UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

Department of Computer and Communications Engineering

UPPER BOUNDS ON THE VALUES OF THE POSITIVE ROOTS OF POLYNOMIALS

by

Panagiotis S. Vigklas

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

University of Thessaly
Volos, Greece
2010

Supervisor	Dr Alkiviadis Akritas, Associate Professor, University of Thessaly
Co-supervisors	Dr Elias Houstis, Professor, University of Thessaly
	Dr Michalis Hatzopoulos, Professor, University of Athens
Doctoral Committee	Dr Elias Houstis, Professor, University of Thessaly
	Dr Michalis Hatzopoulos, Professor, University of Athens
	Dr Panagiotis Sakkalis, Professor, Agricultural University of Athens
	Dr Evangelos Fountas, Professor, University of Piraeus
	Dr Alkiviadis Akritas, Associate Professor, University of Thessaly
	Dr Maria Gousidou-Koutita, Associate Professor, Aristotle Univ. of Thessaloniki
	Dr Panagiota Tsompanopoulou, Assistant Professor, University of Thessaly

To my parents

ACKNOWLEDGEMENTS

Heartfelt thanks go to my scientific adviser, Prof. Alkiviadis Akritas for making me familiar with computer algebra systems and root isolation methods, for his patient support and his indefatigable interest in my thesis and for his friendship.

I am deeply indebted to Adam Strzeboński and Prof. Doru Ştefănescu for their significant contributions towards the completion of this research.

I am grateful to Prof. Elias Houstis and Prof. Michael Hatzopoulos for serving on my thesis committee.

Last but not least I want to thank my parents who encourage me to complete this study.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

This thesis describes new results on computing bounds on the values of the positive roots of polynomials. Bounding the roots of polynomials is an important sub-problem in many disciplines of scientific computing.

Many numerical methods for finding roots of polynomials begin with an estimate of an upper bound on the values of the positive roots. If one can obtain a more accurate estimate of the bound, one can reduce the amount of work used in searching within the range of possible values to find the root (e.g. using a bisection method).

Also, the computation of the real roots of higher degree univariate polynomials with real coefficients is based on their isolation. Isolation of the real roots of a polynomial is the process of finding real disjoint intervals such that each contains one real root and every real root is contained in some interval. To isolate the real positive roots, it is necessary to compute, in the best possible way, an upper bound on the value of the largest positive root. Although, several bounds are known, the first of which were obtained by Lagrange and Cauchy, this thesis revealed that there was much room for improvement on this topic. Today, two of the algorithms presented in this thesis, are regarded as the best (one of linear computational complexity and the other of quadratic complexity) and have already been incorporated in the source code of major computer algebra systems such as `Mathematica` and `Sage`.

A certain part of this thesis is also devoted to the analytical presentation of the continued fraction real root isolation method. Its algorithm and its underlying components are presented thoroughly along with a new implementation of the method using the above mentioned bounds. Intensive computational tests verify that this

implementation makes the continued fraction real root isolation method the fastest among its rivals.

After almost thirty years of usage and development, the continued fractions real root isolation algorithm, introduced back in 1976 by A. Akritas, continues today to efficiently tackle a basic but still important mathematical problem, the solution of a polynomial equation. The revived interest in this algorithm is motivated by the need to solve, in real time, polynomial equations of higher degrees in such diverse scientific fields as control theory, financial theory, signal processing, robotics, computer vision, computer-aided-design, geometric modeling, industrial problems, to name a few. The usage of the continued fraction real root isolation algorithm from major commercial and open source mathematical solvers proves its robustness. This thesis has contributed towards this direction.

ABSTRACT (in greek)

Αυτή η διατριβή παρουσιάζει νέα αποτελέσματα σε ότι αφορά τον υπολογισμό των άνω ορίων των τιμών των θετικών ριζών των πολυωνυμικών εξισώσεων. Ο υπολογισμός αυτών των άνω ορίων αποτελεί ένα σημαντικό πρόβλημα σε πολλά διαφορετικά πεδία των επιστημονικών υπολογισμών και εφαρμογών.

Υπάρχουν σήμερα πολλές αριθμητικές μέθοδοι για την εύρεση των ριζών των πολυωνυμικών εξισώσεων που ξεκινούν με μια εκτίμηση του άνω ορίου των τιμών των θετικών ριζών. Αν κάποιος μπορούσε να υπολογίσει με μεγαλύτερη ακρίβεια αυτό το άνω όριο, θα μείωνε δραστικά των αριθμό των υπολογισμών που θα χρειαζόταν για την αναζήτηση της ρίζας του πολυωνύμου μέσα σε ένα συγκεκριμένο εύρος τιμών, (π.χ. κάνοντας χρήση μιας μεθόδου διχοτόμησης).

Επίσης, ο υπολογισμός των πραγματικών ριζών πολυωνυμικών εξισώσεων μιας μεταβλητής μεγάλου βαθμού με πραγματικούς συντελεστές βασίζεται στη μέθοδο απομόνωσής τους. Η απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων αφορά στην εύρεση πραγματικών μη συνεχόμενων διαστημάτων τέτοιων ώστε καθένα από αυτά να περιέχει μια ρίζα και κάθε πραγματική ρίζα να περιέχεται σε κάποιο από αυτά. Για να απομονώσουμε τις πραγματικές θετικές ρίζες, είναι απαραίτητο καταρχάς, να υπολογίσουμε, με τον καλύτερο δυνατό τρόπο, ένα άνω όριο στη τιμή της μεγαλύτερης θετικής ρίζας. Αν και υπάρχουν αρκετές τέτοιες μέθοδοι υπολογισμού, (μερικές από τις οποίες είχαν προταθεί αρχικά από το Lagrange και τον Cauchy), αυτή η διατριβή αποδεικνύει ότι υπάρχουν αρκετά περιθώρια βελτίωσης αυτών των μεθόδων. Σήμερα, δυο από τις αλγοριθμικές μεθόδους που παρουσιάζονται σε αυτή τη διατριβή, θεωρούνται οι

καλύτερες που υπάρχουν (η μια με γραμμική και η άλλη με τετραγωνική υπολογιστική πολυπλοκότητα) και έχουν ήδη ενσωματωθεί στον πηγαίο κώδικα πολύ γνωστών συστημάτων λογισμικού επιστημονικών υπολογισμών όπως π.χ. το ***Mathematica***, ***Sage, Mathemagix,*** κ.α.

Ένα μέρος της παρούσας διατριβής περιλαμβάνει επίσης και την αναλυτική παρουσίαση της μεθόδου απομόνωσης πραγματικών ριζών με συνεχή κλάσματα. Ο αλγόριθμος της μεθόδου περιγράφεται διεξοδικά μαζί με μια νέα υλοποίησή του που ενσωματώνει τις παραπάνω νέες μεθόδους υπολογισμού των ορίων. Εξαντλητικές υπολογιστικές δοκιμές επιβεβαιώνουν ότι η νέα αυτή υλοποίηση του αλγορίθμου κάνει τη μέθοδο απομόνωσης πραγματικών ριζών με συνεχή κλάσματα την ταχύτερη ανάμεσα σε άλλες.

Μετά από τριάντα σχεδόν χρόνια εφαρμογής και ανάπτυξης, η μέθοδος απομόνωσης πραγματικών ριζών με συνεχή κλάσματα, που προτάθηκε το 1976 από τον Α. Ακρίτα, συνεχίζει και σήμερα να αντιμετωπίζει αποτελεσματικά ένα βασικό αλλά ωστόσο πολύ σημαντικό μαθηματικό πρόβλημα, αυτό της επίλυσης της πολυωνυμικής εξίσωσης. Το έντονο ενδιαφέρον που έδειξε η ερευνητική κοινότητα τελευταία για τη μέθοδο αυτή, πηγάζει από την ανάγκη ύπαρξης μιας αξιόπιστης και αποδοτικής μεθόδου για τη λύση, σε πραγματικό χρόνο, πολυωνυμικών εξισώσεων μεγάλου βαθμού σε ποικίλα επιστημονικά πεδία όπως η θεωρία ελέγχου, οικονομική θεωρία, επεξεργασία σήματος, ρομποτική, υπολογιστική όραση, γραφικά υπολογιστών, υπολογιστική γεωμετρία, βιομηχανικά προβλήματα, κλπ. Η υιοθέτηση της μεθόδου απομόνωσης πραγματικών ριζών με συνεχή κλάσματα από μεγάλα εμπορικά και ανοικτού κώδικα μαθηματικά πακέτα λογισμικού αποδεικνύει τη δύναμή της και τις δυνατότητές της. Δύναμη και δυνατότητες που οφείλονται εν μέρει και στα αποτελέσματα αυτής της διατριβής.

CHAPTER I

Introduction

1.1 Historical Note

One of the oldest and maybe for centuries the only area of study in Algebra had been polynomial equations. The problem was to find formulas that could give the roots of polynomials in terms of their coefficients.

It has been found, from historical searches, that the ancient Babylonians, who created their civilization in 2000 B.C. in Mesopotamia, knew how to find the roots of 1st and 2nd degree polynomials. Also they could approximate the square roots of numbers. They formulated the problems and their solutions mostly verbally.

The next big step was done by the ancient Greeks. A group of mathematicians called Pythagoreans (5th century B.C.), proved that the square roots that appeared in the study of 2nd degree equations resulted in irrational numbers.

The ancient Greeks were using geometrical designs for solving polynomial equations of the 1st, 2nd and 3rd degree. That is geometrical designs made with a ruler and a pair of compasses. Traces of algebraic representation for solving 2nd degree equations did not exist until 100 B.C. The mathematician Diofante in 250 B.C. introduced a form of algebraic symbolism. The arithmetic of Diofante is for algebra of the same importance as the elements of Euclid for geometry. The Arabians improved algebraic calculus but did not manage to solve 3rd degree equations.

In the Middle Age, European mathematicians improved the things they learned from the Arabs, the most famous of them being Al-Khwarismi and introduced new symbols. During the Renaissance, the development of algebra was remarkable, like all other branches of mathematics.

Approximately at the end of the 15th century the University of Bologna in Italy, was one of the most famous in Europe. This fame was related with the attempt of the Bolognese mathematicians to solve 3rd and 4th degree equations.

It seems that Professor Scipio del Ferro, who died in 1526 managed to solve the equation of the 3rd degree, without ever publishing his work. Niccolo Fontana known as Tartaglia found again the solution of the 3rd degree equation. This particular project of Fontana was published in 1545 from a polymath doctor in Milan, Hieronimo Cardano in his work *Ars Magna* (The Great Art). *Ars Magna* also includes a method for solving polynomial equations of degree four, by reducing them to equations of degree three.

Of course, after that discovery, the effort was concentrated in finding formulas which would give the roots of equations of degree 5 or greater than 5.

In the 18th century Josheph Louis Lagrange, influenced drastically the theory of equations and approximately three years later C.F. Gauss (1777-1855) based on Lagrange's conclusions proved The Fundamental Theorem of Algebra.

The proof of the fact that there is not a formula to compute the roots of equations of degree 5 was given by Paolo Ruffini (1804), who preceded Horner by about 15 years. The Norwegian mathematician, N.H. Abel (1802-1829), in 1824, generalized Ruffini's work by showing the impossibility of solving the general quintic equation by means of radicals, thus finally put to rest a difficult problem that had puzzled mathematicians for many years. Of course there was still the problem of finding the conditions that such an equation must satisfy in order to be solved. Abel was working on this problem until his death in 1829.

Eventually this problem was solved by the young French mathematician Evariste Galois (1811-1832). His theory virtually contains the solution of this problem. Galois wrote his conclusions in an illegible manuscript 31 pages long, the night before he died at the age of 20. This manuscript became well known when Joseph Liouville presented it in the French Academy in 1843.

Since then (and for some time before in fact), researchers have concentrated on numerical (iterative) methods such as the famous Newton's method of the 17th century, Bernoulli's method of the 18th, and Graeffe's method of the early 19th. During the same period, Fourier conceived the idea to split the problem, of the higher degree equation solving, in two subproblems; that is, fist to isolate the real roots, and then to approximate them to any desired degree of accuracy. The major problem was isolation, which attracted immediately the attention of the mathematicians. To isolate the roots two theorems were initially proposed: Budan's (1807) and Fourier's (1820) theorems on which Vincent's (1836) and Sturm's (1829) theorems were based later on. Vincent's (1836) theorem, was, in turn, the foundation of the Akritas' continued fractions method of 1978, a method that is considered the most efficient today[1].

[1]For Descartes', Budan's, Fourier's, Vincent's and Sturm's theorems, see the Appendix. For details on Vincent's theorem, see Chapter IV.

CHAPTER II

Bounds

2.1 Definitions

2.1.1 Univariate Polynomials

A polynomial is a mathematical expression of the form

$$p(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + ... + \alpha_{n-1} x + \alpha_n, \quad (\alpha_0 > 0) \tag{2.1}$$

If the highest power of x is x^n, the polynomial is said to be of degree n. It was proved by Gauss in the early 19th century that every polynomial of positive degree has at least one zero (i.e. a value z which makes $p(z)$ equal to zero), and that a polynomial of degree n has n zeros (not necessarily distinct). Often we use x for a real variable, and z for a complex one. A zero of a polynomial is synonymous to the "root" of the equation $p(x) = 0$. A zero may be real or complex, and if the "coefficients" α_i are all real, then complex zeros occur in conjugate pairs $\alpha + i\beta$, $\alpha - i\beta$. The purpose of the first part of this study is to describe methods which have been developed to find bounds for the real positive roots of polynomials with real coefficients.

2.1.2 Bounds on the Values of the Roots of Polynomials

In attempting to find the roots of a polynomial equation it is advantageous to narrow the region within which they must be sought. So, our aim is to establish sharp bounds, for the positive and negative roots $x_1, x_2, ..., x_m$, $1 \leq m \leq n$, of the equation $p(x) = 0$. It is sufficient to restrict ourselves to finding the upper bound, ub, of only the positive roots of polynomials of type (2.1). Here is why:
Consider along with (2.1) the transformed equations

$$p_1(x) \equiv x^n p(\frac{1}{x}) = 0 \tag{2.2a}$$

$$p_2(x) \equiv x^n p(-x) = 0 \tag{2.2b}$$

$$p_3(x) \equiv x^n p(-\frac{1}{x}) = 0 \tag{2.2c}$$

and let the upper bounds of their positive roots be ub_1, ub_2 and ub_3 respectively. Then the number $\frac{1}{ub_1}$ is clearly a *lower* bound on the values of the positive roots of equation (2.1), that is, all positive roots x^+ of this equation, if they exist, satisfy the inequality

$$\frac{1}{ub_1} \leq x^+ \leq ub \tag{2.3}$$

Similarly, the numbers $-ub_2$ and $-\frac{1}{ub_3}$ are, respectively, *lower* and *upper* bounds of the *negative* roots of (2.1), that is, all negative roots x^- of this equation, if they exist, satisfy the inequality

$$-ub_2 \leq x^- \leq -\frac{1}{ub_3} \tag{2.4}$$

It should be emphasized here that bounds on the values of just the *positive* roots of polynomials are scarce in the literature. Especially, in the English literature, only

bounds on the absolute values (positive and negative) of the roots existed until 1978. As Akritas points out, he was able to find Cauchy's bound (described below) on the values of the positive roots in Obrechkoff's book, (*Obreschkoff*, 1963). Bounds on the values of the positive roots of polynomials are important, because it is only those bounds that can be used in the root isolation process described in Chapter IV.

2.2 Classical Methods for Computing Bounds

In this section we first present the two classical theorems by Cauchy and Lagrange-MacLaurin. Until recently, the first was the only method used for computing the bounds, on the values of the positive roots of polynomials. In addition, we include, Kioustelidis' bound, (*Kioustelidis*, 1986), which is closely related to the one by Cauchy.

2.2.1 Cauchy's Method

Theorem II.1. *Let $p(x)$ be a polynomial as in* (2.1)*, of degree $n > 0$, with $\alpha_{n-k} < 0$ for at least one k, $1 \leq k \leq n$. If λ is the number of negative coefficients, then an upper bound on the values of the positive roots of $p(x)$ is given by*

$$ub = \max_{\{1 \leq k \leq n : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{\lambda \alpha_k}{\alpha_0}}$$

Note that if $\lambda = 0$ there are no *positive roots.*

Proof. From the definition above we have

$$ub^k \geq \left(-\frac{\lambda \alpha_{n-k}}{\alpha_0} \right)$$

for every k such that $\alpha_{n-k} < 0$. For these k, the inequality above could be written

$$ub^n \geq \left(-\frac{\lambda\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

Summing for all k's we have

$$\lambda ub^n \geq \lambda \sum_{1\leq k\leq n:\alpha_{n-k}<0} \left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

or

$$ub^n \geq \sum_{1\leq k\leq n:\alpha_{n-k}<0} \left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

i.e., dividing $p(x) = 0$ by α_0, making unitary the leading coefficient, and replacing x with ub, $x \leftarrow ub$, the first term, i.e. ub^n, would be *greater than*, or *equal to*, the sum of the absolute values of the terms with negative coefficient. Hence, for all $x > ub$, $p(x) > 0$. □

Even though the proof is sound, and easy to follow, it gives us no insight on what is going on. Hence, we cannot improve on it. The same holds for the following theorem.

