Exploring the distribution of roots of polynomials with bounded integer coefficients

David Grilec

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2019

Abstract

This is an example of `infthesis` style. The file `skeleton.tex` generates this document and can be used to get a "skeleton" for your thesis. The abstract should summarise your report and fit in the space on the first page. You may, of course, use any other software to write your report, as long as you follow the same style. That means: producing a title page as given here, and including a table of contents and bibliography.

Acknowledgements

Acknowledgements go here.

Table of Contents

Chapter 1

Introduction

My supervisor has analysed the breadth of the continued fractions root finding method [4]. §2.1 c Under the root distribution assumption §2.1 my mentor has given an upper bound on the breadth of the tree obtained by performing the transformations used by continued fraction method. The main aim of this project was to implement a program to test an alternative root distribution assumption §2.1, under which we could provide a stronger upper bound on the breadth of the tree.

The program consist of two main stages. In the first stage sample polynomials are created. For this I have used Axiom - computer algebra system [1], because it is both fast and reliable with many convenient algebraic functions already defined and free to use. Sample polynomials were created with a specific degree and coefficients chosen uniformly at random from the range $[-B,B]$ for some $B \geq 1$. For the purpose of this project the polynomials had to be square free, to guarantee that all the roots are unique and primitive to avoid having polynomials with the identical roots. We also provide experimental ratios of square free versus non-square free poly and primitive versus non-primitive §3.1.

In the second stage we use two numerical methods for finding roots, QR method §2.2, which was implemented for C in GSL - GNU Scientific Library[2] and JenkinsTraub algorithm commonly known as RPOLY §2.3 which was implement in C++ by CRBond[6]. Both methods were used in the main program, which was written in C/C++. The choice for C/C++ derived naturally when considering the importance of speed of calculation for big samples and higher degree polynomials.

If both methods agree on all the roots of a polynomial within a specified accuracy we keep the roots, if not, the polynomial is discarded. This way we avoid the numerical instability of these methods, while at the same time we are able to keep the computing speed that comes with numerical methods. The methods agreed with over 95% on all polynomials §4.2, so there is no risk of biasing the sample.

Once the roots were obtained, we were able to test the alternative assumption. The assumption has been tested for polynomials of different degrees and different bounds on the coefficients. The results of the experiments support the alternative root distribution assumption.

After providing evidence in support of the alternative root distribution assumption, I have upgraded my program to produce a heat map matrix of the roots. To display colourful images of the roots I added a third stage, an R script [5], because of its predefined functions, which allow for quick and easy plotting of the heat maps of roots, providing a great insight on how roots are distributed on the coordinate system.

While comparing the heat maps we have noticed an unexpected and yet unexplained trend, lack of roots around the points -1 and 1. This suggests that further investigation is needed.

Chapter 2

Background

2.1 Previous work

This sections is extensively based on my supervisors's work [4].

Throughout we consider non-zero polynomials in z of degree n with coefficients a_i, where $|a_i| \leq B$ for some bound $B \in \mathbb{Z}$.

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0$$

The continued fraction method for finding the roots of a square free f is based on the transformations

$$f^T = f(z+1) = a_n(z+1)^n + a_{n-1}(z+1)^{n-1} + ... + a_1(z+1) + a_0$$

and

$$f^I = f\left(\frac{1}{z+1}\right)(z+1)^n = a_n + a_{n-1}(z+1) + ... + a_1(z+1)^{n-1} + a_0(z+1)^n$$

If $f^T(0) = 0$ then we replace $f^T(z)$ by $f^T(z)/z$ and similarly for $f^I(z)$. This way we make sure that 0 is not a root of f^T nor of f^I. For this reason we will assume that $a_0 \neq 0$.

"Vincent's theorem [7] shows that if f is square free then for all sufficiently long sequences of transformations involving T and I we produce a polynomial with variation of coefficients 0 and 1. This is false if the polynomial is not square free, for example if $f = (2z^2 - 1)^2$ then $f^{IIT} = f^I = z^4 + 4z^3 + 2z^2 - 4z + 1$."

For a polynomial f we say that f has a binary tree denoted by $tree(f)$, of the possible sequences of transformations.

If f has a sequence of coefficients $a_n, a_{n-1}, ..., a_0$ where the sign changes less than twice, then we say that f is terminal, the sequence of transformations stops at the branch and tree is empty, otherwise the root is labelled with f.

