Instituto Nacional de Matemática Pura e Aplicada

D.Sc. thesis

Methods for bounding and isolating the real roots of univariate polynomials

Eric Javier Biagioli

Supervised by

Dr. Roberto Imbuzeiro Oliveira (IMPA)
Dr. Luis Peñaranda (UFRJ)

Thursday, March 3, 2016 9:30 AM

Abstract

This work addresses the problem of finding the real roots of univariate polynomials, covering two related subareas. One of them is related to the problem of computing an upper (lower) bound for the value of the maximum (minimum) positive root of a univariate polynomial. In this part, we present a detailed state-of-the art survey and we propose a method that improves the quality of the bounds produced by other existing methods, without introducing significative extra amount of computational effort. The other part is related to the problem of *isolating the positive roots of a univariate polynomial* P (i.e.: computing a set S of disjoint intervals, each one of them containing exactly one positive root of P, all positive roots of P being contained by some interval in S). In this part, the main results related to the problem are presented, explained and compared; and two results are introduced: one adaptation of one of the underlying theorems (Fourier's theorem), which requires less amount of computational effort in the cases in which P is a *fewnomial* (i.e.: a sparse polynomial, a polynomial in which the degree is much higher than the number of terms), and is also introduced one method for root isolation which, while relying only on elementary and intuitive results, proves to have good performance in most of the testcases. In addition to our theoretical approach, implementations and extensive tests of our methods are presented. Along with these two mentioned results, we also expose two ideas that, although we have not been able to obtain concrete results from them, we find them promising. One of them is related to the Sturm idea, and the other is related to Fourier's theorem.

Acknowledgements

I want to express my deepest gratitude to my advisors Luis Peñaranda and Roberto Imbuzeiro Oliveira. For innumerous reasons. For their comments and moral support, which were essential to me. For the multiple discussions, for the multiple ideas, and for being there all the time. I am deeply grateful to them. It has been a honor for me to work with them. Without them, this thesis would not have been possible.

I want to also thank Prof. Luiz Velho, head of the VisGraf laboratory, for letting me be part of the Lab. I greatly appreciated such a stimulating work environment.

I want also to thank my colleagues and friends.

Contents

Chapter 1

Introduction

The solution of numerical equations has been a focus of interest for algebraists and geometers since the origins of algebra up to the present. It is one of the oldest and most studied problems in Mathematics. Starting with ancient civilizations like the Babylonians, and for centuries, mathematicians have looked for ways to compute the roots of polynomials from the numerical values of its coefficients. Much more modern results (i.e.: Abel, 1802–1829) [4, 5] proved that there is no general algebraic solution for the roots of a quintic equation, or any general polynomial equation of degree greater than four, in terms of explicit algebraic operations. Lagrange (1736–1813) did suspect this, but he was not capable of proving his suspicion.

Lagrange published, in 1798, an extensive book on the problem of finding roots of *numerical equations* [72]. As it can be seen in the next quotation, a *numerical* equation is an equation of the form $p(x) = 0$, where p is a polynomial in which the coefficients are *numbers*, not symbols. An equation in which the coefficients are symbols is called an *algebraic* equation. Lagrange saw the subject of computing solutions of equations as divided in two parts, concerning either numerical or algebraic equations:

> *La solution de tout problème déterminé se réduit, en dernière analyse, à la résolution d'une ou de plusieurs équations, dont les coefficiens sont donnés en nombres, et qu'on peut appeler* équations numériques. *Il est donc important d'avoir des méthodes pour résoudre complètement ces équations, de quelque degré qu'elles soient. . . .*
>
> *Il faut bien distinguer la résolution des équations numériques de ce qu'on appelle en Algèbre la résolution générale des équations. La première est,*

à proprement parler, une opération arithmétique, fondée à la vérité sur les principes généraux de la théorie des équations, mais dont les résultats ne sont que des nombres, où l'on ne reconnaît plus les premiers nombres qui ont servi d'éléments, et qui ne conservent aucune trace des différentes opérations particulières qui les ont produits. L'extraction des racines carrées et cubiques est l'opération la plus simple de ce genre; c'est la résolution des équations numériques du second es du troisième degré, dans lesquelles tous les termes intermédiaires manquent. [72]

This book of Lagrange was published in 1798 and revised in 1808. It is, essentially, a reproduction in six chapters of two extensive previous papers of him, from 1769/1770; with voluminous additional notes. The book contains methods for finding all the real roots of *numerical* equations.

With respect to algebraic equations, and also by 1770, Lagrange published his work *Reflexions sur la Resolution Algébrique des Equations* [71], in which he tried to extend existing work on the old problem of solving algebraic equations by formulas involving radicals, and expressed and supported his view that the situation beyond degree four looked unpromising. This latter manuscript represents an important milestone in setting the stage for the later work of Abel and Galois, whose new methods and discoveries confirmed the truth of Lagrange's pessimistic assessment about algebraic solutions.

The work of Lagrange set the initial from which the problem started to evolve much faster than previously. In chapter 2 we show a historical line starting at him.

1.1 The problem

This thesis is about the solution of numerical equations, in the modern sense of the phrase. The current approaches (and, in fact, ancient approaches too) to compute the solutions for a numerical equation involves two steps: *isolate the solutions* and *refine the isolation*. The problem of *isolating* the roots of a given polynomial p is the problem of computing a set of disjoint intervals, each containing *exactly one* real root of p, which together contain all roots. The problem of *refining* an isolating interval is the problem in which we have an interval I that contains exactly one root of p and we want to compute a narrower interval, contained in I, which also contains the root.

In modern bibliographies [13, 32, 85, 87], the problem of *isolating the real roots of* $p(x)$ is sometimes referred to as *solving the polynomial* $p(x)$.

In this thesis we analyze the problem of isolating the real roots of polynomials. When analyzing this problem, there are some sub-problems that appear in a natural way; among them, is the problem of giving lower and upper bounds for the positive roots of a polynomial. Nowadays, the most efficient methods for root isolation can be significatively improved through improving the root bounding algorithms that they use as auxiliary tools.

1.2 Contributions

The main contributions on this thesis are:

Root bounding problem:

- We propose a unified way to understand almost all of the current approaches.
- Based on this insight we propose, implement and extensively test a new method, on many significant scenarios, obtaining a technique that yields significant improvements in the results while not introducing a significant time penalty.

Root isolation problem:

- An adaptation of a theorem stated by Fourier, which is better for sparse polynomials (often referred to as *fewnomials*).
- A method which relies on elementary and intuitive concepts, and shows quite acceptable performance in most cases; resulting faster than Sturm and only improved by the fastest approach, the VAS, which is more complicated from a conceptual point of view.