2.2.2 The Lagrange–MacLaurin Method

Theorem II.2. *Suppose α_{n-k}, $k \geq 1$, is the first of the negative coefficients*[1] *of a polynomial $p(x)$, as in* (2.1)*. Then an upper bound on the values of the positive roots of $p(x)$ is given by*

$$ub = 1 + \sqrt[k]{\frac{B}{\alpha_0}},$$

where B is the largest absolute value of the negative coefficients of the polynomial $p(x)$.

[1] If there is no negative coefficient then $p(x)$ has no positive roots.

Proof. Set $x > 1$. If in $p(x)$ each of the nonnegative coefficients $\alpha_1, \alpha_2, \ldots,$ α_{k-1} is replaced by zero, and each of the remaining coefficients $\alpha_k, \alpha_{k+1}, \ldots, \alpha_n$ is replaced by the negative number $-B$, we obtain

$$p(x) \geq \alpha_0 x^n - B(x^{n-k} + x^{n-k-1} + \ldots + 1) = \alpha_0 x^n - B\frac{x^{n-k+1} - 1}{x - 1}$$

Hence for $x > 1$ we have

$$\begin{aligned} p(x) > \alpha_0 x^n - \frac{B}{x-1} x^{n-k+1} &= \frac{x^{n-k+1}}{x-1}(\alpha_0 x^{k-1}(x-1) - B) \\ &> \frac{x^{n-k+1}}{x-1}(\alpha_0 (x-1)^k - B) \end{aligned}$$

Consequently for

$$x \geq 1 + \sqrt[k]{\frac{B}{\alpha_0}} = ub$$

we have $p(x) > 0$ and all the positive roots x^+ of $p(x)$ satisfy the inequality $x^+ < ub$. □

2.2.3 Kioustelidis' Method

Theorem II.3. *Let $p(x)$ be a polynomial as in (2.1), of degree $n > 0$, with $\alpha_{n-k} < 0$ for at least one k, $1 \leq k \leq n$. Then an upper bound on the values of the positive roots of $p(x)$ is given by*

$$ub = 2 \max_{\{1 \leq k \leq n : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{\alpha_{n-k}}{\alpha_0}}.$$

Proof. From the definition above we have

$$ub^k \geq \frac{1}{2^k}\left(-\frac{\alpha_{n-k}}{\alpha_0}\right)$$

for every k such that $\alpha_{n-k} < 0$. For these k, the inequality above could be written

$$ub^n \geq \frac{1}{2^k}\left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

Summing for all k's we have

$$ub^n \geq \sum_{1\leq k\leq n:\alpha_{n-k}<0} \frac{1}{2^k}\left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

or

$$ub^n \geq (1-\frac{1}{2^n}) \sum_{1\leq k\leq n:\alpha_{n-k}<0} \left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

and because $(1-2^{-n}) < 1$ we get

$$ub^n \geq \sum_{1\leq k\leq n:\alpha_{n-k}<0} \left(-\frac{\alpha_{n-k}}{\alpha_0}\right) ub^{n-k}$$

i.e., dividing $p(x) = 0$ by α_0, making unitary the leading coefficient, and replacing x with ub, $x \leftarrow ub$, the first term, i.e. ub^n, would be *greater than*, or *equal to*, the sum of the absolute values of the terms with negative coefficient. Hence, for all $x > ub$, $p(x) > 0$.

□

In the next chapter, we will present a theorem by Ştefănescu, (*Ştefănescu*, 2005), that gives some insight into the nature of how these bounds are computed. Extending Ştefănescu's theorem, (*Akritas and Vigklas*, 2006), (*Akritas, Strzeboński, and Vigklas*, 2006), we obtain a general theorem, which includes the above three methods as special cases, and from which new, sharper, bounds can be derived.

CHAPTER III

A General Theorem for Computing Bounds on the Positive Roots of Univariate Polynomials

3.1 Preliminaries

In the following discussion we shall consider polynomials with integer or rational coefficients of any (arbitrary) bit–length. The methods that will be presented here are methods of infinite precision (based on exact arithmetic) and must not be confused with numerical or other approximate methods where someone has to take under consideration various types of errors that infiltrate the computation process and progressively degrade the final results.

3.2 Ştefănescu's Theorem and its Extension

Despite the fact that in the literature one can find many formulas[1] that estimate an upper bound on the largest *absolute* value of the real or complex roots, (*Yap*, 2000), (*Mignotte*, 1992), the most recent addition for a method to compute bound on the *positive* roots of polynomials, that is of importance to us, has been by Ştefănescu. Namely, in (*Ştefănescu*, 2005), the following theorem is proved:

[1] A bibliographical search till 2005 gives over 50 articles or books which give such bounds.

Theorem III.1 (Ştefănescu's, 2005)**.** *Let $p(x) \in \mathbb{R}[x]$ be such that the number of variations of signs of its coefficients is* ***even****. If*

$$p(x) = c_1 x^{d_1} - b_1 x^{m_1} + c_2 x^{d_2} - b_2 x^{m_2} + \ldots + c_k x^{d_k} - b_k x^{m_k} + g(x), \tag{3.1}$$

with $g(x) \in \mathbb{R}_+[x]$, $c_i > 0, b_i > 0, d_i > m_i > d_{i+1}$ for all i, the number

$$B_3(p) = \max\left\{ \left(\frac{b_1}{c_1}\right)^{1/(d_1-m_1)}, \ldots, \left(\frac{b_k}{c_k}\right)^{1/(d_k-m_k)} \right\} \tag{3.2}$$

is an upper bound for the positive roots of the polynomial p for any ***choice*** *of $c_1, \ldots, c_k$.*

We point out that Ştefănescu's theorem introduces the concept of *matching* or *pairing* a positive coefficient with a negative coefficient of a lower order term. That is, to obtain an upper bound, we match each negative coefficient–in fact we match a negative term, with a positive one, but for short we mention coefficient–with a preceding positive one, and take the maximum. Clearly, Ştefănescu's theorem has limited use since it works only for polynomials with an *even* number of sign variations[2].

The following theorem generalizes Theorem III.1, in the sense that it applies to polynomials with any number of sign variations. To accomplish this, we introduce the new concept of *breaking-up* a positive coefficient into several parts to be paired with negative coefficients (of lower order terms)[3].

[2]In (*Tsigaridas and Emiris*, 2006), Tsigaridas and Emiris mention slightly different the same theorem "*Moreover, when the number of negative coefficients is even then a bound due to Ştefănescu can be used which is much better*". Unfortunately, still with this version of the theorem its weakness remains.

[3]After the publication of this work, (*Akritas, Strzeboński, and Vigklas*, 2006), Ştefănescu also extended Theorem III.1 in (*Ştefănescu*, 2007).

Theorem III.2. *Let*

$$p(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_0, \quad (\alpha_n > 0) \tag{3.3}$$

be a polynomial with real coefficients and let $d(p)$, $t(p)$ denote the degree and the number of its terms, respectively. Moreover, assume that $p(x)$ can be written as

$$p(x) = q_1(x) - q_2(x) + q_3(x) - q_4(x) + \ldots + q_{2m-1}(x) - q_{2m}(x) + g(x), \tag{3.4}$$

where all polynomials $q_i(x)$, $i = 1, 2, \ldots, 2m$ and $g(x)$ have only positive coefficients. In addition, assume that for $i = 1, 2, \ldots, m$ we have

$$q_{2i-1}(x) = c_{2i-1,1} x^{e_{2i-1,1}} + \ldots + c_{2i-1,t(q_{2i-1})} x^{e_{2i-1,t(q_{2i-1})}} \tag{3.5}$$

and

$$q_{2i}(x) = b_{2i,1} x^{e_{2i,1}} + \ldots + b_{2i,t(q_{2i})} x^{e_{2i,t(q_{2i})}}, \tag{3.6}$$

where $e_{2i-1,1} = d(q_{2i-1})$ and $e_{2i,1} = d(q_{2i})$ and the exponent of each term in $q_{2i-1}(x)$ is greater than the exponent of each term in $q_{2i}(x)$. If for all indices $i = 1, 2, \ldots, m$, we have

$$t(q_{2i-1}) \geq t(q_{2i}), \tag{3.7}$$

then an upper bound of the values of the positive roots of $p(x)$ is given by

$$ub = \max_{\{i=1,2,\ldots,m\}} \left\{ \left(\frac{b_{2i,1}}{c_{2i-1,1}} \right)^{\frac{1}{e_{2i-1,1} - e_{2i,1}}}, \ldots, \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}} \right)^{\frac{1}{e_{2i-1,t(q_{2i})} - e_{2i,t(q_{2i})}}} \right\}, \tag{3.8}$$

for any permutation of the positive coefficients $c_{2i-1,j}$, $j = 1, 2, \ldots, t(q_{2i-1})$. Other-

wise, for each of the indices i for which we have[4]

$$t(q_{2i-1}) < t(q_{2i}) \tag{3.9}$$

we ***break-up*** *one of the coefficients of* $q_{2i-1}(x)$ *into* $t(q_{2i}) - t(q_{2i-1}) + 1$ *parts, so that now* $t(q_{2i}) = t(q_{2i-1})$ *and apply the same formula (3.8) given above.*

Proof. Suppose $x > 0$. We have

$$\begin{aligned}
|p(x)| \;\geq\; & c_{1,1}x^{e_{1,1}} + \ldots + c_{1,t(q_1)}x^{e_{1,t(q_1)}} - b_{2,1}x^{e_{2,1}} - \ldots - b_{2,t(q_2)}x^{e_{2,t(q_2)}} \\
+ & \\
& \vdots \\
+ \;\; & c_{2m-1,1}x^{e_{2m-1,1}} + \ldots + c_{2m-1,t(q_{2m-1})}x^{e_{2m-1,t(q_{2m-1})}} \\
- \;\; & b_{2m,1}x^{e_{2m,1}} - \ldots - b_{2m,t(q_{2m})}x^{e_{2m,t(q_{2m})}} + g(x) \\
= \;\; & x^{e_{2,1}}(c_{1,1}x^{e_{1,1}-e_{2,1}} - b_{2,1}) + \ldots \\
+ \;\; & x^{e_{2m,t(q_{2m})}}(c_{2m-1,t(q_{2m})}x^{e_{2m-1,t(q_{2m})}-e_{2m,t(q_{2m})}} - b_{2m,t(q_{2m})}) + g(x)
\end{aligned}$$

which is strictly positive for

$$x > \max_{\{i=1,2,\ldots,m\}} \left\{ \left(\frac{b_{2i,1}}{c_{2i-1,1}}\right)^{\frac{1}{e_{2i-1,1}-e_{2i,1}}}, \ldots, \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}}\right)^{\frac{1}{e_{2i-1,t(q_{2i})}-e_{2i,t(q_{2i})}}} \right\}$$

□

Remark 1. Pairing positive with negative coefficients and breaking-up a positive coefficient into the required number of parts—to match the corresponding number of negative coefficients—are the key ideas of this theorem. In general, formulae analogous to (3.8) hold for the cases where: (a) we pair coefficients from the non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$, for $1 \leq l < i$, and (b) we break-up one or more

[4] A partial extension of Theorem III.1, presented in (*Akritas and Vigklas*, 2007), does not treat the case $t(q_{2i-1}) < t(q_{2i})$.

positive coefficients into several parts to be paired with the negative coefficients of lower order terms. In the following section we present several implementations of Theorem III.2.

3.3 Algorithmic Implementations of the Generalized Theorem

Theorem III.2 is stated in such a way, that it is amenable to several implementations; to wit, the positive-negative coefficient pairing is not unique and can be done in several ways[5].

Moreover, we have quite a latitude in choosing the positive coefficient to be broken up; and once that choice has been made, we can break it up in equal or unequal parts. We explore some of these choices below.

We begin with the most straightforward approach for implementing Theorem III.2, which is to first take care of all the cases where $t(q_{2i-1}) < t(q_{2i})$, and then, for all $i = 1, 2, \ldots, m$, to pair a positive coefficient of $q_{2i-1}(x)$ with a negative coefficient of $q_{2i}(x)$—starting with the coefficients $c_{2i-1,1}$ and $b_{2i,1}$ and moving to the right (in non-increasing order of exponents), until the negative coefficients have been exhausted.

Example 1. Consider the polynomial

$$p_1(x) = x^9 + 3x^8 + 2x^7 + x^6 - 4x^4 + x^3 - 4x^2 - 3$$

[5]An example of the worst possible pairing strategy is the rule by Lagrange and MacLaurin, (*Akritas and Vigklas*, 2006), that was mentioned in (§ 2.2.2)

for which we have

$$\begin{aligned} q_1(x) &= x^9 + 3x^8 + 2x^7 + x^6 \\ -q_2(x) &= -4x^4 \\ q_3(x) &= x^3 \\ -q_4(x) &= -4x^2 - 3. \end{aligned}$$

A direct application of Theorem III.2 pairs the terms $\{x^9, -4x^4\}$ of $q_1(x)$ and $q_2(x)$, and *ignores* the last three terms of $q_1(x)$. It then splits the coefficient of x^3 into two, say equal parts to account for the two negative terms of $q_4(x)$ and forms the pairs $\{\frac{x^3}{2}, -4x^2\}$ and $\{\frac{x^3}{2}, -3\}$. The resulting upper bound is 8, whereas the *maximum positive real root* of the polynomial is 1.06815.

Another way of applying Theorem III.2 would be to pair each of the terms of $q_1(x)$ with $-4x^4$ of $q_2(x)$, and pick the minimum; that is, we pick the minimum of the terms $\{x^9, -4x^4\}$, $\{3x^8, -4x^4\}$, $\{2x^7, -4x^4\}$ and $\{x^6, -4x^4\}$, which is $\sqrt[4]{4/3} = 1.07457$. Then, we pair each of the negative terms of $q_4(x)$ with all of the unmatched positive terms of $q_1(x)$ and $q_3(x)$ and pick the minimum. That is, for the term $-4x^2$ we pick the minimum of $\{x^9, -4x^2\}$, $\{2x^7, -4x^2\}$, $\{x^6, -4x^2\}$ and $\{x^3, -4x^2\}$ which is $\sqrt[5]{2} = 1.1487$, whereas for the term -3 we pick the minimum of $\{x^9, -3\}$, $\{x^6, -3\}$ and $\{x^3, -3\}$ which is $\sqrt[9]{3} = 1.12983$. Finally, the bound is the $max\{\sqrt[4]{4/3}, \sqrt[5]{2}, \sqrt[9]{3}\} = 1.1487$.

This last approach is also encountered in (*Hong*, 1998) and (*Ştefănescu*, 2005). The computed bound is close to the optimal value, due to the quadratic complexity of this method, whereas the first one was linear. In the sequel, we first present implementation methods of Theorem III.2 that are linear in complexity and the computed bounds are close to the optimal value.

3.4 Linear Complexity Bounds

Bounds that we meet most often in the literature, such as Cauchy's and Kioustelidis', (§ 2.2.1, § 2.2.3), are of *linear* complexity.

The General Idea of the Linear Complexity Bounds: These bounds are computed as follows:

- *each* negative coefficient of the polynomial is paired with *one* of the preceding *unmatched* positive coefficients;
- the maximum of all the computed radicals is taken as the estimate of the bound.

In general, we can obtain better bounds if we pair coefficients from non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$, for $1 \leq l < i$. The earliest known implementation of this type is Cauchy's rule, that was described in (§ 2.2.1). Using Theorem III.2 we obtain the following interpretation of Cauchy's and Kioustelidis' theorems:

Definition 1: Cauchy's "*leading–coefficient*" implementation of Theorem III.2. For a polynomial $p(x)$, as in Eq. (2.1), with λ negative coefficients, Cauchy's method first breaks-up its leading coefficient, α_n, into λ *equal* parts and then pairs each part with the first unmatched negative coefficient. That is, we have:

$$ub_C = \max_{\{1 \leq k \leq n : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{\lambda \alpha_{n-k}}{\alpha_0}}$$

or, equivalently,

$$ub_C = \max_{\{1 \leq k \leq n : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{\alpha_{n-k}}{\frac{\alpha_0}{\lambda}}}.$$

So, in *Example 1* we form the pairs $\{\frac{x^9}{3}, -4x^4\}$, $\{\frac{x^9}{3}, -4x^2\}$ and $\{\frac{x^9}{3}, -3\}$, and obtain as upper bound the value 1.64375. This improvement in the estimation of the

bound is due to the fact that the radicals that come into play, namely $\sqrt[5]{12}$, $\sqrt[7]{12}$, and $\sqrt[9]{9}$, (obtained from the pairs mentioned above) are of higher order and hence the numbers computed are smaller.

From (§ 2.2.3) we obtain the following:

Definition 2: Kioustelidis' "*leading–coefficient*" implementation of Theorem III.2.