At any vertex v labelled with the polynomial g, if g^I is not terminal then there is a left child with the edge labelled I and the vertex labelled with g^I. Similarly if g^T is not terminal there is a right child with the edge labelled T and the vertex labelled with g^T.

We define the depth of a vertex to be the number of edges from the root to the vertex. Thus the root has depth 0 and its children, if any, have depth 1. The breath at depth d of the tree is the number of vertices of depth d. The breadth of a tree is 0 if it is empty otherwise it is the maximum breadth over all depths (if this is unbounded we take the breadth to be ∞, this does not happen in our case). We denote the breadth of $tree(f)$ by $br(f)$.

Further we define two open discs parametrized by m. S_m is the disc given by

$$\left| z - \frac{2m^2+8m+7}{2(m+2)(m+3)} \right| < \frac{1}{2(m+2)(m+3)}$$

L_m is the disc given by

$$|z| < \frac{m+2}{m+3}$$

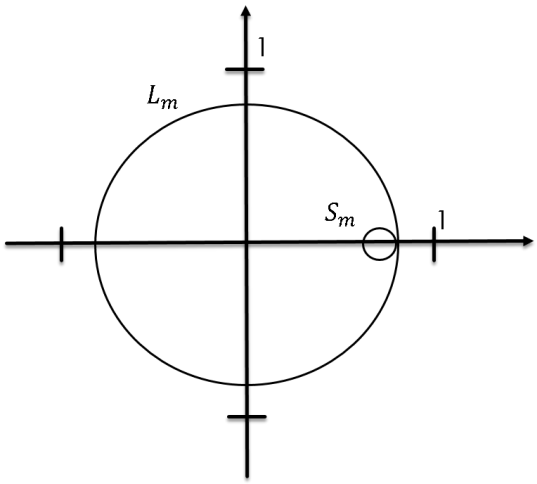

Figure 2.1: Figure shows circles L_1 and S_1, with slightly adjusted sized for visual purposes.

Note that L_m is the smallest disc centred at the origin that encloses and touches S_m on the real line.

Root Distribution Assumption: Let N_m be the expected number of roots in L_m of a square free primitive polynomial selected uniformly at random, more on selection in §2.2. There is a constant m_0 such that for all $m \geq m_0$ the expected number of roots in S_m is at most $N_m(s_m/l_m)^2$ where s_m is the radius of S_m and l_m is the radius of L_m.

Under this assumption Kalorkoti[4] has proven that the upper bound on the breadth of the $tree(f)$ for a square free primitive polynomial f is

$$\mathbb{E}[br(f)] < (4.44 + m_0)\log(n+1)B + 1.45\log B + 5$$

Alternative Assumption: The experimental evidence from [4] suggest that for all $m \geq 4$ we can provide a stricter upper bound estimate $N_m(s_m/l_m)^{1+\varepsilon}$ for some $\varepsilon > 0$. Under this assumption it Kalorkoti[4] has proven, that the upper bound on the breadth of the $tree(f)$ for a square free primitive polynomial f is

$$\mathbb{E}[br(f)] < \left(4.59 + \frac{1}{\varepsilon \log 2}\right)\log(n+1)B + 1.45\log B + 5$$

The aim of the project is to build a software and provide experimental support to this assumption, or to disprove it.

2.2 Stage 1: Polynomial generation

The first stage of the pipeline for the program is an Axiom program for generation of square free primitive polynomials. Given the degree of polynomial, the bound for the coefficients and desired sample size, the program generates sample polynomials.

For each polynomial to be constructed, coefficients are chosen uniformly at random in the ranges, $[1,B]$ for leading coefficients, $[-B,-1]\cup[1,B]$ for the trailing coefficient and $[-B,B]$ for the other coefficients, with the help of Axiom's *random* function, which uses random seeds and gives a good randomization.

Once the polynomial p had been constructed we test if p is square free and primitive. The ratio's of polynomials being square free, primitive and neither can be seen in §4.4. If p is not square free we discard it. If p is square free, we first divide p by a constant to make it square free, call this polynomial q.