Application:

- An application of the approaches for the root isolation problem, which allows, in the area of simulation of physical systems, to numerically solve a new family of situations, improving significatively the quality of the numerical results obtained in some simulations.

Expository:

- An ordered text which collects the main results and algorithms of the area, all of them explained with elementary arguments. Although almost all of the results can be explained with quite elementary concepts, there exist some places in current literature where we can find quite complicated explanations.

1.3 Outline of the chapters

Chapter 2: Historical overview. We give an historical but not exhaustive overview of the problem. This chapter focuses on more ancient results and ideas; modern ideas will be visited later.

Chapter 3: Root bounding. We survey existing approaches for the problem, and propose a unified framework to see all of them in the same way. We propose a new technique that improves the current methods without introducing significative computational effort.

Chapter 4: Root isolation. We show an adaptation of the theorem of Fourier; we survey existing methods for the root isolation problem, proposing elementary explanations for all of them, and we propose a method which, while relying on elementary and intuitive concepts, shows to be quite efficient for some particular input cases.

Chapter 5: One application in simulation of physical systems. In this chapter, whose main ideas were developed in collaboration with Dr. Federico Bergero (CIFASIS, Rosario, Argentina), the algorithms mentioned in chapter 4 are adapted to be efficient for isolating the minimum positive root of a given polynomial, not necessarily sparse; and these adaptations are inserted as auxiliary tools inside the current simulation methods. Due to the lack of analytic solutions for systems of high degree, these simulation methods focus on systems of order 4 or less. However, it is not the analytic solution that is needed, but the numerical one. With our adaptations, we show how a family of higher–order systems become integrable.

Chapter 2

Historical overview

This chapter shows a historical survey of the problem of computing numerical solutions of univariate polynomials, stating and explaining the most important results, chronologically. From Descartes up to nowadays, giving more importance to ancient results, since we will revisit modern ones in the next chapter of this thesis. As it will be shown, the problem has evolved a lot. Currently there exists research effort in many sub-problems of that initial problem, all of them with significative impact in different areas. Among them we can point, for example, the problem of *bounding* the real roots of a univariate polynomial, the problem of *counting* the number of real roots of a univariate polynomial in a given interval, the problem of bounding the size of the separation between any two roots of a univariate polynomial (known as *repulsion*), the problem of *isolating* the real roots of a univariate polynomial, the problem of *refining* an interval containing exactly one real root of a univariate polynomial. All of these problems are, of course, related. All of them appear in a natural way when trying to solve the first mentioned problem: computing the real roots of univariate polynomials. The subsequent chapters analyze some of these sub-problems.

Contributions in this chapter The main goal of this chapter is to give a chronological survey of the main contributions to the problem of computing the real roots of a univariate polynomial and its related sub-problems. It begins with a brief timeline, putting the main results in historical perspective and, after that, it shows them in more detail. In some cases, additional interpretations are given, with the intention of simplifying the reading and understanding.

2.1 Historical review

In 1637, in the appendix *La géometrie* of his *Discours de la méthode*, Descartes formulated without any proof, the oldest and by far the most famous theorem related to counting real roots of univariate polynomials: Descartes' rule of signs (see the original statement on page 11). It gives an upper bound for the number of real positive roots that a polynomial p can have, considering only the signs of its coefficients.

In 1798, Lagrange proposed, in his *De la résolution des equations numériques de tous les degrés* [72] (with most of its content republished in 1826, in his *Traité de la résolution des equations numériques de tous les degrés* [73]), the first systematic method to address the problem of computing the real roots of a univariate polynomial. It consists in evaluating the input polynomial p at an increasing sequence of numbers, starting at a lower bound of the roots of p, and ending at an upper bound of them. The values at that sequence must be chosen in a way that ensures that p cannot have more than one root between any two consecutive elements of it. Looking the changes of sign at the results of these evaluations, this procedure can tell which intervals have exactly one root. The condition that p cannot have more than one root between any two consecutive elements of the sequence is fulfilled by taking as sequence an arithmetic progression whose ratio Δ is a lower bound on the distances between any two roots of p. Lagrange proposed four ways to compute a valid Δ, with and without explicit computation of the auxiliary *equation of differences*, whose roots are the differences between all ordered pairs of distinct roots of p.

Although this method was the first *algorithm* and has strong theoretical importance, it was not very successful in practice due to large amount of computational effort that it required. This aspect of the method was strongly criticized by Fourier, who stated that it was *highly impractical* [19, 17].

In 1800, in his *Nouvelle méthode pour la résolution des équations numériques d'un degré quelconque* [25], Budan stated a theorem (see page 15) for computing an upper bound on the number of real roots a polynomial has in an open interval by counting the number of sign changes in the sequences of its coefficients. This book was republished (under the same name) in 1807.

In 1820, Fourier [47, 46] stated a theorem from which it was possible to conclude as corollary the Descartes' rule of signs (stated without any proof, as we already said, by

Descartes in 1637, in his *Geometry*). This theorem has been shown to be equivalent to the one stated by Budan 20 years before. After Fourier's reformulation of the theorem, the version of Budan was almost completely replaced in the literature by Fourier's version and the latter has been referred to under various names, including Budan's. This can be seen, for example, in Prasolov's book [82, page 27], where the theorem is stated in Fourier's formulation and named Budan–Fourier.

In 1829, based on the proof that Fourier gave to his version of Budan's theorem, following a reasoning that mimics Fourier's one, Sturm [90, 91] proposed a method that allows to count *exactly* the number of roots in some interval (a, b). It does not produce *a bound* on the number of root of the polynomial: it actually computes the *exact number of roots that the polynomial p has in the given interval*. This method is one of the most reliable methods and is still widely used to count and isolate roots.

In 1834, Vincent [94, 95] proposed a theorem based on Budan's original formulation of the Budan–Fourier's theorem. This method, due to the fact that the problem had already been considered *solved* after Sturm's work, was almost completely forgotten until the middle of the 20th century. However, a new family of algorithms based on his work appeared during that century. Nowadays, Vincent's work is the basis of the fastest known method for isolating the real roots of a univariate polynomial, and for a family of algorithms based on it [32, 13, 19].

So, Fourier's formulation of Budan's theorem has been the basis for Sturm's method. Budan's original formulation has been the basis for Vincent's theorem, which has been forgotten, brought back to life about one century later, and now serves as basis for the fastest method among the current approaches.