For a polynomial $p(x)$, as in Eq. (2.1), Kioustelidis' method matches the coefficient $-\alpha_{n-k}$ of the term $-\alpha_{n-k}x^{n-k}$ in $p(x)$ with $\frac{\alpha_n}{2^k}$, the leading coefficient divided by 2^k.

$$ub_K = 2 \max_{\{1\leq k\leq n:\alpha_{n-k}<0\}} \sqrt[k]{-\frac{\alpha_{n-k}}{\alpha_0}}$$

or, equivalently,

$$ub_K = \max_{\{1\leq k\leq n:\alpha_{n-k}<0\}} \sqrt[k]{-\frac{\alpha_{n-k}}{\frac{\alpha_0}{2^k}}}.$$

Kioustelidis' "leading-coefficient" implementation of Theorem III.2, differs from that of Cauchy's only in that the leading coefficient is now broken up in *unequal* parts, by dividing it with different powers of 2, *Kioustelidis* (1986).

So, in *Example 1* with Kioustelidis' method we form the pairs $\{\frac{x^9}{2^5}, -4x^4\}$, $\{\frac{x^9}{2^7}, -4x^2\}$ and $\{\frac{x^9}{2^9}, -3\}$, and obtain as upper bound the value 2.63902.

We can still improve the estimation of the upper bound, if we use *Remark 1* and we pair the two negative terms of $q_4(x)$ with the first two (of the three) ignored positive terms of $q_1(x)$. In this way, we obtain an upper bound of 1.31951, which is very close to 1.06815, the maximum positive root of $p_1(x)$. This new improvement is explained by the fact that the radicals $\sqrt[5]{4}$, $\sqrt[6]{4/3}$, and $\sqrt[7]{3/2}$, obtained from the pairs $\{x^9, -4x^4\}$, $\{3x^8, -4x^2\}$ and $\{2x^7, -3\}$, yield even smaller numbers.

Moreover, extensive experimentation confirmed that by pairing coefficients from the non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$ of $p(x)$, where $1 \leq l < i$, we ob-

tain bounds which are the same as, or better than, the bounds obtained by direct implementation of Theorem III.2, and in *most* cases better than those obtained by Cauchy's and Kioustelidis' rules.

Therefore, using Theorem III.2, a new linear complexity method, *first–*λ, was developed for computing upper bounds on the values of the positive roots of polynomials.

Definition 3: "*first–*λ" implementation of Theorem III.2.
For a polynomial $p(x)$, as in (3.3), with λ negative coefficients we first take care of all cases for which $t(q_{2i}) > t(q_{2i-1})$, by breaking-up the last coefficient $c_{2i-1,t(q_{2i})}$, of $q_{2i-1}(x)$, into $t(q_{2i}) - t(q_{2i-1}) + 1$ *equal* parts. We then pair each of the first λ positive coefficients of $p(x)$, encountered as we move in non-increasing order of exponents, with the first unmatched negative coefficient.

Although this bound is a significant improvement over the other two bounds by Cauchy and Kioustelidis, even this approach can lead, in some cases, to an overestimation of the upper bound, as seen in the following example, which highlights the importance of suitable pairing of negative and positive coefficients.

Example 2. Consider the polynomial

$$p(x) = x^3 + 10^{100}x^2 - 10^{100}x - 1.$$

which has one sign variation and, hence, only one positive root, $x = 1$.

Cauchy's "*leading–coefficient*" implementation of Theorem III.2 forms the pairs $\{\frac{x^3}{2}, -10^{100}x\}$ and $\{\frac{x^3}{2}, -1\}$, and taking the maximum of the radicals computed, we obtain a bound estimate of 1.41421×10^{50}; Kioustelidis' "*leading–coefficient*" implementation of Theorem III.2 forms the pairs $\{\frac{x^3}{2^2}, -10^{100}x\}$ and $\{\frac{x^3}{2^3}, -1\}$ yielding an upper bound of 2×10^{50}; and finally our "*first–*λ" implementation pairs the terms $\{x^3, -10^{100}x\}$ and $\{10^{100}x^2, -1\}$ yielding an upper bound of 10^{50}.

A "possible solution" to this problem could also be to scan the positive coefficients backwards (in non-decreasing order of exponents) in which case the pairs $\{10^{100}x^2, -10^{100}x\}$ and $\{x^3, -1\}$ are formed, yielding an upper bound of 1.

From the above example, it becomes obvious that in addition to the already presented implementations of Theorem III.2 we also need another, different pairing strategy to take care of cases in which these three approaches perform poorly.

However, the "possible solution" outlined above, may well take care of *Example 2*, but it picks coefficients from the adjacent polynomials $q_{2i-1}(x)$ and $q_{2i}(x)$ of $p(x)$, with all the associated weaknesses, mentioned above.

Therefore, we did not pick this "possible solution" as our fourth implementation of Theorem III.2. Instead, we chose the "*local-max*" pairing strategy, which is defined as follows:

Definition 4: "*local-max*" implementation of Theorem III.2.
For a polynomial $p(x)$, as in (3.3), the coefficient $-\alpha_k$ of the term $-\alpha_k x^k$ in $p(x)$ —as given in Eq. (3.3)— is paired with the coefficient $\frac{\alpha_m}{2^t}$, of the term $\alpha_m x^m$, where α_m is the largest positive coefficient with $n \geq m > k$ and t indicates the number of times the coefficient α_m has been used.

Note that our "*local-max*" strategy can pair coefficients of $p(x)$ from the non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$ of $p(x)$, where $1 \leq l < i$, and breaks-up positive coefficients also in *unequal* parts. Moreover, binary fractions of *only* the coefficient α_m get paired with each negative coefficient; this process continues until we encounter a greater positive coefficient.

Applying our "*local-max*" implementation to *Example 2* we form two pairs $\{\frac{10^{100}}{2}x^2, -10^{100}x\}$ and $\{\frac{10^{100}}{2^2}x^2, -1\}$, from which we obtain an upper bound of 2. Therefore, we return the value $2 = min\{10^{50}, 2\}$, which is the minimum of our "*first–λ*" and "*local-max*" implementations.

3.4.1 The Pseudocode

Below we present the pseudocode for the four different implementations of Theorem III.2. Cauchy's "*leading–coefficient*" implementation is described in Algorithm 1, lines 1–14, and the output is ub_C. Kioustelidis' "*leading–coefficient*" implementation is described in Algorithm 2, lines 1–14, and the output is ub_K. (These two bounds are presented here for completion.) The "*local-max*" implementation is described in Algorithm 3, lines 1–20, and the output is ub_{LM}. The "*first–λ*" implementation is described in Algorithms 4 & 5, lines 1–77, and the output is ub_{FL}. The final upper bound is $ub = min\{ub_{FL}, ub_{LM}\}$.

Input: A univariate polynomial $p(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + \ldots + \alpha_n, \quad (\alpha_0 > 0)$

Output: An upper bound, ub_C, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {α0, α1, α2, ..., αn−1, αn};
3  λ ⟵ the number of negative elements of cl;
4  if n + 1 <= 1 or λ = 0 then return ub_C = 0;
5  j = n + 1;
7  for i = 1 to n do
9      if cl(i) < 0 then
10         tempub = (λ(−cl(i)/cl(j)))^(1/(j−i));
11         if tempub > ub then ub = tempub;
12     end
13 end
14 ub_C = ub
```

Algorithm 1: Cauchy's "*leading–coefficient*" implementation of Theorem III.2.

Input: A univariate polynomial $p(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + \ldots + \alpha_n, \quad (\alpha_0 > 0)$

Output: An upper bound, ub_K, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {α_0, α_1, α_2, ..., α_{n-1}, α_n};
3  λ ⟵ the number of negative elements of cl;
4  if n + 1 <= 1 or λ = 0 then return ub_K = 0;
5  j = n + 1;
7  for i = 1 to n do
9      if cl(i) < 0 then
10         tempub = 2((−cl(i)/cl(j)))^{1/(j−i)};
11         if tempub > ub then ub = tempub;
12     end
13 end
14 ub_K = ub
```

Algorithm 2: Kioustelidis' "*leading–coefficient*" implementation of Thm. III.2.

Input: A univariate polynomial $p(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_0, \quad (\alpha_n > 0)$

Output: An upper bound, ub_{LM}, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {α_0, α_1, α_2, ..., α_{n-1}, α_n};
3  if n + 1 <= 1 then return ub_LM = 0;
4  j = n + 1;
5  t = 1;
7  for i = n to 1 step −1 do
9      if cl(i) < 0 then
10         tempub = (2^t(−cl(i)/cl(j)))^{1/(j−i)};
11         if tempub > ub then ub = tempub;
12         t++;
13     else
14         if cl(i) > cl(j) then
15             j = i;
16             t = 1
17         end
18     end
19 end
20 ub_LM = ub
```

Algorithm 3: The "*local-max*" implementation of Theorem III.2.

Input: A univariate polynomial $p(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_0, \quad (\alpha_n > 0)$

Output: An upper bound, ub_{FL}, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {α_0, α_1, α_2, ..., α_{n-1}, α_n};
3  λ ⟵ the number of negative elements of cl;
4  if n + 1 <= 1 or λ = 0 then return ub_FL = 0;
5  j = n + 1;
7  while j > 1 do // make sure t(q_{2i-1}) ≥ t(q_{2i}) holds for all i
9      while j > 1 and (cl(j) = 0 or cl(j) > 0) do // compute t(q_{2i-1})
11         flag = 0;
12         while j > 1 and cl(j) > 0 do
13             flag = 1;
14             posCounter + +;
15             j − −
16         end
17         if flag = 1 then LastPstvCoef = j + 1;
18         while j > 1 and cl(j) = 0 do
19             j − −
20         end
21     end
22     if j = 1 and cl(j) > 0 then posCounter + +;
23     while j > 1 and (cl(j) = 0 or cl(j) < 0) do // compute t(q_{2i})
25         while j > 1 and cl(j) < 0 do
26             negCounter + +;
27             j − −
28         end
29         while j > 1 and cl(j) = 0 do
30             j − −
31         end
32     end
33     if j = 1 and cl(j) < 0 then negCounter + +;
34     if negCounter > posCounter then // replace last coefficient by a list
           cl(LastPstvCoef) = { cl(LastPstvCoef) / (negCounter − posCounter + 1), ... }
                              \_____ negCounter−posCounter+1 _____/
35
36     end
37     negCounter = 0;
38     posCounter = 0;
39 end
```

Algorithm 4: The first part of the "*first–λ*" implementation of Theorem III.2.

```
40  i = j = n + 1;
42  while i > 0 and j > 0 and λ > 0 do // pair coefficients and process pairs
44      while cl(j) ≤ 0 do
45          j − −
46      end
48      if cl(j) is a list element then // cl(j) is a list element
49          while (cl(i) ≥ 0 or cl(i) is a list) and i > 1 do
50              i − −
51          end
52          tempub = (−cl(i)/cl(j))^{1/(j−i)};
53          λ − −;
54          if tempub > ub then ub = tempub;
55          i − −;
56          j − −;
57      end
58  end
59  if cl(j) is a list then // cl(j) is a list
60      k = the number of elements of cl(j);
61      temp = cl(j, 1);
63      if k > λ then
64          k = λ
65      end
66      for ν = 1 to k do
67          while (cl(i) ≥ 0 or cl(i) is a list) and i > 1 do
68              i − −
69          end
70          tempub = (−cl(i)/temp)^{1/(j−i)};
71          λ − −;
72          if tempub > ub then ub = tempub;
73          i − −;
74      end
75      j − −;
76  end
77  ub_FL = ub
```

Algorithm 5: The second part of the "*first–λ*" implementation of Theorem III.2.

3.4.2 Testing Linear Complexity Bounds

In this section, we present some examples using the same classes of polynomials, as in (*Akritas and Strzeboński*, 2005) in order to evaluate our new combined implementation, min{"*first–λ*", "*local-max*"}, of Theorem III.2 and to compare it with Cauchy's and Kioustelidis' "*leading–coefficient*" implementations.

Table 3.1, "uRandom" indicates a random polynomial whose leading coefficient is one[6], whereas "sRandom" indicates a random polynomial obtained with the randomly chosen seed 1001; the average size of the coefficients ranges from -2^{20} to 2^{20}. Additionally, Kioustelidis' name was shortened to "K" and a "star" indicates that the bound obtained by "*local-max*" was the minimum of the two. *MPR* stands for the maximum positive root, computed numerically.

[6]For exact mathematical formulas of the benchmark polynomials, please see the Appendix.

Table 3.1: Linear complexity bounds of positive roots for various types of polynomials.

		Degrees									
Polynomial	Bounds	10	100	200	300	400	500	600	700	800	900
Laguerre	Cauchy(ub_C)	500	5×10^5	4×10^6	1.35×10^7	3.2×10^7	6.25×10^7	1.08×10^8	1.72×10^8	2.56×10^8	3.65×10^8
	K (ub_K)	200	2×10^4	8×10^4	18×10^4	32×10^4	50×10^4	72×10^4	98×10^4	1.28×10^6	1.62×10^6
	$min(ub_{FL}, ub_{LM})$	100	1×10^4	4×10^4	9×10^4	16×10^4	25×10^4	36×10^4	49×10^4	64×10^4	81×10^4
	MPR	29.92	374.98	767.82	1162.8	1558.81	1955.44	2352.5	2749.87	3147.48	3545.29
ChebyshevI	Cauchy(ub_C)	2.74	25	50	75	100	125	150	175	200	225
	K (ub_K)	3.16	10	14.14	17.32	20	22.36	24.49	26.46	28.28	30
	$min(ub_{FL}, ub_{LM})$	1.58	5	7.07	8.66	10	11.18	12.25	13.23	14.14	15
	MPR	0.987688	0.999877	0.999969	0.999986	0.999992	0.999995	0.999997	0.999997	0.999998	0.999998
ChebyshevII	Cauchy(ub_C)	2.60	24.87	49.87	74.87	99.87	124.86	149.88	174.88	199.88	224.88
	K (ub_K)	3	9.95	14.11	17.29	19.98	22.34	24.47	26.44	28.27	29.98
	$min(ub_{FL}, ub_{LM})$	1.5	4.97	7.05	8.65	9.99	11.17	12.24	13.22	14.13	14.99
	MPR	0.959493	0.999516	0.999878	0.999945	0.999969	0.99998	0.999986	0.99999	0.999992	0.999994
Wilkinson	Cauchy(ub_C)	275	252500	2.01×10^6	6.77×10^6	1.6×10^7	3.13×10^7	5.4×10^7	8.59×10^7	1.28×10^8	1.82×10^8
	K (ub_K)	110	10100	40200	90300	160400	250500	360600	490700	640800	810900
	$min(ub_{FL}, ub_{LM})$	55	5050	20100	45150	80200	125250	180300	245350	320400	405450
	MPR	10	100	200	300	400	500	600	700	800	900
Mignotte	Cauchy(ub_C)	1.778	1.048	1.024	1.016	1.012	1.009	1.008	1.007	1.006	1.005
	K (ub_K)	3.26	2.081	2.040	2.026	2.020	2.016	2.013	2.011	2.0098	2.0087
	$min(ub_{FL}, ub_{LM})$	1.63	1.041	1.020	1.013	1.0099	1.0079	1.0066	1.0056	1.0049	1.0044
	MPR	1.5763	1.0362	1.0177	1.0117	1.0088	1.0070	1.0058	1.0050	1.0044	1.0039
uRandom	Cauchy(ub_C)	1892	42535	7.04×10^6	5282.2	9.62×10^7	11801.2	5.25×10^7	17389	17199.7	513.4
	K (ub_K)	1892	1810	135426	2001.73	1.01×10^6	1441.75	373400	1851.05	1746.37	133.8
	$min(ub_{FL}, ub_{LM})$	946	1810	135426*	4.92*	506494	29.3*	186700	20.4*	3.08*	2.57*
	MPR	944.962	905.528	67721.9	1.40192	506493	13.7921	186698	10.6972	0.998305	1.21821
Random	Cauchy(ub_C)	2.02	52	11.62	156.95	7.15	122.6	258.6	45.8	10.48	993.1
	K (ub_K)	2.23	2.24	2.28	2.04	2.41	2.32	3.49	4.88	2.08	4.54
	$min(ub_{FL}, ub_{LM})$	3.11*	2.15*	1.4	1.98*	1.68*	2.43*	3.47	2.44	1.82*	4.54*
	MPR	1.1843	1.64514	1.00699	1.22919	1.00248	1.39784	2.69568	1.00576	1.02541	3.39394
usRandom	Cauchy(ub_C)	602.6	17.61	205.1	1.50×10^8	100.4	13574	7.31×10^7	6.28×10^7	2.20×10^8	636.6
	K (ub_K)	602.6	18.61	91.19	2.06×10^6	54.17	1752.4	493872	364264	1.12×10^6	165.9
	$min(ub_{FL}, ub_{LM})$	1.48*	1.90*	1.73*	1.03163×10^6	1.99*	17.37*	493872*	364264*	557783	1.99*
	MPR	∄(-0.236)	∄(-0.236)	∄(-0.236)	1.03162×10^6	1.20669	9.69017	246938	182136	557782	1.06084
sRandom	Cauchy(ub_C)	13.6	152.5	303.1	458.9	87.2	513	6.03	5.16	18.36	8.65
	K (ub_K)	4.54	5.65	5.56	6.33	2.18	3.95	2.89	2.38	2.00	2.25
	$min(ub_{FL}, ub_{LM})$	4.54*	5.65*	5.56	3.17	1.61	3.64	1.44	1.67*	1.99*	1.99*
	MPR	2.40372	4.8321	3.5684	2.7936	1.02576	1.01633	1.00183	1.0038	1.01238	1.00061

From Table 3.1, we see that Kioustelidis' method is, in general, better (or much better) than that of Cauchy. This is not surprising given the fact that Kioustelidis breaks-up the leading coefficient in *unequal* parts, whereas Cauchy breaks it up in *equal* parts.

