8-17-2016 12:00 AM

A comparison of solution methods for Mandelbrot-like polynomials

Eunice Y. S. Chan, *The University of Western Ontario*

Supervisor: Robert M. Corless, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Applied Mathematics

Abstract

We compare two different root-finding methods, eigenvalue methods and homotopy methods, using three test problems: Mandelbrot polynomials, Fibonacci-Mandelbrot polynomials, and Narayana-Mandelbrot polynomials. For the eigenvalue methods, using both MATLAB and Maple, we computed the eigenvalues of a specialized recursively-constructed, supersparse, upper Hessenberg matrix, inspired by Piers Lawrence's original construction for the Mandelbrot polynomials, for all three families of polynomials. This led us to prove that this construction works in general. Therefore, this construction is genuinely a new kind of companion matrix. For the homotopy methods, we used a special-purpose homotopy, in which we used an equivalent differential equation to solve for the roots of all three families of polynomials. To solve these differential equations, we used our own ode solver, based on MATLAB's `ode45` routine, which has pole-vaulting capabilities. We had two versions of this ode solver: one in MATLAB, and the other in C++ that implements Bailey's ARPREC package.

Keywords: Root-finding, companion matrix, eigenvalues, homotopy, Mandelbrot polynomials, Fibonacci-Mandelbrot polynomials, Narayana-Mandelbrot polynomials

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Robert M. Corless, for his infinite wisdom and wonderful guidance. Without him, none of this would have been possible.

I would also like to thank my parents for everything that they have done for me for the past 23 years.

Lastly, my heartfelt appreciation goes to everyone in the Department of Applied Mathematics at Western University for making this such an enjoyable experience.

Contents

Abstract i

Acknowledgements ii

List of Figures v

1 Introduction 1
- 1.1 Conditioning . . . 2
- 1.2 Root-finding methods . . . 5

2 Mandelbrot polynomials and matrices 9
- 2.1 Mandelbrot polynomials . . . 9
- 2.2 Condition numbers and pseudozeros . . . 12
- 2.3 Mandelbrot matrices . . . 13
  - 2.3.1 Using full matrices . . . 16
  - 2.3.2 Using sparse matrices . . . 17
- 2.4 Homotopy method . . . 19
- 2.5 Pole-Vaulting . . . 22
  - 2.5.1 Residues . . . 23
  - 2.5.2 Distinctness . . . 25
- 2.6 Our custom ode solver . . . 27
  - 2.6.1 Runge-Kutta Methods . . . 27
- 2.7 Accuracy of the roots . . . 31
  - 2.7.1 Newton polishing . . . 32
- 2.8 Smallest roots . . . 33
- 2.9 Results for homotopy methods . . . 34

3 Fibonacci-Mandelbrot polynomials and matrices 37
- 3.1 Introduction . . . 37
  - 3.1.1 Fibonacci-Mandelbrot polynomials . . . 38
- 3.2 Condition numbers and pseudozeros . . . 39
- 3.3 Fibonacci-Mandelbrot matrices . . . 40
  - 3.3.1 Results . . . 46
- 3.4 Homotopy methods . . . 48
  - 3.4.1 Distinctness . . . 54
  - 3.4.2 Smallest roots . . . 55

3.4.3 Results 56

4 Narayana-Mandelbrot polynomials and matrices **59**
4.1 Introduction 59
4.1.1 Narayana's cows sequence 59
4.1.2 Narayana-Mandelbrot polynomials 60
4.2 Condition numbers and pseudozeros 61
4.3 Narayana-Mandelbrot matrices 62
4.3.1 Results 67
4.4 Homotopy methods 70
4.4.1 Smallest roots 72
4.4.2 Results 75

5 Concluding Remarks **78**
5.1 Future Work 80

Bibliography **87**

Curriculum Vitae **88**

List of Figures

1.1 Plots of all roots of the Mandelbrot (1.1a), Fibonacci-Mandelbrot (1.1b), and Narayana-Mandelbrot (1.1c) polynomials, with cardioid in grey. 8

2.1 Plot of both mininum and maximum condition numbers against degree of the Mandelbrot polynomials. 12
2.2 Roots of $p_{15}(z)$ with $|p_{15}(z)| = 0.1$ in red. 13
2.3 Different regions the Mandelbrot polynomial with a degree of $16,383$, where the roots are not as well-conditioned, with $|p_{15}(z)| = 0.01$ in red. 14
2.4 All $32,767$ roots of $p_{16}(z)$, produced in Maple 2016. 16
2.5 Time taken for eigenvalue computation of Mandelbrot matrices. The slope of the line of best fit is around 2.3. 17
2.6 Time taken for eigenvalue computation using sparse matrices. The slope of the line of best fit is around 1.3. 19
2.7 Diagram demonstrating pole-vaulting technique (p is the pole, and ρ is the radius of the semicircular arc). 22
2.8 Homotopy paths of $p_3(z)$ and contour where $|p_3(z)| = 1$. 23
2.9 Plots of homotopy paths and contour $|p_n(z)| = 1$ of the Mandelbrot polynomials from $n = 4$ to $n = 9$. 24
2.10 Log-log plots of smallest roots s_n of Mandelbrot polynomials (difference from $1/4$) 34
2.11 Time taken to compute roots of Mandelbrot polynomial using a homotopy method. The line of best fit has a slope about 0.92. 35
2.12 All $2,097,151$ roots of the Mandelbrot polynomial $p_{22}(z)$. These roots were produced in MATLAB using our own ODE solver. 36

3.1 Minimum and maximum condition numbers of the roots for Fibonacci-Mandelbrot polynomials. 40
3.2 All 609 roots of $q_{15}(z)$ with $|q_{15}(z)| = 0.2$ in red. 41
3.3 Different regions the Fibonacci-Mandelbrot polynomials where the roots are not as well-conditioned with $|q_{15}(z)| = 0.05$ in red. 42
3.4 Image visualizations of inverses of Fibonacci-Mandelbrot matrices, where -1, 0 and 1 are black, grey, and white respectively, using Maple 2016. 47
3.5 Plots of eigenvalues using MATLAB's `eig` routine. 48
3.6 Computed eigenvalues of $n = 23$, which has a degree of $28,656$ of the Fibonacci-Mandelbrot matrices using Maple. 49
3.7 Time taken to compute eigenvalues of Fibonacci-Mandelbrot matrices. 49
3.8 Homotopy paths for q_5 and contour where $|q_5(z)| = 1$. 52

3.9 Plots of homotopy paths and contours $|q_n(z)| = 1$ of the Fibonacci-Mandelbrot polynomials from $n = 6$ to $n = 11$. 53
3.10 Log-log plots of smallest roots s_n of Fibonacci-Mandelbrot polynomials (difference from $1/4$). 55
3.11 Time taken to compute roots of $q_n(z)$ using homotopy methods. The line of best fit has slope of 0.82. 57
3.12 Plot of all $3,524,577$ roots of the Fibonacci-Mandelbrot polynomial $q_{33}(z)$. The residuals were all smaller than 10^{-4}. 58

4.1 Condition numbers of the roots of the Narayana-Mandelbrot polynomials, $r_n(z)$. 62
4.2 Roots of $r_{21}(z)$ with $|r_{21}(z)| = 0.1$ in red. 63
4.3 Different regions the Narayana-Mandelbrot polynomials where the roots are not as well-conditioned with $|r_{21}(z)| = 0.05$ in red. 64
4.4 Roots of $r_{27}(z)$, which has a dimension of $18,559$. 68
4.5 Results MATLAB gives when evaluating the eigenvalues of $\mathbf{M}_{28}$ using recursion from Equation (4.10), showing numerical artefacts. 69
4.6 Results Maple gives when evaluating the eigenvalues of $\mathbf{M}_{28}$ using recursion from Equation (4.13), again showing numerical artefacts. 69
4.7 Time taken for computing the eigenvalues of the Narayana-Mandelbrot matrices. 71
4.8 Homotopy paths for $r_6(z)$ and contour where $|r_6(z)| = 1$. 73
4.9 Plots of homotopy paths and contour $|r_n(z)| = 1$ of the Narayana-Mandelbrot polynomials from $n = 7$ to $n = 12$. 74
4.10 Smallest roots of the Narayana-Mandelbrot polynomials (difference from $1/4$). 75
4.11 Time taken to compute roots of $r_n(z)$ using homotopy methods. 76
4.12 Roots of $r_{36}(z)$, which has a degree of $578,948$. 77

Chapter 1

Introduction

In this thesis, we explore two different methods for finding roots of Mandelbrot-like problems: eigenvalue methods and homotopy methods. We are interested in finding out which of the two methods is better in solving multivariate polynomial systems, since there are many useful applications such as computer aided geometric design (CAGD). However, instead of looking into multivariate polynomials system for this thesis, we have decided to retract to a simpler problem, which is only univariate, but with large degree. The three families of polynomials that we explore are the Mandelbrot polynomials, Fibonacci-Mandelbrot polynomials, and Narayana-Mandelbrot polynomials. The last two are new to this thesis.

The recursion for the Mandelbrot polynomials [19] is

$$\begin{aligned} p_0 &= 0 \\ p_{n+1} &= z p_n^2(z) + 1 \,, \end{aligned} \tag{1.1}$$

where $n = 0, 1, 2, \ldots$. Dario Bini et al. used the Mandelbrot polynomials to test their package MPSolve (Multiprecision Polynomial Solver), which was originally presented in [3]. MPSolve is a package for the approximation of the roots of a univariate polynomial using the Aberth method. According to Bini's personal website, they were able to compute around 4 million roots of the Mandelbrot polynomials using their updated package, MPSolve 2.0 (see [4] for more details).

The Fibonacci-Mandelbrot polynomials and Narayana-Mandelbrot polynomials are new families of polynomials that are similar to the Mandelbrot polynomials. However, the recursion for both of these families of polynomials are based on their respective sequences, Fibonacci sequence [22] and Narayana's cows sequence [23]. The recursion for the Fibonacci-Mandelbrot polynomials is

$$\begin{aligned} q_0 &= 0 \\ q_1 &= 1 \\ q_{n+1} &= z q_n(z) q_{n-1}(z) + 1 \,, \end{aligned} \tag{1.2}$$

where $n = 1, 2, 3, \ldots$. Similarly, the recursion for the Narayana-Mandelbrot polynomials is

$$\begin{aligned} r_0 &= 1 \\ r_1 &= 1 \\ r_2 &= 1 \\ r_{n+1} &= z r_n(z) r_{n-2}(z) + 1 \,, \end{aligned} \tag{1.3}$$

where $n = 2, 3, 4, \ldots$.

1.1 Conditioning

Before looking into the two root-finding methods that we applied to our three families of polynomials, we first should look at the condition numbers to see whether the roots of these polynomials are well-conditioned. The following proofs were only done for the Mandelbrot polynomials; however, similar arguments can be made for both the Fibonacci-Mandelbrot and Narayana-Mandelbrot polynomials.

Consider the following for the Mandelbrot polynomials:

$$p_n(z + \Delta z) \doteq p_n(z) + p_n'(z)\Delta z \tag{1.4}$$

If ξ_n is a root of p_n and $\xi_n + \Delta\xi_n$ is a root of $p_n + \Delta p_n$,

$$\begin{aligned} 0 &= (p_n + \Delta p_n)(\xi_n + \Delta\xi_n) \\ &= \cancelto{0}{p_n(\xi_n)} + p_n'(\xi_n)\Delta\xi_n + \Delta p_n(\xi_n) \\ p_n'(\xi_n)\Delta\xi_n &= -\Delta p_n(\xi_n) \\ \therefore \frac{\Delta\xi_n}{\xi_n} &= -\frac{1}{\xi_n p_n'(\xi_n)} \cdot \frac{\Delta p_n(\xi_n)}{1}\,, \end{aligned} \tag{1.5}$$

which means that our absolute condition number is $1/p_n'(z)$ (see [8, Section 3.2.1]). Similarly, our absolute condition numbers for the Fibonacci-Mandelbrot and Narayana-Mandelbrot polynomials are $1/q_n'(z)$ and $1/r_n'(z)$, respectively.

We can also look at the pseudozeros of the Mandelbrot polynomials (see [8, Section 2.4]) to confirm the result that we have gotten from calculating the condition number. Studying the pseudozeros will further help us understand what happens if the coefficients of the Mandelbrot polynomials are somehow changed, such as by measurement error or numerical errors. Given $\varepsilon > 0$, we can define the set of pseudozeros

$$\Lambda_\varepsilon(p_n) = \left\{ z : (p_n + \Delta p_n)(z) = 0 \quad \& \quad |\Delta p_n| \leq \varepsilon \right\} \tag{1.6}$$

Since

$$p_n(z) + \Delta p_n(z) = 0\,, \tag{1.7}$$

this means that

$$|p_n(z)| = |-\Delta p_n(z)| \leq \varepsilon \tag{1.8}$$

by construction. Therefore,

$$\Lambda_\varepsilon(p_n) = \left\{ z : |p_n(z)| \leq \varepsilon \right\}, \tag{1.9}$$

which we can use to confirm the fact that the roots of the Mandelbrot polynomials are well-conditioned and can be located accurately.

However, this is not the only way to do pseudozeros; we can also use *polynomial bases* (which we will denote as $\phi_k(z)$) to compute our pseudozeros. Given the weights $\alpha_k \geq 0$, $\varepsilon > 0$, and $0 \leq k \leq n$, our set of pseudozeros can also be defined as

$$\Lambda_\varepsilon = \left\{ z : |p_n(z)| \leq \varepsilon \cdot B(z) \right\} \tag{1.10}$$

where $\Delta p_n = \sum_{k=0}^{n} \Delta c_k \phi_k(z)$, and each $|\Delta c_k| \leq \varepsilon \cdot \alpha_k$, and $B(z) = \sum_{k=0}^{n} \alpha_k |\phi_k(z)|$. In the monomial basis, $B(z)$ grows exponentially in d_n, doubly exponentially in n; essentially, small changes in the coefficients force large changes in the value of $p_n(z)$. The same argument can be made for both the Fibonacci-Mandelbrot and Narayana-Mandelbrot polynomials to show that their roots are also well-conditioned (when the monomial basis is not used).

Additionally, we need to look into the numerical stability of our recurrence relation. The rounding errors made in the computation of the floating-point evaluation of the Mandelbrot polynomials is, using the IEEE model [14]

$$\text{fl}(zp_n^2 + 1) = z \cdot \text{fl}(p_n)^2(1 + \delta_1)(1 + \delta_2)(1 + \delta_3) + 1 \cdot (1 + \delta_3), \tag{1.11}$$

where $(1 + \delta_j)$ represents the rounding error (due to floating point numbers) that is introduced when an arithmetic operation is performed. We can rewrite Equation (1.11) as

$$\hat{p}_{n+1}(z) = \text{fl}\left(p_{n+1}(z)\right) = z \cdot \text{fl}\left(p_n(z)\right)^2 (1 + \theta_3) + (1 + \theta_1) \ . \tag{1.12}$$

The notation θ_3 is taken from [14]. It means that roughly 3 rounding errors,

$$|\theta_n| \leq \frac{n\mu}{1 - n\mu}\,, \tag{1.13}$$

where μ is machine precision. Knowing that the recursion for the Mandelbrot polynomials is $p_{n+1}(z) = zp_n^2(z) + 1$, we get the following

$$\begin{aligned}
\hat{p}_{n+1}(z) - p_{n+1}(z) &= z\hat{p}_n^2(z)\,(1 + \theta_3) + 1 + \theta_1 - zp_n^2(z) - 1 \\
&= z\,(\hat{p}_n(z) + p_n(z)) \cdot (1 + \theta_3) \cdot (\hat{p}_n(z) - p_n(z)) + \theta_1 \\
e_{n+1} &\doteq 2zp_n(z) \cdot e_n\,(1 + \theta_3) + \theta_1\,.
\end{aligned} \tag{1.14}$$

Therefore,

$$e_n = (2z)^{n-1} \cdot (p_{n-1}(z) \cdot p_{n-2}(z) \cdots p_1(z))\,e_1 + O(\theta_n)\,. \tag{1.15}$$

From Equation (1.15), we are particularly interested in the $(2z)^{n-1} \cdot (p_{n-1}(z) \cdot p_{n-2}(z) \cdots p_1(z))$ part. Since we know that the largest root of the Mandelbrot polynomials is approximately -2, substituting this into $(2z)^{n-1}$, where n is, for example, 8, we get quite a large number, which is around $16,000$. Therefore, we need to see whether $(p_{n-1}(z) \cdot p_{n-2}(z) \cdots p_1(z))$ is small enough to keep e_n small. Using Maple, we find that the computed values of $(2z)^{n-1} \cdot (p_{n-1}(z) \cdot p_{n-2}(z) \cdots p_1(z))$ ranges from 0 to about $10,000$ when $n = 8$. Since our largest value is less than 4^{n-1}, this means that our recursion is mildly unstable: d_n^2 is an acceptable factor, about what you would expect for a random polynomial.

1.2 Root-finding methods

One of the methods that we explored for root-finding is using eigenvalue methods, using these three families of polynomials as test problems. For this method, we computed the eigenvalues of companion matrices in order to solve for the roots of the polynomials. Instead of using

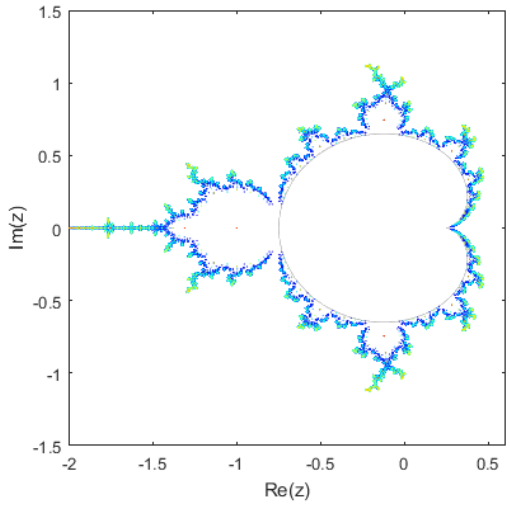
conventional companion matrix constructions, such as the ones found in [13], [8, Chapter 6], and [6], we used a supersparse [1] recursively-constructed upper Hessenberg companion matrix, inspired by Piers Lawrence. In his paper [9], he introduced this construction for the Mandelbrot polynomials. We then used a similar construction for the Fibonacci-Mandelbrot polynomials and Narayana-Mandelbrot polynomials. The surprising analogy between all three families of supersparse companion matrices led us to prove that this construction works in general (proof in Chapter 5), leading us to a genuinely a new kind of companion matrix that can offer better numerically-conditioned for unimodular polynomials.

This new kind of companion matrix also fall into the class of Bohemian matrices, which stands for BOunded HEight Matrix of Integers [11]. As will be seen later, the companion matrices for these families of polynomials only contain elements $\{-1, 0\}$ for the Mandelbrot matrices, and $\{-1, 0, 1\}$ for the Fibonacci-Mandelbrot and Narayana-Mandelbrot matrices.

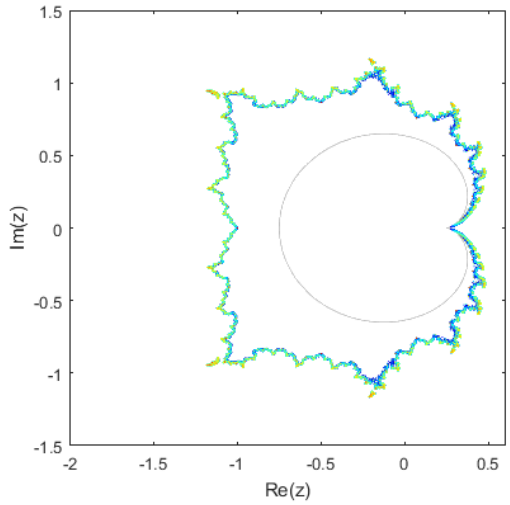
The second method that we looked into for finding roots is *homotopy continuation methods*. One early work about homotopy methods is [17], which claims that the amount of computational work is roughly proportional to the number of solutions. For each family of polynomials, we used a special-purpose homotopy equivalent to a differential equation. Using the previous roots as our initial conditions, the solutions of these differential equations end up being the roots of the polynomial that we are trying to solve. When we first started, to solve these differential equations, we used Shampine & Reichelt's Odesuite in MATLAB (more specifically `ode45`) [20]. However, because we needed to avoid singularities when integrating, we decided to write our own ode solver (described in Chapter 2), which is very similar to MATLAB's `ode45` routine, but has pole-vaulting capabilities. Realizing that we need to use multiple precision to compute the roots of higher iterations, we also implemented a version that uses Bailey's ARPREC package [2] in C++. Using homotopy methods, we were also able to look at the smallest roots of the Mandelbrot, Fibonacci-Mandelbrot, and Narayana-Mandelbrot polynomials.

[1] A matrix is supersparse if, it is sparse and its nonzero elements are drawn from a small set, e.g. $\{-1, 1\}$

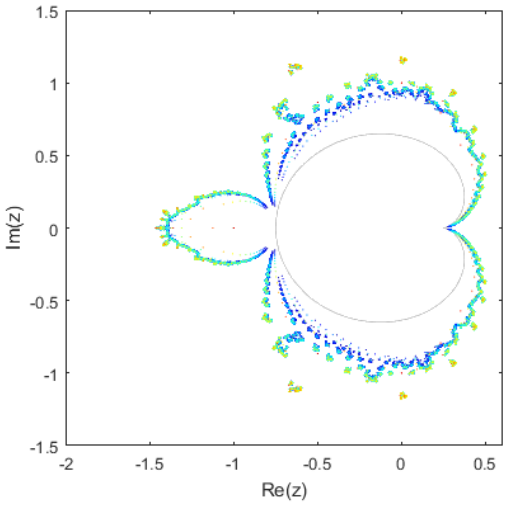
Altogether, these two different methods for root-finding were tested in three families of polynomials: Mandelbrot polynomials (Chapter 2), Fibonacci-Mandelbrot polynomials (Chapter 3), and Narayana-Mandelbrot polynomials (Chapter 4). Figure 1.1 shows all of the roots (overlaid on top of each other with the roots for the largest degree at the bottom) for each family of polynomials, each with a grey cardioid to allow comparison of the size of the roots of each family.



(a) Roots of Mandelbrot polynomials from $n = 3$ to $n = 19$.



(b) Roots of Fibonacci-Mandelbrot polynomials from $n = 4$ to $n = 30$.



(c) Roots of Narayana-Mandelbrot polynomials from $n = 4$ to $n = 36$.

Figure 1.1: Plots of all roots of the Mandelbrot (1.1a), Fibonacci-Mandelbrot (1.1b), and Narayana-Mandelbrot (1.1c) polynomials, with cardioid in grey.

Chapter 2

Mandelbrot polynomials and matrices

2.1 Mandelbrot polynomials

Mandelbrot polynomials, which are used by Bini et al. [4, 3] as a test problem, are based on the Mandelbrot set [19]

$$z_{n+1} = z_n^2 + c \,, \tag{2.1}$$

where $z_0 = 0$ and c is a complex constant. We can simplify Equation (2.1) by dividing everything by c, removing the trivial root $c = 0$:

$$\frac{z_{n+1}}{c} = c\left(\frac{z_n}{c}\right)^2 + 1 \,. \tag{2.2}$$

From this, we can rename the variables in Equation (2.2) to get the Mandelbrot polynomial, which is defined by the recurrence relation

$$\begin{aligned} p_0(z) &= 0 \\ p_{n+1}(z) &= z p_n^2(z) + 1 \,, \end{aligned} \tag{2.3}$$

where $n = 0, 1, 2, 3, \ldots$.

Expanding Equation (2.3) using the monomial basis expansion, the first six polynomials of $p_n(z)$ are

$$
\begin{aligned}
p_0(z) &= 0 \\
p_1(z) &= 1 \\
p_2(z) &= z + 1 \\
p_3(z) &= z^3 + 2z^2 + z + 1 \\
p_4(z) &= z^7 + 4z^6 + 6z^5 + 6z^4 + 5z^3 + 2z^2 + z + 1 \\
p_5(z) &= z^{15} + 8z^{14} + 28z^{13} + 60z^{12} + 94z^{11} + 116z^{10} + 114z^9 \\
&\quad + 94z^8 + 69z^7 + 44z^6 + 26z^5 + 14z^4 + 5z^3 + 2z^2 + z + 1 \, .
\end{aligned} \tag{2.4}
$$

A few noticeable properties of the Mandelbrot polynomials include [9]:

1. The degree of $p_n(z)$ for $k > 0$ is $d_n = 2^{n-1} - 1$.

2. The roots of $p_n(z)$ are periodic points of the Mandelbrot set with period n.

3. The coefficients of $p_n(z)$ when expressed in the monomial basis are nonnegative integers. Indeed $p_n(z)$ is unimodular [1].

4. Derivatives can be computed from the recurrence relation by $p'_0(z) = 0$ and for all $n \geq 0$ by

$$p'_{n+1}(z) = p_n(z) \left(p_n(z) + 2z p'_n(z) \right) \, .$$

5. $p_n(z)$ and $p'_n(z)$ can simultaneously be evaluated by their recurrence relations at a cost of $O(n)$ or $O(\ln d_n)$ operations.