Let the number of different polynomials we could obtain by multiplying q by a constant, while still no having coefficient absolutely bigger than the bound B, be w. We accept q with the probability of $1/w$. Note than on average we will still have the same number of square free primitive polynomial as we would otherwise, this way however, we neutralize any possible fluctuations from polynomials that have been generated. However, as we found §4.4, this procedure quickly became irrelevant with higher bounds and higher degree, as vast majority of polynomials were primitive and such that by multiplying them with a constant, the resulting polynomial was no longer within the bounds.

The resulting polynomials were then written to a *.txt* file to be used in the second stage.

2.3 QR method

QR method requires polynomial p to have real coefficients and to be monic. Note that our polynomials are not monic, however dividing them by the leading coefficients will produce monic polynomials with real coefficients.

From the monic polynomial $p(z) = a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0$ we construct a companion matrix

$$C_0 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{pmatrix}$$

Next step is to perform QR factorization $C = QR$, where Q is a matrix with orthogonal columns and R is an invertible upper triangular matrix with positive entries on its diagonal.

We repetitively perform these two operations

$$C_0 = Q_0 R_0$$
$$R_0 Q_0 = C_1$$

The matrix C_n is converging to an upper triangular matrix and diagonal entries are converting towards roots of polynomial p. We stop when the entries in C_n below the main diagonal are sufficiently small.

2.4 RPOLY algorithm

The JenkinsTraub algorithm or RPOLY for short is an algorithm that finds a linear or quadratic factor of a polynomial working exclusively in real arithmetic. The polynomial is then deflated by a corresponding linear or quadratic factor and the process is repeated.

The algorithm is quite long and it involves 3 stages, that can be restarted if the conversion is not fast enough. If you are truly interested in details how algorithm work, I highly suggest to have a look at the paper by Jenkins and Traub [3].

2.5 Stage 2: Assumption Testing

In stage 2 the program reads a file of generated polynomials. Each polynomial is then fed to both QR and RPOLY method. The roots obtained are then sorted and compared.

If for all the roots of a given polynomial the methods agree within the precision of 10^{-12} then we kept the roots, otherwise not. After that, the program checks if the roots fall in or on the circles S_m and L_m as characterized in §2.1 for different values of parameter m.

The results of this experiment are recorded in §4.4.

Aside from the counting the number of roots that can be found within S_m and L_m, the program also constructs a big matrix with a arbitrary size to represent the coordinate system. The entries of this matrix are then printed to a *.txt* file and turned into pixels in the next stage.

2.6 Stage 3: image generation

Once we have catched the distribution of zeros in a *.txt* file, R program reads it and displays the pictures in a *.jpeg* file for easier interpretation. The darker the color, the more roots have been found in the pixel. The pictures can be seen in §4.4.

When observing the pictures the reader should bear in mind that the colouring scale is not linear. Linear colouring scale produced almost entirely white picture with the exception of real line, which had varying hues of purple. The drastic difference between real line and the rest of the picture is because on the real line there are too many roots, in fact so many that make that the number of non-real roots is insignificant. The images still carried some information related to the distribution of roots along the real line.

However, to achieve better insights into the distribution of roots on complex plane, we have to minimize the effect of extreme value on the hue scale. In other words, we have to pull up the color in pixels with less roots.

Graphing function in R takes the minimum and maximum value of the input and normalizes the whole input to the range $[0,1]$, in order to allocate hue colouring based on the value of each entry.

So to pull up the hue color of less dense pixels, I had to increase their values relative to the pixels with higher density. log function is perfect for this. After testing several different scales, I have settled down for $\log_{100}$. This means that every entry of the matrix had a $\log_{100}$ applied to it, before proceeding to the graphing function.