Our "*first–λ*" implementation, as the name indicates, uses additional coefficients and, therefore, it is not surprising that it is, in general, better (or much better) than both previous methods. In the few cases where Kioustelidis' method is better than "*first–λ*", the "*local-max*" method takes again the lead.

Therefore, given their linear cost of execution, we propose that one could safely use only the last two implementations of Theorem III.2 in order to obtain the best bounds possible. Certainly, this is worth trying in the continued fractions real root isolation method in order to further improve its performance. We will carry on this endeavor in Chapter IV of this study.

Last but not least, it should be noted that these new bounds, "*first–λ*", "*local-max*", as well as the min{"*first–λ*", "*local-max*"} have already been implemented into one of the newest open-source[7] mathematics software system, "`SAGE`", (*SAGE*, 2004–2010). A demonstration of a "`SAGE`" session calculating bounds on the values of the positive roots of some polynomials can be found in the next section.

3.4.3 `Sage` Session Demonstration of New Bounds

In `Sage` reference manual, (*SAGE*, 2004–2010), three methods are defined as:

sage.rings.polynomial.real_roots.`cl_maximum_root_first_lambda(cl)`,

sage.rings.polynomial.real_roots.`cl_maximum_root_local_max(cl)`,

sage.rings.polynomial.real_roots.`cl_maximum_root(cl)`

[7] Another implementation of our bounds can be found in the computer algebra system `Mathemagix`, (*Hoeven, Lecerf, Mourrain, and Ruatta*, 2008).

implementing our linear complexity bounds "*first–λ*", "*local-max*" and min{"*first–λ*", "*local-max*"}, described earlier, (*Akritas, Strzeboński, and Vigklas*, 2006). Given a polynomial represented by a list of its coefficients, `(cl)` (as *RealIntervalFieldElements*, *RIF*), an upper bound on its largest real root is being computed. Computing for instance the upper bound of the polynomial equation:

$$x^5 - 10x^4 + 15x^3 + 4x^2 - 16x + 400 = 0$$

we have

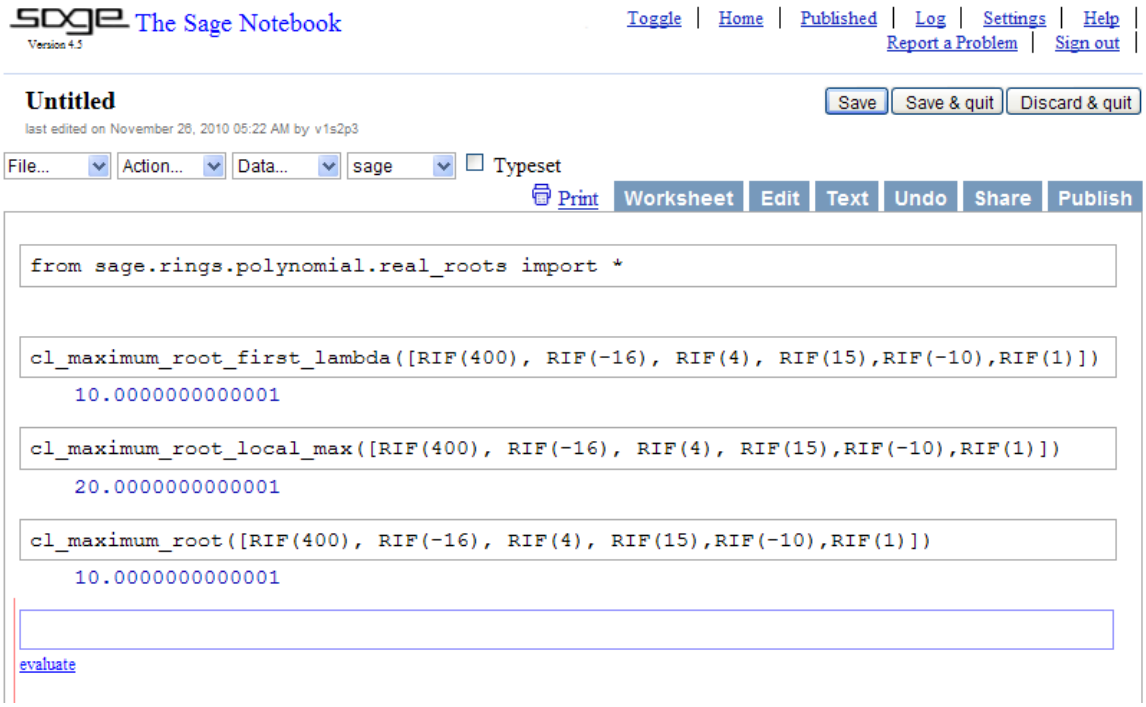

Figure 3.1: Screen capture of `Sage` software calculating bounds using the algorithms proposed in (*Akritas, Strzeboński, and Vigklas*, 2006).

The bounds above correspond to $ub_{FL} = 10$, $ub_{LM} = 20$, $min\{ub_{FL}, ub_{LM}\} = 10$ respectively, whereas the maximum positive root of the polynomial computed numerically is $MPR = 7.9945$.

3.5 Quadratic Complexity Bounds

To further investigate the new proposed bounds it was decided to define, in addition, new bounds of quadratic complexity this time (based on the linear complexity counterparts), hoping that their improved estimates *should* compensate for the extra time needed to compute them. These bounds are based on the following idea:

The General Idea of the Quadratic Complexity Bounds: These bounds are computed as follows:

- *each* negative coefficient of the polynomial is paired with *all* the preceding positive coefficients and the minimum of the computed values is taken;
- the maximum of all those minimums is taken as the estimate of the bound.

In general, the estimates obtained from the quadratic complexity bounds are less than or equal to those obtained from the corresponding linear complexity bounds, as the former are computed after much greater effort and time. The quadratic complexity bounds described below are all extensions of their linear complexity counterparts.

Thus, we have:

Definition 5: "*Cauchy Quadratic*" implementation of Theorem III.2.
For a polynomial $p(x)$, as in Eq. (2.1), each negative coefficient $a_i < 0$ is "paired" with *each* one of the preceding positive coefficients a_j divided by λ_i — that is, *each* positive coefficient a_j is "broken up" into *equal* parts, as is done with *just* the leading coefficient in Cauchy's bound; λ_i is the number of negative coefficients to the right of, and including, a_i — and the minimum is taken over all j; subsequently, the maximum is taken over all i.
That is, we have:

$$ub_{CQ} = \max_{\{a_i<0\}} \min_{\{a_j>0:j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{\lambda_i}}}.$$

Example 2, continued: For *Cauchy Quadratic* we first compute

- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2}, -10^{100}x\}$ and $\{\frac{10^{100}x^2}{2}, -10^{100}x\}$ which is 2,
- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2}, -1\}$ and $\{\frac{10^{100}x^2}{2}, -1\}$ which is $\frac{\sqrt{2}}{10^{50}}$,

and we then obtain as a bound estimate the value $max\{2, \frac{\sqrt{2}}{10^{50}}\} = 2$.

Definition 6: "*Kioustelidis' Quadratic*" implementation of Theorem III.2.
For a polynomial $p(x)$, as in Eq. (2.1), each negative coefficient $a_i < 0$ is "paired" with *each* one of the preceding positive coefficients a_j divided by 2^{j-i} — that is, *each* positive coefficient a_j is "broken up" into *unequal* parts, as is done with *just* the leading coefficient in Kioustelidis' bound — and the minimum is taken over all j; subsequently, the maximum is taken over all i.
That is, we have:

$$ub_{KQ} = 2 \max_{\{a_i<0\}} \min_{\{a_j>0:j>i\}} \sqrt[j-i]{-\frac{a_i}{a_j}},$$

or, equivalently,

$$ub_{KQ} = \max_{\{a_i<0\}} \min_{\{a_j>0:j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{j-i}}}}.$$

Example 2, continued: For *Kioustelidis' Quadratic* we first compute

- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2^2}, -10^{100}x\}$ and $\{\frac{10^{100}x^2}{2}, -10^{100}x\}$ which is 2,

- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2^3}, -1\}$ and $\{\frac{10^{100}x^2}{2^2}, -1\}$ which is $\frac{2}{10^{50}}$,

and we then obtain as a bound estimate the value $max\{2, \frac{2}{10^{50}}\} = 2$.

Definition 7: "*first*–λ *Quadratic*" implementation of Theorem III.2.

For a polynomial $p(x)$, as in (3.3), with λ negative coefficients we first take care of all cases for which $t(q_{2\ell}) > t(q_{2\ell-1})$, by breaking-up the last coefficient $c_{2\ell-1,t(q_{2\ell})}$, of $q_{2\ell-1}(x)$, into $d_{2\ell-1,t(q_{2\ell})} = t(q_{2\ell}) - t(q_{2\ell-1}) + 1$ *equal* parts. Then each negative coefficient $a_i < 0$ is "paired" with *each* one of the preceding $min(i, \lambda)$ positive coefficients a_j divided by d_j — that is, *each* of the preceding $min(i, \lambda)$ positive coefficient a_j is "broken up" into d_j *equal* parts, where d_j is initially set to 1 and its value changes *only* if the positive coefficient a_j is broken up into equal parts, as stated in Theorem III.2; $u(j)$ indicates the number of times a_j can be used to calculate the minimum, it is originally set equal to d_j and its value decreases each time a_j is used in the computation of the minimum — and the minimum is taken over all j; subsequently, the maximum is taken over all i.

That is, we have:

$$ub_{FLQ} = \max_{\{a_i<0\}} \min_{\{a_j>0: j>min(i,\lambda): u(j)\neq 0\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{d_j}}}.$$

From the above descriptions it is clear that u_{FLQ} tests *just* the first $min(i, \lambda)$ positive coefficients, whereas *all* the other quadratic complexity bounds test *every* preceding positive coefficient. Hence, u_{FLQ} is faster (or quite faster) than all of them.

Example 2, continued: For *first*–λ *Quadratic* we first compute

- the minimum of the two radicals obtained from the pairs of terms $\{x^3, -10^{100}x\}$ and $\{10^{100}x^2, -10^{100}x\}$ which is 1 — evaluated from the second pair of terms,

- the minimum of the two radicals obtained from the pairs of terms $\{x^3, -1\}$ and $\{10^{100}x^2, -1\}$ which is 1,

and we then obtain as a bound estimate the value $max\{1, 1\} = 1$. Note that once a term with a positive coefficient has been used in obtaining the minimum, it cannot be used again!

Definition 8: "*local-max Quadratic*" implementation of Theorem III.2.
For a polynomial $p(x)$, as in (3.3), each negative coefficient $a_i < 0$ is "paired" with *each* one of the preceding positive coefficients a_j divided by 2^{t_j} — that is, *each* positive coefficient a_j is "broken up" into *unequal* parts, as is done with *just* the locally maximum coefficient in the local max bound; t_j is initially set to 1 and is incremented each time the positive coefficient a_j is used — and the minimum is taken over all j; subsequently, the maximum is taken over all i.
That is, we have:

$$ub_{LMQ} = \max_{\{a_i<0\}} \min_{\{a_j>0:j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{t_j}}}}.$$

Since $2^{t_j} \leq 2^{j-i}$ — where i and j are the indices realizing the max of min; equality holds when there are *no* missing terms in the polynomial — it is clear that the estimates computed by **"*local-max Quadratic*"** are sharper by the factor $2^{\frac{j-i-t_j}{j-i}}$ than those computed by **"*Kioustelidis' Quadratic*"**.

Example 2, continued: For "*local-max Quadratic*" we first compute

- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2}, -10^{100}x\}$ and $\{\frac{10^{100}x^2}{2}, -10^{100}x\}$ which is 2,

- the minimum of the two radicals obtained from the pairs of terms $\{\frac{x^3}{2^2}, -1\}$ and $\{\frac{10^{100}x^2}{2^2}, -1\}$ which is $\frac{2}{10^{50}}$,

and we then obtain as a bound estimate the value $max\{2, \frac{2}{10^{50}}\} = 2$.

3.5.1 The Pseudocode

Below we present the pseudocode for ub_{LMQ} and ub_{FLQ} quadratic implementations of Theorem III.2. We decided to omit ub_{CQ} and ub_{KQ} implementations since both previous theoretical analysis and empirical data establish the better performance of ub_{LMQ} and ub_{FLQ} over these two in every case. The ub_{LMQ}, **"*local-max Quadratic*"** implementation is described in Algorithm 6, lines 1–18, whereas the ub_{FLQ}, **"*first*–λ *Quadratic*"** implementation is described in Algorithms 7 and 8, lines 1–66.

Input : A univariate polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0, (a_n > 0)$

Output: An upper bound ub_{LMQ}, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {a_0, a_1, a_2, ..., a_{n-1}, a_n};
3  timesused ⟵ {1, 1, 1, ..., 1};
4  ub = 0;
5  if n + 1 ≤ 1 then return ub = 0;
6  for m ⟵ n to 1 do
7      if cl(m) < 0 then
8          tempub = ∞;
9          for k ⟵ n + 1 to m + 1 do
10             temp = ( -cl(m) / ( cl(k) / 2^timesused(k) ) )^(1/(k-m));
11             timesused(k) + +;
12             if tempub > temp then tempub = temp;
13         end
14         if ub < tempub then ub = tempub;
15     end
16 end
17 ub_LMQ = ub;
18 return ub_LMQ;
```

Algorithm 6: The "*local-max Quadratic*" implementation of Theorem III.2.

Input : A univariate polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0, (a_n > 0)$

Output: An upper bound ub_{FLQ}, on the values of the positive roots of the polynomial

initializations;
$cl \longleftarrow \{a_0, a_1, a_2, \ldots, a_{n-1}, a_n\}$;
$\lambda \longleftarrow$ number of negative elements of cl;
$usedVector \longleftarrow \{0, 0, 0, \ldots, 0\}$;
for $i \longleftarrow 1$ **to** $n + 1$ **do**
    if $cl(i) > 0$ **then** $usedVector(i) = 1$;
end
if $n + 1 \leq 1$ **or** $\lambda = 0$ **then return** $ub = 0$;
$i = n + 1$;
$templamda = 0$;
$flag = 0$;
while $templamda < \lambda$ **do** // make sure $t(q_{2i-1}) \geq t(q_{2i})$ holds for all i
    if $cl(i) > 0$ **then**
        if $flag = 0$ **then** $posCounter++$;
        else if $flag = 1$ **then**
            if $negCounter > posCounter$ **then**
                $usedVector(positionLastPositiveCoef) = negCounter - posCounter + 1$;
            end
            $negCounter = 0$;
            $posCounter = 1$;
            $flag = 0$;
        end
        $positionLastPositiveCoef = i$;
    else if $cl(i) < 0$ **then**
        $flag = 1$;
        $negCounter++$;
        $templamda++$;
    end
    $i--$;
end
if $negCounter > posCounter$ **then**
    $usedVector(positionLastPositiveCoef) = negCounter - posCounter + 1$;
end

Algorithm 7: 1st part of "*first–λ Quadratic*" implementation of Theorem III.2.