There exist different proofs of Fourier's theorem in the literature. Fourier's original work was split into two parts [47, 46]; one part proved the theorem assuming a given condition and the other proved that the theorem remains true without that assumption. In this work we will give a proof of the theorem that is based on the ideas presented by Fourier in the first of his two parts. Fourier's theorem allows to compute an upper bound on the number of roots of a polynomial in a given interval (a, b) and establishes that the actual number of roots differs from that bound by an even non-negative quantity. If the bound is 0 or 1, it is actually *the exact number of roots*.

At the present, the most widely mentioned algorithms in the bibliography, based on

Vincent's work, are the VAS (Vincent, Akritas, Strzeboński) [13], the VCA (Vincent, Collins, Akritas) [32] and the VAG (Vincent, Alesina, Galuzzi) [17, 18, 19]. The last one is, among these three, the clearest and simplest conceptually; but it is not efficient. The best implementation of VCA is the one given by Rouillier and Zimmermann [85], and it was the preferred method (in practice) to compute zeros of univariate polynomials, until the discovery of the VAS, which has shown to be faster [13].

So, nowadays, there exist two main approaches for the problem: Sturm's theorem and Vincent's theorem. While Vincent's theorem is based on Budan's work, Sturm's theorem mimics Fourier's proof of an equivalent formulation of it.

Let us get back to the historical line, to 1834, after Vincent stated his theorem. Nobody cared about it for about one century, until the book of Uspensky [92]. Uspensky himself, writing about Vincent's theorem says:

> This remarkable theorem was published by Vincent in 1836, in the first volume of *Liouville's Journal*, but later so completely forgotten that no mention of it is found even in such a capital work as *Enzyclopädie der mathematischen Wissenschaften*.

Uspensky exposed, in his book, the method that was proposed by Vincent. It had the disadvantage that could have exponential complexity in some cases. Collin and Akritas, in the work in which they introduced the VCA [32], named this method as *Uspensky method* and the method they were presenting (now known as VCA) as *Modified Uspensky method*. Some years later, Akritas realized that the method he had taken from the book of Uspensky was not *Uspensky's*; he realized that it was in fact Vincent's method with slight modifications. When Akritas realized about this, he published the article *There is no Uspensky's method* [9], in which he explains this name confusion. Anyway, even not being the author of the implementation used in the VCA article for comparisons, Uspensky's contribution was remarkable: he was the first author, after around one century, who brought back to life the forgotten Vincent's theorem. The fastest algorithms for *root finding* of nowadays are based on this theorem and, by middle of the 20th century, it was forgotten.

In fact Akritas explains that this name confusion began when he read Vincent's theorem and method *from the book of Uspensky*, the only place in which it could have been found at that time.

There are, nowadays, a lot of important articles, with significative contributions to the problem. Table 2.1 is not intended to give a complete map of all the related bibliography, it is supposed just to depict a high–level map of the works, in a chronological way. In the next sections we will go into further details on some of these contributions.

2.1.1 On the Descartes' rule of signs (1637)

There are in the literature many statements *similar*, but not equal, to the original Descartes' formulation of his rule. For example, Prasolov, in [82, page 28], states Descartes' rule is defined as follows:

> The number of positive roots of the polynomial $f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ does not exceed the number of sign changes in the sequence $a_0, a_1, \ldots, a_n$.

Collins and Akritas in [32, page 273], states:

> Descartes' rule of signs is a theorem which asserts that the number of positive real roots (multiplicities counted) of a real polynomial A is equal to $\text{var}(A) - 2k$, for some non-negative integer k[1].

Rouillier and Zimmermann in [85, page 35], states (in their *theorem 2*), a definition equivalent to Collins and Akritas' one:

> Let $P(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial in $\mathbb{R}[x]$. If we denote by $V(P)$ the number of sign changes in the list $(a_0, \ldots, a_d)$ and $\text{pos}(P)$ the number of positive real roots of P counted with multiplicities, then $\text{pos}(P) \leq V(P)$, and $V(P) - \text{pos}(P)$ is even.

Sagraloff, in [87, page 298, 2nd column, 2nd paragraph] states:

> For an arbitrary polynomial $p(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{R}[x]$, the number m^* of positive real roots of p is bounded by the number v^* of sign variations in its coefficients sequence $(p_0, \ldots, p_n)$ and, in addition, $v^* \equiv m^* \mod 2$.

[1] where $\text{var}(A)$ is the number of sign variations in the list of coefficients of A

TABLE 2.1 Main contributions to the problem of computing the real roots of a univariate polynomial, and its associated subproblems.

1637	Descartes states without any proof, in his *La géometrie*, his *rule of signs* (see page 11).
1798	Lagrange proposed his method, which required a large amount of computational effort. He also introduced methods to compute the root repulsion of the input polynomial [72, 73] (see page 13).
1800	Budan introduced his theorem, which allowed the computation of an upper bound on the number of real roots that a given univariate polynomial has on a given interval, based on analyzing changes of signs in the sequence of coefficients and performing changes of variable of the form $x \leftarrow x + 1$ [25] (see page 15).
1820	Fourier presented his theorem, which turned to be a reformulation of Budan's theorem [47, 46, 82] (see page 50).
1829	Sturm presented a method strongly based on Fourier's formulation, and whose proof mimics Fourier's proof. This method was the first one that computed the exact number of roots instead of an upper bound. Sturm method is still used nowadays [90, 91] (see page 55).
1834	Vincent presented his theorem and method, based on Budan's original formulation of his theorem. Vincent's theorem was forgotten, mainly due to the existence of Sturm's method. Vincent's method had exponential cost on the worst case [94, 95] (see page 58).
1948	Uspensky brings Vincent's theorem back to life [92, 9].
1976	Collins and Akritas, based on Uspensky's book, proposed a method to isolate the roots of a polynomial using Vincent's theorem; this method was much better than Vincent's, because it (unlike Vincent's) had no exponential complexity in the worst case, it was polynomial. They compared their method to *Uspensky's* method. Later, they realized that the comparison was indeed against a method that was already proposed by Vincent in his work [32, 9] (see page 4.4.1).
2004	Rouillier and Zimmermann proposed the fastest implementation of the VCA algorithm. They made clever considerations about the implementation, which shown to impact its performance significatively [85].
2005	Akritas and Strzeboński compared the bisection approach used in the implementation of Rouillier and Zimmermann with a method based on continued fractions. The new approach shown to be faster than the fastest implementation of VCA [13] (see page 68).

Although all of them can be proven to be true, none of these statements is the original statement. The original rule was published by Descartes in 1637, in his work *La géometrie*, without any proof. A complete proof of the rule was given only in 1828 by Gauss [19]. At the first part of his work, Descartes claims that a polynomial $p(x)$ has a root α if and only if it is divisible by $x - \alpha$. After that, he states without a proof:

> As a result, it is possible to know how many true[2] roots and false[3] roots an equation[4] can have. Namely, it can have as many true roots as the signs + and − alternate, and as many false roots as two signs + or two signs − follow one another.