Proof of Property 1 From the recurrence relation, shown in Equation (2.3), it can easily be

[1] Polynomials that are unimodular have coefficients that increase to a unique maximum then decrease when the terms are ordered by their degree.

seen that the degree of the polynomials is

$$d_n = 2 \cdot d_{n-1} + 1 \, . \tag{2.5}$$

We can prove by induction that the degree of the Mandelbrot polynomials can be expressed as

$$d_n = 2^{n-1} - 1 \, . \tag{2.6}$$

First, we can substitute $n = 1$ into Equation (2.6):

$$\begin{aligned} d_1 &= 2^{1-1} - 1 \\ &= 2^0 - 1 \\ &= 1 - 1 = 0 \, , \end{aligned} \tag{2.7}$$

which we can see from $p_1(z)$ in Equation (2.4) is true. Assuming when $n = k$, $d_k = 2^{k-1} - 1$ is true. Then, when $n = k + 1$,

$$\begin{aligned} d_{k+1} &= 2 \cdot d_k + 1 \\ &= 2 \cdot (2^{k-1} - 1) + 1 \\ &= 2 \cdot 2^{k-1} - 2 + 1 \\ &= 2 \cdot 2^{k-1} - 1 \\ &= 2^{k+1-1} - 1 \end{aligned} \tag{2.8}$$

which matches to Equation (2.6). This, therefore, proves that the degree of the Mandelbrot polynomials is $d_n = 2^{n-1} - 1$.

Sketch of Property 2 The roots of $p_n(z)$ are periodic points of the Mandelbrot set with period n. For example, $p_n(-1) \in \{0, 1\}$. Indeed, $p_2(-1) = 0$, $p_3(-1) = 1$, $p_4(-1) = 0$, and the cycle repeats after that.

2.2 Condition numbers and pseudozeros

We will first look at the condition numbers of the roots of the Mandelbrot polynomials. As mentioned in Chapter 1, the condition number for the roots is $1/p_n'(z)$. Figure 2.1 shows the log-log plot of both the minimum and maximum condition numbers against the degree of the Mandelbrot polynomials. Here, the circles represent the maximum condition numbers, in which the minimum $\left|p_n'(\xi_n)\right|$, where ξ_n is a root of $p_n(z)$, were used to compute our condition numbers. It can be seen that none of the condition numbers here exceed the value of 1. On the other hand, the crosses are the minimum condition numbers. Looking at the line of best fit running through these data points, some roots become better conditioned as the degree of the polynomials increase. The slope of this line is around -2 (the lower line with a slope of -2 is there for reference).

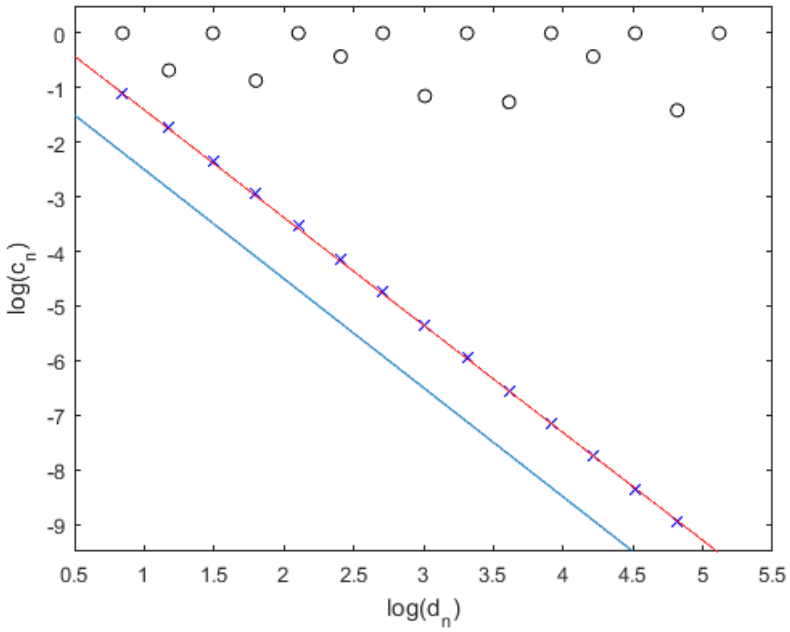


Figure 2.1: Plot of both mininum and maximum condition numbers against degree of the Mandelbrot polynomials.

We can also plot the pseudozeros of the Mandelbrot polynomials by the plotting the contours of $p_n(z) = \varepsilon$, where ε is an arbitrarily small number. Figure 2.2 illustrates this idea: it shows where the contours $|p_{15}(z)| = 0.1$ lie. Most of the contours that surround each of the

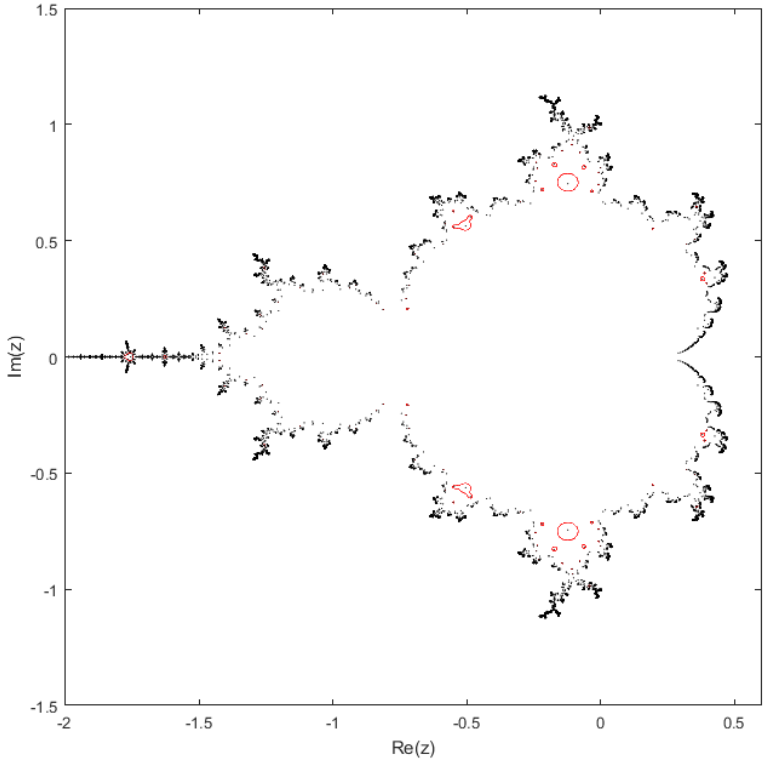


Figure 2.2: Roots of $p_{15}(z)$ with $|p_{15}(z)| = 0.1$ in red.

roots are too small to be seen, meaning that these roots are well-conditioned. However, the roots in which the contours that surrounds it can be seen are those roots that are not as well-conditioned. Figure 2.3 shows $|p_{15}(z)| = \varepsilon$, where $\varepsilon = 0.01$, in red, at the most interesting parts of the Mandelbrot polynomials, where the contours can be seen in Figure 2.2, but has the contour $|p_{15}(z)| = 0.01$ plotted instead.

Knowing that the roots are well-conditioned, we can now look at two different methods in solving the Mandelbrot polynomials: an eigenvalue method and a homotopy method.

2.3 Mandelbrot matrices

The Mandelbrot matrices, first thought of by Piers Lawrence [9], are recursively-constructed upper Hessenberg matrices that only contains $\{-1, 0\}$, in which the eigenvalues are the roots of

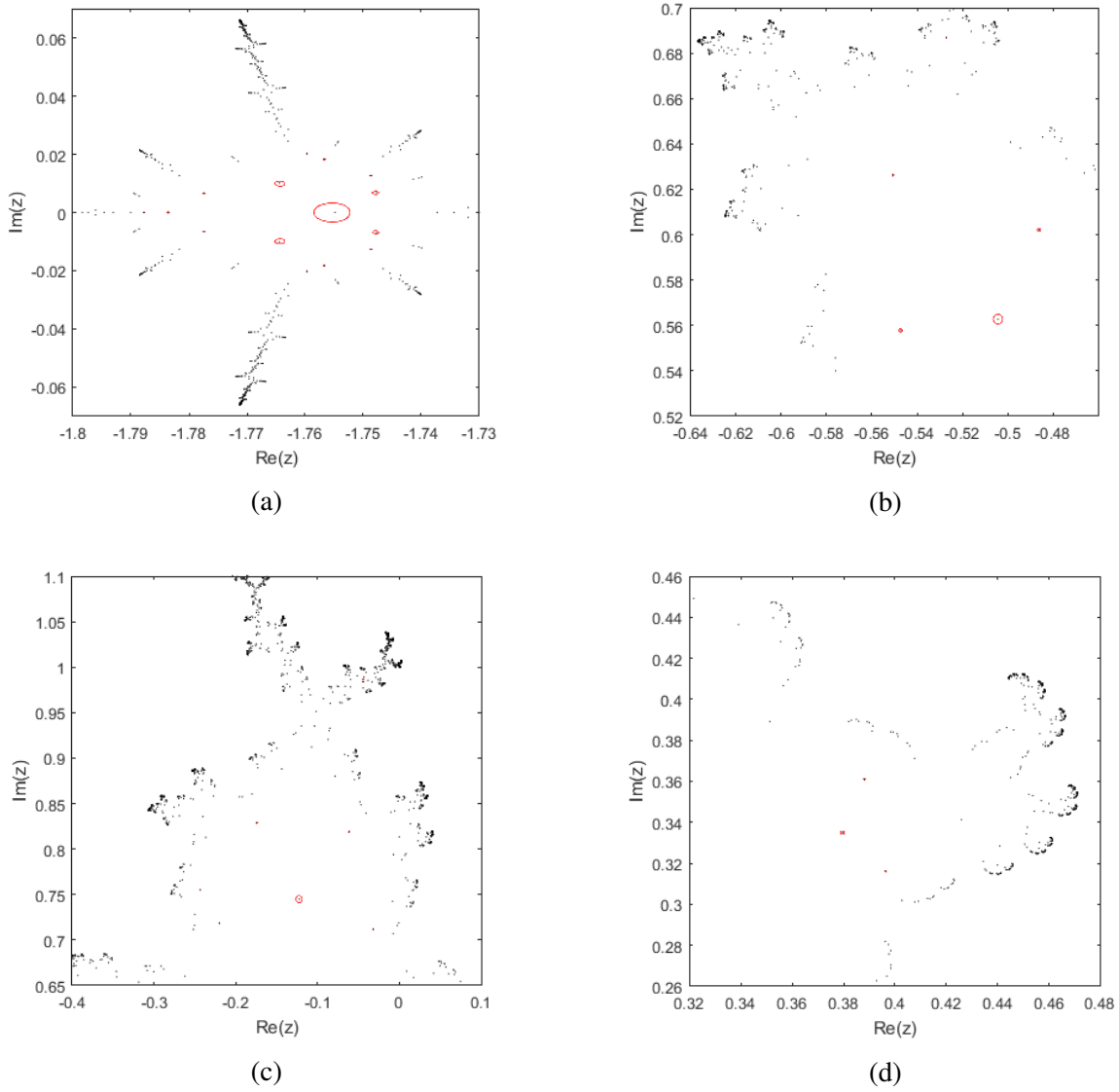


Figure 2.3: Different regions the Mandelbrot polynomial with a degree of $16,383$, where the roots are not as well-conditioned, with $|p_{15}(z)| = 0.01$ in red.

the corresponding Mandelbrot polynomials. We begin our recursive matrices with

$$\mathbf{M}_2 = \begin{bmatrix} -1 \end{bmatrix}, \tag{2.9}$$

which corresponds to $p_2(z)$. It is obvious that the eigenvalue of Equation (2.9) is -1, which is clearly the root of $p_2(z) = z + 1$. Let $\mathbf{r}_n = \begin{bmatrix} 0 & 0 & \dots & 1 \end{bmatrix}$ and $\mathbf{c}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$, where the length of both vectors are $d_n = 2^{n-1} - 1$. Then, our matrix construction is

$$\mathbf{M}_{n+1} = \begin{bmatrix} \mathbf{M}_n & & -\mathbf{c}_n\mathbf{r}_n \\ -\mathbf{r}_n & 0 & \\ & -\mathbf{c}_n & \mathbf{M}_n \end{bmatrix},$$

for all $n > 1$. The first few Mandelbrot matrices are

$$\mathbf{M}_3 = \begin{bmatrix} -1 & 0 & -1 \\ -1 & 0 & 0 \\ & -1 & -1 \end{bmatrix}, \tag{2.10}$$

and

$$\mathbf{M}_4 = \begin{bmatrix} -1 & 0 & -1 & & & & -1 \\ -1 & 0 & 0 & & & & \\ & -1 & -1 & & & & \\ & & -1 & & & & \\ & & & -1 & -1 & 0 & -1 \\ & & & & -1 & 0 & 0 \\ & & & & & -1 & -1 \end{bmatrix}. \tag{2.11}$$

Evaluating the eigenvalues of both $\mathbf{M}_3$ and $\mathbf{M}_4$, we can see that the eigenvalues are the roots of $p_3(z)$ and $p_4(z)$ respectively. We can show that $p_n(z) = \det(z\mathbf{I} - \mathbf{M}_n)$ for all $n > 0$ using induction and the Schur complement [24], which will be shown in Chapter 5.

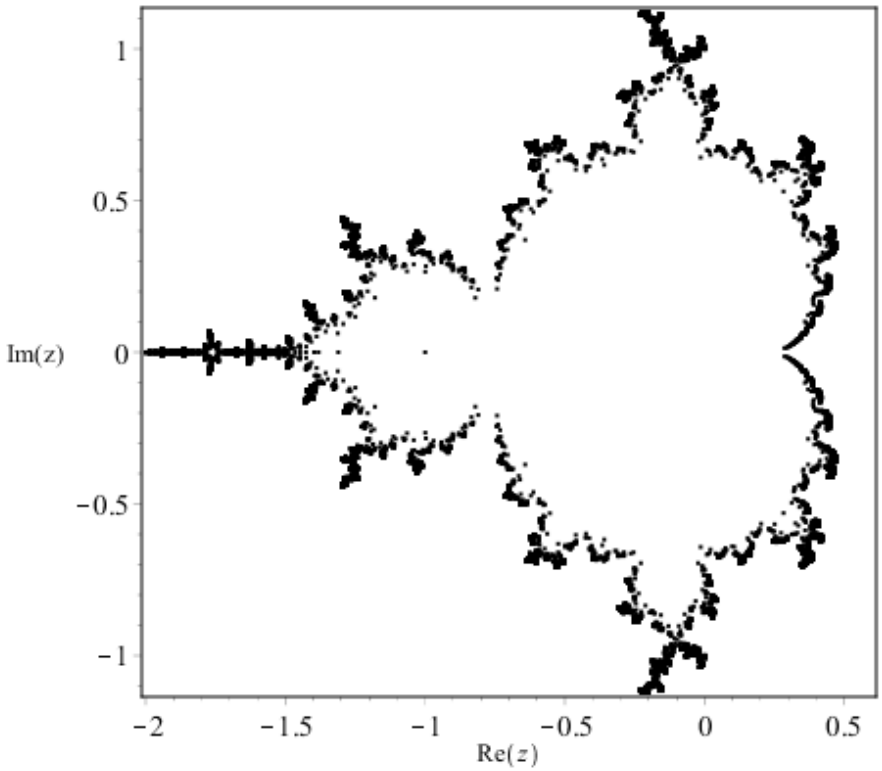


Figure 2.4: All 32,767 roots of $p_{16}(z)$, produced in Maple 2016.

2.3.1 Using full matrices

Using Maple 2015, we were able to compute up to $n = 16$, which is 32,767 roots using a machine with 32 GB of memory, shown in Figure 2.4. Although the Mandelbrot matrices are sparse, initially, for simplicity's sake, we used full matrices in our eigenvalue computation. Figure 2.5 shows the time taken to compute the eigenvalues of the Mandelbrot matrices as the dimension d_n of the matrix increases. As you can see from the figure, the line fitted to the data (the bottom red line) is not as steep as the reference line (the top line), which has a slope of 3. In fact, the slope of the line of fit is around 2.3. Therefore, this method has a time complexity of order less than $O(d_n^3)$. This fact is surprising because we expected this method to have a time complexity of exactly $O(d_n^3)$. We believe that the time complexity calculated here is less than what we expected because the algorithm effectively uses a divide and conquer approach to solve for the eigenvalues due to the structure of these companion matrices, thus reducing the time complexity. In detail, we believe (but have not proved) that the matrices reduce quickly,

breaking into roughly equal halves.

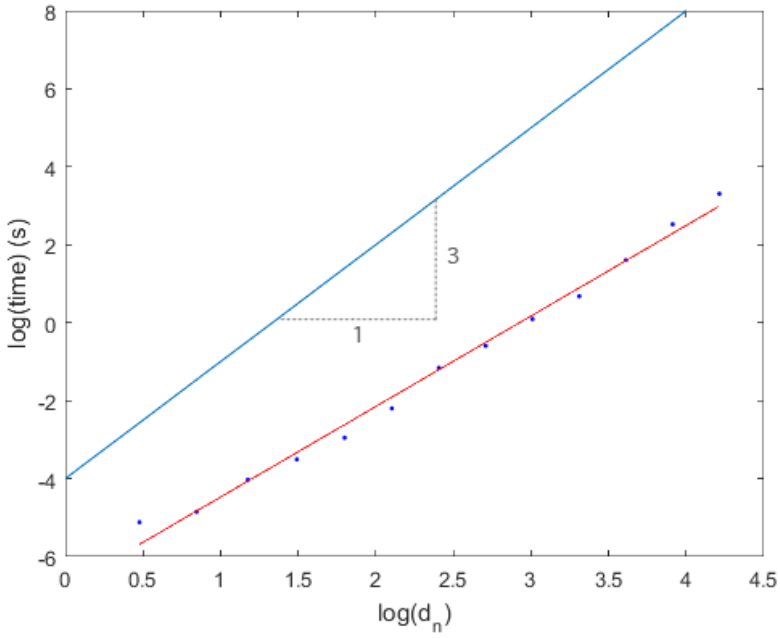


Figure 2.5: Time taken for eigenvalue computation of Mandelbrot matrices. The slope of the line of best fit is around 2.3.

Since we are using full matrices in our eigenvalue computations, this means that the space complexity is quite large, $O(d_n^2)$, and since the matrices that we are working with are quite sparse, most of the numbers that are stored would actually be zeros. Therefore, using the sparse data type would help us save space when storing our matrices, and hopefully help us compute more roots. However, computing eigenvalues using sparse data structures do present some difficulties which will be discussed in the next subsection.

2.3.2 Using sparse matrices

To take advantage of the sparseness of the matrices when solving for the eigenvalues, we can use MATLAB's `eigs` routine, which uses Arnoldi iteration to solve for eigenvalues. Unlike solving for eigenvalues using full matrices, we cannot solve for all of the eigenvalues at once. Instead, we have to look at different regions and compute the eigenvalues that are in each region of interest, and then piece all of the results together. This introduces some difficulties when

trying to find all of the eigenvalues of the Mandelbrot matrices.

The first issue is determining what regions to look at that would help us compute our eigenvalues. Borrowing the idea from homotopy methods (which will be discussed later in this chapter), we can use the roots from the previous iteration to help us locate the new roots. However, for matrices of higher dimensions, it becomes increasingly difficult to locate all of the eigenvalues. Therefore, as the dimension of the matrices increases, the number of eigenvalues that need to be found at each region also increases. Also, since we are computing several eigenvalues from each region, it means that we will end up getting duplicates. This introduces another challenge of eliminating all of the duplicate eigenvalues that were computed, so that each eigenvalue only appears once. Since these eigenvalues are computed numerically, the eigenvalue duplicates may not necessarily be exactly the same. This imposes another challenge when comparing two results in determining whether they are in fact duplicates of each other or whether they are distinct roots, but located near each other. Unfortunately, using this routine, we were only able to compute all of the roots up to $n = 13$, which is only $4,095$ roots. In the interest of time, we decided not to pursue this any further.

The time taken to compute these eigenvalues of the Mandelbrot polynomials using sparse matrices can be seen in Figure 2.6. The slope of the line that passes through the data points is around 1.3; the line above it is a reference for the steepness of a slope of 2. Although, here, it shows that the time complexity is of order $O(d_n^{1.3})$, which is less than the time complexity when using full matrices, it does not seem quite correct. However, we are lacking some data since we are only considering the time computed for up to $n = 13$. Once higher dimensions are achieved using this method, the time complexity may increase; it might possibly have a higher time complexity than the method that uses full matrices. Therefore, more research would need to be done.

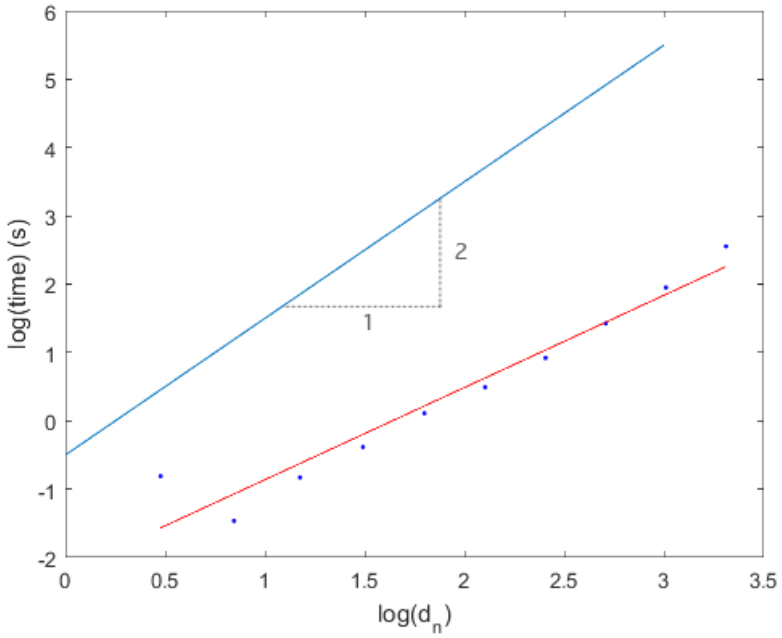


Figure 2.6: Time taken for eigenvalue computation using sparse matrices. The slope of the line of best fit is around 1.3.

2.4 Homotopy method

Consider this special-purpose homotopy

$$H_n(\zeta, \tau) = \zeta(p_{n-1}(\zeta))^2 + \tau^2 \,. \tag{2.12}$$

When $\tau = 0$, it is clear that the zeros of the homotopy is $\zeta = 0$, and the zeros (twice) of $p_{n-1}(z)$, which, in other words, are the roots ξ_{n-1} of the previous polynomials. On the other hand, when $\tau = 1$, $H_n(\zeta, \tau)$ is the same as our Mandelbrot polynomials, which means that the roots of our homotopy are equivalent to the roots of $p_n(z)$. Therefore, by differentiating both sides of Equation (2.12) with respect to τ, we obtain the following differential equation, which

describes a path from the roots of $\zeta(p_{n-1}(\zeta))^2$ to the roots of $p_n(\zeta)$, provided that $p_n'(\zeta) \neq 0$:

$$\begin{aligned}
0 &= H_n(\zeta, \tau) \\
0 &= \frac{\mathrm{d}}{\mathrm{d}\tau} H_n(\zeta, \tau) \\
&= \frac{\mathrm{d}\zeta}{\mathrm{d}\tau} \cdot p_{n-1}^2(\zeta) + 2\zeta \cdot p_{n-1}(\zeta) \cdot p_{n-1}'(\zeta) \frac{\mathrm{d}\zeta}{\mathrm{d}\tau} + 2\tau \\
&= \frac{\mathrm{d}\zeta}{\mathrm{d}\tau} \left(p_{n-1}^2(\zeta) + 2\zeta p_{n-1}(\zeta) \right) + 2\tau \\
&= p_n'(\zeta) \frac{\mathrm{d}\zeta}{\mathrm{d}\tau} + 2\tau \,. \qquad (2.13)
\end{aligned}$$

Thus,

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = -\frac{2\tau}{p_n'(\zeta)}, \quad \tau \in [0, 1] \,. \qquad (2.14)$$

However, since our initial conditions include the roots of the previous polynomial, ξ_{n-1}, it means that we encounter a singularity when we first solve this differential equation numerically. To reiterate, $p_n'(z) = p_{n-1}(z)(p_{n-1}(z) + 2z p_{n-1}'(z))$, and since ξ_{n-1} is a double root of $z p_{n-1}^2(z)$, this means that $p_n'(\xi_{n-1}) = 0$. Additionally, since ξ_{n-1} is a double root, this also means that two new roots ξ_n will stem from this one initial condition. In order to achieve this, we can use Taylor series expansion to perturb our initial conditions so that our solutions will go on two different paths, thus giving us two distinct solutions for each previous root. Let $\zeta = \xi_{n-1} + a\tau + O(\tau^2)$, where τ is arbitrarily small. To find what the coefficient a is, we can do the following:

$$\begin{aligned}
p_{n-1}(\zeta) &= p_{n-1}(\xi_{n-1} + a\tau) \\
p_{n-1}(\xi_{n-1} + a\tau) &= p_{n-1}(\xi_{n-1}) + a\tau p_{n-1}'(\xi_{n-1}) + O(\tau^2) \\
p_{n-1}(\xi_{n-1} + a\tau) &= a\tau p_{n-1}'(\xi_{n-1}) + O(\tau^2) \,. \qquad (2.15)
\end{aligned}$$

Substituting this into the right hand side of Equation (2.12) and setting this expression to 0, we get the following:

$$0 = (\xi_{n-1} + a\tau)(a\tau p_{n-1}'(\xi_{n-1}))^2 + \tau^2 + O(\tau^3) \,. \qquad (2.16)$$

From here, we can collect the coefficients of powers of 2 for τ, and solve for a, which is

$$a = \pm \frac{1}{p'_{n-1}(\xi_{n-1})} \sqrt{\frac{-1}{\xi_{n-1}}}. \tag{2.17}$$

As we can see from Equation (2.17), a can either be positive or negative. This means that we can perturb our initial conditions in two different directions. Instead of using

$$\zeta(0) = \xi_{n-1}\,, \tag{2.18}$$

as our initial condition, we would use

$$\zeta(\tau) = \xi_{n-1} + \frac{\tau}{p'_{n-1}(\xi_{n-1})} \sqrt{\frac{-1}{\xi_{n-1}}} \tag{2.19}$$

and

$$\zeta(\tau) = \xi_{n-1} - \frac{\tau}{p'_{n-1}(\xi_{n-1})} \sqrt{\frac{-1}{\xi_{n-1}}}\,, \tag{2.20}$$

resulting in finding two different roots. Therefore, using this technique, we are able to perturb our initial condition, ξ_{n-1}, to avoid the singularity that we encounter when $\tau = 0$. This is essentially the Puiseux expansion of $\zeta(\tau)$:

$$\zeta(\tau) = \xi_{n-1} \pm a\tau + O(\tau^2) \tag{2.21}$$

and the reason we used τ^2 instead of just simply τ in the homotopy.