```
sumPosCoeff = 0;
i = n + 1;
// Last of the first-λ coefficients
while sumPosCoeff < λ do
    if usedVector(i) ≠ 0 then
        sumPosCoeff+ = usedVector(i);
        flPos = i;
    end
    i − −;
end
/* If the last of the first-λ coefficients is a broken one (usedVector(flPos) > 1), there might
   be a chance that the sum of the positive coefficients (including broken ones) is more than
   λ. For Example: Let the signs of p be + + + - + + - - - + + + - the 5^th positive
   coefficient will be broken into 2 pieces (usedVector(8) = 2). However, the sum of the
   first-λ (5 non broken) positive coefficients is 6 (incl. broken). As a result we are going
   to use the last of the positive first-λ coefficients timesToUse(8) − (sum − λ) = 1 time only.
   */
timesToUse(flpos)− = (sumPosCoeff − λ);
denomVector ⟵ usedVector;
m = n;
ub = 0;
while λ > 0 do
    if cl(m) < 0 then
        tempub = ∞;
        for k = n + 1 to max(m + 1, flPos) do
            if usedVector(k) > 0 then
                tempB = ( −cl(m) / cl(k) / denomVector(k) )^(1/(k−m));
                if tempub > tempB then
                    tempub = tempB;
                    tempN = k;
                end
            end
        end
        usedVector(tempN) − −;
        λ − −;
        if ub < tempub then ub = tempub;
    end
    m − −;
end
ub_FLQ = ub;
return ub_FLQ;
```

Algorithm 8: 2nd part of "*first–λ Quadratic*" implementation of Theorem III.2

3.5.2 Testing Quadratic Complexity Bounds

In this section, we present some results using the same classes of polynomials[8], as in (*Akritas and Strzeboński*, 2005) in order to compare **"*first*–λ *Quadratic*"** and **"*local-max Quadratic*"** implementation of Theorem III.2.

In Table 3.2, **"*first*–λ *Quadratic*"** and **"*local-max Quadratic*"** names were shortened to ub_{FLQ} and 'ub_{LMQ} respectively. Also, in parenthesis the respective computation time is given for each algorithm, whereas *MPR* stands for the maximum positive root, computed numerically.

[8]For exact mathematical formulas of the benchmark polynomials, please see the Appendix.

Table 3.2: Quadratic complexity bounds of positive roots for various types of polynomials.

		Degrees									
Polynomial	Bounds	10	100	200	300	400	500	600	700	800	900
Laguerre	ub_{LMQ}	200 (0.)	2×10^4 (0.563)	8×10^4 (3.562)	18×10^4 (11.187)	32×10^4 (25.594)	50×10^4 (49.782)	72×10^4 (87.344)	98×10^4 (142.453)	128×10^4 (220.766)	162×10^4 (329.719)
	ub_{FLQ}	100 (0.)	10^4 (0.015)	4×10^4 (0.031)	9×10^4 (0.078)	16×10^4 (0.109)	25×10^4 (0.172)	36×10^4 (0.25)	49×10^4 (0.328)	64×10^4 (0.406)	81×10^4 (0.5)
	MPR	29.92	374.98	767.82	1162.8	1558.81	1955.44	2352.5	2749.87	3147.48	3545.29
ChecyshevI	ub_{LMQ}	2.23607 (0.)	7.07107 (0.078)	10 (0.515)	12.2474 (1.547)	14.1421 (3.453)	15.8114 (6.453)	17.3205 (10.688)	18.7083 (15.891)	20 (23.406)	21.2132 (33.75)
	ub_{FLQ}	1.58114 (0.)	5 (0.)	7.07107 (0.015)	8.66025 (0.047)	10 (0.62)	11.1803 (0.11)	12.2474 (0.141)	13.288 (0.172)	14.1421 (0.234)	15 (0.281)
	MPR	0.987688	0.999877	0.999969	0.999989	0.999992	0.999995	0.999997	0.999997	0.999998	0.999998
ChecyshevII	ub_{LMQ}	2.12132 (0.015)	7.03562 (0.079)	9.97497 (0.531)	12.227 (1.563)	14.1244 (3.453)	15.7956 (6.468)	17.3061 (10.719)	18.6949 (16.687)	19.9875 (24.656)	21.2014 (34.734)
	ub_{FLQ}	1.5 (0.)	4.97494 (0.016)	7.05337 (0.015)	8.64581 (0.047)	9.98749 (0.078)	11.1692 (0.109)	12.2372 (0.141)	13.2193 (0.187)	14.1333 (0.219)	14.9917 (0.265)
	MPR	0.959493	0.999516	0.999878	0.999945	0.999969	0.99998	0.999986	0.99999	0.999992	0.999994
Wilkinson	ub_{LMQ}	110 (0.)	10100 (6.391)	40200 (52.234)	90300 (179.75)	160400 (438.516)	250500 (878.)	360600 (1549.89)	490700 (2508.52)	640800 (3833.52)	810900 (5569.42)
	ub_{FLQ}	55 (0.)	5050 (0.016)	20100 (0.047)	45150 (0.078)	80200 (0.014)	125250 (0.203)	180300 (0.282)	245350 (0.359)	320400 (0.516)	405450 (0.656)
	MPR	10	100	200	300	400	500	600	700	800	900
Mignotte	ub_{LMQ}	1.77828 (0.)	1.04811 (0.)	1.02353 (0.015)	1.01557 (0.)	1.01164 (0.)	1.00929 (0.016)	1.00773 (0.)	1.00662 (0.)	1.00579 (0.)	1.00514 (0.015)
	ub_{FLQ}	1.63069 (0.)	1.04073 (0.016)	1.01995 (0.)	1.01321 (0.016)	1.00988 (0.15)	1.00789 (0.016)	1.00656 (0.016)	1.00562 (0.015)	1.00491 (0.016)	1.00437 (0.015)
	MPR	1.5763	1.0362	1.0177	1.0117	1.0088	1.0070	1.0058	1.0050	1.0044	1.0039
sRandom	ub_{LMQ}	2.01011 (0.)	14.3673 (0.219)	1.39904 (1.016)	3.54546 (2.11)	3.65744 (3.828)	7.75602 (6.)	2.5257 (8.625)	1.53975 (11.735)	1.70317 (14.765)	1.65478 (18.734)
	ub_{FLQ}	2.45417 (0.)	25.1062 (0.031)	1.04472 (0.157)	3.54546 (0.281)	2.5862 (0.547)	7.75602 (0.843)	2.03158 (1.157)	1.6409 (1.656)	1.48873 (1.5)	1.17328 (2.375)
	MPR	1.6173	8.15106	0.982276	1.1221	1.71921	1.012339	0.983633	0.983628	1.010844	0.983628
usRandom	ub_{LMQ}	1.57532 (0.)	1916790 (0.25)	1.40849 (1.032)	1.78915 (2.125)	105264 (3.797)	1272940 (5.891)	1803160 (8.5)	12.6533 (11.531)	1197500 (14.875)	790432 (18.765)
	ub_{FLQ}	3.16629 (0.016)	1916790 (0.047)	1.06293 (0.156)	15.6504 (0.266)	105264 (0.531)	1272940 (0.828)	901580 (1.156)	6.32665 (1.641)	598750 (2.046)	790432 (2.328)
	MPR	0.925381	1.018871	0.982509	0.9841	1.026689	1.011621	1.331235	1.755769	1.013423	1.228396
pRandom	ub_{LMQ}		5984.55 (10.297)	8818.63 (75.453)			14435.4 (1053.55)				
	ub_{FLQ}		4231.72 (0.562)	5555.39 (1.453)			9093.75 (21.156)				
	MPR		998	998			1019				

From the data presented in the Table 3.2, it becomes obvious that the sharpness of the estimates of both ub_{FLQ} and ub_{LMQ} is about the same, but ub_{FLQ} in most cases runs faster (or quite faster) than ub_{LMQ}. So, when it comes to quadratic complexity bounds the ub_{FLQ} algorithm is undoubtedly the best choice regarding sharpness to speed of computation, ratio. However, examining both Table 3.2 and Table 3.1, one must be very careful in his choice of quadratic versus linear complexity bounds because then he has to trade-off between a slightly better bound estimation and a greater algorithmic complexity and execution time. This last remark seems to have been exploited by the commercial computational software program, `Mathematica`, (*Wolfram Research*, 2008), as we can see in the following section.

3.5.3 `Mathematica` Session Demonstration of New Bounds

The `Mathematica`'s real root isolation source code, *by default*, uses the better bound from each category, i.e. **"*first*–λ"**, ub_{FL}, from the linear complexity bounds and **"*local-max Quadratic*"**, ub_{LMQ} from the quadratic complexity ones, (*Strzeboński*, 2010). However, its source code also contains implementations of Cauchy's, (§ 2.2.1), and Hong's bound, (*Hong*, 1998), which we call "*Kioustelidis' Quadratic*" in our work as well as the **"*local-max*"**, ub_{LM} and **"*first*–λ *Quadratic*"**, ub_{FLQ}, bounds. There is an undocumented system variable which allows to change the bound used to any combination of those bounds. The new bounds have been added in `Mathematica` version 7.

These bounds are always *implicity* used for calculating intervals isolating polynomial real roots. Intervals are given by command `RootIntervals`. Real root isolation is used by *any* `Mathematica` function that requires real algebraic number computation.

See the following examples:

```
In[1]:= poly = x^5 - 3 x + 1;
        (* Find real and complex roots of the above polynomial:*)

In[2]:= RootIntervals[poly]
        (* Isolate the real roots; multiple roots are indicated in the second part of the output: *)

Out[2]= {{{-2, 0}, {0, 1}, {1, 3}}, {{1}, {1}, {1}}}

In[3]:= Reduce[poly == 0, x, Reals]
        (* Use Reduce to find the real roots; multiple roots are given once:*)

Out[3]= x == Root[1 - 3 #1 + #1^5 &, 1] || x == Root[1 - 3 #1 + #1^5 &, 2] || x == Root[1 - 3 #1 + #1^5 &, 3]

In[4]:= RootIntervals[poly, Complexes]
        (* Isolate the complex roots; multiple roots are indicated in the second part of the output:*)

Out[4]= {{{-2, 0}, {0, 1}, {1, 3}, {-3 - 3 i, 3}, {-3, 3 + 3 i}}, {{1}, {1}, {1}, {1}, {1}}}

In[5]:= Reduce[poly == 0, x]
        (* Use Reduce to find the complex roots; multiple roots are given once:*)

Out[5]= x == Root[1 - 3 #1 + #1^5 &, 1] || x == Root[1 - 3 #1 + #1^5 &, 2] || x == Root[1 - 3 #1 + #1^5 &, 3] || x == Root[1 - 3 #1 + #1^5 &, 4] || x == Root[1 - 3 #1 + #1^5 &, 5]

In[6]:= Solve[poly == 0, x]
        (* Use Solve to find the complex roots with multiplicities: *)

Out[6]= {{x → Root[1 - 3 #1 + #1^5 &, 1]}, {x → Root[1 - 3 #1 + #1^5 &, 2]}, {x → Root[1 - 3 #1 + #1^5 &, 3]}, {x → Root[1 - 3 #1 + #1^5 &, 4]}, {x → Root[1 - 3 #1 + #1^5 &, 5]}}

In[7]:= NSolve[poly == 0, x]
        (* Use NSolve to find numerical approximate solutions of the polynomial: *)