So, translating into more modern words, Descartes' rule of signs, as stated by Descartes, is composed by the two following parts:

1. A real polynomial has *no more* positive roots than alternations of signs between two consecutive coefficients.

2. A real polynomial has *no more* negative roots than permanences of signs between two consecutive coefficients.

Descartes' statement was attacked by several contemporaries, pointing that a real polynomial can have fewer positive roots than the number of sign alternations in the list of its coefficients. The counterargument is just the fact that Descartes was meaning *at most* as many positive roots as alternations of signs and *at most* as many negative roots as permanences of signs. This is actually what Descartes explained in his 77^{th} refutation letter, directed against Roberval [40]. Wallis, another contemporary of Descartes, tried to attribute the rule to Harriot, an English geometer postdating Viète and predating Descartes. However, De Gua [36] pointed that Wallis' thesis cannot be seriously argued. The arguments of De Gua were solid and well documented.

Now that we have stated original Descartes' rule of signs, let us stop briefly and analyze the following example:

$$p(x) = x^2 - 1 = (x - 1)(x + 1)$$

[2] by *true* he was meaning *positive*
[3] by *false* he was meaning *negative*
[4] a polynomial

has, apparently, 1 alternation of signs and 0 permanences of signs! So at a first glance, part 2 of Descartes' rule of signs seems to be incorrect. What is the problem? What did we miss?

The point is that in the most intuitive way in which we count alternations and permanences of signs in the list of coefficients, we just remove all zeros and proceed to count. But what was Descartes' meaning? What is the meaning of *consecutive terms*? Let us show Descartes' presumed interpretation [21]. First, let us note that Descartes' statement is in fact correct *if one assigns, in any manner, signs to the lacunary coefficients*. For example, in the previous polynomial, we could write

$$p(x) = +x^2 + 0x - 1$$

and p would have 1 alternation and 1 permanence. And with these quantities, Descartes' rule of signs is correct for this case. Or we could write

$$p(x) = +x^2 - 0x - 1$$

and, again, p would have 1 alternation and 1 permanence and again the Descartes' rule is correct. It is clear, given that Descartes' rule produces *upper bounds* for the number of positive (and negative) roots, that it is preferable to assign the signs of lacunary coefficients in a way that minimizes these quantities. Descartes' presumed convention can be synthetized into two rules [21]:

1. In the context of counting sign alternations of p, the sign of a non-zero coefficient limiting a sequence of null coefficients on the left propagates to the entire sequence.

2. In the context of counting sign permanences of p, the sign of a sequence of null coefficients alternates, starting from the first non-zero coefficient limiting the sequence on the left. This means that a sequence of coefficients of the form $a_i, 0, 0, \ldots, 0, a_j$ with $a_i a_j \neq 0$ contributes 1 permanence if a_i and a_j are of opposite signs and the number of 0's is odd; or if a_i and a_j are of the same signs and the number of 0's is even. Otherwise, it contributes no permanence.

For example, let us count with this convention the number of sign alternations and sign permanences on the polynomials $p(x) = +3x^4 - x$ and $p(x) = +3x^5 - x$

$$\begin{aligned} &\textit{alternations:} \quad p(x) = +3x^4 + 0x^3 + 0x^2 - x^1 - 0x^0 \implies 1 \text{ alternation} \\ &\textit{permanences:} \quad p(x) = +3x^4 + 0x^3 - 0x^2 - x^1 - 0x^0 \implies 3 \text{ permanences} \end{aligned}$$

$$\begin{aligned} &\textit{alternations:} \quad p(x) = +3x^5 + 0x^4 + 0x^3 + 0x^2 - x^1 - 0x^0 \implies 1 \text{ alternation} \\ &\textit{permanences:} \quad p(x) = +3x^5 + 0x^4 - 0x^3 + 0x^2 - x^1 - 0x^0 \implies 2 \text{ permanences} \end{aligned}$$

An intuitive way of seeing the second part of the convention is: *when counting sign permanences, assign to each lacunary term of the form* $0x^i$ *the same sign of the expression* $(-x)^i$.

It might seem strange to use two different assignations of signs to the null coefficients in order to count permanences and alternations, but this method has two advantages: the number of alternations and permanences is minimized and the two methods are dual in the sense that the number of permanences of $p(x)$ is the number of alternations of $p(-x)$ and vice versa.

2.1.2 Lagrange (1798)

In 1798, in his book *De la résolution des équations numériques de tous les degrés* [72, 73] , Lagrange exposed a complete method for isolating and approximating all real (and complex) roots of a polynomial with real coefficients.

As it was already mentioned at the begin of this historical survey, the key idea of Lagrange's method is to evaluate the input polynomial p at an increasing sequence of numbers, starting at a lower bound of the roots of p, and ending at an upper bound of them, chosen in a way that ensures that p cannot have more than one root between any two consecutive sequence elements. From the changes of signs at the results of these evaluations we can tell which intervals have exactly one root.

In order to fulfill the condition that p cannot have more than one root between any two consecutive elements of the sequence, the sequence is chosen to be an arithmetic progression with a ratio Δ being a lower bound on the distances between any two roots of p.

Lagrange proposed four ways to compute a valid Δ, with and without explicit com-

putation of the auxiliary *equation of differences*, whose roots are the differences between all ordered pairs of distinct roots of p.

The problem with this approach proposed by Lagrange was that it is very expensive in terms of computation. Lagrange himself realized that and tried to improve this point. He proposed techniques to make the algorithm more efficient and, although the algorithm itself has been replaced by other much more efficient methods, some of the powerful ideas he used when trying to improve his algorithm foreshadowed methods developed much later in geometry and abstract algebra.

Lagrange's algorithm

Lagrange's method aims to *compute the positive roots of a polynomial equation*

$$p(x) = x^n - a_{n-1}x^{n-1} + a_{n-2}x^{n-2} - a_{n-3}x^{n-3} + \cdots = 0 \tag{2.1}$$

where p has only simple roots. This condition can be achieved by dividing p by $\gcd(p, p')$. Lagrange's algorithm is essentially divided in three steps:

1. Compute a lower bound Δ for the distance between roots of p.

2. Using Δ, together with rescaling techniques, isolate roots of p. This step is the most computational-effort-demanding.

3. Refine the intervals produced in step 2.

For step 1, Lagrange introduced the *equation of differences*, whose roots are the differences of all ordered pairs of distinct roots of the input polynomial.