Unfortunately, these are not the only singularities that we encounter when solving Equation (2.14). As an example, we can look at the singularities that we encounter when $n = 3$. The differential equation for $p_3(z)$ is

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = \frac{-2\tau}{3\zeta^2 + 4\zeta + 1}\,, \tag{2.22}$$

where the initial conditions are $\zeta(0) = 0, -1$. Looking at Equation (2.22), it is quite obvious

that we will encounter singularities when $\zeta(\tau) = -1$ and $-\frac{1}{3}$, which is when $\tau \doteq 0.3849$. This will give us problems when trying to integrate along the real τ-axis. Therefore, we need to use some method, such as the pole-vaulting technique, in order to avoid these singularities.

2.5 Pole-Vaulting

The pole-vaulting technique is a way to avoid singularities by backing off from the pole slightly, and then going in a semicircular arc in the complex τ-plane (see [8, Section 12.11.1]), also visually shown in Figure 2.7. Our semicircular path can be defined as

$$\tau = p - \rho e^{i\theta}, \tag{2.23}$$

where p is the location of our singularity (which is a pole), and ρ is the radius of our semicircular arc (or mathematically, $\rho = p - \tau$). From Equation (2.23), we can see that when $\theta = 0$, $\tau = p - \rho$, and when $\theta = \pi$, $\tau = p + \rho$. This means that we will be hopping over the pole by taking $0 \leq \theta \leq \pi$.

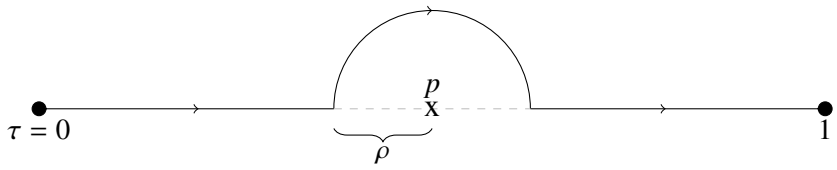


Figure 2.7: Diagram demonstrating pole-vaulting technique (p is the pole, and ρ is the radius of the semicircular arc).

Therefore, our ordinary differential equation for pole-vaulting becomes

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\theta} = \frac{\mathrm{d}\zeta}{\mathrm{d}\tau}\frac{\mathrm{d}\tau}{\mathrm{d}\theta} = \frac{-2i(p - \rho e^{i\theta})\rho e^{-i\theta}}{p_n'(\zeta)}, \quad \theta \in [0, \pi]. \tag{2.24}$$

Using this pole-vaulting technique, we can easily integrate Equation (2.22). Figure 2.8 shows the paths (in black) that were taken to achieve the roots for $p_3(z)$. In the figure, the blue circles are the starting points, $\zeta = 0$ and $\zeta = \xi_2 = -1$, and the red crosses are the final roots, ξ_3. The

grey line is the contour $|p_3(\zeta)| = 1$. Notice that in this figure, two of the final roots stem from our initial point $\zeta = \xi_2 = -1$, and only one root comes from $\zeta = 0$. This matches our previous comment about needing to perturb our initial conditions so that we can get two separate paths from this initial point. Figure 2.9 shows the homotopy paths of the Mandelbrot polynomials from $n = 4$ to $n = 9$.

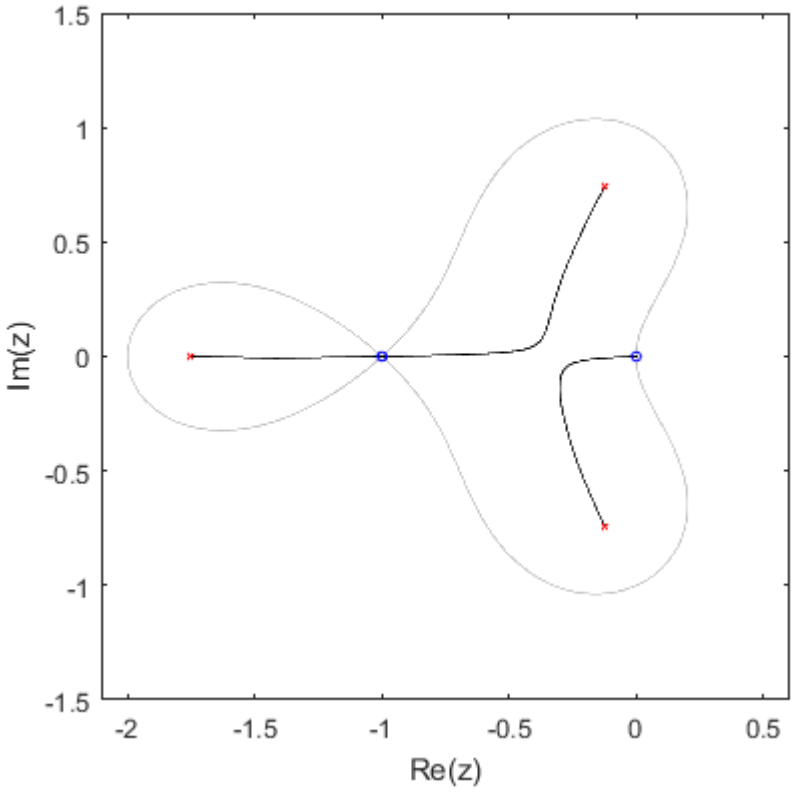
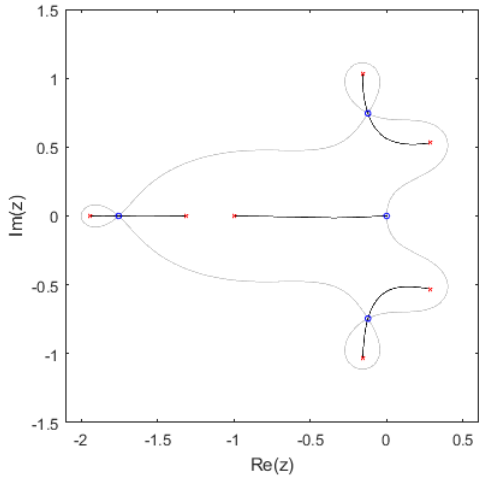


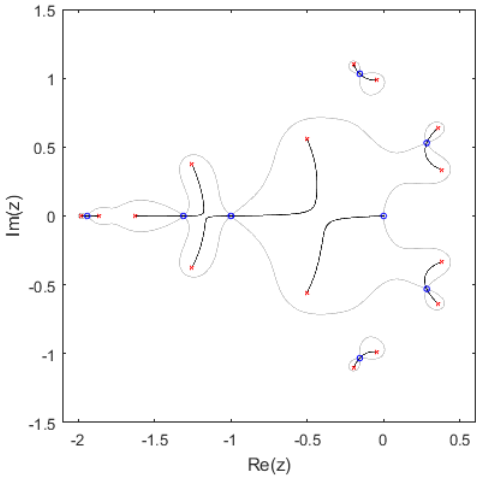
Figure 2.8: Homotopy paths of $p_3(z)$ and contour where $|p_3(z)| = 1$.

2.5.1 Residues

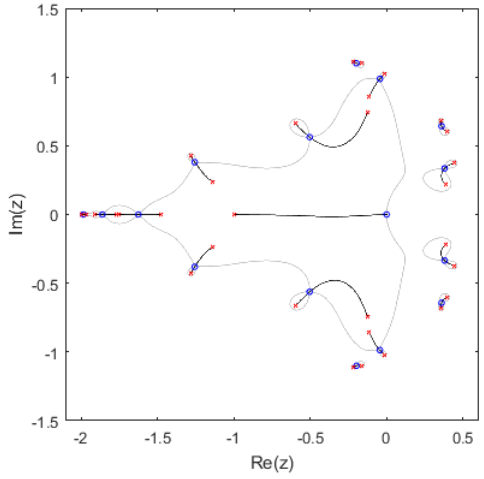
Our discussion about pole-vaulting thus far only looks at the case where we are integrating in a semi-circular arc above the real axis. However, we are also interested in knowing whether integrating along a semi-circular path below the real-axis would give us the same result. To determine whether there is any difference, we can compute the residues by integrating around the pole ($\theta \in [0, 2\pi]$), and see whether the z values when $\theta = 0$ is the same as the z values



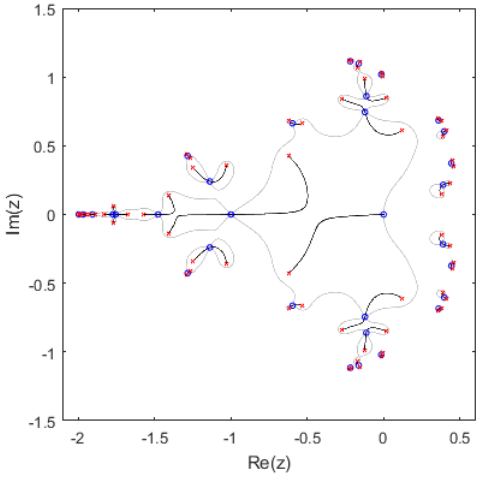
(a) n = 4



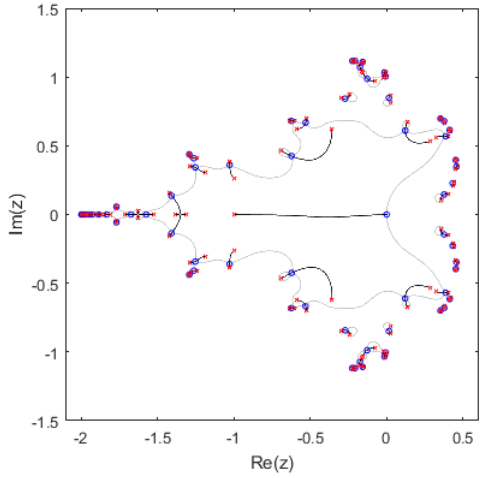
(b) n = 5



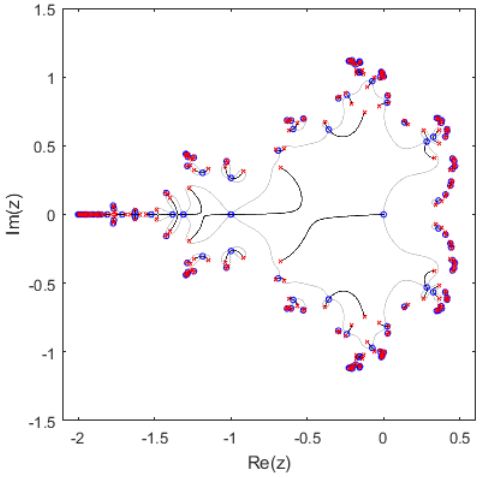
(c) n = 6



(d) n = 7



(e) n = 8



(f) n = 9

Figure 2.9: Plots of homotopy paths and contour $|p_n(z)| = 1$ of the Mandelbrot polynomials from $n = 4$ to $n = 9$.

when $\theta = 2\pi$. In MATLAB, using our own ode solver (which will be discussed later in this chapter), the residues are of order $O(10^{-2})$. Since this value is much smaller than the distance from where we start integrating our differential equation in the complex plane to our pole, we believe that it does not matter whether we use a pole-vaulting technique to integrate above or below the real axis. We also used multiple precision to compute the residues to ensure that the values that we computed for our residues were just simply due to the numerical technique that was used. The residues computed with higher precision are $O(10^{-5})$, which is less than the value that we computed when using MATLAB, as expected. Therefore, this confirms that we should be able to compute all of the roots of the Mandelbrot polynomials regardless of whether we are integrating above or below the real axis.

2.5.2 Distinctness

From the paper [10], we learn that all of the roots in this family of polynomials are distinct. To ensure that this statement is true, we need to make sure that the paths of integration of all routes taken to find the roots do not cross, thus leading us to find all the roots without getting any duplicates. To illustrate this, we can see that none of the homotopy paths cross in each plot of Figure 2.9. However, this is not a strong enough argument to confirm that all paths are distinct. We also need to prove uniqueness, which requires the Lipschitz condition in a domain $\mathcal{R}$ to be satisfied in order for there to be at most one solution [5, Chapter 6.3]. We, however, already know that the solutions of the differential equations are not unique if our initial condition is $\zeta(0) = \xi_{n-1}$, since it is a double root. Therefore, in place of $\zeta(0) = \xi_{n-1}$ as our initial conditions, we will use our perturbed initial conditions (see Equations (2.19) and (2.20)).

Consider the following general initial value problem:

$$x' = f(t, x), \quad x(\tau) = A\,. \tag{2.25}$$

We say that f satisfies a Lipschitz condition in a region $\mathcal{R}$ if there is a constant $L \geq 0$ such that [5, Chapter 6.2]

$$|f(t,u) - f(t,v)| \leq L\,|u - v|\,, \quad \text{if } (t,u),\ (t,v) \in \mathcal{R}\,. \tag{2.26}$$

Letting our region $\mathcal{R} = \mathbb{C} \sim S$, where S is a small region surrounding each singularity, and applying this definition to our homotopy for the Mandelbrot polynomials,

$$\begin{aligned} |f(t,u) - f(t,v)| \leq L\,|u - v| &= \left| \frac{-2t}{p_n'(u)} - \frac{-2t}{p_n'(v)} \right| \\ &= \left| \frac{-2t\,(p_n'(v) - p_n'(u))}{p_n'(u)p_n'(v)} \right|. \end{aligned} \tag{2.27}$$

Defining

$$\begin{aligned} p_n'(v) &= p_n'(u + v - u) \\ &= p_n'(u) + p_n''(u)(v-u) + O(v-u)\,, \end{aligned} \tag{2.28}$$

we get

$$\begin{aligned} L\,|u - v| &= \left| \frac{-2t\,(p_n'(u) + p_n''(u)(v-u) + O(v-u))}{p_n'(u)p_n'(v)} \right| \\ &\doteq \frac{-2tp_n''(u)}{p_n'(u)p_n'(v)}\,|v - u|\ . \end{aligned} \tag{2.29}$$

Therefore,

$$L = \frac{-2tp_n''(u)}{p_n'(u)p_n'(v)}\,. \tag{2.30}$$

However, since $p_n'(u) \approx p_n'(v)$, we can rewrite Equation (2.30) as

$$L = \frac{-2tp_n''(u)}{(p_n'(u))^2}\,. \tag{2.31}$$

As long as we avoid the singularities, when $p_n'(z) = 0$, we will be following a continuous path, which means that the second derivative is bounded. From this, we know that L is also bounded, thus satisfying the Lipschitz condition. Therefore, this shows that the solutions to our initial value problems are in fact unique.

2.6 Our custom ode solver

When we first started, we used MATLAB's `ode45` routine to solve our differential equations to find the roots of the Mandelbrot polynomials. However, instead of using the pole-vaulting technique, we decided to integrate in the complex plane using a triangular pathway. We were able to compute all of the roots up to $n = 20$, which is $524,288$ roots, until some pathways diverged off to infinity when computing some of the roots when $n = 21$, due to the increased density of the singularities. Therefore, we decided to write our own ode solver in MATLAB (and eventually in C++) to solve these differential equations so that we can step around the roots by using pole-vaulting. Although we could have used MATLAB's ode45 routine for the pole-vaulting technique, we did not want to compute all of the singularities as it becomes computationally expensive for high degrees. Therefore, our ode solver is based on MATLAB's `ode45` routine, which uses the Runge-Kutta-Fehlberg algorithm, but instead of terminating the program when it reaches a singularity, it is able to step around the singularity using the pole vaulting technique described in the previous section and continue integrating until it reaches the final point, which in our case is $\tau = 1$.

2.6.1 Runge-Kutta Methods

Euler's method is the simplest method for solving an ordinary differential equation numerically. However, it is not very accurate (or at least not accurate enough for our liking) since it is only a first order method. To improve on this, we could use Taylor series expansion to increase the order of our solution by using higher order derivatives. Since $p_n'(z)$, $p_n''(z)$ and higher order

approximations are also available in $O(n)$, i.e. $O(\ln d_n)$, flops, Taylor series methods could well be viable for this problem. However, we decided to take another approach: to evaluate the derivative function $f(t, x(t))$ more than once at different points, and then use a weighted average of the values thus obtaining an approximation of the slope of the secant. This idea gives what are called the Runge-Kutta (RK) methods, which we have found to be adequate for our needs.

There are different RK methods in use [8, Chapter 13]. If we let the weights be b_i, the time step be h, and $i = 1, 2, 3, \ldots, s$, the general form for an explicit Runge-Kutta methods is

$$\mathbf{x}_{k+1} = \mathbf{x}_0 + h(b_1\mathbf{k}_1 + \cdots + b_s\mathbf{k}_s) = x_k + h\sum_{i=1}^{s} b_i\mathbf{k}_i\,, \tag{2.32}$$

where the stages are

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}\,(t_k, \mathbf{x}_k)\\
\mathbf{k}_2 &= \mathbf{f}\,(t_k + c_2h, \mathbf{x}_k + a_{21}\mathbf{k}_1)\\
\mathbf{k}_3 &= \mathbf{f}\,(t_k + c_3h, \mathbf{x}_k + a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2)\\
&\vdots\\
\mathbf{k}_s &= \mathbf{f}\,\big(t_k + c_sh, \mathbf{x}_k + a_{s1}\mathbf{k}_1 + a_{s2}\mathbf{k}_2 + \cdots + a_{s,s-1}\mathbf{k}_{s-1}\big)\,,
\end{aligned} \tag{2.33}$$

which can also be expressed as

$$\mathbf{k}_i = \mathbf{f}\left(t_k + c_ih, \mathbf{x}_k + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j\right), \tag{2.34}$$

Here, the c_i and a_{ij} are the weights of our previously computed values of $\mathbf{k}_j$, $j < i$. We can conveniently summarize all of the coefficients in a tableau called a *Butcher tableau*, which has

the following form:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}, \tag{2.35}$$

where **A** is a lower-triangular matrix with zeros as diagonal entries.

For robustness and to ensure quality of the solution, we need an adaptive step size scheme. The goal of an adaptive step scheme is to take the largest stepsize possible while ensuring that the absolute local truncation error is less than a tolerance given by the user for every step. Here, we have decided to use the Runge-Kutta-Fehlberg method (RKF45) [12], which is a fourth-order method with an error estimate of order $O(h^5)$. The *Butcher tableau* for the Runge-Kutta-Fehlberg method, where the first row of coefficients at the bottom of the table gives the fifth-order accurate method, and the last row gives the fourth-order accurate method, is

$$\begin{array}{c|cccccc} 0 & & & & & & \\ 1/4 & 1/4 & & & & & \\ 3/8 & 3/32 & 9/32 & & & & \\ 12/13 & 1932/2197 & -7200/2197 & 7296/2197 & & & \\ 1 & 439/216 & -8 & 3680/513 & -845/4104 & & \\ 1/2 & -8/27 & 2 & -3544/2565 & 1859/4104 & -11/40 & \\ \hline & 16/135 & 0 & 6656/12825 & 28561/56430 & -9/50 & 2/55 \\ & 25/216 & 0 & 1408/2565 & 2197/4104 & -1/5 & 0 \end{array}. \tag{2.36}$$

From Equation (2.36), we can see that the fourth-order approximation is

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{25}{216}\mathbf{k}_1 + \frac{1408}{2565}\mathbf{k}_3 + \frac{2197}{4104}\mathbf{k}_4 - \frac{1}{5}\mathbf{k}_5 \tag{2.37}$$

and our fifth-order approximation is

$$\tilde{\mathbf{x}}_{i+1} = \mathbf{x}_i + \frac{16}{135}\mathbf{k}_1 + \frac{6656}{12825}\mathbf{k}_3 + \frac{28561}{56430}\mathbf{k}_4 - \frac{9}{50}\mathbf{k}_5 + \frac{2}{55}\mathbf{k}_6 \,. \tag{2.38}$$

The local truncation error is

$$\tau_{i+1} = \tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1} \,. \tag{2.39}$$

We can use the truncation error to help us calculate the size of the next step. By Taylor's theorem, our local truncation error is proportional to

$$\tau_{i+1} = kh^{p+1} \tag{2.40}$$

for some constant k and where p, in our case, is 4. So, we can multiply a scalar q with h to get the following:

$$\begin{aligned} \tau_{i+1}(qh) &= kq^p h^p \\ &= q^p k h^p \\ &= q^p \tau_{i+1}(h) \\ &\approx q^p \left(\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1}\right) \,. \end{aligned} \tag{2.41}$$

Therefore, to make

$$|\tau_{i+1}(qh)| \approx |q^p \left(\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1}\right)| < \text{tol} \,, \tag{2.42}$$

we want

$$q \leq \left[\frac{\text{tol}}{|\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1}|}\right]^{1/p} \,, \tag{2.43}$$

but in practice, we use

$$q \leq 0.8 \left[\frac{\text{tol}}{|\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1}|} \right]^{1/p} . \tag{2.44}$$

Therefore, the optimal step size for the next step is qh. This is a very primitive way of step-size control, but has worked adequately for these problems, most notably because we may improve the approximate answer by a Newton step on $p_n(z) = 0$.

We can take advantage of the automation of the step sizes to help us locate singularities that are in the way. In our ode solver, once the step size falls below a certain size, say 10^{-6}, the method will start integrating around the singularity using the Runge-Kutta-Fehlberg method once again. As a reminder, the differential equation that is used to integrate around the singularity is given as

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\theta} = \frac{\mathrm{d}\zeta}{\mathrm{d}\tau}\frac{\mathrm{d}\tau}{\mathrm{d}\theta} = \frac{-2i(p - \rho e^{i\theta})\rho e^{-i\theta}}{p_n'(\zeta)} , \quad \theta \in [0, \pi] ,$$

which can also be found as Equation (2.24). The problem that we encounter here is that we actually do not know what p or ρ are since we do not know where the exact location of the singularities are. Therefore, to estimate ρ, which is the distance that we are from the pole, we can use Newton's method

$$\rho = \zeta_i - \frac{p_n'(\zeta_i)}{p_n''(\zeta_i)} , \tag{2.45}$$

since the denominator of our differential equation is just simply $p_n'(\zeta)$, and we can easily calculate $p_n'(\zeta_i)$ and $p_n''(\zeta_i)$.

Once the semi-circular path is finished integrating (when $\theta = \pi$), our ode solver continues integrating along its original path until it reaches $\tau = 1$.

2.7 Accuracy of the roots

To check for the accuracy of these roots, we can calculate the residuals by evaluating $p_n(\xi_n)$. We expect that we can use Newton's method to reduce the size of the residuals of each of the roots, which will be discussed in the following subsection.

2.7.1 Newton polishing

To improve on the accuracy of our roots, we expect that we can use Newton's method to polish the roots:

$$\xi_n = \xi_n - \frac{p_n(\xi_n)}{p_n'(\xi_n)} \,. \tag{2.46}$$

We only polish each root once as we fear polishing the roots any more than that may cause the roots to skip over to another root. However, Newton's method may not actually give us better results. As we can see from the paper [10] by Corless and Lawrence, Newton's method has problems when $p_n''(z)$ is too large, which the authors found out when trying to find the largest roots of the Mandelbrot polynomials.

The authors began with the observation that the largest root is quite close to, but slightly closer to zero than, -2. In order to use Newton's method, the derivatives need to be calculated, which they found to be (at $z = -2$)

$$p_n'(z) = \frac{4^{n-1} - 1}{3} \,, \tag{2.47}$$

which resulted in the Newton estimate to be

$$z_n \doteq -2 + \frac{3}{4^{n-1} - 1} \,. \tag{2.48}$$

However, the Newton estimate above is not quite right: the error of Equation (2.48) is $O(4^{-n})$, but the guess is already $O(4^{-n})$ accurate, so taking a Newton step hardly improves this estimate. Therefore, we need to look at the growth of higher derivatives. The Newton estimate is based on the expansion

$$p_n(-2 + \varepsilon) = p_n(-2) + p_n'(-2)\varepsilon + \frac{1}{2} p_n''(-2)\varepsilon^2 + \cdots \,, \tag{2.49}$$

but neglects the terms of $O(\varepsilon^2)$, since they are usually benign. Using Maple's `rsolve`, the

second derivative of the Mandelbrot polynomial evaluated at -2 is

$$p_n''(-2) = -\frac{1}{27}4^{2n} + \left(\frac{1}{3} - \frac{k}{9}\right)4^n - \frac{8}{27}\,, \tag{2.50}$$

which exposes the problem with Newton's method. Here, $p_n''(-2)$ is $O(\varepsilon^{-2})$, so we cannot neglect the $O(\varepsilon^2)$ term in this case. In [10], the authors go on to find an analytical expression for the largest magnitude real roots of the Mandelbrot polynomials, but we shall not need that here.