Out[7]= {{x → -1.38879}, {x → -0.0802951 - 1.32836 i}, {x → -0.0802951 + 1.32836 i}, {x → 0.334734}, {x → 1.21465}}
```

Figure 3.2: Screen capture of `Mathematica` software using the proposed bounds in various commands.

CHAPTER IV

Application of the New Bounds to Real Root Isolation Methods

4.1 Introduction

In this chapter we apply the newly proposed *linear* and *quadratic* complexity (upper) bounds on the values of the positive roots of polynomials on a method for the isolation of real roots of polynomials. Although there are many root isolations methods (based on continued fractions, bisection, exclusion, etc) that could greatly benefit from our new sharper bounds we decided to study their impact on the performance of the Vincent-Akritas-Strzeboński (`VAS`) method for the isolation of real roots of polynomials. The `VAS` real root isolation method is based on continued fractions and till today is considered the fastest among its rivals, having already being incorporated in major mathematical software packages.

Computing (lower) bounds on the values of the positive roots of polynomials is a crucial operation in the `VAS` method. Therefore, we begin by reviewing some basic facts about this method, which is based on Vincent's theorem[1], (*Vincent*, 1836):

[1]For a complete overview of Vincent's theorem of 1836 and its implications to root isolation, see (*Akritas*, 2010).

Theorem IV.1 (Vincent, 1836)**.** *If in a polynomial, $p(x)$, of degree n, with rational coefficients and without multiple roots we perform sequentially replacements of the form*

$$x \leftarrow \alpha_1 + \frac{1}{x}, x \leftarrow \alpha_2 + \frac{1}{x}, x \leftarrow \alpha_3 + \frac{1}{x}, \ldots$$

where $\alpha_1 \geq 0$ is an arbitrary non negative integer and $\alpha_2, \alpha_3, \ldots$ are arbitrary positive integers, $\alpha_i > 0$, $i > 1$, then the resulting polynomial either has no sign variations or it has one sign variation. In the last case the equation has exactly one positive root, which is represented by the continued fraction

$$\alpha_1 + \cfrac{1}{\alpha_2 + \cfrac{1}{\alpha_3 + \cfrac{1}{\ddots}}}$$

whereas in the first case there are no positive roots.

The thing to note is that the quantities α_i (the partial quotients of the continued fraction) are computed by repeated application of a method for estimating *lower* bounds[2] on the values of the positive roots of a polynomial.

Therefore, the efficiency of the `VAS` continued fractions method heavily depends on how good these estimates are.

4.2 Algorithmic Background of the `VAS` Method

In the sequel we present the `VAS` algorithm –as found in (*Akritas and Strzeboński*, 2005)– and correct a misprint in Step 5 that had appeared in that presentation; moreover, we explain where the new bound on the positive roots is used.

[2]A *lower* bound, ℓb, on the values of the positive roots of a polynomial $f(x)$, of degree n, is found by first computing an *upper* bound, ub, on the values of the positive roots of $x^n f(\frac{1}{x})$ and then setting $\ell b = \frac{1}{ub}$, see (§ 2.1.2).

4.2.1 Description of the VAS-Continued Fractions Algorithm

Using the notation of the paper *Akritas and Strzeboński* (2005), let $f \in \mathbb{Z}[x] \setminus \{0\}$. By $sgc(f)$ we denote the number of sign changes in the sequence of nonzero coefficients of f. For nonnegative integers a, b, c, and d, such that $ad - bc \neq 0$, we put

$$intrv(a, b, c, d) := \Phi_{a,b,c,d}((0, \infty))$$

where

$$\Phi_{a,b,c,d} : (0, \infty) \ni x \longrightarrow \frac{ax + b}{cx + d} \in (min(\frac{a}{c}, \frac{b}{d}), max(\frac{a}{c}, \frac{b}{d}))$$

and by *interval data* we denote a list

$$\{a, b, c, d, p, s\}$$

where p is a polynomial such that the roots of f in $intrv(a, b, c, d)$ are images of positive roots of p through $\Phi_{a,b,c,d}$, and $s = sgc(p)$.

The value of parameter α_0 used in step 4 below needs to be chosen empirically. In our implementation $\alpha_0 = 16$.

Algortihm Continued Fractions (VAS).

Input: A squarefree polynomial $f \in \mathbb{Z}[x] \setminus \{0\}$

Output: The list *rootlist* of the isolation intervals of the positive roots of f.

1. Set *rootlist* to an empty list. Compute $s \leftarrow sgc(f)$. If $s = 0$ return an empty list. If $s = 1$ return $\{(0, \infty)\}$. Put interval data $\{1, 0, 0, 1, f, s\}$ on *intervalstack*.

2. If *intervalstack* is empty, return *rootlist*, else take interval data $\{a, b, c, d, p, s\}$ off *intervalstack*.

3. Compute a lower bound $\alpha \in \mathbb{Z}$ on the positive roots of p.

4. If $\alpha > \alpha_0$ set $p(x) \leftarrow p(\alpha x)$, $a \leftarrow \alpha a$, $c \leftarrow \alpha c$, and $\alpha \leftarrow 1$.

5. If $\alpha \geq 1$, set $p(x) \leftarrow p(x+\alpha)$, $b \leftarrow \alpha a + b$, and $d \leftarrow \alpha c + d$. If $p(0) = 0$, add $[b/d, b/d]$ to *rootlist*, and set $p(x) \leftarrow p(x)/x$. Compute $s \leftarrow sgc(p)$. If $s = 0$ go to step 2. If $s = 1$ add $intrv(a, b, c, d)$ to *rootlist* and go to step 2.

6. Compute $p_1(x) \leftarrow p(x+1)$, and set $a_1 \leftarrow a$, $b_1 \leftarrow a + b$, $c_1 \leftarrow c$, $d_1 \leftarrow c + d$, and $r \leftarrow 0$. If $p_1(0) = 0$, add $[b_1/d_1, b_1/d_1]$ to *rootlist*, and set $p_1(x) \leftarrow p_1(x)/x$, and $r \leftarrow 1$. Compute $s_1 \leftarrow sgc(p_1)$, and set $s_2 \leftarrow s - s_1 - r$, $a_2 \leftarrow b$, $b_2 \leftarrow a + b$, $c_2 \leftarrow d$, and $d_2 \leftarrow c + d$.

7. If $s_2 > 1$, compute $p_2(x) \leftarrow (x+1)^m p(\frac{1}{x+1})$, where m is the degree of p. If $p_2(0) = 0$, set $p_2(x) \leftarrow p_2(x)/x$. Compute $s_2 \leftarrow sgc(p_2)$.

8. If $s_1 < s_2$, swap $\{a_1, b_1, c_1, d_1, p_1, s_1\}$ with $\{a_2, b_2, c_2, d_2, p_2, s_2\}$.

9. If $s_1 = 0$ goto step 2. If $s_1 = 1$ add $intrv(a_1, b_1, c_1, d_1)$ to *rootlist*, else put interval data $\{a_1, b_1, c_1, d_1, p_1, s_1\}$ on *intervalstack*.

10. If $s_2 = 0$ goto step 2. If $s_2 = 1$ add $intrv(a_2, b_2, c_2, d_2)$ to *rootlist*, else put interval data $\{a_2, b_2, c_2, d_2, p_2, s_2\}$on *intervalstack*. Go to step 2.

Please note that the lower bound[3], α, on the positive roots of $p(x)$ is computed in Step 3, and used in Step 5.

4.2.2 The Pseudocode of the VAS-Continued Fractions Algorithm

We present the pseudocode of the VAS-Continued Fractions Root Isolation Method, below in Algorithm 9, lines 1–13. Note the repeated use of the lower bound *lb* in lines 4–5.

[3]As mentioned in (§ 2.2), Cauchy's bound was the only one known and the first one to be used in VAS, in 1978. This of course changed, in 2006, after we developed the new bounds.

Input: The square-free polynomial $p(x) \in \mathbb{Z}[x], p(0) \neq 0$, and the Möbius transformation

$$M(x) = \frac{ax+b}{cx+d} = x,\ a, b, c, d \in \mathbb{Z}$$

Output: A list of isolating intervals of the positive roots of $p(x)$

1 $var \longleftarrow$ the number of sign changes of $p(x)$;
2 **if** $var = 0$ **then RETURN** $\emptyset$;
3 **if** $var = 1$ **then RETURN** $\{]a, b[\}$ `// a = min(M(0),M(∞)), b = max(M(0),M(∞));`
4 $\ell b \longleftarrow$ a lower bound on the positive roots of $p(x)$;
5 **if** $\ell b > 1$ **then** $\{p \longleftarrow p(x + \ell b), M \longleftarrow M(x + \ell b)\}$;
6 $p_{01} \longleftarrow (x+1)^{deg(p)} p(\frac{1}{x+1}), M_{01} \longleftarrow M(\frac{1}{x+1})$ `// Look for real roots in` $]0, 1[$;
7 $m \longleftarrow M(1)$ `// Is 1 a root?` ;
8 $p_{1\infty} \longleftarrow p(x+1), M_{1\infty} \longleftarrow M(x+1)$ `// Look for real roots in` $]1, +\infty[$;
9 **if** $p(1) \neq 0$ **then**
10 | **RETURN** `VAS`$(p_{01}, M_{01}) \bigcup$ `VAS`$(p_{1\infty}, M_{1\infty})$
11 **else**
12 | **RETURN** `VAS`$(p_{01}, M_{01}) \bigcup \{[m, m]\} \bigcup$ `VAS`$(p_{1\infty}, M_{1\infty})$
13 **end**

Algorithm 9: `VAS`-Continued Fractions Algorithm.

4.2.3 Example of the Real Root Isolation Method

Executing *Algorithm* 9 for the polynomial $8x^4 - 18x^3 + 9x - 2$ which has one negative, $-\sqrt{2}/2$ and three positive real roots, $1/4$, $\sqrt{2}/2$ and 2, we have:

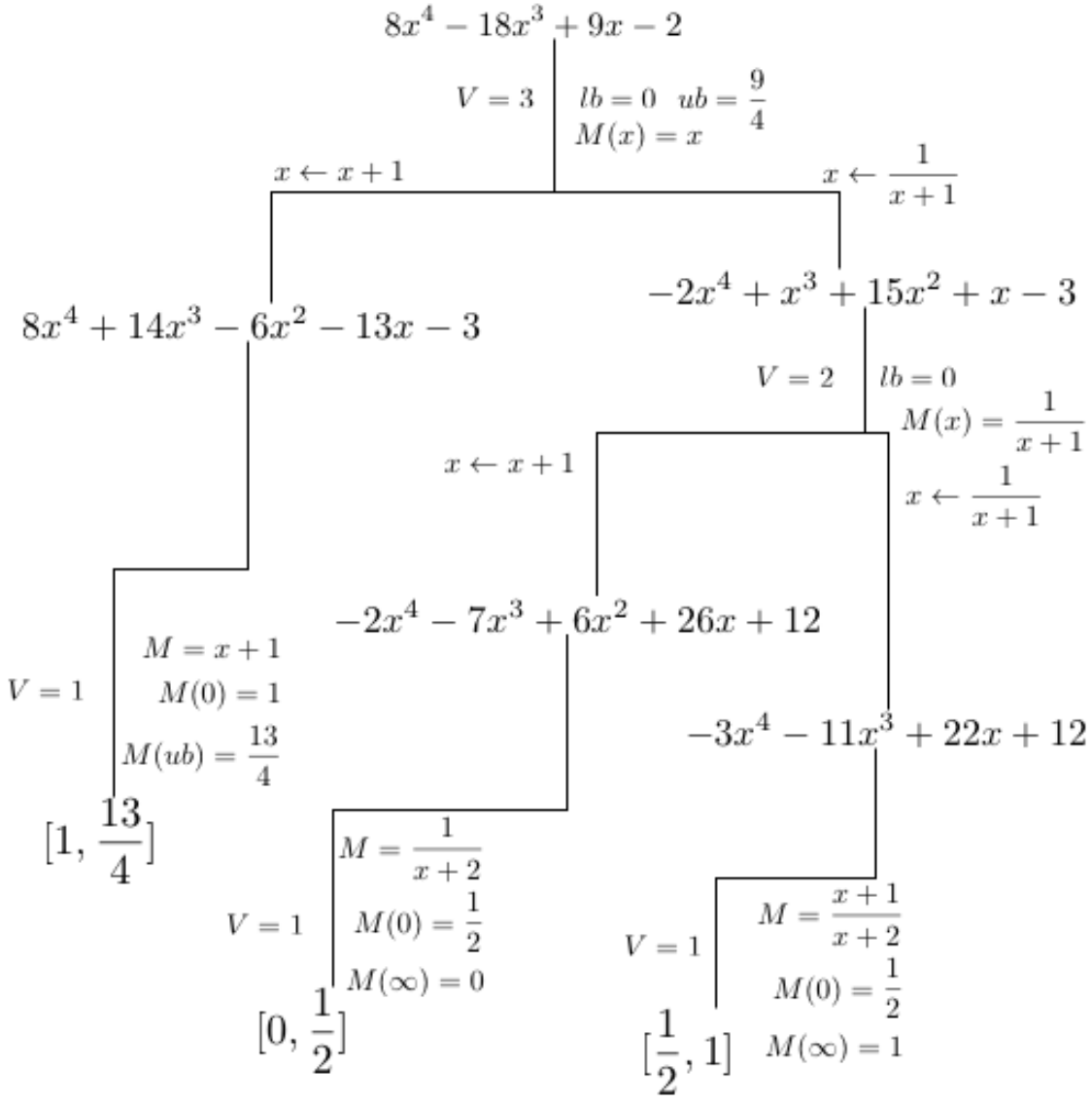

Figure 4.1: Tree-diagram of the VAS-CF Real Root Isolation Algorithm 9, for the polynomial $8x^4 - 18x^3 + 9x - 2$.

4.3 Benchmarking `VAS` with New Bounds

In this section we compare four implementations of the `VAS` real root isolation method using two linear and two quadratic complexity bounds on the values of the positive roots of polynomials.

The two linear complexity bounds are: Cauchy's, ub_C and $min(ub_{FL}, ub_{LM})$, the minimum of **"*first*–λ"** and **"*local-max*"** bounds, (*Akritas, Strzeboński, and Vigklas*, 2006), whereas the two quadratic complexity ones are: ub_{KQ}, the Quadratic complexity variant of Kioustelidis' bound, studied by Hong, (*Hong*, 1998), and ub_{LMQ}, the Quadratic complexity version of the **"*local-max*"** bound, (*Akritas, Strzeboński, and Vigklas*, 2008a).

Our choice of the various bounds in the implementations of `VAS` is justified as follows:

1. From the linear complexity bounds we included:

   (a) Cauchy's bound, ub_C, to be used as a point of reference, since it has been in use for the past 30 years, and

   (b) $min(ub_{FL}, ub_{LM})$ bound, (*Akritas, Strzeboński, and Vigklas*, 2006), which is the best among the linear complexity bounds, in order to see when it's implementation will outperform that of the two quadratic complexity bounds.

2. From the quadratic complexity bounds we included:

   (a) Kioustelidis' bound, ub_{KQ} and

   (b) ub_{LMQ} bound in order to compare their performance; as explained in the previous chapter ub_{LMQ} computes sharper estimates than ub_{KQ}.

They all use the same implementation of Shaw and Traub's algorithm for Taylor shifts (*von zur Gathen and Gerhard*, 1997). We followed the standard practice and used as benchmark the Laguerre, Chebyshev (first and second kind), Wilkinson and Mignotte polynomials[4], as well as several types of randomly generated polynomials of degrees $\{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000\}$. For the random polynomials the size of the coefficients ranges from -2^{20} to 2^{20}.

[4]For exact mathematical formulas of the benchmark polynomials, please see the Appendix.

Table 4.1: Special polynomials of some indicative degrees.

Polynomial Class	Degree	ub_C $Time(s)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$	$ub_{KQ}(Hong)$ $Time(s)$	ub_{LMQ} $Time(s)$
Laguerre	100	0.23	0.19	0.19	0.17
Laguerre	1000	979	665	729	633
Laguerre	1500	7194	4903	5356	4569
Laguerre	2000	27602	21007	22712	19277
ChebyshevI	100	0.19	0.17	0.16	0.11
ChebyshevI	1000	517	460	496	299
ChebyshevI	1500	3681	3333	3381	2188
ChebyshevI	2000	16697	15010	14571	10473
ChebyshevII	100	0.42	0.17	0.15	0.10
ChebyshevII	1000	529	437	443	296
ChebyshevII	1500	3772	3198	3190	2166
ChebyshevII	2000	16559	14492	14370	10184
Wilkinson	100	0.03	0.03	0.03	0.03
Wilkinson	1000	54.6	44.5	43.7	43.3
Wilkinson	1500	339	295	270	265
Wilkinson	2000	1361	1305	1241	1242
Mignotte	100	0.008	0.004	0.008	0.004
Mignotte	1000	0.79	0.78	0.81	0.66
Mignotte	1500	2.05	2.12	2.06	1.77
Mignotte	2000	4.52	4.37	4.47	3.69

Table 4.2: Polynomials with random 10-bit coefficients.

Degree	No. of roots $Avg(Min/Max)$	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	4.4 (2/6)	0.01 (0.00/0.01)	0.01 (0.01/0.02)	0.01 (0.01/0.02)	0.01 (0.01/0.01)
200	4.0 (2/8)	0.06 (0.02/0.18)	0.06 (0.03/0.16)	0.05 (0.03/0.14)	0.04 (0.03/0.09)
300	4.8 (4/6)	0.14 (0.07/0.24)	0.12 (0.06/0.22)	0.13 (0.07/0.19)	0.09 (0.07/0.13)
400	4.4 (4/6)	0.17 (0.12/0.21)	0.18 (0.12/0.25)	0.17 (0.12/0.20)	0.16 (0.12/0.20)
500	4.8 (2/8)	0.70 (0.21/1.96)	0.54 (0.20/1.22)	0.35 (0.21/0.56)	0.32 (0.20/0.50)
600	5.2 (4/6)	0.96 (0.46/1.41)	0.86 (0.51/1.25)	0.60 (0.42/0.84)	0.53 (0.42/0.72)
700	4.0 (2/6)	0.95 (0.45/1.68)	0.81 (0.44/1.33)	0.82 (0.44/1.25)	0.69 (0.50/0.91)
800	5.2 (4/8)	1.97 (0.67/4.09)	1.68 (0.74/3.33)	1.22 (0.71/2.25)	1.02 (0.72/1.70)
900	3.6 (2/6)	2.56 (0.68/7.15)	2.27 (0.72/6.13)	1.44 (0.71/2.55)	1.19 (0.67/1.87)
1000	6.4 (4/8)	4.07 (1.63/9.02)	3.56 (1.54/7.64)	2.86 (1.57/4.51)	2.06 (1.38/3.18)
1500	4.0 (2/6)	10.6 (2.73/26.1)	7.51 (2.33/13.9)	5.78 (2.35/10.1)	5.24 (2.43/7.77)
2000	6.8 (4/12)	53.8 (7.54/137)	45.5 (7.90/118)	23.3 (7.67/53.9)	19.1 (7.61/40.2)

Table 4.3: Polynomials with random 1000-bit coefficients.

Degree	No. of roots $Avg(Min/Max)$	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	4.0 (4/4)	0.01 (0.00/0.02)	0.01 (0.00/0.02)	0.01 (0.00/0.02)	0.01 (0.00/0.02)
200	3.6 (2/6)	0.06 (0.03/0.12)	0.05 (0.02/0.10)	0.04 (0.02/0.06)	0.03 (0.01/0.06)
300	4.8 (2/8)	0.12 (0.04/0.32)	0.11 (0.04/0.28)	0.10 (0.04/0.23)	0.09 (0.04/0.17)
400	4.4 (2/6)	0.29 (0.06/0.54)	0.25 (0.06/0.44)	0.24 (0.06/0.44)	0.16 (0.06/0.25)
500	5.2 (4/8)	0.68 (0.16/1.20)	0.55 (0.17/0.95)	0.45 (0.21/0.92)	0.32 (0.21/0.48)
600	3.6 (2/4)	0.76 (0.19/2.09)	0.54 (0.18/0.96)	0.43 (0.19/0.66)	0.39 (0.18/0.52)
700	3.6 (0/6)	1.26 (0.25/2.82)	1.28 (0.19/2.51)	0.85 (0.19/1.54)	0.68 (0.19/1.29)
800	4.4 (2/6)	3.03 (0.29/5.50)	2.53 (0.26/4.76)	1.08 (0.34/1.68)	0.93 (0.27/1.53)
900	5.6 (4/8)	4.55 (1.05/9.32)	3.72 (1.02/7.53)	2.23 (1.00/3.09)	1.59 (0.76/2.68)