2.1.3 Budan (1800)

Budan's work [25] is a complete book including not only the theorem now known by his name, but also a method for isolating the roots of a univariate polynomial and to compute the coefficients of a polynomial after a variable *shifting*. That is: the coefficients of $p(x + 1)$ given the coefficients of $p(x)$.

Budan's theorem can be stated as follows:

Theorem 1. *Given a polynomial $p(x)$, of degree n, and given the real numbers l and r, such that $l < r$ and $p(r) \neq 0$, then the number V_l of alternations of signs in the list of coefficients of the polynomial $p(x+l)$ cannot be smaller than the number V_r of alternations of signs in the list of coefficients of the polynomial $p(x+r)$; and $V_l - V_r$ is an upper bound for the number or roots of p in the interval (l, r), and it can differ from the actual number of roots by an even amount only.*[5]

2.1.4 Fourier (1820)

Fourier's theorem can be formulated as follows [47, 46]:

Theorem 2. *Let $N(x)$ be the number of sign changes in the sequence $f(x)$, $f'(x)$, ..., $f^{(n)}(x)$, where f is a polynomial of degree n. If $a < b$, $f(a) \neq 0$, $f(b) \neq 0$, then $N(a) \geq N(b)$, and the number of roots of f (multiplicities counted) between a and b does not exceed $N(a) - N(b)$. Moreover, the number of roots can differ from $N(a) - N(b)$ by an even number only.*

This theorem will be revisited in more detail in chapter 4.

2.1.5 Equivalence between Fourier's and Budan's formulations

The equivalence follows trivially from the observation that, given a polynomial p of degree n, the $n+1$ terms of the Fourier sequence $p(l)$, $p'(l)$, ..., $p^{(n)}(l)$ have the same signs than the coefficients of $p(x+l) = \sum_{i=0}^{n} \frac{p^{(i)}(a)}{i!} x^i$.

Alesina and Galuzzi [17, 18, 19], the authors of the clearest modern method, pointed out that the controversy over priority rights of Budan or Fourier is pointless from a modern point of view. Alesina and Galuzzi states that Budan had "an amazingly modern understanding of the relevance of reducing the algorithm to translate a polynomial by $x \leftarrow x + p$ by simple additions".

2.1.6 Sturm (1829)

Sturm [90] presented, in 1829, a method which was strongly based on Fourier's proof of his theorem; but it had a substantial difference. Sturm's method allowed counting

[5] The reason why the literature often uses the letter V to denote these quantities is that such number of alternations of signs in the list of coefficients of a polynomial is often referred as *variations* instead of *alternations*.

exactly the number of different roots that a polynomial had in a given interval. Sturm's defined an alternative sequence for the input polynomial (instead of the sequence of its derivatives of the Fourier's theorem) and, with this new list of functions, the difference in the number of sign variations, when the list is evaluated at two points a and b is the *exact* number of *different* roots (multiple roots count just as 1) of p, instead of the upper bound produced by Fourier's theorem. We will return to this theorem in chapter 4.

2.1.7 Vincent (1834)

In 1834, Vincent, in his work *Note Sur la résolution des Équations numériques* [94, 95], stated the following theorem:

> **Theorem** (Vincent)**.** *Si dans une équation numérique rationelle en x dépourvue de racines égales, on fait successivement, et conformément au procédé de Lagrange,*
>
> $$x = a + \frac{1}{x'}, \quad x' = b + \frac{1}{x''}, \quad x'' = c + \frac{1}{x'''}, \quad \ldots$$
>
> *on parvient toujours par la suite des transformations , et quels que soient d'ailleurs les nombres $a, b, c, \ldots$ [supposés toutefois positifs et ¿ 1], à une équation transformée qui se trouve dans l'un de ces deux cas : ou de neu plus avoir que des permanences, ou de ne plus offrir qu'UNE variation ; dans ce seconde cas, l'équation en x a une racine réelle positive représentée par la fraction continue*
>
> $$a + \cfrac{1}{b + \cfrac{1}{c + \cfrac{1}{d + \ldots}}}$$
>
> *et n'en a qu'une seule de cette valeur; le premier cas , au contraire , arrive toutes les fois que l'équation n'a aucune racine susceptible de l'expression indiquée.*

The phrase "et conformément au procédé de Lagrange" in the previous theorem is important; and the lack of it makes many of the restatements of Vincent theorem be incorrect. For example, Alesina and Galuzzi's article [17, page 219] states that *the resulting polynomial after the variable change* has at most one sign variation. The same mistake is present

in Uspensky's statement of the theorem. This is incorrect, since the resulting expression after the sequence of substitutions *is not a polynomial*. This mistake is, unfortunately, widely spread in the bibliography. This is unfortunate, especially for people interested in understanding the details of the subject. A carefully inspection of both Vincent's and Lagrange's works leads us to understand that the *elimination of the denominator* in the resulting expression is, precisely, the meaning of the phrase "et conformément au procédé de Lagrange".

Having said this, the theorem can be restated as follows.

Theorem (Vincent)**.** *Let $f(x)$ be a polynomial of degree n with rational coefficients and without multiple roots. The sequence of h successive variable changes*

$$x \leftarrow a_1 + \frac{1}{x}, \quad x \leftarrow a_2 + \frac{1}{x}, \quad x \leftarrow a_3 + \frac{1}{x}, \quad \ldots, x \leftarrow a_h + \frac{1}{x}$$

can be seen as one only change of the form

$$x \leftarrow a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots + \cfrac{1}{a_h + \cfrac{1}{x}}}}$$

which is,

$$x \leftarrow \frac{Ax + B}{Cx + D}$$

for some nonnegative integers A, B, C, D. For h sufficiently large, the polynomial

$$(Cx + D)^n f\left(\frac{Ax + B}{Cx + D}\right)$$

has either 0 or 1 sign variations in its list of coefficients. Moreover, in the first case f has no roots in the interval whose endpoints are $\frac{B}{D}$ and $\frac{A}{C}$, while in the second case it has exactly 1 root in it.

This result, known nowadays as *theorem of Vincent*, is used to prove that Vincent's method (proposed in the same work) terminates. These ideas served as basis for a family of modern algorithms, in which is included the fastest known algorithm for root isolation. Vincent's algorithm and method will be revisited in chapter 4.