2.8 Smallest roots

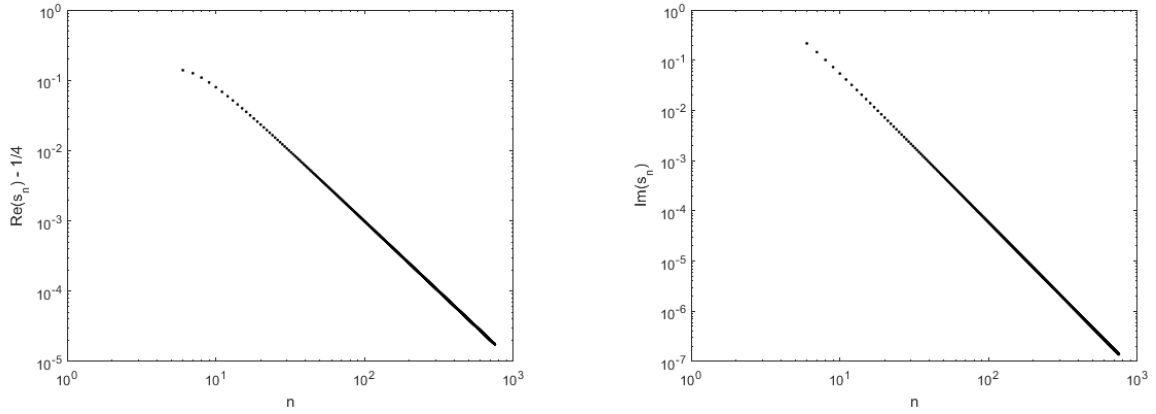
We can also use homotopy methods, as described above, to find the smallest roots, s_n, of the Mandelbrot polynomials by just simply using the smallest root from the previous iteration as our starting point. We empirically deduce from our computations that that s_n has the form:

$$s_n = \frac{1}{4} + \alpha_{\mathrm{Re}} n^{-\beta_{\mathrm{Re}}} \pm i\alpha_{\mathrm{Im}} n^{-\beta_{\mathrm{Im}}} + h.o.t. \tag{2.51}$$

We can plot the real (minus 1/4) and imaginary part of the smallest roots in a log-log plot, shown in Figure 2.10 to compute β_{Re} and β_{Im}, which turn out to be, to the accuracy that we use, 2 and 3 respectively. However, we are not sure that these are exact. Therefore, we can approximate

$$s_n \doteq \frac{1}{4} + \frac{\alpha_{\mathrm{Re}}}{n^2} \pm \frac{\alpha_{\mathrm{Im}}}{n^3}\,. \tag{2.52}$$

Knowing what β_{Re} and β_{Im} are, we can compute α_{Re} and α_{Im}, which are approximately 9.869 and 58.81, respectively. Corless and Lawrence [10] conjectured that α_{Re} is π^2. This work does not confirm that conjecture, but at least it does not contradict it, because $\pi^2 \doteq 9.8696044\ldots$.



(a) Real part (b) Imaginary part

Figure 2.10: Log-log plots of smallest roots s_n of Mandelbrot polynomials (difference from $1/4$)

2.9 Results for homotopy methods

We were able to compute in MATLAB, using our own ode solver, all the roots of the Mandelbrot polynomials up to $n = 22$, which has a degree of $2,097,151$ to the order of $O(10^{-4})$ precision (see Figure 2.12). However, according to Bini's personal website, they were able to solve for around 4 million roots, which is one more iteration that what we have computed.

The time that it took to compute the roots using a homotopy method can be seen in Figure 2.11. In this figure, the line of best fit that runs through the data points has a slope of around 0.92, which is less than 1 (the line above our data is a reference to show the steepness of a line with a slope of 1). Computationally speaking, this does not make any sense. There is a lower bound on the complexity of $O(d_n)$ because we have to output d_n roots; further, evaluating a residual at each root costs $O(\ln d_n)$ flops, making an overall lower bound of $O(d_n \ln d_n)$. What must be happening here is that the "constant" hidden by the O symbol is larger for the first few n, and only is asymptotically constant. The roots are getting easier to find for larger d_n.

As we compute the roots using our custom ode solver in MATLAB, we notice that we lose accuracy as the iteration increases. We believe that the residuals are getting larger because of the mild instability of the recurrence relation. Therefore, we need to use multiple precision in order to compute the roots of higher iterations. We used David Bailey's ARPREC package [2]

for arbitrary precision in C++. Unfortunately, the ARPREC package does not lend itself to OpenMP parallelization since it is not entirely thread safe [1]. Despite this, we were able to compute up to $n = 19$, which has a degree of $262,143$ thus far with the maximum residual of $O(10^{-11})$.

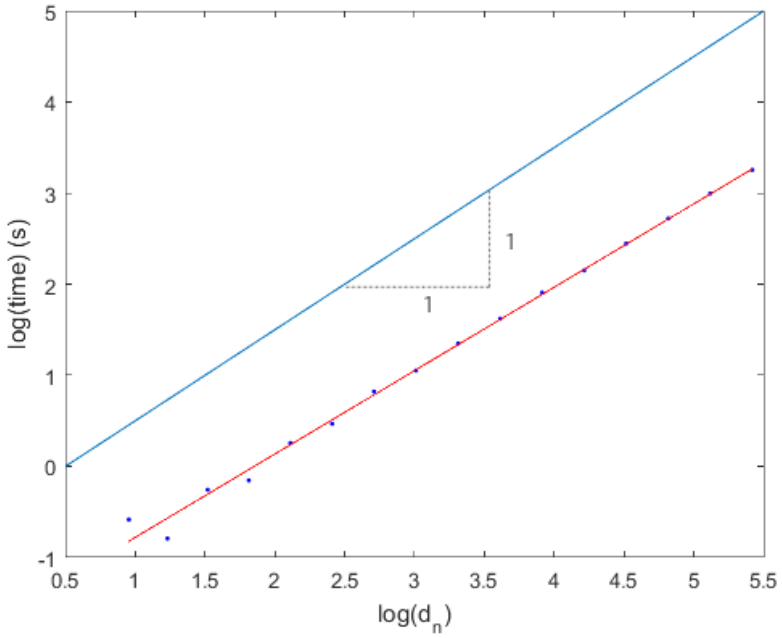


Figure 2.11: Time taken to compute roots of Mandelbrot polynomial using a homotopy method. The line of best fit has a slope about 0.92.

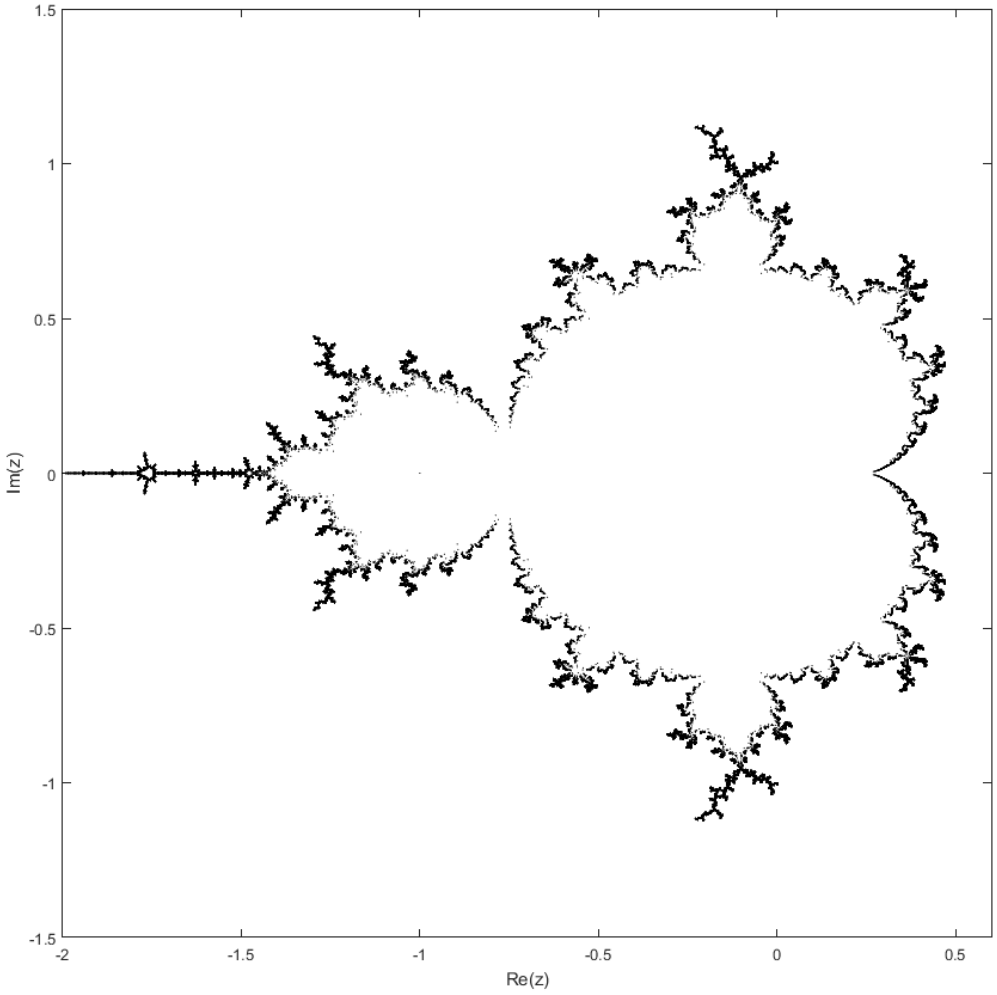


Figure 2.12: All $2,097,151$ roots of the Mandelbrot polynomial $p_{22}(z)$. These roots were produced in MATLAB using our own ODE solver.

Chapter 3

Fibonacci-Mandelbrot polynomials and matrices

3.1 Introduction

The Fibonacci sequence, which is Sequence A000045 of the Online Encyclopedia of Integer Sequences [22] is a widely known sequence. It begins

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots \tag{3.1}$$

and is generated by the recursion

$$F_n = F_{n-1} + F_{n-2} \tag{3.2}$$

with $F_0 = 0$ and $F_1 = 1$. There is a plethora of resources such as [16], [18], and the references therein, that talk about the Fibonacci sequence which can be referred to if the reader wants to learn more about the Fibonacci sequence.

3.1.1 Fibonacci-Mandelbrot polynomials

The Fibonacci-Mandelbrot polynomials are very similar to the Mandelbrot polynomials, described in the previous chapter, but are slightly different. As a reminder, the recursion for the Mandelbrot polynomials is

$$p_{n+1}(z) = zp_n^2 + 1\,, \tag{3.3}$$

where $p_0 = 0$. The Fibonacci-Mandelbrot polynomials, on the other hand, have the recursion

$$\begin{aligned} q_0(z) &= 0\,, \quad q_1(z) = 1 \\ q_{n+1}(z) &= zq_n(z)q_{n-1}(z) + 1, \end{aligned} \tag{3.4}$$

where $n = 1, 2, 3, \ldots$. Instead of taking the polynomial from the previous iteration and squaring it, we are multiplying the polynomials from the previous two iterations together. This is the reason why it is called the Fibonacci-Mandelbrot polynomials.

Expanding Equation (3.4) using the monomial basis expansion, we can get the first few polynomials:

$$\begin{aligned} q_0(z) &= 0 \\ q_1(z) &= 1 \\ q_2(z) &= 1 \\ q_3(z) &= z + 1 \\ q_4(z) &= z^2 + z + 1 \\ q_5(z) &= z^4 + 2z^3 + 2z^2 + z + 1 \\ q_6(z) &= z^7 + 3z^6 + 5z^5 + 5z^4 + 4z^3 + 2z^2 + z + 1 \\ q_7(z) &= z^{12} + 5z^{11} + 13z^{10} + 22z^9 + 28z^8 + 28z^7 + 23z^6 + 16z^5 + 10z^4 + 5z^3 + 2z^2 + z + 1\,. \end{aligned} \tag{3.5}$$

Some properties of the Fibonacci-Mandelbrot polynomials include:

1. The leading and trailing coefficients are 1.
2. All coefficients are positive integers.
3. The polynomials are unimodular.
4. The next-to-leading coefficient is a Fibonacci number.
5. Put $d_n = \deg q_n$. Then $d_1 = 0$, $d_2 = 0$, $d_{n+1} = d_n + d_{n-1} + 1$ or $d_n = F_n - 1$, where F_n is a Fibonacci number (see Equation (3.2)).
6. The roots of $q_n(z)$ lead to periodic points of $q_{n+1}(z) = zq_n(z)q_{n-1} + 1$, of period $n - 2$. For instance, $q_3(-1) = 0$, $q_4(-1) = 1$, $q_5(-1) = 1$, and then repeats: $q_n(-1) = \{0, 1, 1\}$.
7. The coefficients of q_n grow doubly exponentially: $O(\phi^{\phi^n})$, $\phi = \frac{1+\sqrt{5}}{2} \doteq 1.618\ldots$.

3.2 Condition numbers and pseudozeros

Similar to the Mandelbrot polynomials, the absolute condition number of the roots is $1/q_n'(z)$. Figure 3.1 shows the minimum and maximum condition numbers of the roots of the Fibonacci-Mandelbrot polynomials. The maximum condition numbers are represented by the circles, and are computed by taking the reciprocal of the minimum value of $\left|q_n'(z)\right|$. On the other hand, the minimum condition numbers are represented by the crosses, and computed by taking the reciprocal of the maximum value of $\left|q_n'(z)\right|$. Just as we have seen for the Mandelbrot polynomials, the maximum condition number for the Fibonacci-Mandelbrot polynomials is also 1, which means that the roots are well-conditioned. The slope for the line of best fit for the minimum condition number for the Fibonacci-Mandelbrot polynomials is around -1.9, which is slightly greater than -2. The line that is below the lines of best fit is for reference; it has a slope of -2.

Additionally, we can look at the pseudozeros of the Fibonacci-Mandelbrot polynomials by plotting the contours at fairly small values of the polynomials and see where these contours lie

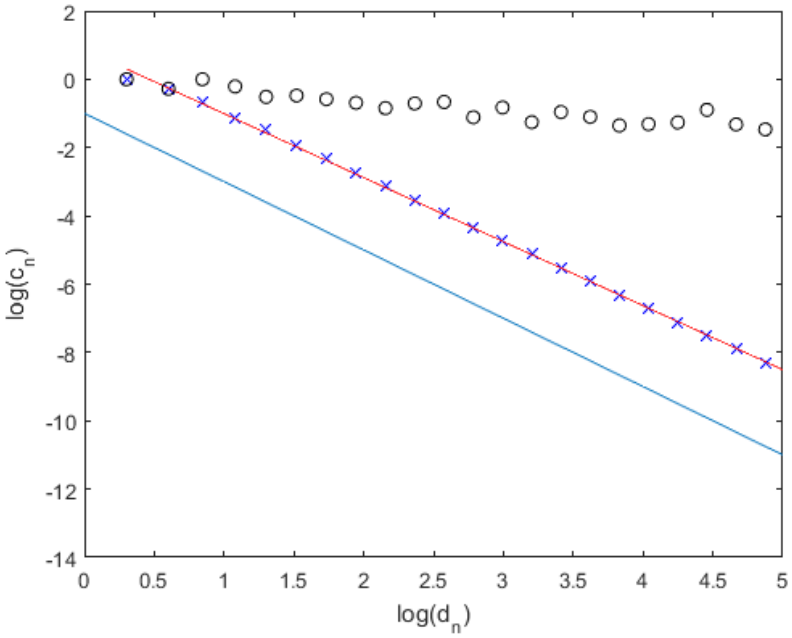


Figure 3.1: Minimum and maximum condition numbers of the roots for Fibonacci-Mandelbrot polynomials.

with respect to the location of the roots. Figure 3.2 shows the roots of $q_{15}(z)$ with the contours of $|q_{15}(z)| = 0.2$ in red. Here, we can see that the contours that are visible encircles the roots quite closely, which mean that the roots are well-conditioned. We can look closer into some of the more interesting regions (that contain more red) of the roots of Fibonacci-Mandelbrot polynomials, and reduce the size of the contour that we are looking into for these particular regions. In Figure 3.3, we zoom into 4 different regions of the roots of $q_{15}(z)$, and plotted $|q_{15}(z)| = 0.05$ instead of $|q_{15}(z)| = 0.2$.

3.3 Fibonacci-Mandelbrot matrices

Using Piers Lawrence's idea of using supersparse companion matrices to compute the roots of the Mandelbrot polynomials, $p_n(z)$, we can create analogous supersparse matrices for the

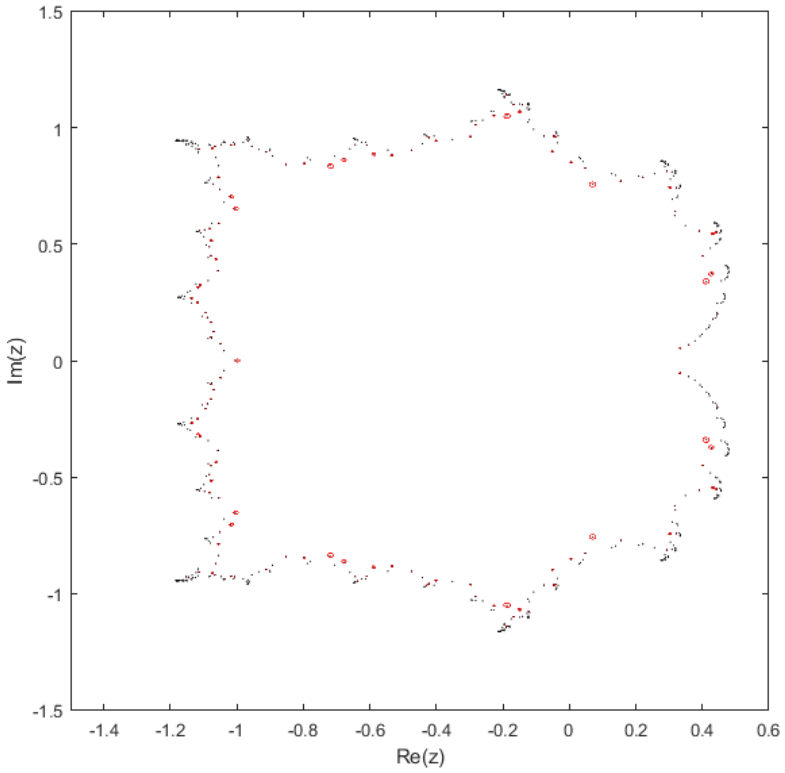


Figure 3.2: All 609 roots of $q_{15}(z)$ with $|q_{15}(z)| = 0.2$ in red.

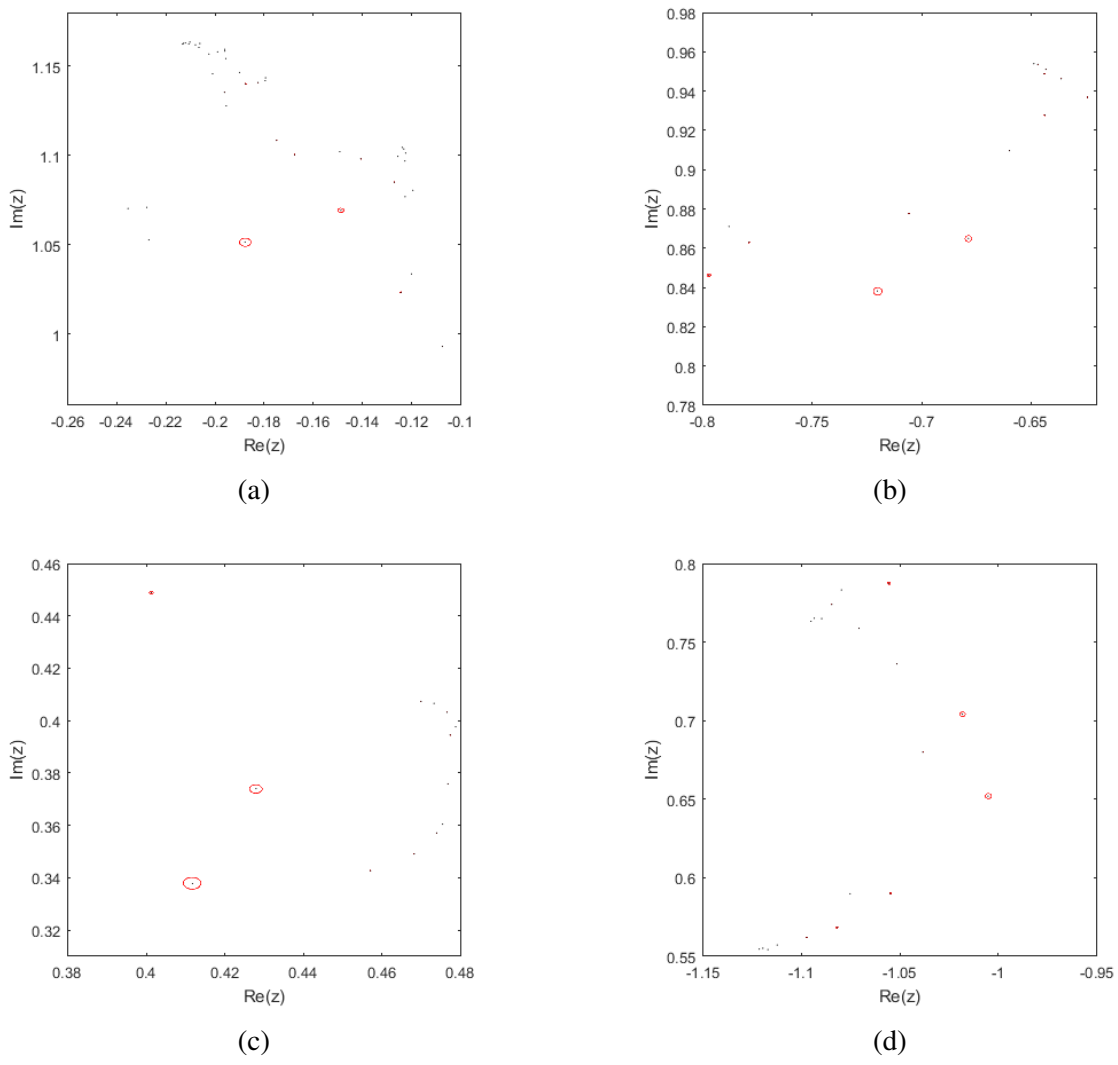


Figure 3.3: Different regions the Fibonacci-Mandelbrot polynomials where the roots are not as well-conditioned with $|q_{15}(z)| = 0.05$ in red.

Fibonacci-Mandelbrot polynomials, $q_n(z)$. We start with

$$\mathbf{M}_3 = [-1], \tag{3.6}$$

in which the eigenvalue, -1, is the root of $q_3(z) = z + 1$ and

$$\mathbf{M}_4 = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}, \tag{3.7}$$

where the eigenvalues, $-\frac{1}{2} \pm \frac{\sqrt{3}i}{2}$, are the roots of $q_4(z) = z^2 + z + 1$. Also, note that

$$\mathbf{M}_4^T = \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix} \tag{3.8}$$

also leads to a similar family. However, we decided to use Equation (3.7) so that the subdiagonal of these family of companion matrices will always be -1.

Let $\mathbf{r}_n = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}$ and $\mathbf{c}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$ be row and column vectors of length d_n, where d_n is the degree of the polynomial, $q_n(z)$. Then, our matrix construction would be

$$\mathbf{M}_{n+1} = \begin{bmatrix} \mathbf{M}_n & & (-1)^{d_{n+1}}\mathbf{c}_n\mathbf{r}_{n-1} \\ -\mathbf{r}_n & 0 & \\ & -\mathbf{c}_{n-1} & \mathbf{M}_{n-1} \end{bmatrix} \tag{3.9}$$

for all $n > 2$. The first few Fibonacci-Mandelbrot matrices are

$$\mathbf{M}_5 = \begin{bmatrix} 0 & 1 & & 1 \\ -1 & -1 & & \\ & -1 & & \\ & & -1 & -1 \end{bmatrix}, \tag{3.10}$$

and

$$\mathbf{M}_6 = \begin{bmatrix} 0 & 1 & 0 & 1 & & & -1 \\ -1 & -1 & 0 & 0 & & & \\ & -1 & 0 & 0 & & & \\ & & -1 & -1 & & & \\ & & & -1 & & & \\ & & & & -1 & 0 & 1 \\ & & & & & -1 & -1 \end{bmatrix}. \tag{3.11}$$

Computing the characteristic polynomials for both Equations (3.10) and (3.11), they both match the Fibonacci-Mandelbrot polynomials, $q_5(z)$ and $q_6(z)$, respectively. We can also construct the Fibonacci-Mandelbrot matrices slightly differently: we can swap $\mathbf{M}_n$ and $\mathbf{M}_{n-1}$, and change $\mathbf{r}_n$ and $\mathbf{c}_n$ to the correct lengths. Thus, the recursion for this companion matrix is

$$\mathbf{M}_{n+1} = \begin{bmatrix} \mathbf{M}_{n-1} & & (-1)^{d_{n+1}}\mathbf{c}_{n-1}\mathbf{r}_n \\ -\mathbf{r}_{n-1} & 0 & \\ & -\mathbf{c}_n & \mathbf{M}_n \end{bmatrix}, \tag{3.12}$$

where $\mathbf{M}_3$ and $\mathbf{M}_4$ are the same as above. Therefore, the next few Fibonacci-Mandelbrot ma-

trices using the recursion shown in Equation (3.12) are

$$\mathbf{M}_5 = \begin{bmatrix} -1 & & & & 1 \\ -1 & & & & \\ & -1 & 0 & 1 & \\ & & -1 & -1 & \end{bmatrix}, \tag{3.13}$$

$$\mathbf{M}_6 = \begin{bmatrix} 0 & 1 & & & & & -1 \\ -1 & -1 & & & & & \\ & -1 & & & & & \\ & & -1 & -1 & 0 & 0 & 1 \\ & & & -1 & 0 & 0 & 0 \\ & & & & -1 & 0 & 1 \\ & & & & & -1 & -1 \end{bmatrix}, \tag{3.14}$$

in which the characteristic polynomials of both $\mathbf{M}_5$ (Equation (3.13)) and $\mathbf{M}_6$ (Equation (3.14)) also match $q_5(z)$ and $q_6(z)$ respectively. It can be shown that $q_n(z) = \det(z\mathbf{I} - \mathbf{M}_n)$ for all $n > 3$ using induction and the Schur complement [24], which will be shown in Chapter 5.

