 2 variables, axis limits,
%% resolution and step size.  Returns Matrix corresponding to
%% basins of attraction for Newton's Method in R^2 applied
%% to func.

function [u,Xaxis,Yaxis] = EULERbasin(func,xmin,xmax,ymin,ymax,res,delta)

%Create vectors to define the X and Y axis as well as a vector to
%hold fixed points as they are found.
Xaxis = xmin:1/res:xmax;
Yaxis = ymin:1/res:ymax;
fixed = [inf, -inf;0,0];
% minimum threshold at which two numbers are considered equal
eps = 0.0000000001;
%fixed(n) = inf;
tic
```

```
%Paint is the matrix that will hold a integer corresponding to
%one of the fixed points found by EULER2v1.m
Xsiz = length(Xaxis);
Ysiz = length(Yaxis);

Paint = zeros(length(Yaxis),length(Xaxis));

% Here we define the D.E. that Euler's meth will be applied to
% This DE is continuous newton's method for the function
syms x y D F;
fx = diff(func,x);
fy = diff(func,y);

D = [fx,fy];

F = -D^-1*func;

h = waitbar(0,'Please wait');

%For each of the rows and columns, EULER2v1.m is run,
%then a check is performed to see if the fixed point found is
%already in 'fixed'.  If it is, the corresponding entry in 'Paint'
%is changed to reflect the fixed point.  If not, the
%final is added to fixed.
for k = 1:Xsiz
    for j = 1:Ysiz

        waitbar( ((k-1)*Ysiz + j)/(Ysiz*Xsiz),h);

        start = [Xaxis(k) ; Yaxis(j)];

        final = EULER2v1(F,start,delta);
        found = false;

        %% check to see if the fixed point calculated is already known
        for n = 1:size(fixed,2)
            if ( norm(fixed(:,n) - final) < eps || isequal(fixed(:,n),final))
               found = true;
               Paint(j,k) = n;
            end
```

```
        end

        % in the case of a new fixed point, the new point is added to
        % fixed, then fixed is reordered, then the indices in Paint
        % are changed to match foo. Finally, fixed is changed to match foo
        if (found == false)
            fixed = [fixed,final];
            foo = mysort2(fixed);

            for a = 1:size(foo,2)

                for b = 1:size(fixed,2)

                    if (fixed(:,b) == foo(:,a))
                        Old = find(Paint == b);
                        Paint(Old) = a;
                    end

                end

            end
            fixed = foo
        end

    end
end

close(h);
u = Paint;
toc
end
```

6.2 EULER2v1

EULERbasin.m utilizes an auxiliary routine named *EULER2v1.m* to iterate the Newton map (1.6) in $\mathbb{R}^2$.

```
%% EULER2v1
%% Owen Lewis
```

```
%% returns the fixed point obtained if Newton's Method for 'func'
%% is iterated using seed value 'start' and step size 'delta'

function fixed = EULER2v1(func,start,delta)
%Method takes in a symbolic function, a starting
%position for the method, and a step size 'epsilon'

%Several variables are initialized including the current
%position of the method (Xpos,Ypos)
n = 2;
Xpos = start(1);
Ypos = start(2);

syms x y;
F = func;
ticker = 0;
run = true;
% max number of iterations
max = floor(40/delta)*(10*n)^2;
% Threshold at which two points are considered equal
eps = 0.0000000000001;


%Euler method is run until a forced 'return' or the iteration
%limiter is reached

while ( ticker < max )

        x = Xpos;
        y = Ypos;

        Step = delta*eval(F);

        % this is a check to see if the method has reached a singular pt
        if (isnan(norm(Step)))
            'Singularity'
            fixed = [inf;0];
            return
        end
```

```
        % This is a check to see if the method has reached an equilibrium
        if ( norm(Step) < eps )
            %fixed = roundoffn([Xpos;Ypos],8);
            fixed = [Xpos;Ypos];
            return
        end

        % This is a check to see if the method has failed to reach
        % and equilibrium before the iteration limiter is reached.
        if ( ticker == max - 1 )
            'Transient Orbit'
            fixed = [-inf; 0];
            return
        end

        Next = [Xpos;Ypos] + Step;
        Xpos = Next(1);
        Ypos = Next(2);
        ticker = ticker + 1;

end


return;
```

6.3 RK2v1

RK2v1.m is a routine much like *EULER2v1,* used to calculate the fixed point of (1.7) when utilizing a Runge-Kutta approximation. It takes inputs identical to those of *EULER2v1* and returns the same output type, thus the two routines are substitutable.