2.1.8 Modern results: from Uspensky up to nowadays

The theorem of Vincent was completely forgotten, probably due to the existence of Sturm's method. But in 1948, Uspensky, in his book *Theory of Equations*, restated Vincent's theorem and method for root isolation. After him, Collins and Akritas, based on Uspensky book, proposed the first algorithm based on Vincent's work which handled the exponential worst-case. They showed that the proposed method was faster than Sturm's and Uspensky's methods. Some years later, Akritas realized that the method that they called *Uspensky's* should have been called in fact *Vincent's method*. The algorithm they proposed is now known as VCA. In 2004, Rouillier and Zimmermann gave the fastest implementation of this method. Their considerations for it impacted very significatively the performance of the VCA. The last contribution to the problem of isolating the roots of a polynomial was made in 2005, by Akritas and Strzeboński [13]. The algorithm that they proposed is strongly based on Vincent's method; it is known as VAS and is the default root isolation method in most of the widely used algebra systems (Mathematica, Sage, SymPy, Xcas).

Chapter 3

Bounding the real roots of univariate polynomials

The first, and in many cases the most expensive, step in most methods for computing the real roots of a polynomial is to *isolate* them. That is, to compute a set of disjoint intervals containing them (exactly one root on each interval and all roots contained in disjoint intervals).

Root bounding is a central problem to real root isolation algorithms (i.e.: to this first step). Among the most widely-used root isolation algorithms, there are the Vincent–Akritas–Strzeboński (VAS) and Vincent–Collins–Akritas (VCA) algorithms[1]. In particular, VAS computes a *lower bound* on the positive roots of a polynomial on each execution of the main loop.

It has been shown that even slight improvements on the quality of this lower bound, and even investing much computational effort when computing it, impact the performance of the VAS algorithm [15, 93].

There are two main variables of root bounding which impact in the performance of VAS: the *accuracy of the bounds* and the *efficiency of the root bounding method*. This chapter introduces an idea intended to address both variables. A new method will be presented, which improves both the accuracy and the performance of the current methods.

So, throughout the chapter we consider the problem of obtaining upper bounds for the positive real roots of polynomials. In section 1, we present a survey of the most

[1]Nowadays, VAS is the default root isolation method in many widely used computer algebra systems, such as Mathematica, SageMath, SymPy and XCAS. VCA was, for many years, the default method many computer algebra systems, but it is currently only used by Maple.

important existing algorithms. Section 2 exposes a simple unified framework that allows seeing almost all these algorithms from a common point of view. For that, the concept of *killing graph* is introduced. In section 3, a new root bounding method is introduced, which is actually a way to *improve the result obtained by any other method*. Any method can be chosen and then, after it is executed, the present method can be run *on* the obtained results and improve them.

The new method works as follows: at the beginning, it computes an upper bound for the positive roots of the input polynomial by using some other existent method (for example *first-*λ). During this computation, it also creates the *killing graph* associated to the solution produced by that other method. This structure allows improving the answer generated by the auxiliary already existent bounding method.

The complexity of the method being introduced in next sections is $O(t + \log^2(d))$, where t is the number of nonzero monomials the input polynomial has and d is its degree. Since the other methods are either linear or quadratic in t, the new method does not introduce a significant complexity overhead. To better improve the solution, it can be executed many times. It will be shown in this chapter that, with only *two* runs, the new method improves all the current algorithms (even those whose complexity is quadratic in t).

Sections 4 and 5 are related to the implementation, experiments, and analysis of the obtained results. Section 6 is the conclusion.

A subset of the ideas presented in this chapter were exposed at SIBGRAPI'2015, in the WIP track [23].

Contributions in this chapter The main contributions in this chapter are:

- Putting together, in the same text, in the same context, all the most used approaches and methods to compute an upper bound for the real roots of a given univariate polynomial.

- Proposing a framework to derive almost all of them in a unified way, from a general idea. It is clear, from the texts of related bibliography, that some authors already have this idea [14, 12, 15, 89]. In some places it is even proposed in an *implicit* way; in this work we propose the concept in a *more explicit* way.

- Proposing, implementing and extensively testing a new method, on many significant scenarios, obtaining a technique that yields a significative improvement in the results while not introducing a significant time penalty.

A note on notation Throughout this chapter, we will refer to terms with positive coefficients as *positive terms*, to terms with negative coefficients as *negative terms*, and to terms with null coefficients as *null terms*.

3.1 Related Work

Consider a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \qquad (a_n > 0) \tag{3.1}$$

The problem of computing an upper bound for the largest real positive root of p has been studied by many authors. In this section we will give an overview of the most relevant methods. The next section will intend to give a way to see most of these methods in the same unified way. The present section does not intend to give deep details on each particular method, but to give an overview of all of them. The work of Vigklas includes an excellent and detailed overview of all these methods [93].

The most important approaches present in the literature for the problem of computing an upper bound for the real positive roots of a polynomial p defined as in 3.1 can be classified by their complexity. Some of them require a computational effort proportional to the number of terms of the input polynomial (and we call them *linear*), and some of them require a computational effort proportional to the square of this quantity (and we call them *quadratic*).

Among the most famous *linear* methods, we can mention the bounds given by Cauchy's leading coefficient method [14, 16, 93], Lagrange [16, 93, 72], Kioustelidis [59], Stefanescu [89], first-λ method, [14] local-max method [14]. Among the *quadratic* ones, we can mention the quadratic variants of Cauchy [93], Kioustelidis [93], first-λ [12, 93] and local-max [15, 93].

As is shown in [93], the best linear–time bounds are first-λ and local-max; and its quadratic versions. The best current approach with linear complexity is taking the min-

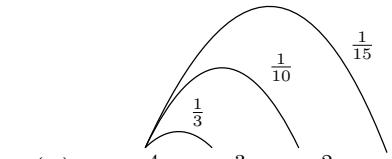
imum of the bounds produced by these two approaches, and the same happens with quadratic complexity. The method that we will present in this chapter improves in all cases the linear results and even in *almost all* cases the quadratic approaches, and is still linear.

In this section, for the sake of completeness, we will show most of these methods.

3.1.1 Assignments, costs and induced bounds

In some of the following subsections we will use the concepts of *assignment* and *induced bound*.

Imagine that we assign to each negative term of $p(x)$ a fraction of a positive term of higher degree (also of $p(x)$), taking care that no positive term is assigned *more than entirely*. For example, in the figure, the fraction assigned of x^4 is $\frac{1}{3} + \frac{1}{10} + \frac{1}{15} = \frac{1}{2} < 1$. If it was larger than 1, then it would have been assigned *more than entirely*. The following figure shows an example of assignment of negative terms to fractions of positive terms of higher degree. The edges show the relation *assigned to* and the values at them indicate the fraction of the term being assigned.