Unlike the Mandelbrot matrices, notice that the Fibonacci-Mandelbrot matrices contain $\{-1, 0, 1\}$, whereas the Mandelbrot matrices contain just the values $\{0, -1\}$. What is also interesting is that the inverses of these Fibonacci-Mandelbrot companion matrices have inverses that are also supersparse, and only contain $\{-1, 0, 1\}$ as well. For example, if we take the inverse of

$\mathbf{M}_6$ (Equation (3.11)) that follows the recursion found in Equation (3.9),

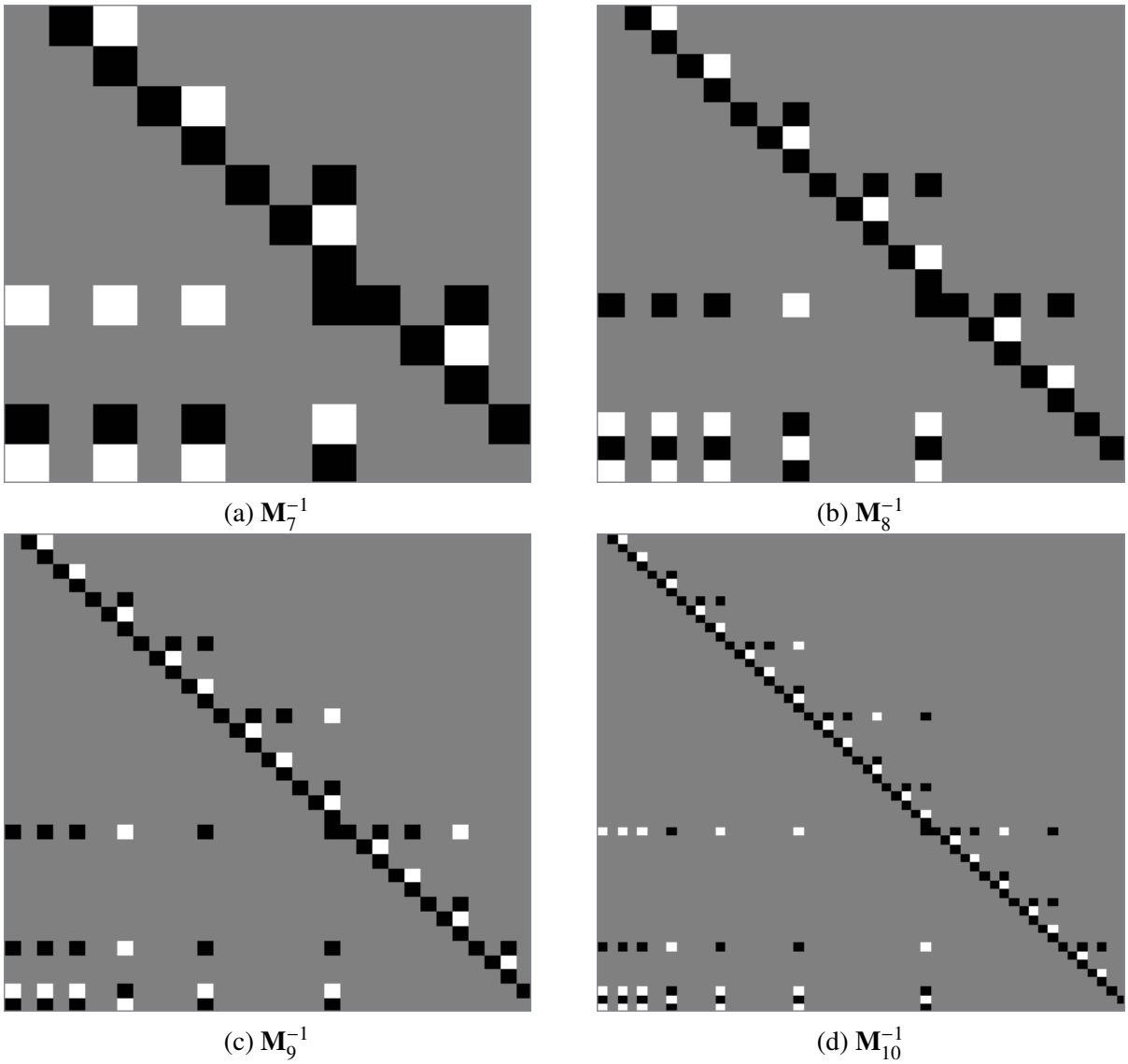
$$\mathbf{M}_6^{-1} = \begin{bmatrix} 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.15}$$

We can also use Maple to help us visualize the next few inverses, shown in Figure 3.4, where -1 is black, 0 is grey, and 1 is white. It is obvious from these plots that there is clearly a pattern for the inverses of the Fibonacci-Mandelbrot polynomials. More research is required to learn more about the inverses of these companion matrices and will be left to future work.

3.3.1 Results

Using our first matrix construction (Equation (3.9)), MATLAB's `eig` routine was able to compute the eigenvalues of $\mathbf{M}_{22}$, which has a dimension of $17,710$, correctly (see Figure 3.5a). However, it was not able to successfully compute the roots of $q_{23}(z)$ correctly, shown in Figure 3.5b. In MATLAB's `eig` routine, the default for `balanceOption` is `'balance'`, which enables balancing. In most cases, the balancing step improves the conditioning of the matrix to produce more accurate results. However, in our case, it did not give us the correct results. Therefore, we computed the eigenvalues once again with `'nobalance'`, but unfortunately, produced the same (incorrect) results. Additionally, we did not attempt to solve for the eigenvalues of sparse matrices even though it is very likely that it can help us find more roots using this method.

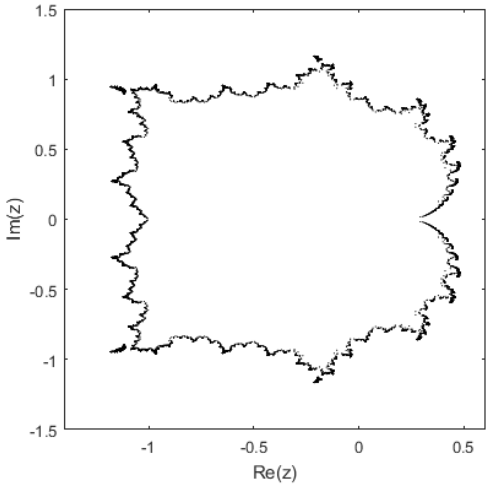
We also tried computing the eigenvalues of the Fibonacci-Mandelbrot matrices using both Maple 2015 and Maple 2016. Surprisingly, the different versions of Maple gave us different



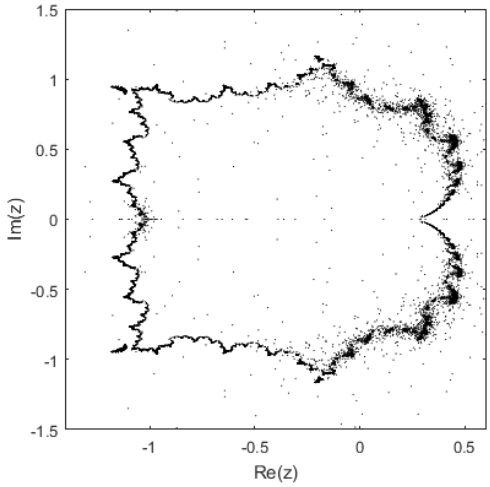
(a) $\mathbf{M}_7^{-1}$ (b) $\mathbf{M}_8^{-1}$

(c) $\mathbf{M}_9^{-1}$ (d) $\mathbf{M}_{10}^{-1}$

Figure 3.4: Image visualizations of inverses of Fibonacci-Mandelbrot matrices, where -1, 0 and 1 are black, grey, and white respectively, using Maple 2016.



(a) Computed eigenvalues of $\mathbf{M}_{22}$, which has a dimension of 17, 710.



(b) Computed eigenvalues computed of $\mathbf{M}_{23}$, which has a dimension of 28, 656.

Figure 3.5: Plots of eigenvalues using MATLAB's `eig` routine.

results. Maple 2015 actually gives us the results that we were expecting (see Figure3.6a), whereas Maple 2016 gives us inaccurate results (see Figure 3.6b).

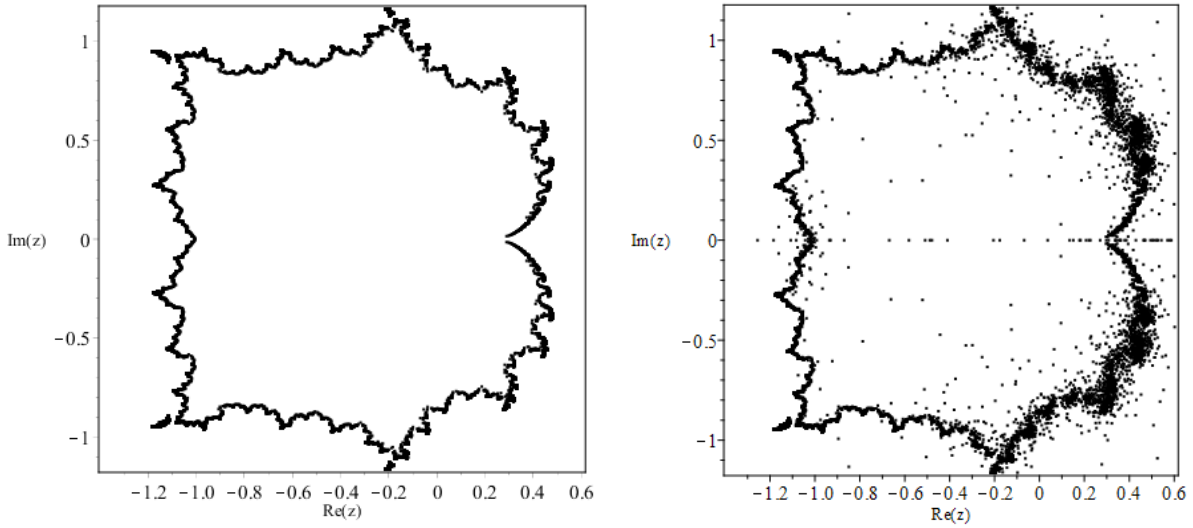
From Figure 3.7, we can see that the time complexity is around $O(d_n^{2.3})$, which is very similar to the time complexity that we computed when using the eigenvalue method on the Mandelbrot matrices. As a reference, the top line has a slope of 3, which is the slope that we expect our line of best fit to have.

3.4 Homotopy methods

We can also use homotopy methods to solve for the roots of the Fibonacci-Mandelbrot polynomials. Consider the following homotopy:

$$H_n(\zeta, \tau) = \zeta q_{n-1}(\zeta) q_{n-2}(\zeta) + \tau. \tag{3.16}$$

Comparing this homotopy (Equation (3.16)) to the homotopy used for the Mandelbrot polynomials (Equation (2.12)), we can see that they are quite similar. However, the main difference



(a) Using Maple 2015

(b) Using Maple 2016

Figure 3.6: Computed eigenvalues of $n = 23$, which has a degree of $28,656$ of the Fibonacci-Mandelbrot matrices using Maple.

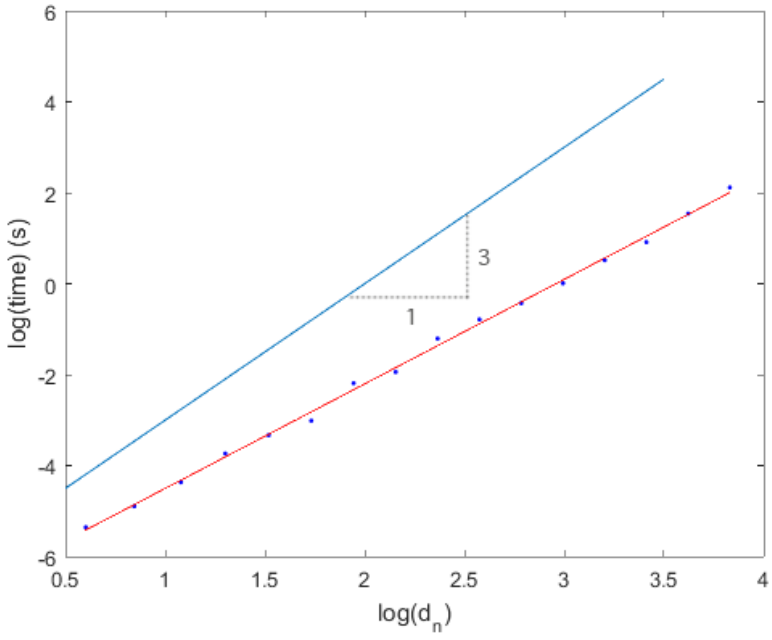


Figure 3.7: Time taken to compute eigenvalues of Fibonacci-Mandelbrot matrices.

is that the variable τ in the homotopy for the Mandelbrot polynomials is squared, whereas here in Equation (3.16), is just simply τ. This is because the zeros when $\tau = 0$ are simple at $\zeta = 0$, the roots of $q_{n-1}(\zeta)$, and the roots of $q_{n-2}(\zeta)$: we do not start at any double roots.

Similarly to the Mandelbrot polynomials, we can differentiate the right-hand side of Equation (3.16) with respect to τ to give us the following differential equation:

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = \frac{-1}{q_n(\zeta)}, \tag{3.17}$$

where we integrate $0 \leq \tau \leq 1$. Just as we did for the homotopy method used for the Mandelbrot polynomials, we can use the zeros of $\zeta q_{n-1}(\zeta) q_{n-2}(\zeta)$ as our initial conditions to help us find the roots of $q_n(\zeta)$.

Unfortunately, just as when solving the differential equations numerically for the Mandelbrot polynomials, we encounter singularities along the real-axis when solving Equation (3.17) for the Fibonacci-Mandelbrot polynomials. As an example for this case, we can look at the singularities when $n = 4$. The differential equation for $q_4(z)$ is

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = \frac{-1}{2\zeta + 1}, \tag{3.18}$$

where $\zeta(0) = 0$ and $\zeta(0) = -1$ (since it is the root of $q_3(z)$). There are no roots for $q_2(z)$, so we do not include $q_2(z)$ in our initial conditions. From Equation (3.18), it is obvious that we will encounter a singularity when $\zeta = -\frac{1}{2}$. Also, since Equation (3.18) is separable, we can easily find the value of τ when $\zeta = -\frac{1}{2}$, and check that τ lies on the real-axis between 0 and 1.

$$\begin{aligned} \frac{\mathrm{d}\zeta}{\mathrm{d}\tau} &= \frac{-1}{2\zeta + 1} \\ (2\zeta + 1)\,\mathrm{d}\zeta &= -\mathrm{d}\tau \\ \zeta^2 + \zeta &= -\tau + C, \quad \text{where } C \text{ is a constant .} \end{aligned} \tag{3.19}$$

It is obvious that when $\zeta = 0$ and $\tau = 0$, our constant $C = 0$. To find the value of our constant

when $\zeta = -1$ and $\tau = 0$, we can substitute the corresponding values to Equation (3.19):

$$\begin{aligned}(-1)^2 + (-1) &= C \\ 1 - 1 &= C \\ C &= 0. \end{aligned} \tag{3.20}$$

Therefore, when $\zeta = -1$ and $\tau = 0$, our constant C is also 0. Knowing that our constant $C = 0$, we can now solve for τ when $\zeta = -\frac{1}{2}$ to see at what value of τ we encounter a singularity for Equation (3.18):

$$\begin{aligned}\left(-\frac{1}{2}\right)^2 + \left(-\frac{1}{2}\right) &= -\tau \\ \frac{1}{4} - \frac{1}{2} &= -\tau \\ -\frac{1}{4} &= -\tau \\ \tau &= \frac{1}{4}. \end{aligned} \tag{3.21}$$

This shows that we do in fact encounter a singularity if we integrate along the real-axis, which means that we need to use the pole-vaulting technique described in the previous chapter (see Section 2.5) in order to avoid the singularities.

Since we are not starting from double roots for the Fibonacci-Mandelbrot polynomials, this means that we do not need to perturb our initial condition, which we had to do for the Mandelbrot polynomials. Instead, we can simply use the zeros of $\zeta q_{n-1}(\zeta)q_{n-2}(\zeta)$, as mentioned before. This means we only will get one root from each initial condition, unlike in the Mandelbrot polynomials, where we get 2 roots from the zeros of $p_{n-1}(\zeta)$ (remember that we only got 1 root from $\zeta = 0$ for the Mandelbrot polynomials). As demonstrated in Figure 3.8, created in MATLAB, we can see the homotopy paths taken from our initial points to our roots, ξ_5. In this figure, the root, $\xi_3 = -1$, is indicated by a triangle, the roots, $\xi_4 = -0.5 \pm 0.86603\ldots$, are diamonds, and $\zeta = 0$ is a circle, and they each lead us to a root, ξ_5, which are squares.

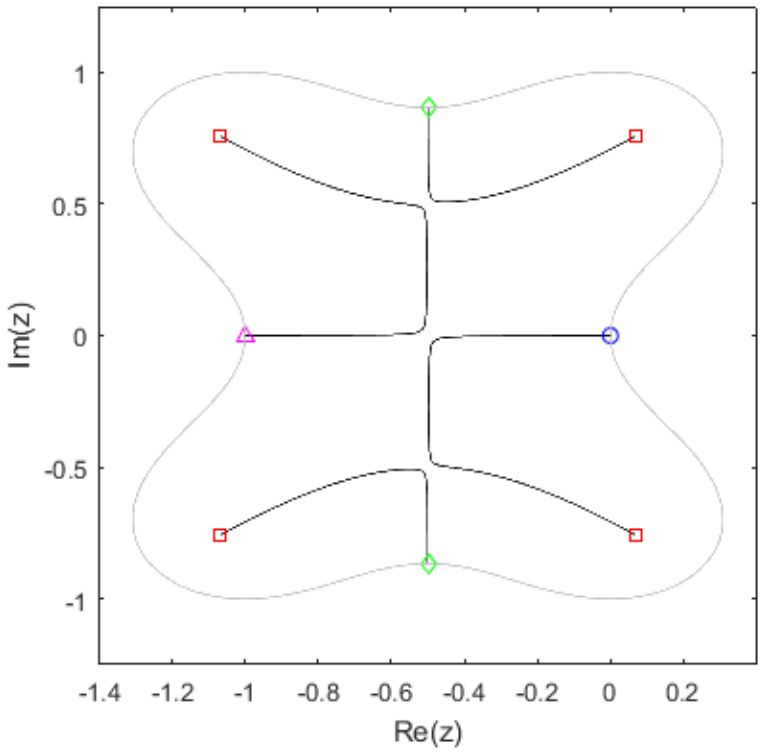
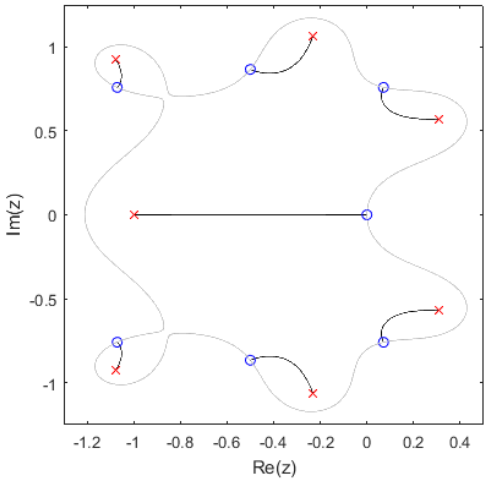


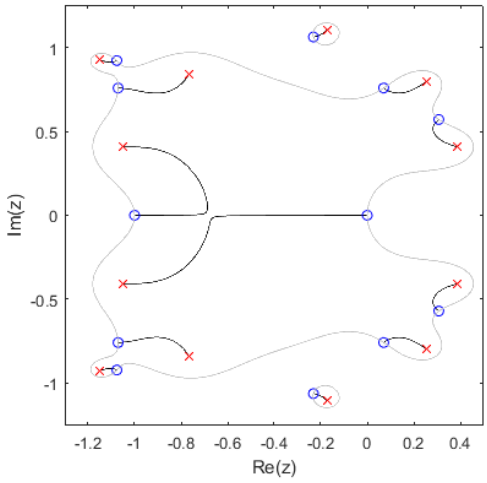
Figure 3.8: Homotopy paths for q_5 and contour where $|q_5(z)| = 1$.

The grey line that surround the roots is the contour, $|q_5(z)| = 1$. Notice in this figure that three singularities are avoided by pole-vaulting.

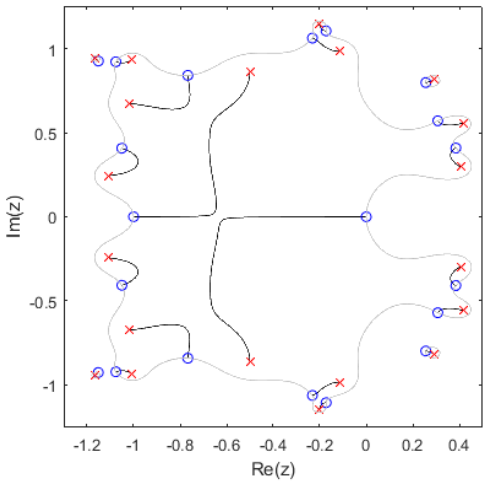
Figure 3.9 shows the homotopy paths of the Fibonacci-Mandelbrot polynomials from $n = 6$ to $n = 11$. To simplify the plots, all of the initial points are blue circles (instead of showing where each initial point comes from), while the final points are red crosses. These plots clearly show that only one root stems from each initial point, unlike the Mandelbrot polynomials, seen in Figure 2.9. One can prove that the gcd of $q_n(z)$ and $q_{n-1}(z)$ is 1: they can have no roots in common because each would be periodic with period n and $n - 1$ and hence a fixed point, but there are no fixed points in this iteration.



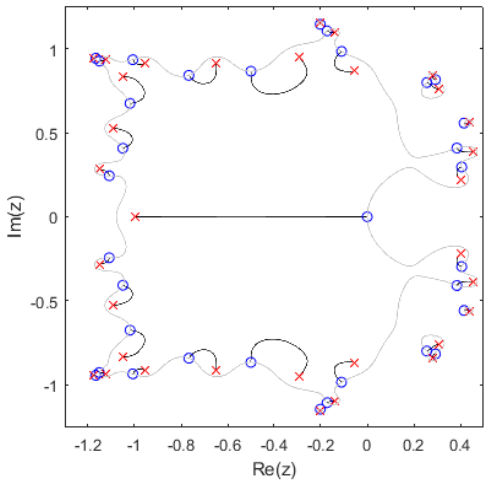
(a) n = 6



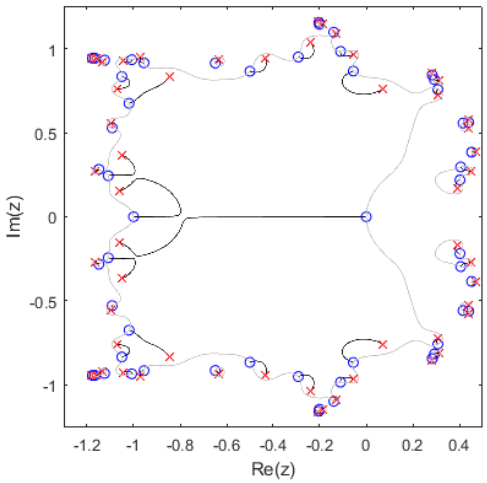
(b) n = 7



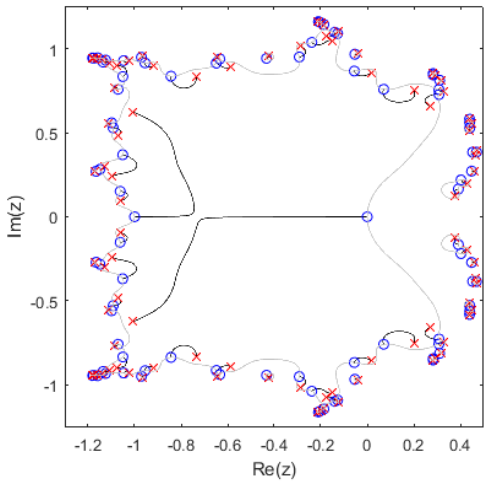
(c) n = 8



(d) n = 9



(e) n = 10



(f) n = 11

Figure 3.9: Plots of homotopy paths and contours $|q_n(z)| = 1$ of the Fibonacci-Mandelbrot polynomials from $n = 6$ to $n = 11$.

3.4.1 Distinctness

Just as we have seen with the homotopy paths for the Mandelbrot polynomials, the paths for the Fibonacci-Mandelbrot polynomials also do not cross (see Figures 3.8 and 3.9). We can also prove, just like in Section 2.5.2, that each initial value problem is unique as long as the singularities are avoided. However, this time, we do not need to be concerned about the initial condition, and can start with the zeros of $\zeta q_{n-1}(\zeta)q_{n-2}(\zeta)$, since these are not double roots.