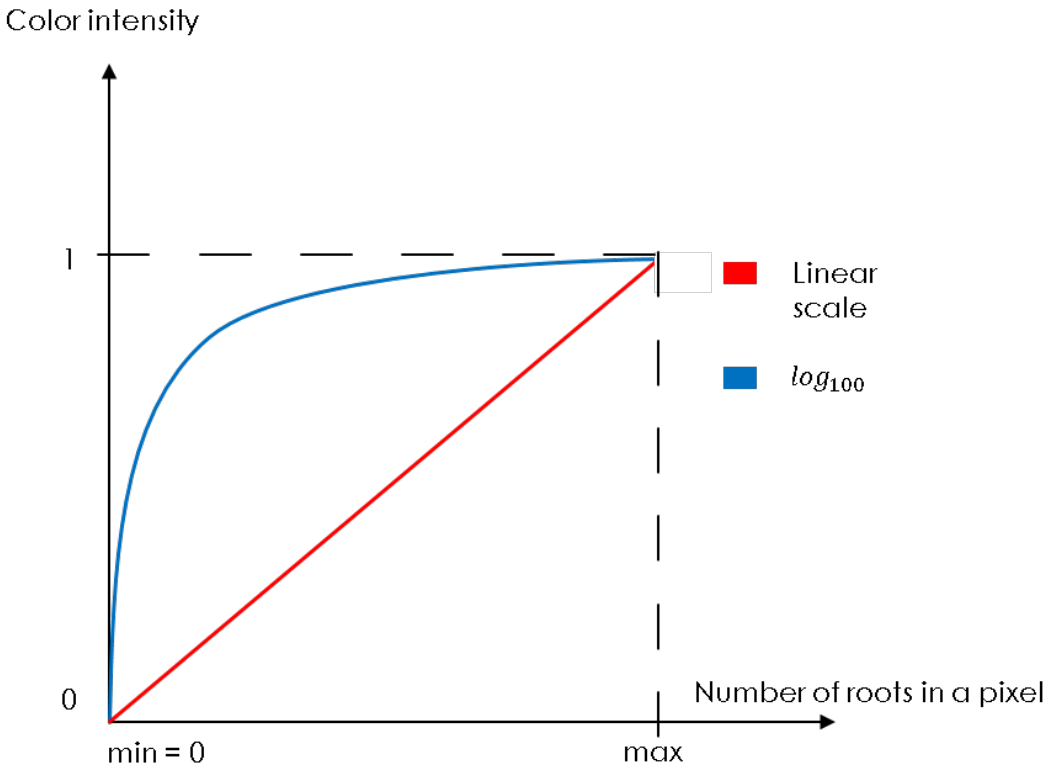

Figure 2.2: This figure illustrates the effect of applying $\log_{100}$ on the hue scale.

Chapter 3

Results

Chapter 4

Extra

THEOREM: Given a set A, of all the polynomials of degree n and coefficients bounded by B, let $p \in A$. Then z_0 is a root of $p(z_0)$ if and only if z_0^{-1} is a root of some polynomial $r(z_0^{-1}) \in A$.

PROOF:

Let

$$r(z) = a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0$$

then

$$\begin{aligned} r(z_0^{-1}) &= a_n z_0^{-n} + a_{n-1} z_0^{-n+1} + ... + a_1 z_0^{-1} + a_0 \quad / \cdot z^n \\ &= a_0 z_0^n + a_1 z_0^{n-1} + ... + a_{n-1} z_0 + a_n \\ &= p(z_0) \end{aligned}$$

Now we turn our attention to the effect of the bound B on the annulus that contains all the roots of polynomials of degree n with coefficients bounded by B.

THEOREM: The set A of all the roots of all polynomial of degree n with coefficients bounded by B. Then A is contained in the annulus $1/B+1 < |z| < B+1$.

PROOF:

$$
\begin{aligned}
p(z) &= a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0 \\
-a_0 &= a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z \\
1 &= \frac{-1}{a_0}(a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z) \\
1 &< |a_n| z^n + |a_{n-1}| z^{n-1} + \ldots + |a_1| z \\
1 &< B|z|^n + B|z|^{n-1} + \ldots + B|z| \\
1 &< B(|z|^n + |z|^{n-1} + \ldots + |z|) \\
\frac{1}{B} &< \frac{|z|}{1-|z|} \\
|z| &> \frac{1}{B+1}
\end{aligned}
$$

Since z is a root of $p \in A$ if and only if z^{-1} is a root of some polynomial $r \in A$, it follows that $|z| < B+1$. Hence, $1/B+1 < |z| < B+1$.

Bibliography

[1] AXIOM-TEST.

[2] GSL-TEST.

[3] M. A. Jenkins and J. F. Traub. *A three-stage algorithm for real polynomials using quadratic iteration*. 12 1970.

[4] Kyriakos Kalorkoti. Tree breadth of the continued fractions root finding method.

[5] R Language-TEST.

[6] RPOLY-TEST.

[7] A. J. H. Vincent. *Sur la resolution des equations numeriques, Journal de mathematiques pures et appliquees*. 1 1836.