1000	3.6 (2/6)	2.42 (0.46/4.62)	2.06 (0.44/3.92)	1.27 (0.40/2.00)	1.04 (0.42/1.68)
1500	5.6 (4/8)	16.1 (2.30/40.2)	9.41 (1.99/18.2)	7.17 (2.10/11.9)	5.63 (1.96/10.8)
2000	5.2 (4/6)	23.3 (4.12/79.8)	19.4 (4.08/65.4)	13.2 (4.33/33.4)	10.4 (4.11/20.2)

Table 4.4: Monic polynomials with random 10-bit coefficients.

Degree	No. of roots $Avg(Min/Max)$	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	4.8 (2/8)	0.01 (0.01/0.02)	0.01 (0.00/0.02)	0.01 (0.01/0.02)	0.01 (0.00/0.02)
200	5.6 (4/6)	0.08 (0.03/0.16)	0.06 (0.03/0.13)	0.06 (0.03/0.09)	0.05 (0.03/0.08)
300	4.8 (4/6)	0.12 (0.08/0.22)	0.12 (0.08/0.21)	0.12 (0.08/0.15)	0.10 (0.08/0.15)
400	4.8 (4/6)	0.19 (0.19/0.29)	0.19 (0.16/0.26)	0.18 (0.15/0.26)	0.17 (0.16/0.20)
500	5.2 (4/10)	0.44 (0.18/1.38)	0.42 (0.18/1.19)	0.33 (0.18/0.74)	0.32 (0.19/0.63)
600	5.6 (4/8)	0.99 (0.30/2.04)	0.76 (0.30/1.21)	0.65 (0.31/0.94)	0.49 (0.30/0.72)
700	5.2 (4/8)	1.14 (0.43/1.63)	0.99 (0.42/1.47)	0.92 (0.46/1.30)	0.73 (0.49/0.94)
800	5.6 (4/8)	1.45 (0.66/1.99)	1.29 (0.64/1.62)	1.22 (0.65/1.42)	0.90 (0.69/1.03)
900	4.4 (2/6)	1.01 (0.74/1.18)	1.01 (0.71/1.31)	1.05 (0.69/1.36)	1.09 (0.72/1.33)
1000	5.6 (4/8)	3.40 (1.18/7.09)	3.02 (1.03/5.94)	2.41 (1.10/4.28)	1.72 (1.10/2.70)
1500	6.8 (6/8)	14.8 (6.06/27.3)	11.8 (5.86/17.1)	8.43 (5.98/12.9)	6.80 (4.90/9.24)
2000	7.6 (4/14)	54.8 (6.12/137)	47.6 (6.09/120)	23.9 (6.15/56.0)	19.4 (6.03/42.2)

Table 4.5: Monic polynomials with random 1000-bit coefficients.

Degree	No. of roots $Avg(Min/Max)$	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	6.0 (4/8)	0.03 (0.02/0.04)	0.01 (0.01/0.03)	0.01 (0.00/0.02)	0.01 (0.00/0.02)
200	5.2 (4/8)	0.09 (0.02/0.22)	0.07 (0.02/0.19)	0.04 (0.02/0.11)	0.04 (0.03/0.06)
300	5.6 (4/8)	0.19 (0.06/0.46)	0.14 (0.07/0.28)	0.12 (0.06/0.24)	0.14 (0.07/0.26)
400	5.2 (4/8)	0.41 (0.08/1.00)	0.24 (0.06/0.54)	0.21 (0.06/0.44)	0.15 (0.06/0.28)
500	5.6 (4/8)	0.62 (0.18/1.00)	0.39 (0.12/0.68)	0.45 (0.12/0.74)	0.26 (0.12/0.37)
600	4.8 (4/6)	1.03 (0.24/3.09)	0.52 (0.17/1.21)	0.37 (0.17/0.68)	0.32 (0.17/0.59)
700	5.2 (2/10)	1.27 (0.20/2.67)	1.02 (0.21/1.84)	0.86 (0.19/1.43)	0.65 (0.19/1.09)
800	5.6 (4/8)	2.92 (0.43/5.41)	2.40 (0.39/4.46)	1.02 (0.38/2.02)	0.79 (0.38/1.38)
900	6.0 (4/8)	4.22 (0.84/9.86)	2.67 (0.80/5.78)	1.84 (0.88/2.47)	1.43 (0.74/2.22)
1000	5.6 (4/6)	4.23 (2.21/5.86)	2.90 (1.34/4.21)	2.23 (1.34/3.52)	1.44 (1.15/1.84)
1500	6.8 (6/8)	17.1 (26.06/41.8)	11.4 (5.25/28.2)	8.28 (4.86/15.7)	5.44 (3.30/10.4)
2000	6.4 (6/8)	30.6 (4.80/102)	24.0 (4.59/80.9)	16.7 (4.60/47.4)	12.6 (5.02/35.9)

Table 4.6: Products of terms $x^{20} - r$, with random 20-bit r.

Degree	No. of roots	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	10	0.05 (0.02/0.09)	0.05 (0.02/0.09)	0.03 (0.02/0.04)	0.02 (0.02/0.02)
200	20	0.31 (0.18/0.39)	0.27 (0.16/0.38)	0.24 (0.16/0.32)	0.15 (0.12/0.20)
300	30	1.07 (0.58/1.37)	0.89 (0.60/1.11)	0.87 (0.60/1.04)	0.57 (0.56/0.60)
400	40	2.22 (1.92/2.58)	1.97 (1.86/2.27)	1.94 (1.86/2.08)	1.50 (1.35/1.70)
500	50	8.51 (6.03/11.5)	5.32 (4.24/7.28)	4.55 (4.25/5.32)	3.24 (2.87/3.74)
600	60	13.9 (11.7/17.0)	9.15 (8.28/10.2)	8.96 (8.43/9.35)	6.43 (5.96/6.84)
700	70	24.6 (21.7/29.1)	17.2 (13.7/21.2)	16.5 (13.3/19.7)	12.1 (10.6/14.0)
800	80	38.0 (33.7/44.2)	26.3 (23.6/30.4)	24.4 (19.4/30.3)	17.2 (15.1/19.3)
900	90	53.7 (40.4/63.8)	43.5 (37.0/51.5)	37.0 (28.8/45.8)	29.5 (23.1/36.6)
1000	100	89.6 (70.9/103)	69.1 (52.2/78.5)	63.9 (45.4/76.5)	50.0 (42.1/58.9)
1500	150	577 (468/696)	456 (378/533)	429 (360/473)	353 (3.11/402)
2000	200	2228 (1917/2711)	1907 (1674/2342)	1808 (1614/2279)	1464 (1204/1767)

Table 4.7: Products of terms $x^{20} - r$, with random 1000-bit r.

Degree	No. of roots	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
100	10	0.08 (0.05/0.10)	0.08 (0.05/0.12)	0.11 (0.06/0.31)	0.09 (0.03/0.23)
200	20	1.65 (0.96/2.14)	1.42 (0.97/2.09)	1.28 (1.02/0.1.45)	1.31 (1.10/1.50)
300	30	7.54 (5.08/10.8)	5.20 (4.46/5.65)	4.88 (3.67/5.49)	4.24 (3.92/4.69)
400	40	15.7 (10.8/19.7)	15.7 (13.3/17.5)	14.7 (12.7/17.3)	12.7 (11.0/14.1)
500	50	42.4 (29.2/64.7)	44.5 (35.2/48.7)	35.5 (32.8/40.5)	35.0 (27.5/49.7)
600	60	117 (91.9/154)	106 (82.6/134)	103 (90.0/121)	92.0 (86.5/97.0)
700	70	248 (208/332)	252 (221/282)	240 (205/264)	189 (168/205)
800	80	549 (351/753)	481 (410/590)	474 (412/542)	382 (364/432)
900	90	1138 (971/1271)	855 (721/967)	834 (718/931)	670 (646/723)
1000	100	1661 (1513/1913)	1335 (1123/1673)	1265 (1066/1440)	1065 (947/1146)
1500	150	9004 (8233/9705)	8360 (7281/8999)	8230 (7357/9652)	6141 (5659/6470)

Table 4.8: Products of terms $x - r$ with random integer r.

Bit-length of roots	Degree	ub_C $Time(s)$ $Avg(Min/Max)$	$min(ub_{FL}, ub_{LM})$ $Time(s)$ $Avg(Min/Max)$	$ub_{KQ}(Hong)$ $Time(s)$ $Avg(Min/Max)$	ub_{LMQ} $Time(s)$ $Avg(Min/Max)$
10	100	0.46 (0.28/0.94)	0.24 (0.18/0.28)	0.34 (0.27/0.41)	0.34 (0.30/0.41)
10	200	1.46 (1.24/1.85)	1.40 (1.28/1.69)	1.41 (1.26/1.71)	1.40 (1.20/1.69)
10	500	18.1 (16.5/18.9)	18.1 (16.6/18.8)	21.2 (17.5/24.4)	22.1 (18.7/24.2)
1000	20	0.07 (0.04/0.14)	0.02 (0.02/0.03)	0.03 (0.02/0.04)	0.03 (0.02/0.04)
1000	50	3.69 (2.38/6.26)	0.81 (0.60/1.28)	0.88 (0.52/1.28)	0.81 (0.52/1.11)
1000	100	47.8 (37.6/56.9)	13.8 (10.3/19.2)	17.6 (12.4/25.9)	15.8 (11.3/21.3)

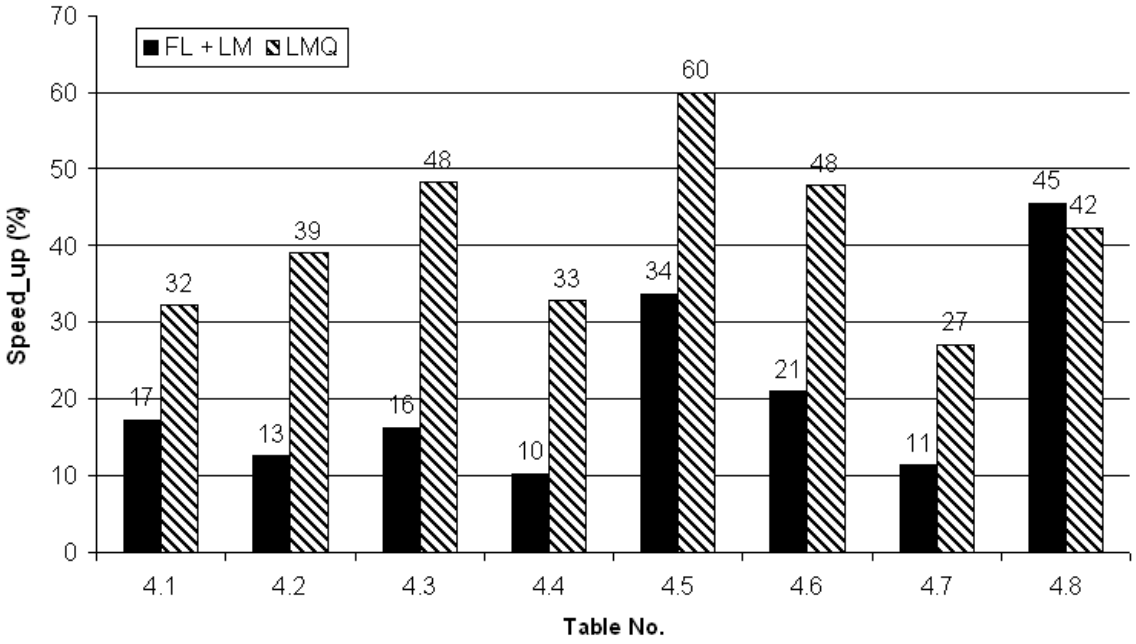

Figure 4.2: The average $speed - up$ of the `VAS` algorithm for each Table (4.1–4.8) using the $min(ub_{FL}, ub_{LM})$ and ub_{LMQ} against Cauchy's bound, ub_C.

Summarizing the testing results[5] from Tables 4.1 through 4.8 above we have the Figure 4.2. The time gain for $min(ub_{FL}, ub_{LM})$ and ub_{LMQ} against Cauchy's bound, ub_C, was calculated for every row using the formulas: $Speed - up = 100 \cdot |min(ub_{FL}, ub_{LM}) - ub_C|/ub_C$ and $Speed - up = 100 \cdot |ub_{LMQ} - ub_C|/ub_C$, respectively. Then, for every Table, the average value of $Speed - up$ was computed giving a rough overall estimation on time gain that `VAS` algorithm received after the incorporation of the new bounds.

Taking these test results into consideration, one could safely conclude that using the new proposed bounds (linear or quadratic) in `VAS` real root isolation algorithm would see a average overall improvement in computation time of about 20% for $min(ub_{FL}, ub_{LM})$ and about 40% for ub_{LMQ} bound.

Also, notice that ub_{LMQ} is fastest for all classes of polynomials tested, except for the case of very many very large roots, Table 4.8. In the case of very many very large roots `VAS` using ub_{LMQ} is a very close second to `VAS` using our linear complexity bound $min(ub_{FL}, ub_{LM})$[6].

We end this chapter by presenting the following graph, Figure 4.3. This graph depicts the overall time of `VAS-CF` in comparison with the time spent for computing bounds. Especially, the left scale shows the total time in seconds (bars) needed by `VAS-CF` to isolate the roots of a certain class of polynomials (Laguerre) using both ub_{LM}, the ***"local-max"*** bound, and ub_{LMQ}, its quadratic version. The right scale is associated with the two curves which show the total time spent by `VAS-CF` in computing just these bounds.

[5]The change in memory use was negligible in every case, and hence, it is not included. Also, timing results are subject to measurement error, which especially affect small timings.

[6]For additional discussion on these conclusions, see (*Akritas, Strzeboński, and Vigklas*, 2007), (*Akritas, Strzeboński, and Vigklas*, 2008b) and (*Akritas*, 2009).

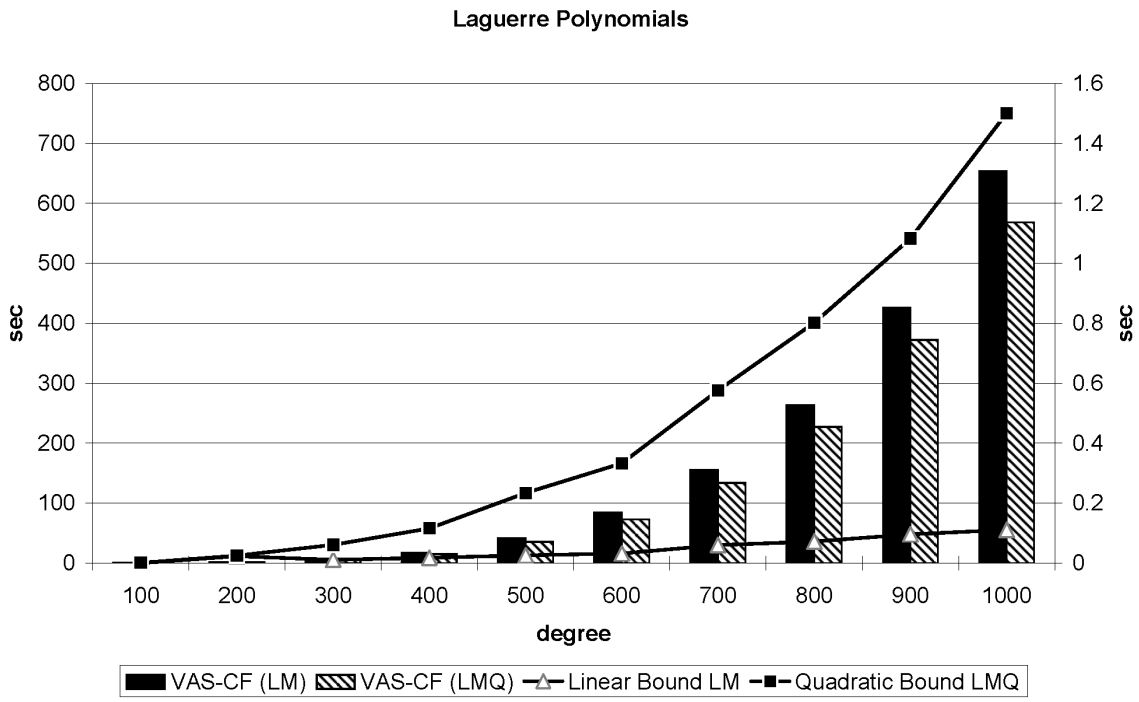

Figure 4.3: Computation times for the Laguerre polynomials of degree (100...1000). The `VAS-CF(LM)`, `VAS-CF(LMQ)`, `(LM)`, and `(LMQ)` are described above in the text. Note that the bars are scaled to the left Y axis whereas the lines to the right one.

CHAPTER V

Conclusions

5.1 Final Note

This thesis was motivated by an old, but still important for many modern scientific fields, problem in polynomial algebra, namely, the determination of upper bounds on the values of the positive roots of polynomials. The widespread development and use of computer algebra systems, (CAS), along with an increasing interest in root isolation methods, provided a fruitful context for reexamination of this classical problem.

We have presented and analyzed a variety of algorithms both of linear and quadratic computational complexity that take advantage of a theorem that we extended, in order to establish a unified general framework in which the classical ones fitted perfectly and new ones came out naturally. These algorithms are simple in implementation, and in most cases outperform in speed and accuracy the established preexisting methods.

The incorporation of these new bounds in the Vincent-Akritas-Strzeboński, (`VAS`), continued fractions polynomial real root isolation algorithm offered a significant speed-up to an already fast method extending significant the range of its applicability and its robustness.

The immediate adoption of our new algorithms by major mathematical software systems, such as *Mathematica* and *Sage*, bear witness to their usefulness and their effectiveness. However, this thesis is by no means exhaustive and questions such as:

- Is there an optimal way to break up the coefficients in the"*first–λ*" method?
- Is it possible to extend these bounds to multivariate polynomials with similar success?
- Can these bounds be suitably modified to constitute a complex analogue for polynomials with complex coefficients?

and many others, seek to be answered by our long term on-going research.

APPENDICES

APPENDIX A

Theorems on the Number of Real Roots of a Polynomial

A.1 Number of Real Roots of a Polynomial in an Interval

After the bounds of the positive and negative real roots of the polynomial equation $p(x) = 0$ have been calculated according to the methods presented in Chapter 3.2 the next question that arises concerns the number of real roots of a polynomial in a given interval (a, b). A picture of the number of real roots of equation $p(x) = 0$ in an interval (a, b) is shown in Figure A.1, for the function $y = p(x)$, where the roots x_1, x_2, x_3 are found as the points of intersection of the graph with the x-axis. We note that (a) If $p(a)p(b) < 0$, then on the interval (a, b) there is an odd number of roots of $p(x)$, counting multiplicities, (b) If $p(a)p(b) > 0$, then on the interval (a, b) there are either no roots of $p(x)$ or there is an even number of such roots. The question of the number of real roots of an algebraic equation in a given interval is solved completely by the Sturm method, (*Kurosh*, 1988). Before going into that let us introduce the notion of the number of sign changes in a set of numbers.

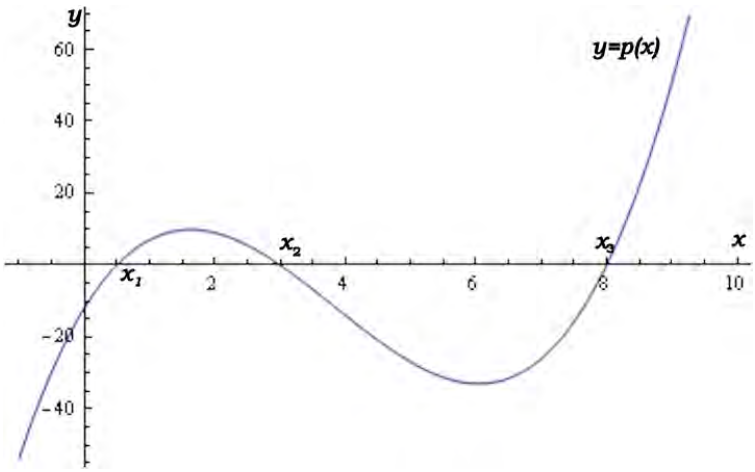

Figure A.1: A polynomial with three positive real roots.

Definition A.1. Suppose we have an ordered finite set of real numbers different from zero:

$$r_1, r_2, ...r_l \quad (l \geq 2) \tag{A.1}$$

We say that there is a change of sign for a pair of two successive elements r_k, r_{k+1} of (A.1) if these elements have opposite signs, that is,

$$r_k r_{k+1} < 0 \tag{A.2}$$

and there is no change of sign if the signs are the same:

$$r_k r_{k+1} > 0 \tag{A.3}$$

The total number of changes of sign in all pairs of successive elements $r_k, r_{k+1} (k = 1, 2, ...l-1)$ of (A.1) is called the number of sign changes or variations of sign, (Var), in (A.1).

Example A.2. Consider the polynomial $p(x) = x^3 - 7x^2 + 7$. The sequence of its coefficients is $\{1, -7, 0, 7\}$ which shows a number of sign variations equal 2, $(Var = 2)$.

A.1.1 Sturm's Theorem (1827)

For a given polynomial $p(x)$, we can form the *Sturm sequence*

$$p(x), p_1(x), p_2(x), ..., p_m(x) \tag{A.4}$$

where $p_1(x) = p'(x)$, $p_2(x)$ is the remainder, with reversed sign, left after the division of the polynomial $p(x)$ by $p_1(x)$, $p_3(x)$ is the remainder, with reversed sign, after the division of the polynomial $p_1(x)$ by $p_2(x)$, and so on. The polynomials $p_k(x)(k = 2, ..., m)$ may be computed by a modified Euclidean algorithm. If the polynomial $p(x)$ does not have any multiple roots, then the last element $p_m(x)$ in the Sturm sequence is a nonzero real number.

If we represent by $Var(r)$ the number of sign changes in a Sturm sequence for $x = r$, provided that the zero elements of the sequence have been ignored, we have the Sturm's theorem:

Theorem A.3 (Sturm (1827))**.** *If a polynomial $p(x)$ does not have multiple roots and $p(a) \neq 0$, $p(b) \neq 0$, then the number of its real roots $N(a, b)$ in the interval $a < x < b$ is exactly equal to the number of lost sign changes in the Sturm sequence of the polynomial $p(x)$ when going from $x = a$ to $x = b$, that is,*

$$N(a, b) = Var(a) - Var(b) \tag{A.5}$$

Corollary A.4. *If $p(0) \neq 0$, then the number N^+ of positive and the number N^- of negative roots of the polynomial $p(x)$ are respectively:*

$$N^+ = Var(0) - Var(+\infty) \tag{A.6a}$$

$$N^- = Var(-\infty) - Var(0) \tag{A.6b}$$

Corollary A.5. *For all the roots of a polynomial $p(x)$ of degree n to be real, in the absence of multiple roots, it is necessary and sufficient that the following condition holds:*

$$Var(-\infty) - Var(+\infty) = n \tag{A.7}$$