Just as we did for the Mandelbrot polynomials, letting our region $\mathcal{R} = \mathbb{C}$, we can find the Lipschitz constant for the homotopy for the Fibonacci-Mandelbrot polynomials

$$\begin{aligned} |f(t,u) - f(t,v)| \leq L\,|u-v| &= \left| \frac{-1}{q_n'(u)} - \frac{-1}{q_n'(v)} \right| \\ &= \left| \frac{-q_n'(v) + q_n'(u)}{q_n'(u)q_n'(v)} \right| . \end{aligned} \tag{3.22}$$

Let

$$\begin{aligned} q_n'(v) &= q_n'(u + v - u) \\ &= q_n'(u) + q_n''(u)(v-u) + O(v-u) . \end{aligned} \tag{3.23}$$

Substituting Equation (3.23) into Equation (3.22), we get

$$\begin{aligned} L\,|u-v| &= \left| \frac{-q_n'(u) - q_n''(u)(v-u) + q_n'(u) + O(v-u)}{q_n'(u)q_n'(v)} \right| \\ &\doteq \frac{-q_n''(u)}{q_n'(u)q_n'(v)}\,|v-u| \\ &\doteq \frac{-q_n''(u)}{(q_n'(u))^2}\,|v-u| \ . \end{aligned} \tag{3.24}$$

Therefore,

$$L = \frac{-q_n''(u)}{(q_n'(u))^2} . \tag{3.25}$$

As mentioned in the previous chapter, as long as the path that we are taking is continuous, $q_n''(z)$

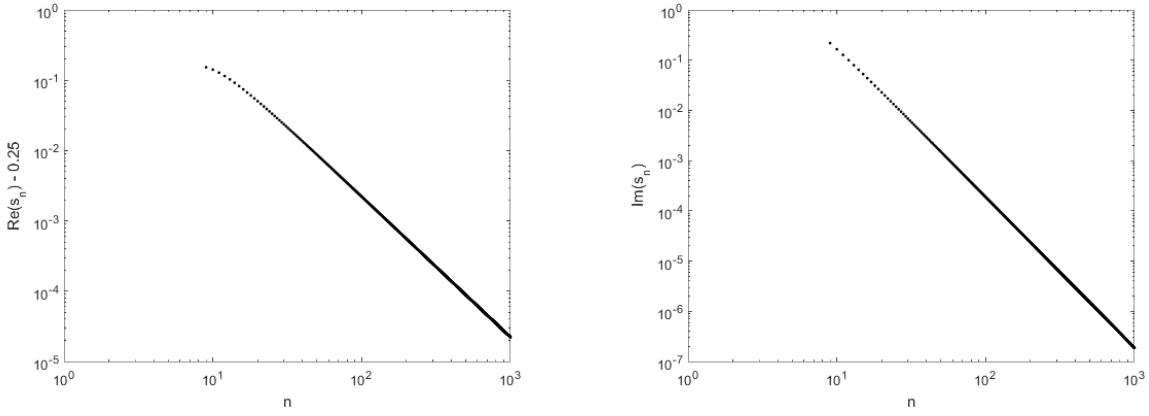
will always be bounded. Since we will be avoiding singularities (whenever $q_n'(z) = 0$) using our custom ode solver, we can ensure that L is in fact bounded, thus satisfying the Lipschitz condition. Therefore, just like the Mandelbrot polynomials, the initial value problems that we use for our homotopy will only give us one solution; hence, it is unique.

3.4.2 Smallest roots

Like the Mandelbrot polynomials, we can use our homotopy method to find the smallest roots of the Fibonacci-Mandelbrot polynomials by using the smallest roots from the previous iteration as our initial point for our differential equation. Again, we deduce that the smallest root has the form

$$s_n = \frac{1}{4} + \alpha_{\mathrm{Re}} n^{-\beta_{\mathrm{Re}}} \pm i\alpha_{\mathrm{Im}} n^{-\beta_{\mathrm{Im}}}. \tag{3.26}$$

Shown in Figure 3.10, we can plot the real part (minus $\frac{1}{4}$) and imaginary part of our smallest root against n, the iteration of the polynomials, in a log-log plot to see what β_{Re} and β_{Im} are. Similar to the Mandelbrot polynomials, β_{Re} and β_{Im} are 2 and 3 respectively, although (again) we are not sure about how exact these values are.



(a) Real part (b) Imaginary part

Figure 3.10: Log-log plots of smallest roots s_n of Fibonacci-Mandelbrot polynomials (difference from $1/4$).

Therefore, like the smallest roots of the Mandelbrot polynomials, the smallest roots of the

Fibonacci-Mandelbrot polynomials are

$$s_n \doteq \frac{1}{4} + \frac{\alpha_{\mathrm{Re}}}{n^2} \pm \frac{\alpha_{\mathrm{Im}}}{n^3}. \tag{3.27}$$

However, α_{Re} and α_{Im} are different: they are around 22.2 and 188.2 respectively.

3.4.3 Results

Using our own ode solver, described in Section 2.6, we were able to compute up to $n = 33$, which is $3,524,577$ roots, of order $\mathcal{O}(10^{-4})$ precision, using a machine with 32 GB of memory. This is shown in Figure 3.12. We were actually able to compute more roots than we did for the Mandelbrot polynomials using the same technique and same machine.

Figure 3.11 shows the time taken to compute the roots of the Fibonacci-Mandelbrot polynomials using our homotopy method. Like the result that we got for the time complexity when computing the roots of the Mandelbrot polynomials using our homotopy method, the slope of the line of best fit is less than 1: the slope is around 0.82. The line above the data is for reference as it has a slope of 1. As mentioned in the previous chapter, the value of less than 1 for the slope of our line of best fit does not make any sense; there is an overall lower bound of $\mathcal{O}(d_n \log(d_n))$ for this method. Therefore, we believe that, like for the Mandelbrot polynomials, it becomes easier to find the roots for higher iterations since the initial guess is closer to the final result.

Similar to the homotopy method that we used when solving for the Mandelbrot polynomials, we notice that for the Fibonacci-Mandelbrot polynomials that the accuracy decreases as the iterations increase, in which we believe is caused by mild instability in the iteration we use. Therefore, we need to to use higher precision in order to calculate higher iterations of the Fibonacci-Mandelbrot polynomials. Once again, we used Bailey's ARPREC package [2] for arbitrary precision in C++. Using this package, we were able to compute up to $n = 31$ ($1,346,268$ roots) thus far within $\mathcal{O}(10^{-12})$ precision.

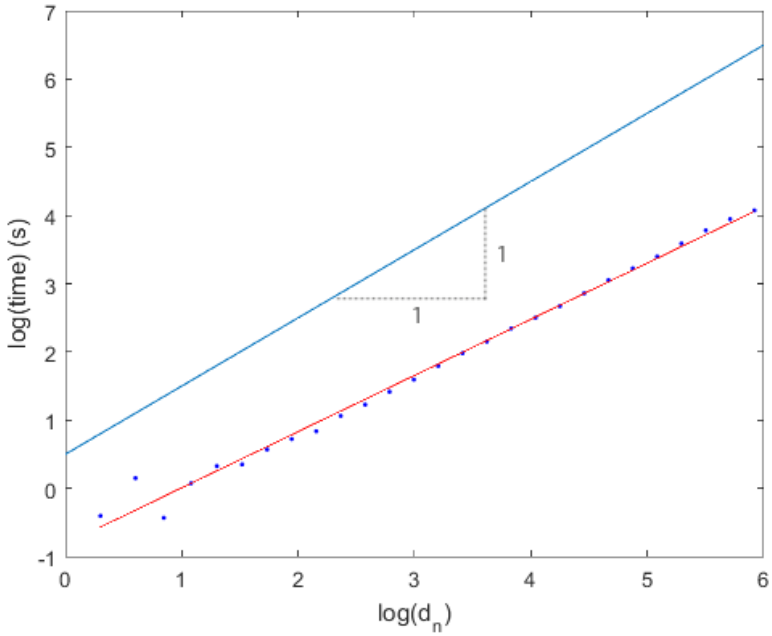


Figure 3.11: Time taken to compute roots of $q_n(z)$ using homotopy methods. The line of best fit has slope of 0.82.

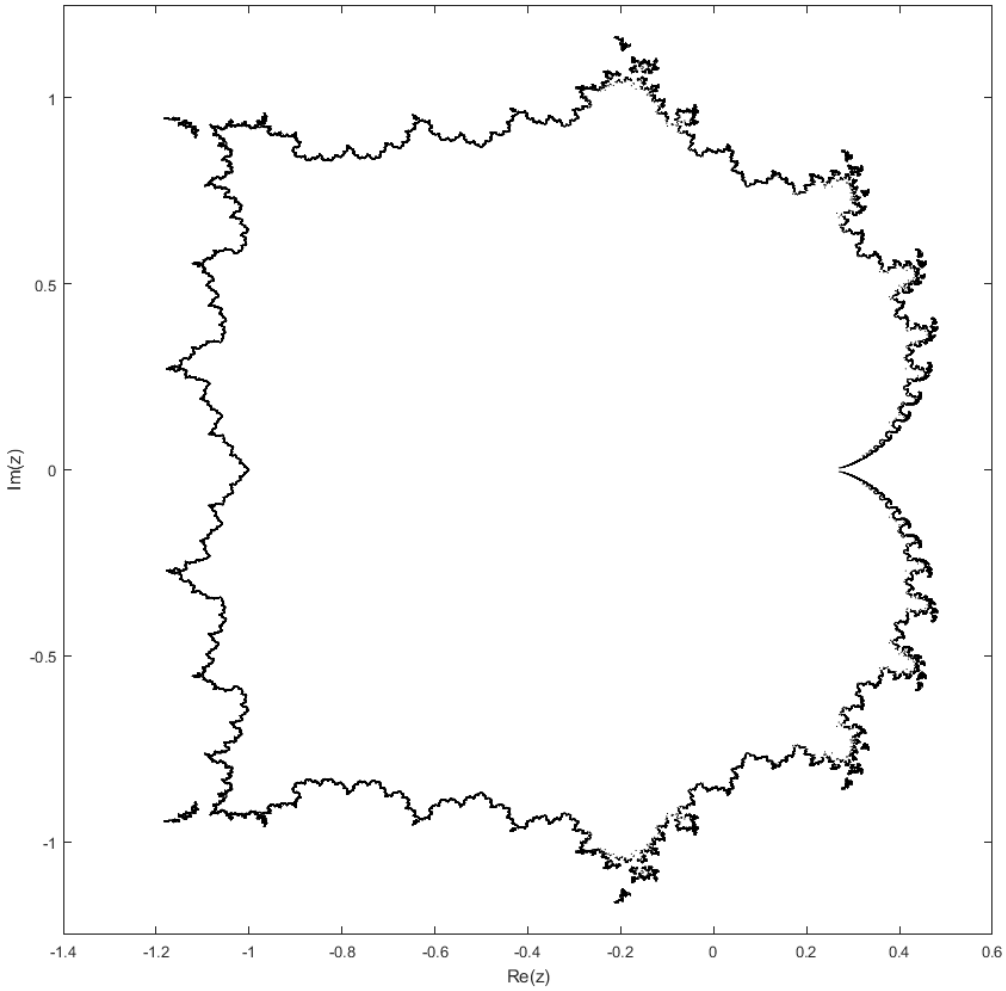


Figure 3.12: Plot of all $3,524,577$ roots of the Fibonacci-Mandelbrot polynomial $q_{33}(z)$. The residuals were all smaller than 10^{-4}.

Chapter 4

Narayana-Mandelbrot polynomials and matrices

4.1 Introduction

Using what we have learned from both the Mandelbrot and Fibonacci-Mandelbrot polynomials, we have decided to apply this knowledge to the Narayana-Mandelbrot polynomials, which are based on the Narayana's cows sequence. We first learned of the Narayana sequence at the Computational Discovery Conference 2016 in a talk by Neil J. A. Sloane.

4.1.1 Narayana's cows sequence

Sequence A000930 of the Online Encyclopedia of Integer Sequences [23], Narayana's cows sequence, begins

$$1, 1, 1, 2, 3, 4, 6, 9, 13, 19, \ldots \tag{4.1}$$

and can be generated by

$$S_{n+1} = S_n + S_{n-2}. \tag{4.2}$$

This sequence is named after a 14th-century Indian mathematician, who proposed the problem to compute the number of cows if a cow produces one calf every year, and in the beginning of its fourth year, each calf produces one calf at the beginning of each year. Many references are given in the OEIS, but see also [21].

4.1.2 Narayana-Mandelbrot polynomials

Similar to the Fibonacci-Mandelbrot polynomials, we can use the recursion from the sequence, which in this case, is the Narayana's cows sequence, to create our family of polynomials. The Narayana-Mandelbrot polynomials have the recursion

$$\begin{aligned} r_0(z) &= 1 \\ r_1(z) &= 1 \\ r_2(z) &= 1 \\ r_{n+1}(z) &= z r_n(z) r_{n-2}(z) + 1. \end{aligned} \tag{4.3}$$

where $n = 2, 3, 4, \ldots$. Using the monomial expansion, we can get the first few Narayana-Mandelbrot polynomials:

$$\begin{aligned} r_0(z) &= 1 \\ r_1(z) &= 1 \\ r_2(z) &= 1 \\ r_3(z) &= z + 1 \\ r_4(z) &= z^2 + z + 1 \\ r_5(z) &= z^3 + z^2 + z + 1 \\ r_6(z) &= z^5 + 2z^4 + 2z^3 + 2z^2 + z + 1 \\ r_7(z) &= z^8 + 3z^7 + 5z^6 + 6z^5 + 5z^4 + 4z^3 + 2z^2 + z + 1. \end{aligned} \tag{4.4}$$

The Narayana-Mandelbrot polynomials share a few properties with the Fibonacci-Mandelbrot polynomials such as

1. The leading and trailing coefficients are 1.

2. All coefficients are positive integers.

3. The polynomials are unimodular.

However, they do have some properties that are unique to this family of polynomials:

1. The next-to-leading coefficient is a number from the Narayana's cows sequence.

2. Put $d_n = \deg r_n$. Then $d_1 = 0$, $d_2 = 0$, and $d_3 = 0$, then $d_{n+1} = d_n + d_{n-2}$ or $d_n = S_n - 1$, where S_n is a number from the Narayana's cows sequence.

4.2 Condition numbers and pseudozeros

Like the other two families of polynomials that we have already seen, the absolute condition number of the roots is the reciprocal of the derivative of our polynomial, $r_n(z)$. Figure 4.1 shows the condition numbers of the Narayana-Mandelbrot polynomials. The circles are our maximum condition number, computed by using the minimum value of $\left|r_n'(z)\right|$ evaluated at its roots, ξ_n. It can be seen that the maximum condition number we encounter is 1, which occurs every other iteration. The crosses, on the other hand, are the minimum condition numbers, calculated by taking the reciprocal of the maximum value of $\left|r_n'(\xi_n)\right|$. The slope of the line running through these points is around -1.8, which is around the results (of around -2; lower line shown as a reference) that we have been getting for the minimum condition numbers of the roots for the Mandelbrot polynomials and the Fibonacci-Mandelbrot polynomials.

Just as we did previously for the other two families of polynomials, we can look at the pseudozeros by plotting the contours of a small value and seeing how tightly they encircle the roots. Figure 4.2 shows the roots of $r_{21}(z)$ with $|r_{21}(z)| = 0.1$ in red. We can see from this figure

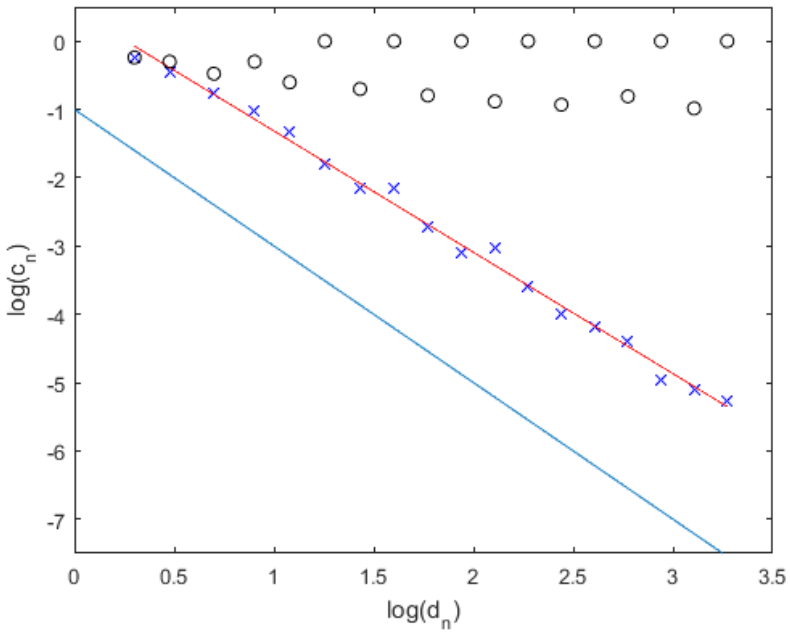


Figure 4.1: Condition numbers of the roots of the Narayana-Mandelbrot polynomials, $r_n(z)$.

that the contour around the root at -1 is quite large. However, considering that the contour is not connected to nearby contours, this shows that the root is still well-conditioned, like the rest of the roots shown here. Zooming into the regions that have visible contours from Figure 4.2 and reducing the value of the contour that we are plotting, we can have a closer look at the contours and how closely they wrap around the root, shown in Figure 4.3. Since all of the contours are very close to the roots where some of them not visible, we can see here that the roots are in fact well-conditioned.

4.3 Narayana-Mandelbrot matrices

Like both the Mandelbrot and Fibonacci-Mandelbrot polynomials, we can produce recursively-constructed supersparse companion matrices for the Narayana-Mandelbrot polynomials. Thus, the Narayana-Mandelbrot companion matrix construction is as follows.

We start off the recursion with

$$\mathbf{M}_3 = [-1]\,, \tag{4.5}$$

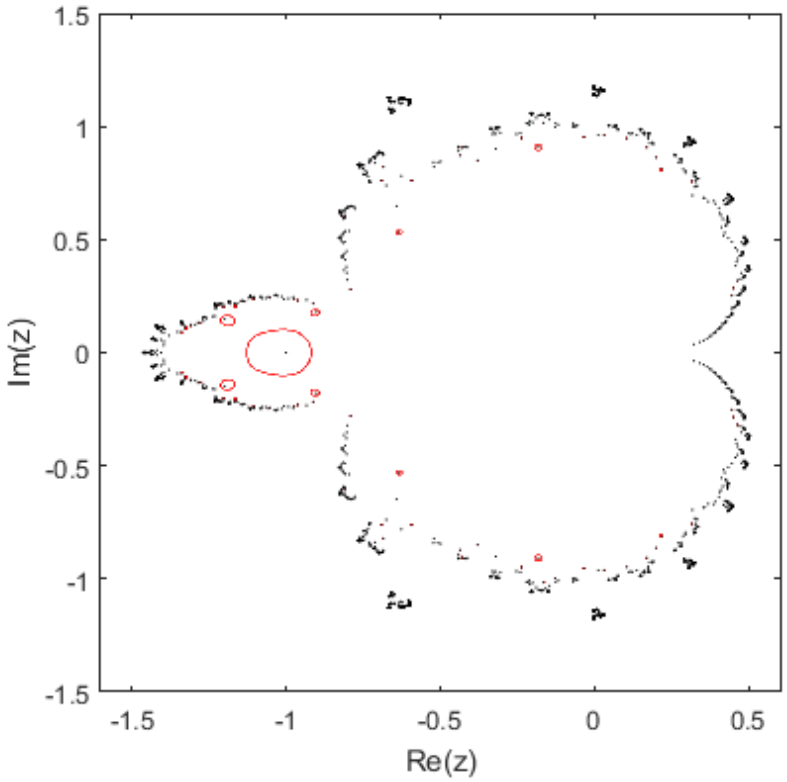


Figure 4.2: Roots of $r_{21}(z)$ with $|r_{21}(z)| = 0.1$ in red.

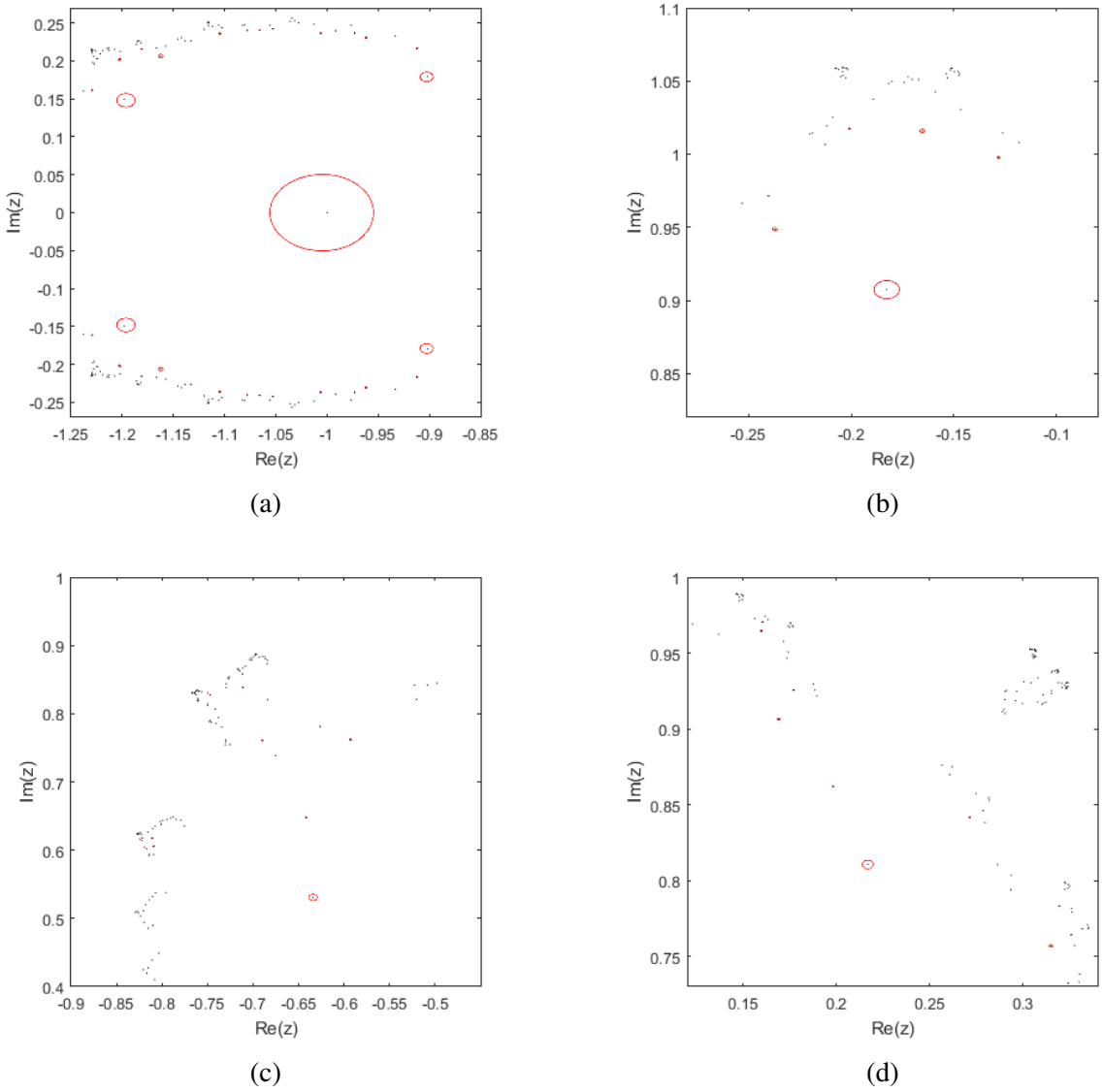


Figure 4.3: Different regions the Narayana-Mandelbrot polynomials where the roots are not as well-conditioned with $|r_{21}(z)| = 0.05$ in red.

just as we did for the Fibonacci-Mandelbrot matrices. Since $r_4(z)$ is the same as $q_4(z)$, we could use the same matrix which we used for the Fibonacci-Mandelbrot matrices:

$$\mathbf{M}_4 = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}. \tag{4.6}$$

The reason why we chose to have $\mathbf{M}_4$ in this formation for the Fibonacci-Mandelbrot matrices is so that our sub-diagonal in our companion matrices are always -1. In this case, we have decided to take the transpose of $\mathbf{M}_4$ from the Fibonacci-Mandelbrot matrices instead. Thus, for the Narayana-Mandelbrot matrices,

$$\mathbf{M}_4 = \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix} \tag{4.7}$$

Since the Narayana-Mandelbrot polynomials take its $r_n(z)$ and $r_{n-2}(z)$ polynomials in its recursion, it means that we will be taking $\mathbf{M}_n$ and $\mathbf{M}_{n-2}$ in order to construct the next matrix. Therefore, we also need

$$\mathbf{M}_5 = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \tag{4.8}$$

as well. Note that we can also use Equation (4.6) for our recursion for $\mathbf{M}_5$ as well so that

$$\mathbf{M}_5 = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix} \tag{4.9}$$

also works. Letting $\mathbf{r}_n = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}$ and $\mathbf{c}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$, where the lengths of

these vectors are of d_n, our construction becomes

$$\mathbf{M}_{n+1} = \begin{bmatrix} \mathbf{M}_n & & -\mathbf{c}_n\mathbf{r}_{n-2} \\ -\mathbf{r}_n & 0 & \\ & -\mathbf{c}_{n-2} & \mathbf{M}_{n-2} \end{bmatrix}. \tag{4.10}$$

As you can see, these matrices are also upper Hessenberg, and the construction of these matrices are quite similar to the construction of the Fibonacci-Mandelbrot matrices: the main difference is that the matrix in the lower right corner is $\mathbf{M}_{n-2}$ instead of $\mathbf{M}_{n-1}$. For this particular construction, the value in the upper right corner is always -1 and is not dependent on the dimension of our matrix, since the number of -1 on the sub diagonal is always even (proof in Chapter 5 will show the relationship between the elements in the subdiagonal and the element in the upper right corner). If we used $\mathbf{M}_4$ from Equation (4.6) and $\mathbf{M}_5$ from Equation (4.9) for our construction so that the subdiagonal only consisted of -1, then the element in the upper right corner would be dependent on the dimension of the matrix.