If we have such an assignment (i.e.: an assignment in which *all* the negative terms are assigned to a fraction of a positive term of higher degree and no positive term is assigned more than entirely), then we also have an upper bound for the roots of $p(x)$ *induced* by the assignment. In fact, we can compute for each negative term a value from which the fraction of positive term to which it is associated starts to be greater than it, and the maximum of these values would be the mentioned upper bound. For example, in the previous example, we have:

$$\left.\begin{array}{ll} \frac{1}{3}x^4 > 3x^3 & \text{for } x > 9 \\ \frac{1}{10}x^4 > 2x^2 & \text{for } x > \sqrt{6} \\ \frac{1}{15}x^4 > 10x & \text{for } x > \sqrt[3]{30} \end{array}\right\} \Longrightarrow p(x) > 0, \text{ for } x > \max\{9, \sqrt{6}, \sqrt[3]{30}\}$$

We call by *cost of the assignment of B to A* and sometimes, depending on the context, just by *cost*, to the value of x at which the positive term $A(x)$ starts to be greater than the positive term $-B(x)$ (being B a negative term). In the previous example, the costs were $9, \sqrt{6}, \sqrt[3]{30}$.

We will be back to these concepts later in this chapter.

3.1.2 Cauchy's *leading coefficient* [93, 15, 89, 16]

Let $p(x)$ be a polynomial as in 3.1. Let λ be the number of negative coefficients in p. Then, an upper bound on the values of the positive roots of $p(x)$ is given by

$$\text{ub} = \max_{a_{n-k}<0} \sqrt[k]{-\frac{\lambda a_{n-k}}{a_n}}$$

Proof. When $\lambda = 0$, p has no real roots and the theorem is trivial. Suppose that $\lambda > 0$. We shall prove that $p(x) > 0$ for every $x > \text{ub}$ and, thus, that ub is an upper bound for the positive roots of p. Consider the sum of the negative terms of $p(x)$. By the statement, the number of such terms is λ. The sum is:

$$S(x) = a_{j_1}x^{j_1} + a_{j_2}x^{j_2} + \cdots + a_{j_\lambda}x^{j_\lambda}$$

Let $x > \text{ub}$. By the definition of ub we have the following:

$$\begin{array}{rcccccc}
x > \text{ub} \geq & \sqrt[n-j_1]{-\frac{\lambda a_{j_1}}{a_n}} & \Rightarrow & a_n x^{n-j_1} > -\lambda a_{j_1} & \Rightarrow & a_n x^n > -\lambda a_{j_1} x^{j_1} \\
x > \text{ub} \geq & \sqrt[n-j_2]{-\frac{\lambda a_{j_2}}{a_n}} & \Rightarrow & a_n x^{n-j_2} > -\lambda a_{j_2} & \Rightarrow & a_n x^n > -\lambda a_{j_2} x^{j_2} \\
\vdots & \vdots & & \vdots & & \vdots & \vdots \\
x > \text{ub} \geq & \sqrt[n-j_\lambda]{-\frac{\lambda a_{j_\lambda}}{a_n}} & \Rightarrow & a_n x^{n-j_\lambda} > -\lambda a_{j_\lambda} & \Rightarrow & a_n x^n > -\lambda a_{j_\lambda} x^{j_\lambda}
\end{array}$$

by summing the inequalities obtained (i.e.: the λ inequalities at the right side of each one of the previous lines), we have:

$$\underbrace{a_n x^n}_{\text{first term of } p} > \underbrace{-a_{j_1}x^{j_1} - a_{j_2}x^{j_2} \cdots - a_{j_\lambda}x^{j_\lambda}}_{\text{absolute value of the sum of negative terms of } p}$$

Then, we have that $p(x) > 0$ for every $x > \text{ub}$ □

3.1.3 Lagrange [93, 15, 89, 16, 72, 73, 71]

Let $p(x)$ be a polynomial as in 3.1. Let a_k, $k < n$, be the first[2] negative term, and let B be the largest of the absolute values of negative coefficients in p. Then, an upper bound on the values of the positive roots of $p(x)$ is given by

$$\text{ub} = 1 + \sqrt[n-k]{\frac{B}{a_n}}$$

Proof. If in $p(x)$ we replace each of the nonnegative coefficients $a_{n-1}, a_{n-2}, \ldots, a_{k+1}$ by zero, and each of the remaining coefficients $a_k, a_{k-1}, \ldots, a_0$ by $-B$, we obtain a new polynomial $p_2(x)$. When $x > 1$, we have the following:

$$p(x) \geq p_2(x) = a_n x^n - B(x^k + x^{k-1} + \cdots + 1) = a_n x^n - B\frac{x^{k+1} - 1}{x - 1}$$

Thus, when $x > 1$, we have:

$$\begin{aligned}
p(x) &\geq a_n x^n - B\frac{x^{k+1} - 1}{x - 1} \\
&= \frac{1}{x-1}\left((x-1)a_n x^n - B(x^{k+1} - 1)\right) \\
&= \frac{1}{x-1}\left((x-1)a_n x^n - Bx^{k+1} + B\right) \\
&> \frac{1}{x-1}\left((x-1)a_n x^n - Bx^{k+1}\right) \\
&= \frac{x^{k+1}}{x-1}\left((x-1)a_n x^{n-k-1} - B\right) \\
&> \frac{x^{k+1}}{x-1}\left((x-1)a_n (x-1)^{n-k-1} - B\right) \\
&= \frac{x^{k+1}}{x-1}\left(a_n (x-1)^{n-k} - B\right)
\end{aligned}$$

Now $\dfrac{x^{k+1}}{x-1} > 0$ when $x > 1$, and $a_n(x-1)^{n-k} - B > 0$ when $x > \text{ub}$. In fact:

$$\begin{aligned}
a_n(x-1)^{n-k} - B &> 0 \\
x - 1 &> \sqrt[n-k]{\tfrac{B}{a_n}} \\
x &> 1 + \sqrt[n-k]{\tfrac{B}{a_n}} = \text{ub}
\end{aligned}$$

Thus, $p(x) > 0$ when $x > \text{ub}$. □

[2]The first term in a set of terms is the one with the highest degree, the one which appears first when reading the expression 3.1 from left to right.

3.1.4 Kioustelidis [93, 15, 89, 16]

Let $p(x)$ be a polynomial as in 3.1. Let $a_{j_1}, \ldots, a_{j_\lambda}$, be the λ negative coefficients in $p(x)$. Then, an upper bound on the values of the positive roots of $p(x)$ is given by

$$\text{ub} = 2 \max_{1 \le w \le \lambda} \sqrt[n-j_w]{-\frac{a_{j_w}}{a_n}}$$

Proof. We will proceed as in the proof of Cauchy's bound. When $\lambda = 0$, p has no real roots and the theorem is trivial. Suppose that $\lambda > 0$. We shall prove that for every $x > \text{ub}$, $p(x) > 0$ and, thus, that ub is an upper bound for the real roots of p.