Example A.6. Let us determine the number of positive and negative roots of the equation $p(x) = x^7 - 7x + 1$. The Sturm sequence is

$$p(x) = x^7 - 7x + 1 \tag{A.8a}$$

$$p_1(x) = x^6 - 1 \tag{A.8b}$$

$$p_2(x) = 6x - 1 \tag{A.8c}$$

$$p_3(x) = 46655 \tag{A.8d}$$

so

$$Var(-\infty) = 3, \quad Var(0) = 2, \quad Var(+\infty) = 0 \tag{A.9}$$

We find that equation $p(x) = x^7 - 7x + 1$ has

$$N^+ = 2 - 0 = 2 \quad \text{positive roots} \tag{A.10a}$$

$$N^- = 3 - 2 = 1 \quad \text{negative roots} \tag{A.10b}$$

and its rest four roots are complex roots. We can easily deduce here a way to isolate the roots of algebraic equations by using Sturm sequence in order to partition the interval (a, b) containing all the real roots of the equation into a finite number of subintervals (α, β) such that $Var(\alpha) - Var(\beta) = 1$.

A.1.2 Fourier's Theorem (1819)

Sturm arrived at his method (1827) by extending an earlier theorem by Fourier (1819). Let us return to the method described above (A.1) for the counting of the number variations of sign in a sequence of numbers:

Definition A.7. Suppose we have a finite ordered sequence of real numbers:

$$r_1, r_2, ..., r_l \tag{A.11}$$

where $r_1 \neq 0$ and $r_l \neq 0$. We define: (a) *lower number of variations* of sign Var_{lo} of the sequence (A.11) for the number of sign changes in an appropriate subsequence that does not contain zero elements and (b) *upper number of variations* of sign Var_{up} of a sequence of numbers (A.11) for the number of sign changes in a transformed sequence of (A.11) where the zero elements

$$r_k = r_{k+1} = ... = r_{k+m-1} = 0 \tag{A.12}$$

$(r_{k-1} \neq 0, r_{k+m} \neq 0)$ are replaced by the elements $\tilde{r}_{k+i} (i = 0, 1, 2, ..., m-1)$ such that

$$sgn(\tilde{r}_{k+i}) = (-1)^{m-i} sgn(r_{k+m}) \tag{A.13}$$

It is clear that if (A.11) has no zero elements, then the number Var of sign changes in the sequence coincides with its lower Var_{lo} and upper Var_{up} number of variation of sign: $Var = Var_{lo} = Var_{up}$ whereas generally $Var_{up} \geq Var_{lo}$.

Example A.8. Let us determine the lower and upper number of changes of sign in the sequence $1, 0, 0, -1, 1$. Omitting the zeros, we have $Var_{lo} = 2$. To calculate Var_{up} using (A.13), we form the sequence $1, -\sigma, \sigma, -1, 1$ where $\sigma > 0$, and $Var_{up} = 4$.

Theorem A.9 (Fourier, 1820)**.** *If the numbers a and b $(a < b)$ are not roots of a polynomial $p(x)$ of degree n, then the number $N(a, b)$ of real roots of the equation*

$p(x) = 0$ *lying between* a *and* b *is equal to the minimal number* ΔVar *of the sign changes lost in the sequence of successive derivatives*

$$p(x), p'(x), ..., p^{n-1}(x), p^n(x) \tag{A.14}$$

when going from $x = a$ *to* $x = b$*, or less that* ΔVar *by an even number:* $N(a, b) = \Delta Var - 2k$ *where* $\Delta Var = Var_{lo}(a) - Var_{up}(b)$ *and* $Var_{lo}(a)$ *is the lower number of variations of sign in the sequence* (A.14) *for* $x = a$*,* $Var_{up}(b)$ *is the upper number of variations of sign in that sequence for* $x = b \quad [k = 0, 1, ..., E(\frac{\Delta Var}{2})]$

Fourier's theorem is the only one found in the literature under the names: Budan, Budan-Fourier, Fourier or Fourier-Budan. We expalin why in the sequel. In the above theorem it is assumed that each root of the equation $p(x) = 0$ is counted according to its multiplicity. If the derivatives $p^k(x) \quad (k = 1, 2, ..., n)$ do not vanish at $x = a$ and $x = b$, then counting the signs is simplified and ΔVar becomes $\Delta Var = Var(a) - Var(b)$.

Corollary A.10. . *If* $\Delta Var = 0$*, then there are no real roots of the equation* $p(x) = 0$ *between* a *and* b*.*

Corollary A.11. . *If* $\Delta Var = 1$*, then there is exactly one real root of the equation* $p(x) = 0$ *between* a *and* b*.*

A.1.3 Descartes' Theorem (1637)

Somewhat easier in applications, but still unable to determine precisely the number of roots, is Descartes' rule of signs (given in his work Geometrie in 1637 and proved by Gauss in 1828), (*Bartolozzi and Franci*, 1993).

Theorem A.12 (Descartes' rule of signs, 1637)**.** *The number of positive roots of an algebraic equation* $p(x) = 0$ *such that a root of multiplicity* m *being counted as* m *roots, is equal to the nummber of variations in sign in the sequence of coefficients*

$$a_n, a_{n-1}, a_{n-2}, ..., a_0 \tag{A.15}$$

(where the coefficients equal to zero are not counted) or less than that by an even integer.

Clearly, Descartes' rule of sign is an application of the Fourier theorem to the interval $(0, +\infty)$. Since

$$p(x) = a_n x^n + ... + a_0 \tag{A.16a}$$

$$p'(x) = n a_n x^{n-1} + ... + a_1 \tag{A.16b}$$

$$p^{(2)}(x) = n(n-1) a_n x^{n-2} + ... + 2a_2 \tag{A.16c}$$

$$... \tag{A.16d}$$

$$... \tag{A.16e}$$

$$... \tag{A.16f}$$

$$p^{(n)}(x) = n! a_n \tag{A.16g}$$

sequence (A.15) is, to within positive factors, a collection of derivatives $p^{(k)}(0) \quad (k = 0, 1, 2..., n)$ written in ascending order, i.e. $a_0, a_1, 2a_2, ..., n!a_n$, therefore, the number of variations in sign in the sequence (A.15) is equal to $Var_{lo}(0)$, zero coefficients not counted. On the other hand, the derivatives $p^{(k)}(+\infty) \quad (k = 0, 1, 2..., n)$ have no sign variations and follows that $Var_{up}(+\infty) = 0$. Then, we have

$$\Delta Var = Var_{lo}(0) - Var_{up}(+\infty) = Var_{lo}(0) \tag{A.17}$$

and on the basis of the Fourier theorem, the number of positive roots of $p(x) = 0$ is either equal to ΔVar or is less than that by an even integer.

Corollary A.13. *If the coefficients of $p(x) = 0$ are different from zero, then the*

number of negative roots of $p(x) = 0$*, counting multiplicities, is equal to the number of non-variations of sign in the sequence* (A.15) *of its coefficients or is less than that by an even integer. The proof of this follows directly from the application of Descartes' rule to the polynomial* $p(-x)$*.*

A.1.4 Budan's Theorem (1807)

Budan's theorem of 1807 is equivalent to, but not the same, as, Fourier's theorem. Due to this equivalence, it was not considered essential and therefore, it was only Fourier's theorem that gained popularity among researchers. Budan's theorem appears in the literature *only* in Vincent's paper, (*Vincent*, 1836) and in Akritas' work, (*Akritas*, 1982). Budan's theorem despite its similarity to Fourier's, leads in a different direction. It states:

Theorem A.14 (Budan, 1807)**.** *If in an algebraic equation* $p(x) = 0$*, we make two distinct substitutions* $x = \alpha + x'$ *and* $x = \beta + x''$*, where* α *and* β *are real numbers and* $\alpha < \beta$*, getting the equations* $A(x') = \sum a_i x^i = 0$ *and* $B(x'') = \sum b_i x^i = 0$*. Then*

- $Var(a_i) \geq Var(b_i)$
- *The number of real roots of* $p(x) = 0$ *between* α *and* β *is:* $N(\alpha, \beta) = Var(a_i) - Var(b_i) - 2k$*, where as above* k *is an integer and* $k \geq 0$*.*

To see that this theorem is equivalent to Fourier's theorem we must replace, in Fourier's sequence, x by any real number c. Then, the $n + 1$ resulting numbers are proportional to the corresponding coefficients of the transformed polynomial equation $p(x + c) = \sum_{0 \leq i \leq n} [p^{(i)}(c)/i!]x^i$ obtained by Taylor's expansion theorem. Budan's theorem is the base of Vincent's theorem that plays an important role to the real root isolation algorithms.

APPENDIX B

Mathematical Formulas of Testing Polynomials

B.1 Mathematical Formulas of the Benchmark Polynomials

Below we present the exact mathematical formulas of the polynomials that were used for the tests, during the computational evaluation of the various bounds. We followed the standard practice and used as benchmark the *Laguerre* polynomials recursively defined as:

$$L_0(x) = 1 \tag{B.1a}$$

$$L_1(x) = 1 - x \tag{B.1b}$$

$$L_{n+1}(x) = \frac{1}{n+1}((2n+1-x)L_n(x) - nL_{n-1}(x)) \tag{B.1c}$$

ChebyshevI of the first kind recursively defined as:

$$T_0(x) = 1 \tag{B.2a}$$

$$T_1(x) = x \tag{B.2b}$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{B.2c}$$

ChebyshevII of the second kind recursively defined as:

$$U_0(x) = 1 \tag{B.3a}$$

$$U_1(x) = 2x \tag{B.3b}$$

$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x) \tag{B.3c}$$

Wilkinson recursively defined as:

$$W(x) = \prod_{i=1}^{n}(x - i) \tag{B.4a}$$

Mignotte recursively defined as:

$$M_n(x) = x^n - 2(5x - 1)^2 \tag{B.5a}$$

as well as several types of randomly generated polynomials of degrees $\{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000\}$. For the random polynomials the size of the coefficients ranges from -2^{20} to 2^{20}. *"uRandom"* indicates a random polynomial whose leading coefficient is one, whereas ' *"sRandom"* indicates a random polynomial obtained with the randomly chosen seed 1001; also *"pRandom"* denotes products of factors (x-randomly generated integer root $p(x) = \prod_{degree}(x - n)$).

BIBLIOGRAPHY

BIBLIOGRAPHY

Akritas, A. G. (1982), Reflections on a pair of theorems by budan and fourier, *Mathematics Magazine*, *55*(5), 292–298.

Akritas, A. G. (2009), Linear and quadratic complexity bounds on the values of the positive roots of polynomials, *Journal of Universal Computer Science*, *15*(3), 523–537.

Akritas, A. G. (2010), Vincent's theorem of 1836: overview and future research, *Journal of Mathematical Sciences*, *168*(3), 309–325.

Akritas, A. G., and A. Strzeboński (2005), A comparative study of two real root isolation methods, *Nonlinear Analysis, Modelling and Control*, *10*(4), 297–304.

Akritas, A. G., and P. Vigklas (2006), Comparison of various methods for computing bounds for positive roots of polynomials, *Abstracts of the 12th International Conference on Applications of Computer Algebra, ACA*, p. 32.

Akritas, A. G., and P. Vigklas (2007), A comparison of various methods for computing bounds for positive roots of polynomials, *Journal of Universal Computer Science*, *13*(4), 455–467.

Akritas, A. G., A. Strzeboński, and P. Vigklas (2006), Implementations of a new theorem for computing bounds for positive roots of polynomials, *Computing*, *78*, 355–367.

Akritas, A. G., A. Strzeboński, and P. Vigklas (2007), Advances on the continued fractions method using better estimations of positive root bounds, *Proceedings of the 10th International Workshop on Computer Algebra in Scientific Computing, CASC*, pp. 24–30.

Akritas, A. G., A. Strzeboński, and P. Vigklas (2008a), Quadratic complexity bounds on the values of positive roots of polynomials, *Abstracts of the International Conference on Polynomial Computer Algebra, PCA 2008*, p. 6.

Akritas, A. G., A. Strzeboński, and P. Vigklas (2008b), Improving the performance of the continued fractions method using new bounds of positive roots, *Nonlinear Analysis Modelling and Control*, *13*(3), 265–279.

Bartolozzi, M., and R. Franci (1993), La regola dei segni dall enunciato di r. descartes (1637) alla dimostrazione di c.f. gauss (1828)., *Archive for History of Exact Sciences*, *45*(4), 335–374.

Ştefănescu, D. (2005), New bounds for positive roots of polynomials, *Journal of Universal Computer Science*, *11*(12), 2132–2141.

Ştefănescu, D. (2007), Bounds for real roots and applications to orthogonal polynomials, in *Proceedings of Computer Algebra in Scientific Computing, 10th International Workshop*, *Lecture Notes in Computer Science, CASC 2007*, vol. 4770, edited by M. W. E. Ganzha G. V. and V. V. E., pp. 377–391, Springer, Bonn, Germany.

Hoeven, J., G. Lecerf, B. Mourrain, and O. Ruatta (2008), Mathemagix computer algebra system,: Version 0.2,
URL `http://www.mathemagix.org/`.

Hong, H. (1998), Bounds for absolute positiveness of multivariate polynomials, *Journal of symbolic Computation*, *25*(5), 571–585.

Kioustelidis, B. (1986), Bounds for positive roots of polynomials, *Journal of Computational and Applied Mathematics*, *16*(2), 241–244.

Kurosh, A. (1988), *Higher Algebra*, Mir Publishers, Moscow.

Mignotte, M. (1992), *Mathematics for Computer Algebra*, Springer-Verlag, New York.

Obreschkoff, N. (1963), *Verteilung und Berechnung der Nullstellen reeller Polynome*, VEB Deutscher Verlag der Wissenschaften, Berlin.

SAGE (2004–2010), Sage, open-source mathematics software, version 4.6,
URL `http://www.sagemath.org/`.

Strzeboński, A. (2010), Wolfram research, inc. personal communication. 23 november.

Tsigaridas, P. E., and Z. I. Emiris (2006), Univariate polynomial real root isolation: continued fractions revisited, in *Proceedings of the 14th conference on Annual European Symposium - Volume 14, (ESA, LNCS 4168)*, edited by Y. Azar and T. Erlebach, pp. 817–828, Springer-Verlag, London, UK.

Vincent, A. J. H. (1836), Sur la resolution des équations numériques, *Journal de Mathématiques Pures et Appliquées*, *1*, 341–372.

von zur Gathen, J., and J. Gerhard (1997), Fast algorithms for taylor shifts and certain difference equations, in *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pp. 40–47, ACM, New York, NY, USA.

Wolfram Research, I. (2008), Mathematica edition: Version 7.0,
URL `http://www.wolfram.com/`.

Yap, C. K. (2000), *Fundamental Problems of Algorithmic Algebra*, Oxford University Press, Oxford.