The following are the next few Narayana-Mandelbrot matrices using the construction from Equation (4.10):

$$\mathbf{M}_6 = \begin{bmatrix} 0 & 0 & -1 & & -1 \\ 1 & 0 & -1 & & \\ & 1 & -1 & & \\ & & -1 & & \\ & & & -1 & -1 \end{bmatrix} \tag{4.11}$$

and

$$\mathbf{M}_7 = \begin{bmatrix} 0 & 0 & -1 & 0 & -1 & & & -1 \\ 1 & 0 & -1 & 0 & 0 & & & \\ & 1 & -1 & 0 & 0 & & & \\ & & -1 & 0 & 0 & & & \\ & & & -1 & -1 & & & \\ & & & & -1 & & & \\ & & & & & -1 & 0 & -1 \\ & & & & & & 1 & -1 \end{bmatrix}. \tag{4.12}$$

4.3.1 Results

Using MATLAB, we were only able to compute up to $n = 27$, which has a dimension of $18,559$ roots, shown in Figure 4.4. Comparing to this result to the other matrices that we have computed the eigenvalues of previously, there is quite a huge difference in the dimension of the roots. For the Mandelbrot matrices, we were able to solve up to $32,767$ roots, whereas for the Fibonacci-Mandelbrot matrices, we were able to solve up to $28,656$ roots for one of our recursive companion matrices. For these two families of matrices, the main problem was the lack of memory that the machine that we were using has (32 GB). However, for the Narayana-Mandelbrot matrices, we actually encounter some problems when evaluating the eigenvalues of $\mathbf{M}_{28}$.

In Figure 4.5, it shows the eigenvalues that MATLAB finds when evaluating $\mathbf{M}_{28}$: on the left is the full plot, and on the right is zoomed-in to the portion where the roots of $r_{28}(z)$ should reside. As mentioned in the previous chapter, we can also switch $\mathbf{M}_n$ and $\mathbf{M}_{n-2}$ around with the correct corresponding $\mathbf{r}_n$ and $\mathbf{c}_n$. As we saw in the previous chapter, this could potentially help us compute either more or less (correct) eigenvalues. The recursion for the matrix construction

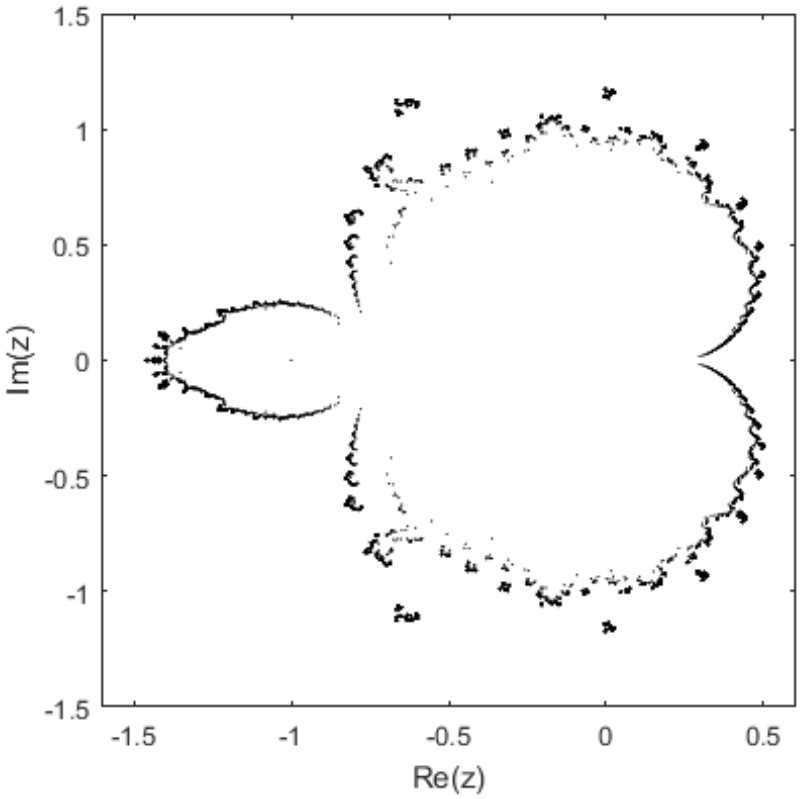
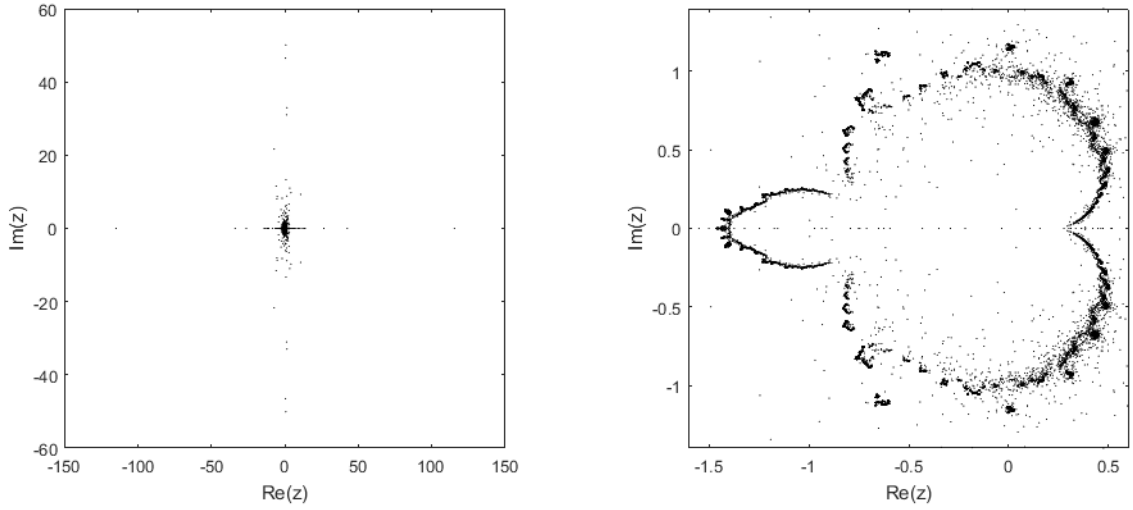


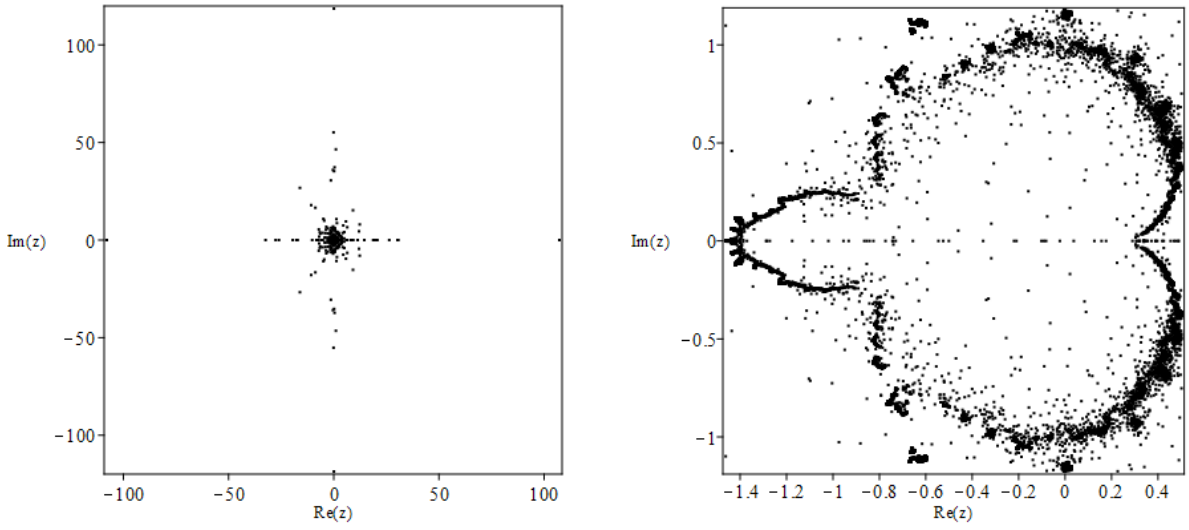
Figure 4.4: Roots of $r_{27}(z)$, which has a dimension of $18,559$.



(a) Overall results

(b) Zoomed version of results.

Figure 4.5: Results MATLAB gives when evaluating the eigenvalues of $\mathbf{M}_{28}$ using recursion from Equation (4.10), showing numerical artefacts.



(a) Overall results

(b) Zoomed version of results.

Figure 4.6: Results Maple gives when evaluating the eigenvalues of $\mathbf{M}_{28}$ using recursion from Equation (4.13), again showing numerical artefacts.

becomes

$$\mathbf{M}_{n+1} = \begin{bmatrix} \mathbf{M}_{n-2} & & -\mathbf{c}_{n-2}\mathbf{r}_n \\ -\mathbf{r}_{n-2} & 0 & \\ & -\mathbf{c}_n & \mathbf{M}_n \end{bmatrix}, \tag{4.13}$$

using the same $\mathbf{M}_3$, $\mathbf{M}_4$, and $\mathbf{M}_5$. This time, using Maple 2016 to solve the eigenvalues of $\mathbf{M}_{28}$ based on the recursion in Equation (4.13), we can see that this also fails to give us the correct roots for $r_{28}(z)$. In an attempt to improve on this result, we tried to increase the number of digits used to compute the eigenvalues of this matrix. Unfortunately, we were unable to retrieve any results for this as it overloaded the CPU of the machine that we used. Note that we did not use Maple 2015, which we saw was able to compute the eigenvalues of the Fibonacci-Mandelbrot matrices correctly up to $n = 23$, to solve for the eigenvalues of the Narayana-Mandelbrot matrices.

We also recorded the time it takes in MATLAB to compute the eigenvalues of the Narayana-Mandelbrot matrices (using the recursion in Equation (4.10)). The slope running through the points in the figure is around 2.3, which is what we have been getting for the time of the other two matrices. The estimated time complexity of $O(d_n^{2.3})$ is less than the expected time complexity of $O(d_n^3)$. To show this, there is a line above our line of best fit, as a reference.

4.4 Homotopy methods

Similar to our previous two families of polynomials, we can use homotopy methods to solve for the roots of the Narayana-Mandelbrot polynomials. Consider the following homotopy:

$$H_n(\zeta, \tau) = \zeta r_{n-1}(\zeta) r_{n-3}(\zeta) + \tau, \tag{4.14}$$

which is very similar to the homotopy used for the Fibonacci-Mandelbrot polynomials (Equation (3.16)). When $\tau = 0$, the zeros of $H_n(\zeta, 0)$ are $\zeta = 0$, the zeros of $r_{n-1}(\zeta)$ and the zeros of $r_{n-3}(\zeta)$. When $\tau = 1$, it is the same equation as Equation (4.3); thus, the zeros of $H_n(\zeta, 1)$ are

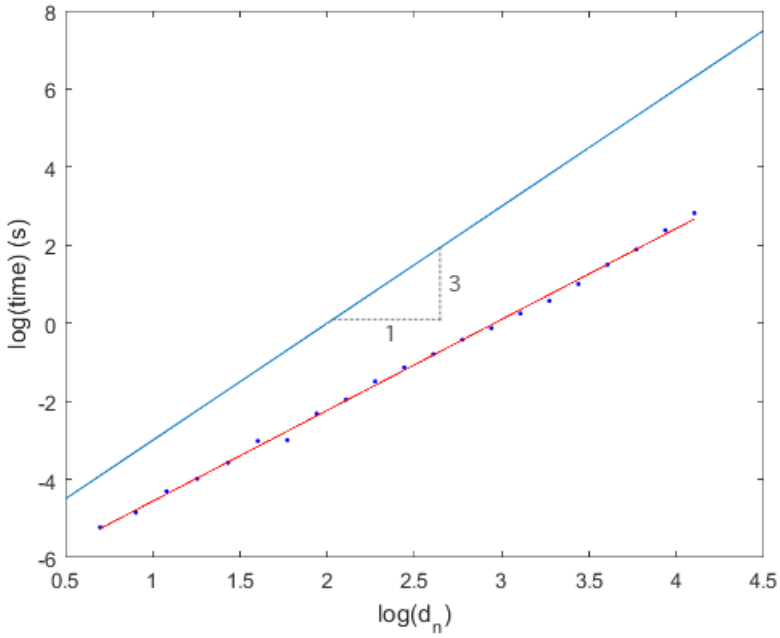


Figure 4.7: Time taken for computing the eigenvalues of the Narayana-Mandelbrot matrices.

the zeros of $r_n(z)$. Differentiating the right-hand side of Equation (4.14) with respect to τ, as we did for the Equation (3.16), we get the following differential equation

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = \frac{-1}{r_n'(\zeta)}, \quad \tau \in [0,1]\,, \tag{4.15}$$

which has essentially the same form as Equation (3.17). Unfortunately, just as with the other two families of polynomials, we do encounter singularities when integrating along the real axis. This could be seen in the previous chapter when we computed the location of the singularities for $q_4(z) = z^2+z+1$. (Note that $r_4(z) = q_4(z)$.) Therefore, we again need to use the pole-vaulting technique described in Section 2.5 to avoid the singularities.

Since the roots are periodic, as mentioned previously when listing properties of the Narayana-Mandelbrot polynomials, we do have to be mindful since we will encounter some duplicate roots when using the previous roots as our initial conditions for our differential equation (Equation (4.3)). Using $n = 6$ as an example, it can be seen that our initial condition for our differ-

ential equation

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\tau} = \frac{-1}{r_6'(z)} = \frac{-1}{5z^4 + 8z^3 + 6z^2 + 4z + 1} \tag{4.16}$$

would be 0, the roots of $r_3(z) = z + 1$ and the roots of $r_5(z) = z^3 + z^2 + z + 1$. It can easily be seen that the root of $r_3(z)$ is -1. The roots of $r_5(z)$ are not as easy, but still fairly simple to evaluate: $\xi_5 = -1, \pm i$. From this, you can see that we start from -1 twice, since -1 is the root of both $r_3(z)$ and $r_5(z)$. Therefore, we can use the same technique (shown in Section 2.4) to perturb our initial conditions for these double roots so that the paths can go off in separate directions to find the two roots that stem from that single point.

In Figure 4.8, we can see the paths taken from our initial points to our roots ξ_6, which are represented by the squares. The circle represents our initial condition $\zeta = 0$, the cross represents ξ_3 and the diamonds represent ξ_5. It can be seen in this figure that there are two pathways that come out from -1, marked with both a cross and a diamond. The plot also has the contour $|r_6(z)| = 1$ in grey.

We also plotted the homotopy paths of the following 6 iterates, shown in Figure 4.9. To simplify the plots, instead of showing which roots each initial condition comes from, all initial points are represented by circles, and the final roots are represented by crosses. From these plots, it can be seen, from the two pathways coming out from one point, that there are other points where the double roots occur, not just at -1.

4.4.1 Smallest roots

We can compute the smallest roots of the Narayana-Mandelbrot polynomials using the homotopy method described previously. Figure 4.10a shows the real part of the smallest roots minus $1/4$, while Figure 4.10b shows the imaginary part of the smallest roots of the Narayana-Mandelbrot polynomials. Like the Mandelbrot and Fibonacci-Mandelbrot polynomials, we

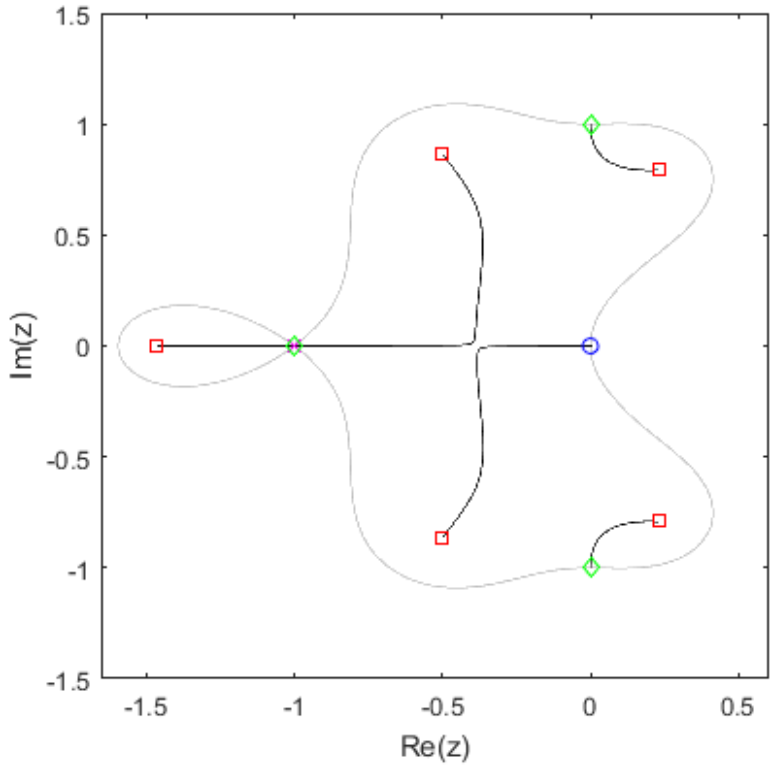
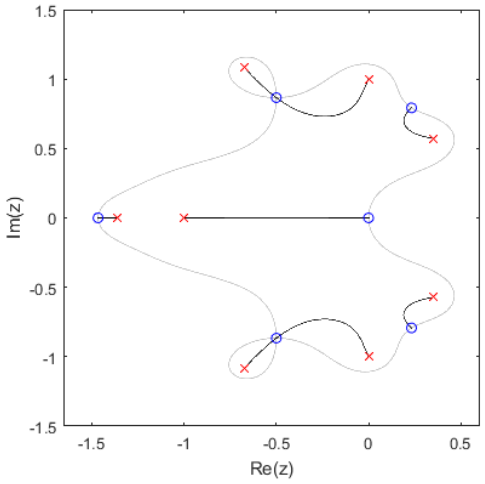
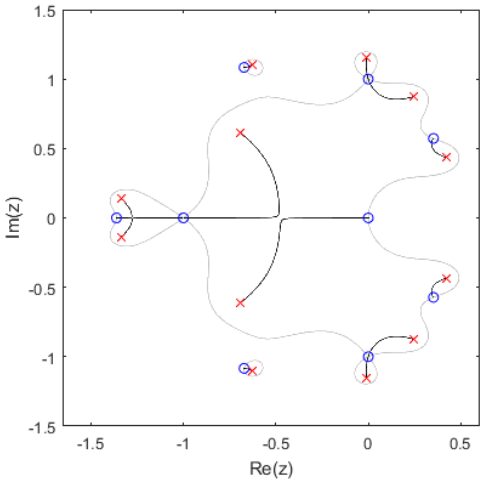


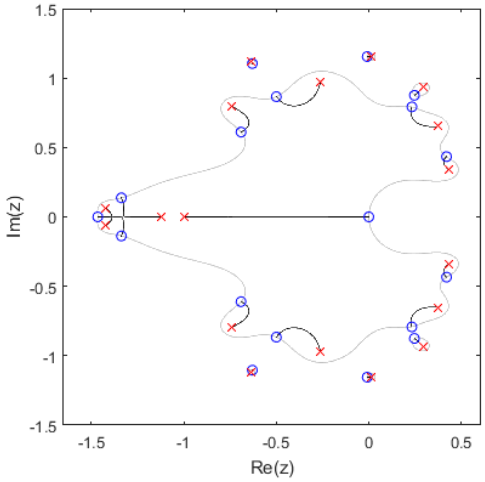
Figure 4.8: Homotopy paths for $r_6(z)$ and contour where $|r_6(z)| = 1$.



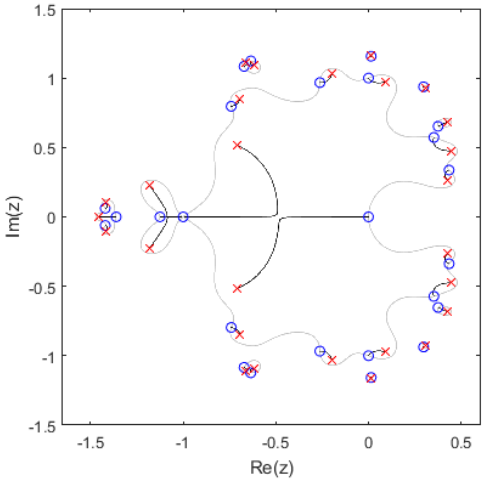
(a) n = 7



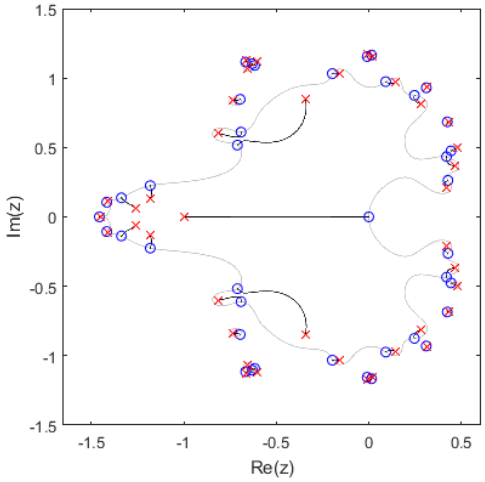
(b) n = 8



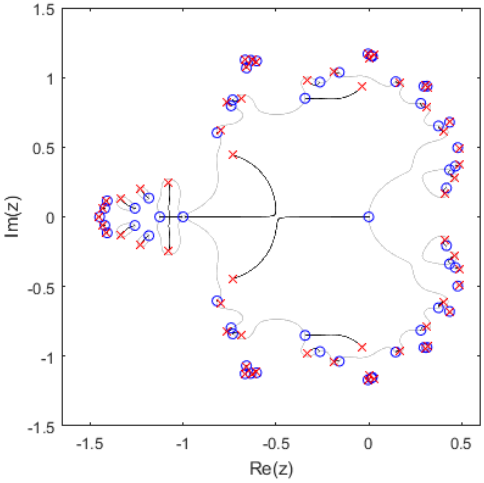
(c) n = 9



(d) n = 10



(e) n = 11



(f) n = 12

Figure 4.9: Plots of homotopy paths and contour $|r_n(z)| = 1$ of the Narayana-Mandelbrot polynomials from $n = 7$ to $n = 12$.

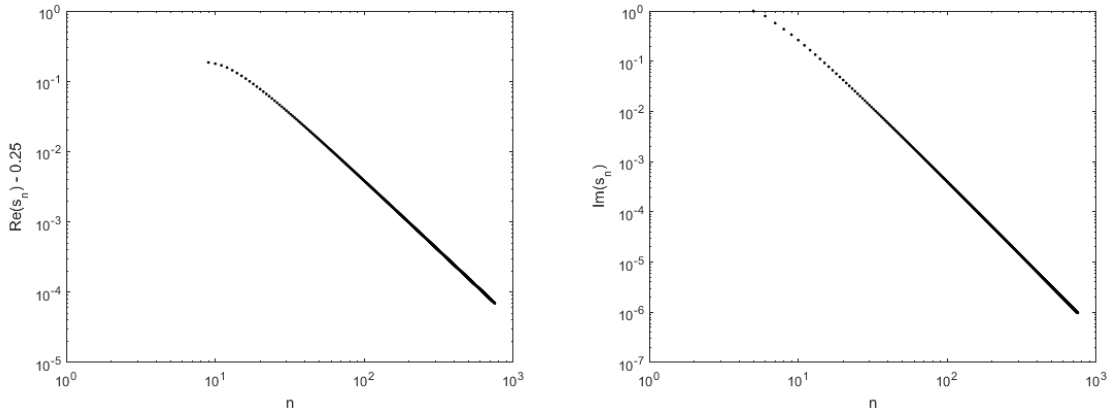
deduced that the smallest roots have the form

$$s_n = \frac{1}{4} + \alpha_{\mathrm{Re}} n^{-\beta_{\mathrm{Re}}} \pm i\alpha_{\mathrm{Im}} n^{-\beta_{\mathrm{Im}}}. \tag{4.17}$$

Similar to the smallest roots of both the Mandelbrot and the Fibonacci-Mandelbrot polynomials the slopes of the real part and the imaginary part are around -2 and -3 respectively, which means that $\beta_{\mathrm{Re}} = 2$ and $\beta_{\mathrm{Im}} = 3$.

Knowing what the β's are, we can now find what the values of our α's by multiplying n^2 or n^3 to the corresponding real part minus $1/4$ or the imaginary part of the smallest roots respectively. From doing so, the smallest roots seem to be

$$s_n \doteq \frac{1}{4} + \frac{39.2}{n^2} \pm \frac{409.5}{n^3}. \tag{4.18}$$



(a) Real part (b) Imaginary part

Figure 4.10: Smallest roots of the Narayana-Mandelbrot polynomials (difference from $1/4$).

4.4.2 Results

We used both MATLAB and David Bailey's ARPREC package in C++ to compute the roots of the Narayana-Mandelbrot polynomials using the homotopy method described above. We only

used our own ode solver (details in Section 2.6) in MATLAB to give us a rough idea of what the time complexity of computing the roots of the Narayana-Mandelbrot polynomials using a homotopy method, since it was able to give us results a lot quicker compared to our C++ code, which uses multiple precision. However, we knew that we would eventually have to use the multiple precision package written in C++ in order to compute the roots of higher degrees. Therefore, we decided not to pursue computing a large number of roots using our ode solver in MATLAB.