```
%% RK2v1
%% Owen Lewis
%% returns the fixed point obtained if an RK approx.of Newton's
%% Method for 'func' is iterated using seed value 'start'
%% and step size 'delta'

function fixed = RK2v1(func,start,delta)
%Method takes in a symbolic function, a starting
```

```
%position for the method, and a step size 'delta'

%Several variables are initialized including the current
%position of the method (Xpos,Ypos)
n = 2;
Xpos = start(1);
Ypos = start(2);

syms x y;
F = func;
ticker = 0;
% max number of iterations
max = floor(40/delta)*(10*n)^2;
% Threshold at which two points are considered equal
eps = 0.00000000001;


%Runga-Kutta method is run until a forced 'return' or the iteration
%limiter is reached

while ( ticker < max )

        x = Xpos;
        y = Ypos;

        k1 = delta*eval(F);

        x = Xpos + k1(1)/2;
        y = Ypos + k1(2)/2;

        k2 = delta*eval(F);

        x = Xpos + k2(1)/2;
        y = Xpos + k2(2)/2;

        k3 = delta*eval(F);

        x = Xpos + k3(1)/2;
        y = Ypos + k3(2)/2;
```

```
        k4 = delta*eval(F);

        Step = (k1 + 2*k2 + 2*k3 + k4)/6;

        % this is a check to see if the method has reached a singular pt
        if (isnan(norm(Step)))
            'Singularity'
            fixed = [inf;0];
            return
        end

        % This is a check to see if the method has reached an equilibrium
        if ( norm(Step) < eps )
            fixed = [Xpos;Ypos];
            return


        % This is a check to see if the method has failed to reach
        % and equilibrium before the iteration limiter is reached.
        elseif ( ticker == max - 1 )
            'Transient Orbit'
            fixed = [-inf; 0];
            return
        end

        Next = [Xpos;Ypos] + Step;
        Xpos = Next(1);
        Ypos = Next(2);
        ticker = ticker + 1;

end


return;
```

6.4 boxcounter

The routine *boxcounter.m* recursively cuts the input array into 4 sub-arrays, and checks to see if each is constant. The vectors returned reflect the radius

of a neighborhood circumscribing the sub-array, and the number of sub-arrays needed to cover the Julia set at each level of recursion.

```
%% boxcounter
%% Owen Lewis
%% Calculates the number of squares N(i) with radius S(i)
%% needed to cover the set of points at which Uin changes value.
%% N and S will be 'n'x1 vectors.

function [N,S] = boxcounter(Uin,Xin,Yin,n)

dims = size(Uin);
proc = find(dims < 2);
Xsiz = length(Xin);
Ysiz = length(Yin);
rad = sqrt((Xin(Xsiz) - Xin(1))^2 + (Yin(Ysiz)-Yin(1))^2)/2;

if n > 1
%% First, we calculate where to divide Uin
    diff = find(Uin ~= Uin(1));
    BreakX = floor(Xsiz/2);
    BreakY = floor(Ysiz/2);

%% and break the Uin as well as the
%% axis in half (as nearly as possible)
    X1 = Xin(1:BreakX);
    Y1 = Xin(1:BreakY);
    U1 = Uin(1:BreakY,1:BreakX);

    X2 = Xin(1:BreakX);
    Y2 = Yin(BreakY+1:Ysiz);
    U2 = Uin(BreakY+1:Ysiz,1:BreakX);

    X3 = Xin(BreakX + 1:Xsiz);
    Y3 = Yin(1:BreakY);
    U3 = Uin(1:BreakY,BreakX + 1:Xsiz);

    X4 = Xin(BreakX + 1:Xsiz);
    Y4 = Yin(BreakY + 1:Ysiz);
    U4 = Uin(BreakY + 1:Ysiz,BreakX + 1:Xsiz);
```

```
%% Boxcounter is run recusively on each of the
%% 4 sub matrices
    [N1,S1] = boxcounter(U1,X1,Y1,n-1);
    [N2,S2] = boxcounter(U2,X2,Y2,n-1);
    [N3,S3] = boxcounter(U3,X3,Y3,n-1);
    [N4,S4] = boxcounter(U4,X4,Y4,n-1);