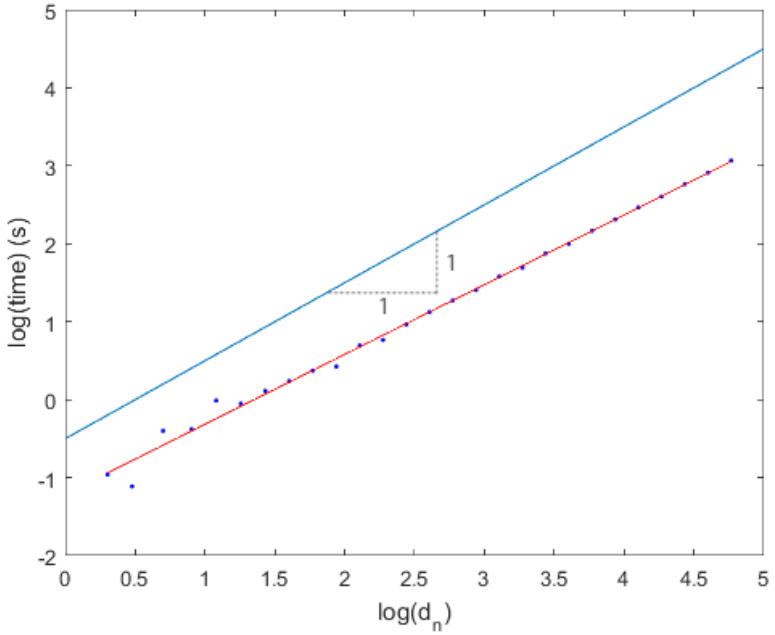


Figure 4.11: Time taken to compute roots of $r_n(z)$ using homotopy methods.

Figure 4.11 shows the amount of time it takes to compute the roots of the Narayana-Mandelbrot polynomials using a homotopy method in MATLAB, using our own ode solver. Similar to the other families of polynomials, the slope of the line running through our data points is around 0.9, which is less than 1. As mentioned before, this is impossible as the homotopy method has a lower limit of $O(d_n \ln d_n)$, and we believe that the "constant" is hidden in the O, and that it decreases for higher iterates.

Using Bailey's ARPREC package [2], we were able to compute up to $n = 36$, which is a degree of $578,948$, thus far, with $O(10^{-9})$ precision, shown in Figure 4.12.

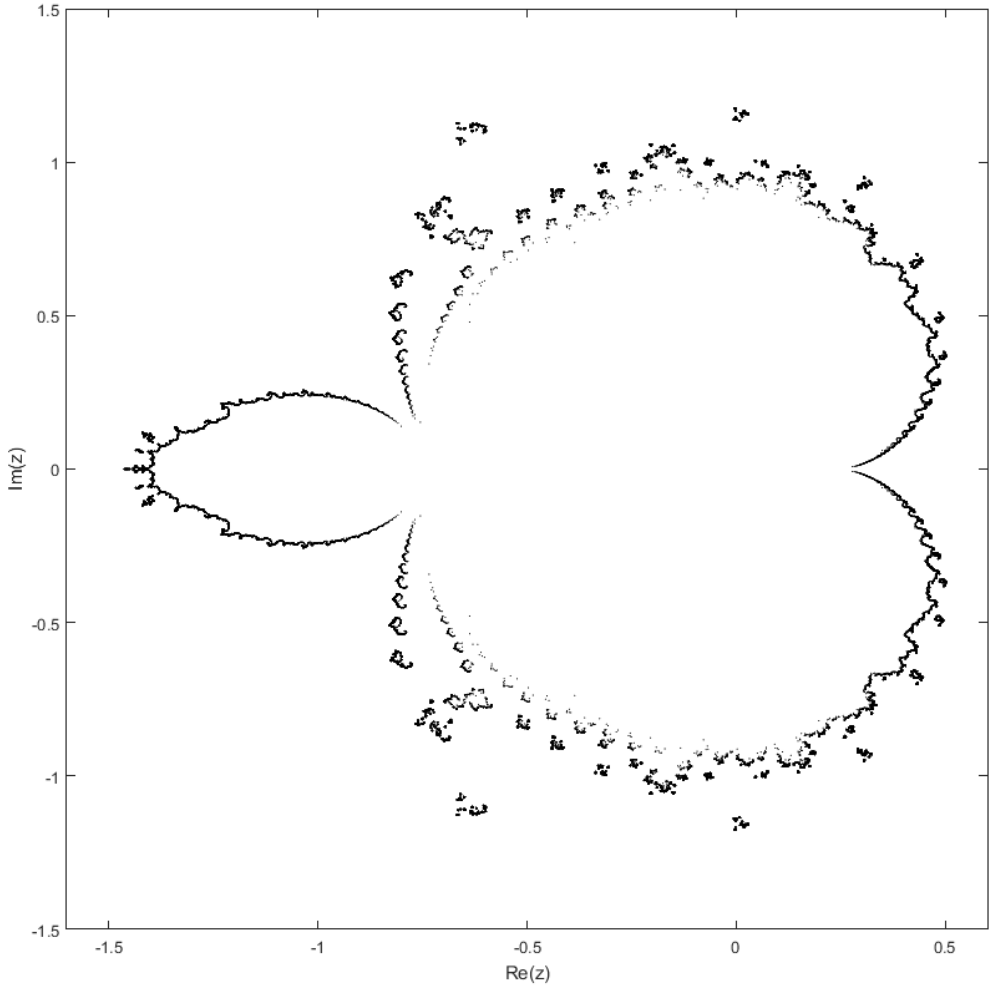


Figure 4.12: Roots of $r_{36}(z)$, which has a degree of $578,948$.

Chapter 5

Concluding Remarks

In this thesis, we explored two different methods for finding the roots of three families of polynomials: Mandelbrot, Fibonacci-Mandelbrot, and Narayana-Mandelbrot polynomials. For the first method, we found the roots of the polynomials by computing the eigenvalues of a supersparse, recursively-constructed companion matrix. Piers Lawrence first introduced this construction for the Mandelbrot matrices, and we have applied a new, similar construction to both the Fibonacci-Mandelbrot and the Narayana-Mandelbrot matrices, thus creating a new kind of companion matrix. For our second method, we used new homotopy methods to find the roots of the families of polynomials.

Comparing the two methods that we explored, homotopy methods are clearly superior in both time and space complexity. As mentioned in our discussions in our previous chapters, the time complexity of the eigenvalue method appeared to be around $O(d_n^{2.3})$, whereas the time complexity that we computed for the homotopy method appear to be around $O(d_n^{0.9})$. Of course in reality, this should have a lower bound of $O(d_n \log d_n)$, which is still less than the time complexity of solving for eigenvalues.

When we say $f(d) = O(d^n)$ as $d \to \infty$, we could mean that there exists a nonzero constant κ such that

$$\lim_{d\to\infty} \frac{f(d)}{d^n} = \kappa \,. \tag{5.1}$$

(This is a simple definition, maybe the simplest, of the order symbol). In fact, we are using "soft-oh," which allows logarithms: there exists a nonzero constant κ and a power β such that

$$\lim_{d\to\infty} \frac{f(d)}{d^n \ln^\beta d} = \kappa\,. \tag{5.2}$$

Experimentally, we try to estimate κ by looking at the value of $f(d)$ for "large" d: say, $d = D$.

$$\lim_{d\to\infty} \frac{f(d)}{d^n} \stackrel{?}{=} \frac{f(D)}{D^n} \tag{5.3}$$

We do not know if D is "large enough" to actually uncover κ accurately, though. For instance,

$$f(d) = 10^{-12} \cdot d^2 + 100 \cdot d \tag{5.4}$$

will look like $O(d)$ for d less than, say, 10^{12}. Similarly,

$$10^{-6} \cdot d \ln d + 100 \cdot d \tag{5.5}$$

will look like $O(d)$ for $d < e^{10^6}$.

Since we are using full matrices in order to compute our eigenvalues, this means that our space complexity for this method is $O(d_n^2)$. On the other hand, for the homotopy method, the space complexity is $O(d_n)$, although it might even be doable in *constant* space. We do realize that this comparison is somewhat unfair, since we can reduce the space complexity of our eigenvalue method by using sparse matrices seeing that these companion matrices are supersparse. However, evaluating these eigenvalues using MATLAB's `eigs` routine to take advantage of the sparseness of the matrices does come with some challenges, such as determining which regions to look at to evaluate the roots, and removing all of the duplicates once we collect all the results together. Due to time constraints, we decided not to look very closely into using sparse matrices to compute our roots; thus, more research would be needed to be done for a

fairer comparison between the two methods.

5.1 Future Work

Of the three families of polynomials that we have studied in this thesis, only the Mandelbrot polynomials have been studied before (very extensively, one might add). Much is already known about the roots of the Mandelbrot polynomials and the properties of the Mandelbrot set (see [19, Chapter 4]). However, very little is known about both the Fibonacci-Mandelbrot and Narayana-Mandelbrot polynomials, since they are both completely new families of polynomials based on the Mandelbrot polynomials and the Fibonacci and Narayana sequences respectively. Therefore, exploring these polynomials further and learning more about the roots of these families of polynomials would be an interesting extension of this work. At MICA 2016, Joachim von zur Gathen suggested to us that these may have applications in random number generation or in primality testing for cryptography.

Additionally, the companion matrix construction first introduced by Piers Lawrence for the Mandelbrot matrices, which we have extended to the Fibonacci-Mandelbrot and Narayana-Mandelbrot matrices, is genuinely a completely new kind of companion matrix. We can prove that the construction is valid by using induction and the Schur determinantal formula. The surprising analogy between all three families of supersparse companions led us to conjecture and prove the following.

Theorem 5.1.1 *Suppose* $a(z) = \det(z\mathbf{I} - \mathbf{A})$, $b(z) = \det(z\mathbf{I} - \mathbf{B})$, *and both* $\mathbf{A}$ *and* $\mathbf{B}$ *are upper Hessenberg matrices with nonzero subdiagonal entries, and*

$$\alpha = \frac{1}{\left(\prod_{j=1}^{d_a-1} a_{j+1,j}\right)\left(\prod_{j=1}^{d_b-1} b_{j+1,j}\right)} \tag{5.6}$$

is the reciprocal of the product of the subdiagonal entries of $\mathbf{A}$ *and* $\mathbf{B}$, *and* $d_a = \deg_z a$ *and* $d_b = \deg_z b$, *so the dimension of* $\mathbf{A}$ *is* $d_a \times d_a$ *and the dimension of* $\mathbf{B}$ *is* $d_b \times d_b$. *Suppose both*

d_a and d_b are at least 1. Then if

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} & & -\alpha c_0 \mathbf{c_a} \mathbf{r_b} \\ -\mathbf{r_a} & 0 & \\ & -\mathbf{c_b} & \mathbf{B} \end{bmatrix} \tag{5.7}$$

where $\mathbf{r_a} = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}$ *of length* d_a, $\mathbf{c_b} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$ *of length* d_b, *we have*

$$c(z) = \det(z\mathbf{I} - \mathbf{C}) = z \cdot a(z)b(z) + c_0. \tag{5.8}$$

Remark *Proving this theorem automatically proves the validity of the constructions of the supersparse companion matrices for p_n, q_n, and r_n.*

Remark *Starting with a polynomial $c(z)$, we see that there are potentially many such $a(z)$ and $b(z)$. This freedom may be quite valuable or, it may be an obstacle.*

Proof Partition

$$z\mathbf{I} - \mathbf{C} = \left[\begin{array}{c:c} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \hdashline \mathbf{C}_{21} & \mathbf{C}_{22} \end{array}\right] \tag{5.9}$$

where $\mathbf{C}_{22} = z\mathbf{I} - \mathbf{B}$ is nonsingular if z is not an eigenvalue of $\mathbf{B}$, i.e. $b(z) \neq 0$. Later we will remove this restriction. Also,

$$\mathbf{C}_{21} = \begin{bmatrix} & 1 \\ & \\ & \end{bmatrix} \tag{5.10}$$

is $d_b \times (d_a + 1)$ and has only one nonzero element, which is a 1 in the upper right corner. Next,

$$\mathbf{C}_{12} = \begin{bmatrix} & \alpha c_0 \\ & \\ & \end{bmatrix} \tag{5.11}$$

is $(1 + d_a) \times d_b$ and again has only one nonzero element, αc_0 in the upper right corner. [In fact, c_0 can be zero.] This leaves

$$\mathbf{C}_{11} = \left[\begin{array}{cc:c} & & 0 \\ & z\mathbf{I} - \mathbf{A} & \vdots \\ & & 0 \\ & & 0 \\ \hdashline & 1 & z \end{array}\right] \tag{5.12}$$

which is $d_a + 1$ by $d_a + 1$.

The Schur factoring is

$$\left[\begin{array}{cc} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{array}\right] = \left[\begin{array}{cc} \mathbf{I} & \mathbf{C}_{12} \\ 0 & \mathbf{C}_{22} \end{array}\right] \left[\begin{array}{cc} \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21} & 0 \\ \mathbf{C}_{22}^{-1}\mathbf{C}_{21} & \mathbf{I} \end{array}\right] \tag{5.13}$$

with the computation of the Schur complement $\mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}$ going to do most of the work in the proof. The Schur determinantal formula [15] is then

$$\det \mathbf{C} = \det\left(\mathbf{C}_{22}\right) \det\left(\mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}\right). \tag{5.14}$$

We have the following propositions.

0. $z\mathbf{I} - \mathbf{A}$ and $z\mathbf{I} - \mathbf{B}$ are upper Hessenberg because $\mathbf{A}$ and $\mathbf{B}$ are.

1. The first d_a columns of $\mathbf{C}_{22}^{-1}\mathbf{C}_{21}$ are zero.

2. The final column of $\mathbf{C}_{22}^{-1}\mathbf{C}_{21}$ is the solution, say $\vec{v}$, of $(z\mathbf{I} - \mathbf{B})\,\vec{v} = \mathbf{e}_1$. Again, $z\mathbf{I} - \mathbf{B}$ is nonsingular.

3. By Cramer's rule, the final entry in $\vec{v}$, say v, is

$$v = \frac{\det\left(\mathbf{C}_{22} \underset{d_b}{\leftarrow} \mathbf{e}_1\right)}{\det\left(\mathbf{C}_{22}\right)} \tag{5.15}$$

where the notation $\mathbf{M} \underset{k}{\leftarrow} \vec{v}$ means replace the kth column of $\mathbf{M}$ with the vector $\vec{v}$ [7].

4. Since $\mathbf{C}_{22} = z\mathbf{I} - \mathbf{B}$ is upper Hessenberg,

$$\mathbf{C}_{22} \underset{d_b}{\leftarrow} e_1 = \begin{bmatrix} * & * & * & \cdots & * & 1 \\ -b_{21} & * & * & \cdots & * & 0 \\ & -b_{32} & * & & \vdots & \vdots \\ & & -b_{43} & \ddots & & \\ & & & \ddots & * & 0 \\ & & & & -b_{d_b,d_b-1} & 0 \end{bmatrix}. \tag{5.16}$$

Laplace expansion about the final column gives

$$\begin{aligned} \det\left(\mathbf{C}_{22} \underset{d_b}{\leftarrow} \mathbf{e}_1\right) &= (-1)^{d_b-1}(-1)^{d_b-1} \prod_{j=1}^{d_b-1} b_{j+1,j} \\ &= \prod_{j=1}^{d_b-1} b_{j+1,j}. \end{aligned} \tag{5.17}$$

Therefore,

$$v = \frac{\prod_{j=1}^{d_b-1} b_{j+1,j}}{b(z)} \tag{5.18}$$

because $\det \mathbf{C}_{22} = \det(z\mathbf{I} - \mathbf{B}) = b(z)$ by hypothesis.

5. Now

$$\mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21} = \begin{bmatrix} & \alpha c_0 & \\ & & \end{bmatrix} \begin{bmatrix} & * & \\ & \vdots & \\ & * & \\ & v & \end{bmatrix} = \begin{bmatrix} & \alpha c_0 v & \\ & & \end{bmatrix} \tag{5.19}$$

is $d_a + 1$ by $d_a + 1$ and has its only nonzero entry, $\alpha c_0 v$, in the upper right corner.

6. The Schur complement is therefore

$$\left[\begin{array}{ccc:c} & & & -\alpha c_0 v \\ & z\mathbf{I}-\mathbf{A} & & 0 \\ & & & \vdots \\ & & & 0 \\ \hdashline 0 & \cdots & 0 \;\; 1 & z \end{array}\right] \tag{5.20}$$

and we compute $\det\left(\mathbf{C}_{11}-\mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}\right)$ by Laplace expansion on the last column:

$$\begin{aligned}
\det\left(\mathbf{C}_{11}-\mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}\right) = &-(-1)^{d_a}\alpha c_0 v \det\begin{bmatrix} -a_{21} & * & * & \cdot & * \\ & -a_{32} & * & & * \\ & & -a_{43} & & \vdots \\ & & & \ddots & \\ & & & & -a_{d_a,d_a-1} \end{bmatrix} \\
&+ z\det(z\mathbf{I}-\mathbf{A}) \\
=&-(-1)^{d_a}\alpha c_0 v \prod_{j=1}^{d_a-1}\left(-a_{j+1,j}\right) + z\cdot a(z) \\
=&\alpha v \prod_{j=1}^{d_a-1} a_{j+1,j}\cdot c_0 + z\cdot a(z) \\
=&\alpha\cdot\frac{\left(\prod_{j=1}^{d_b-1} b_{j+1,j}\right)}{b(z)}\cdot\left(\prod_{j=1}^{d_a-1} a_{j+1,j}\right)\cdot c_0 + z\cdot a(z) \\
=&\frac{c_0}{b(z)} + z\cdot a(z)
\end{aligned} \tag{5.21}$$

by the definition of α.

Therefore by the Schur determinantal formula

$$\begin{aligned}\det(z\mathbf{I}-\mathbf{C}) &= \det(\mathbf{C}_{22})\det\left(\mathbf{C}_{11}-\mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}\right)\\ &= b(z)\left(\frac{c_0}{b(z)}+z\cdot a(z)\right)\\ &= z\cdot a(z)b(z)+c_0. \end{aligned} \tag{5.22}$$

Since the left hand side is a polynomial as is the right hand side, the formula will be true even if $b(z) = 0$, by continuity.

♮

As we have seen in Chapter 3, we are also interested in the inverses of these companion matrices. For the Fibonacci-Mandelbrot matrices, we noticed that the inverses of the companion matrices are also supersparse, containing only elements in $\{-1, 0, 1\}$. We are also interested in looking at the inverses for other companion matrices that follow this construction, particularly the Mandelbrot and Narayana-Mandelbrot matrices, to see whether they are also supersparse, and if any patterns that emerge.

We can also demonstrate this construction on Newton's example polynomial $x^3 - 2x - 5$. We see that $x^3 - 2x - 5 = x(x^2 - 2) - 5 = x(x - \sqrt{2})(x + \sqrt{2}) - 5$, and companion matrices for $x - \sqrt{2}$ and $x + \sqrt{2}$ are just $[+\sqrt{2}]$ and $[-\sqrt{2}]$ respectively. Thus a companion matrix for Newton's polynomial is

$$\begin{bmatrix} \sqrt{2} & & 5 \\ -1 & & \\ & -1 & -\sqrt{2} \end{bmatrix} \tag{5.23}$$

For unimodular polynomials, such companion matrices will be of lower height than the Frobenius or Fiedler [13] companions, and may offer better numerical condition.

We have now established that if $c(z) = z \cdot a(z)b(z) + c_0$ and $\mathbf{A}$ and $\mathbf{B}$ are upper Hessenberg

companion matrices for the polynomials $a(z)$ and $b(z)$ respectively, then

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} & & -\alpha c_0 \mathbf{c_a} \mathbf{r_b} \\ -\mathbf{r_a} & 0 & \\ & -\mathbf{c_b} & \mathbf{B} \end{bmatrix} \tag{5.24}$$

is a companion matrix for $c(z)$. One wonders immediately about a corresponding linearization, $\mathbf{L_C}$, strong or otherwise, for the matrix polynomial ($\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{C}_0 \in \mathbb{C}^{n\times n}$)

$$\mathbf{C}(z) = z\mathbf{A}(z)\mathbf{B}(z) + \mathbf{C_0} \, . \tag{5.25}$$

Suppose $\mathbf{L_A}$ is a block upper Hessenberg linearization for $\mathbf{A}$, $\mathbf{L_B}$ for $\mathbf{B}$. Some very preliminary experiments, where $\mathbf{L_A}$ and $\mathbf{L_B}$ were block upper Hessenberg with all blocks $\mathbf{I}$, so $\alpha = 1$, find that indeed

$$\mathbf{L_C} = \begin{bmatrix} \mathbf{L_A} & & & -\mathbf{C}_0 \\ & -\mathbf{I} & 0 & \\ & & -\mathbf{I} & \\ & & & \mathbf{L_B} \end{bmatrix} \tag{5.26}$$

is a (strong) linearization for $\mathbf{C}(z)$, in the examples we tried. This extension to matrix polynomials will be interesting for applications, if the process is numerically stable (which it might be at least for some problems).

Bibliography

[1] David H. Bailey. A thread-safe arbitrary precision computation package (full documentation). http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf, 2016.

[2] David H. Bailey, Xiaoye S. Li, and Brandon Thompson. Arprec: An arbitrary precision computation package. http://crd.lbl.gov/~dhbailey/dhbpapers/arprec.pdf, 2002.

[3] Dario A. Bini and Giuseppe Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, 23(2-3):127–173, 2000.

[4] Dario A. Bini and Leonardo Robol. Solving secular and polynomial equations: A multiprecision algorithm. *Journal of Computational and Applied Mathematics*, 272:276–292, 2014.

[5] G. Birkhoff and G. C. Rota. *Ordinary Differential Equations*. John Wiley & Sons, New York, 1978.

[6] John P. Boyd. A Fourier companion matrix (multiplication matrix) with real-valued elements: Finding the roots of a trigonometric polynomial by matrix eigensolving. *Numerical Mathematics: Theory, Methods and Applications*, 6(04):586–599, 2013.

[7] David Carlson, Charles R. Johnson, David Lay, and A. Duane Porter. Gems of exposition in elementary linear algebra. *The College Mathematics Journal*, 23(4):299–303, 1992.

[8] Robert M. Corless and Nicolas Fillion. *A graduate introduction to numerical methods*. Springer Science & Business Media, 2014.

[9] Robert M. Corless and Piers W. Lawerence. Mandelbrot polynomials and matrices. *In preparation*.

[10] Robert M. Corless and Piers W. Lawrence. The largest roots of the Mandelbrot polynomials. In *Computational and Analytical Mathematics*, pages 305–324. Springer, 2013.

[11] Robert M. Corless and Steven E. Thornton. The Bohemian eigenvalue project. Poster presentation at ISSAC, will be appearing in CCA, 2016.

[12] Erwin Fehlberg. Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19690021375.pdf, 1969.

[13] Miroslav Fiedler. A note on companion matrices. *Linear Algebra and its Applications*, 372:325–331, 2003.

[14] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. Siam, 2002.

[15] Leslie Hogben and Robert Reams. Partitioned matrices. In *Handbook of Linear Algebra*, pages 10–1 – 10–10. Chapman and Hall/CRC, 2006.

[16] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[17] T. Y. Li, Tim Sauer, and J. A. Yorke. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. *SIAM Journal on Numerical Analysis*, 26(5):1241–1251, 1989.

[18] George Markowsky. Misconceptions about the golden ratio. *The College Mathematics Journal*, 23(1):2–19, 1992.

[19] Heinz-Otto Peitgen and Peter H. Richter. *The beauty of fractals: images of complex dynamical systems*. Springer Science & Business Media, 2013.

[20] Lawrence F. Shampine and Mark W. Reichelt. The MATLAB ODE suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[21] Neil J. A. Sloane. My favorite integer sequences. In *Sequences and their Applications*, pages 103–130. Springer, 1999.

[22] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. published electronically at `https://oeis.org`, 2016. Sequence A000045.

[23] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. published electronically at `https://oeis.org`, 2016. Sequence A000930.

[24] Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

Curriculum Vitae

Name:	Eunice Chan
Post-Secondary Education and Degrees:	Western University London, Ontario 2015 - Present M.Sc. Applied Mathematics 2011-2015 B.Sc. Applied Mathematics
Honours and Awards:	Winner of Best Poster Prize Computational Discovery Conference 2016 London, Ontario Winner of "Distinguished Poster Award" ISSAC 2016 Waterloo, Ontario Western Graduate Research Scholarship (WGRS) 2015-present
Related Work Experience:	Research Assistant Western University (Applied Mathematics) 2015 - present Research Assistant Western University (MEDICI) 2016 - present Teaching Assistant Western University 2015 - present

List of poster presentations:

1. E. Y. S. Chan, R. M. Corless, "Fibonacci-Mandelbrot polynomials and matrices" (Poster), Computation Discovery Conference, (2016).

2. E. Y. S. Chan, R. M. Corless, “Fibonacci-Mandelbrot polynomials and matrices” (Poster), ISSAC, (2016).

List of talks [1]:

1. E. Y. S. Chan, R. M. Corless, “Fibonacci-Mandelbrot polynomials and matrices”, SONAD, (2016).

2. E. Y. S. Chan, R. M. Corless, “Narayana, Mandelbrot, and a new kind of companion matrix”, MICA, (2016).

[1] The underlined name was the speaker.