%% Results are truncated to make sure vector dimensions agree
    short = min([length(S1),length(S2),length(S3),length(S4)]);
    N1 = N1(1:short);
    N2 = N2(1:short);
    N3 = N3(1:short);
    N4 = N4(1:short);
    S1 = S1(1:short);
    S2 = S2(1:short);
    S3 = S3(1:short);
    S4 = S4(1:short);


%% Results for 4 submatrices are combined
    Nrec = N1 + N2 + N3 + N4;
    Srec = (S1 + S2 + S3 + S4)/4;

%% N(1) is maked 0 if Uin is constant, 1 otherwise
%% S is the size of Uin
    N = [~isempty(diff);Nrec];
    S = [rad;Srec];

%% Base case (n = 0)
else
    diff = find(Uin ~= Uin(1));
    N = ~isempty(diff);
    S = rad;
end
```

6.5 *Mex* routines

Because the run-time of *EULERbasin.m* and it's variants became prohibitive when attempting to run large-scale parameter studies, we felt the need to find more efficient means to produce basins of attraction. In order to speed up computation, we utilized the *Mex* functionality of Matlab®. *eulerbasins.c* is a C++ class that generates the same array as *EULERbasin.m* and returns it to Matlab®.

```
/**
 * eulerbasins.C
 *
 * This file generates a matrix indicating the root of convergence
 * for initial points on the grid and passes it back to matlab
 *
 * Written by Owen Lewis
 * April, 2005
 */

#include <math.h>
#include <stdio.h>
#include "mex.h"

/* n_IN is the input double from matlab
   y_out is the matrix returned to matlab */

#define n_IN prhs[0]
#define y_OUT plhs[0]

using namespace std;

using namespace std;

double DELTA = 1;
double XMIN = -1.0;
double XMAX = 1.0;
double YMIN = -1.0;
double YMAX = 1.0;
double TICK = 0.002;
double TOL = 0.0000000001;
```

```
int MAXITS = 1000;
int NUMROOTS = 10;

/* Calculates the real and imaginary parts of p and p' */

double re_f(double a, double b) {
    return (a*a*a - 3*a*b*b - 1);
}

double im_f(double a, double b) {
    return (3*a*a*b - b*b*b);
}

double re_fp(double a, double b) {
    return (3*a*a - 3*b*b);
}

double im_fp(double a, double b) {
    return (6*a*b);
}

/* The iteration of newton's method */

void newton(double *point, double del) {
    int cnt = 0;
    double a = point[0];            /* Components of the input */
    double b = point[1];

    double c = 1;               /* Components of the search */
    double d = 1;               /* direction */

    double ref,imf,rfp,ifp;       /* real and imaginary parts of */
                                  /* f and f' evaluated at (a,b) */

    while ((cnt < MAXITS) && (c*c+d*d > TOL*TOL)) {
        cnt++;
        ref = re_f(a,b);
        imf = im_f(a,b);
        rfp = re_fp(a,b);
        ifp = im_fp(a,b);
```

```
        c = (ref*rfp + imf*ifp) / (rfp*rfp + ifp*ifp);
        d = (rfp*imf - ref*ifp) / (rfp*rfp + ifp*ifp);
        a -= del*c;
        b -= del*d;
        if (c*c+d*d <= TOL*TOL){
}
    }

    point[0] = a;
    point[1] = b;

}
/* runs newton's method for each point of the grid */

void basin(double delta, double matrix[]) {

    double *z = new double[2];
    double *roots = new double[2*NUMROOTS];
    int rootcnt = 0;
    int root = -1;

    double j = YMIN;
    double i = XMIN;


    for (int row = 1000; row > -1;row--) {
        i = XMIN;
        for ( int col = 0; col < 1001;col++) {
    z[0] = i;           /* Set our initial guess */
    z[1] = j;


    newton(z,delta);          /* Run newtons method */

    if (rootcnt == 0) {
        roots[2*rootcnt] = z[0];
roots[2*rootcnt+1] = z[1];
rootcnt++;
root = 0;
```

```
    }
    else {
        root = -1;
for (int k=0; k<NUMROOTS; k++) {
                  double xdiff = z[0] - roots[2*k];
    double ydiff = z[1] - roots[2*k+1];

    if (xdiff*xdiff+ydiff*ydiff < TOL*TOL) {
        root = k;
break;
    }
}
if (root == -1) {
      roots[2*rootcnt] = z[0];
    roots[2*rootcnt+1] = z[1];
    root = rootcnt;
    rootcnt++;
}
    }

            matrix[(col)*1001+row] = root;

            i+=TICK;
}
        j+=TICK;
    }
return;
}

/*********************************************************/
/*(program crashes if number if input arguments is wrong)*/
/*(program crashes if output is not saved to variable)   */
/*    THIS IS A MANDATORY SUBROUTINE                     */
/* sets up MATLAB <-> c parameter passing                */
/*********************************************************/
void
mexFunction(int nlhs,
            mxArray *plhs[],
            int nrhs,
            const mxArray *prhs[])
```

```
{

    double *y, *n; /* why can't n be an int and must be cast? */
    int mm;
    double m;
    double *z;

    /* check if number of params is right, for what little good it does */
    if (nrhs != 1) {
     mexErrMsgTxt("One input arguments required.");
    }
    if (nlhs != 1) {
        mexErrMsgTxt("One output argument required.");
    }

    n = mxGetPr(n_IN);
    m = (double)*n;

    y_OUT = mxCreateDoubleMatrix(1001, 1001, mxREAL);
    z = mxGetPr(y_OUT);

    /* make the actual subroutine call */
     basin(m,z);

    return;
}
```

6.6 Graphic User Interface

To simplify the investigation of Julia-like sets for Newton's Method in $\mathbb{R}^2$, we have developed a simple user interface for Matlab®. The necessary files are *FNewtonGUIv1.m* and *FNewtonGUIv1.fig*. Both can be found at *http://www.math.hmc.edu/ olewis/thesis/* along with a brief explanation of the interface.

Bibliography

[1] H. Brolin, *Invariant sets under iteration of rational functions*, Ark. Mat. **6** (1965), 103–144 (1965).

[2] A. Cayley, *Desiderata and suggestions. No. 3.-the Newton-Fourier imaginary problem*, Amer. J. Math **2** (1879), no. 1, 97.

[3] J. H. Curry, L. Garnett, and D. Sullivan, *On the iteration of a rational function: computer experiments with Newton's method*, Comm. Math. Phys. **91** (1983), no. 2, 267–277.

[4] J. D. Faires and R. Burden, *Numerical methods*, second ed., Brooks/Cole Publishing Co., Pacific Grove, CA, 1998, With 1 IBM-PC floppy disk (3.5 inch; HD).

[5] K. J. Falconer, *The geometry of fractal sets*, Cambridge Tracts in Mathematics, vol. 85, Cambridge University Press, Cambridge, 1986.

[6] P. Fatou, *Sure les equations fonctionnelles*, Bull. Soc. Math., France **47–48** (1919–1920), 161–314.

[7] G. Julia, *Memoire sure l'iteration des fonction rationelles*, J. Math. Pures et Appl. **81** (1918), 47–235.

[8] F. C. Moon, *Chaotic vibrations*, A Wiley-Interscience Publication, John Wiley & Sons Inc., New York, 1987.

[9] J. W. Neuberger, *Continuous Newton's method for polynomials*, Math. Intelligencer **21** (1999), no. 3, 18–23.

[10] H.-O. Peitgen, M. Prüfer, and K. Schmitt, *Global aspects of the continuous and discrete Newton method: a case study*, Acta Appl. Math. **13** (1988), no. 1-2, 123–202.

[11] H.-O. Peitgen, D. Saupe, and F. von Haeseler, *Cayley's problem and Julia sets*, Math. Intelligencer **6** (1984), no. 2, 11–20.

[12] D. Saupe, *Discrete versus continuous Newton's method: a case study*, Acta Appl. Math. **13** (1988), no. 1-2, 59–80.

[13] S. Smale, *A convergent process of price adjustment and global Newton methods*, J. Math. Econom. **3** (1976), no. 2, 107–120.

[14] F. von Haeseler and H.-O. Peitgen, *Newton's method and complex dynamical systems*, Acta Appl. Math. **13** (1988), no. 1-2, 3–58.