Consider the sum of negative terms of $p(x)$. By the statement, the number of such terms is λ. The sum is:

$$S(x) = a_{j_1}x^{j_1} + a_{j_2}x^{j_2} + \cdots + a_{j_\lambda}x^{j_\lambda}$$

By definition of ub we have that, for every $1 \le w \le \lambda$,

$$\text{ub} \ge 2 \sqrt[n-j_w]{-\frac{a_{j_w}}{a_n}} = \sqrt[n-j_w]{2^{n-j_w}\frac{-a_{j_w}}{a_n}} = \sqrt[n-j_w]{\frac{-a_{j_w}}{\frac{a_n}{2^{n-j_w}}}}$$

Let $x > \text{ub}$. By the definition of ub we have the following:

$$\begin{array}{lcccccc}
x > \text{ub} \ge & \sqrt[n-j_1]{\frac{-a_{j_1}}{\frac{a_n}{2^{n-j_1}}}} & \Rightarrow & x^{n-j_1} > \frac{a_{j_1}}{\frac{a_n}{2^{n-j_1}}} & \Rightarrow & \frac{a_n}{2^{n-j_1}}x^n > a_{j_1}x^{j_1} \\
x > \text{ub} \ge & \sqrt[n-j_2]{\frac{-a_{j_2}}{\frac{a_n}{2^{n-j_2}}}} & \Rightarrow & x^{n-j_2} > \frac{a_{j_2}}{\frac{a_n}{2^{n-j_2}}} & \Rightarrow & \frac{a_n}{2^{n-j_2}}x^n > a_{j_2}x^{j_2} \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \\
x > \text{ub} \ge & \sqrt[n-j_\lambda]{\frac{-a_{j_\lambda}}{\frac{a_n}{2^{n-j_\lambda}}}} & \Rightarrow & x^{n-j_\lambda} > \frac{a_{j_\lambda}}{\frac{a_n}{2^{n-j_\lambda}}} & \Rightarrow & \frac{a_n}{2^{n-j_\lambda}}x^n > a_{j_\lambda}x^{j_\lambda}
\end{array}$$

by summing the inequalities obtained (i.e.: the λ inequalities at the right side of each one of the previous lines), we have:

$$a_n \underbrace{\left(\frac{1}{2^{n-j_1}} + \cdots + \frac{1}{2^{n-j_\lambda}}\right)}_{\text{this is } <1} x^n > \underbrace{-a_{j_1}x^{j_1} - a_{j_2}x^{j_2} \cdots - a_{j_\lambda}x^{j_\lambda}}_{\text{absolute value of the sum of negative terms of } p}$$

Then, we have that $p(x) > 0$ for every $x > \text{ub}$ □

3.1.5 Stefanescu [89]

Let $p(x)$ be a polynomial as in 3.1, such that the number of variations of signs in the list of its coefficients (i.e.: in the list $(a_n, a_{n-1}, \ldots, a_0)$) is even. If

$$p(x) = c_1x^{d_1} - b_1x^{m_1} + c_2x^{d_2} - b_2x^{m_2} + \cdots + c_kx^{d_k} - b_kx^{m_k} + g(x)$$

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

$$\text{ub} = \max\left\{\left(\frac{b_1}{c_1}\right)^{\left(\frac{1}{d_1-m_1}\right)}, \ldots, \left(\frac{b_k}{c_k}\right)^{\left(\frac{1}{d_k-m_k}\right)}\right\}$$

is an upper bound.

In other words. This formulation was given by Stefanescu in his paper [89]. The hypotheses of this theorem might seem a bit intricated, but it might be helpful to point out that the hypotheses can be seen as *forbiding consecutive terms with negative coefficient in the list of terms of p, ignoring terms with coefficient* 0. It is also worthwhile to point out that the requirement on the even number of sign changes in the list of coefficients is not necessary and that the reasoning in the proof can be easily adapted for that case. In fact, having an odd number of sign changes would just imply that the last term of p has different sign than the first one, fact that does not matter at all in the proof that Stefanescu himself proposed.

Proof. Suppose $x > 0$. Then

$$\begin{aligned} p(x) &\geq c_1x^{d_1} - b_1x^{m_1} + c_2x^{d_2} - b_2x^{m_2} + \cdots + c_kx^{d_k} - b_kx^{m_k} \\ &= x^{m_1}(c_1x^{d_1-m_1} - b_1) + \cdots + x^{m_k}(c_kx^{d_k-m_k} - b_k) \end{aligned}$$

which is positive for $x > \max\left\{\left(\frac{b_1}{c_1}\right)^{\left(\frac{1}{d_1-m_1}\right)}, \ldots, \left(\frac{b_k}{c_k}\right)^{\left(\frac{1}{d_k-m_k}\right)}\right\}$. □

3.1.6 First-λ [93, 15, 16]

In the polynomial

$$p(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 \qquad (a_n > 0)$$

consider the *blocks (of maximum size) of consecutive terms with coefficients of the same sign*. Firstly, call $g(x)$ to the rightmost block of positive terms (noting that $g(x)$ might be equal to $p(x)$, in the case that $p(x)$ has no negative coefficients). Now, start traversing the remaining blocks from left to right. Call $A_1(X)$ to the first block of positive terms, $-B_1(x)$ to the first block of negative terms, and so on. Refer as a_1 to the number of terms in $A_1(x)$, as b_1 to the number of terms in $B_1(x)$, and so on; as follows:

$$p(x) = \overbrace{a_n x^n + \cdots + a_{n-a_1+1}x^{n-a_1+1}}^{A_1(x)} \overbrace{-a_{n-a_1}x^{n-a_1} - \cdots - a_{n-a_1-b_1+1}x^{n-a_1-b_1+1}}^{-B_1(x)}$$
$$\overbrace{+a_{n-a_1-b_1}x^{n-a_1-b_1} + \cdots + a_{n-a_1-b_1-a_2+1}x^{n-a_1-b_1-a_2+1}}^{A_2(x)} \overbrace{-a_{n-a_1-b_1-a_2}x^{n-a_1-b_1-a_2}}^{-B_2(x)}$$
$$\overbrace{- \cdots - a_{n-a_1-b_1-a_2-b_2+1}x^{n-a_1-b_1-a_2-b_2+1}} + \cdots + g(x)$$

Thus, we have

$$p(x) = A_1(x) - B_1(x) + A_2(x) - B_2(x) + \cdots + A_k(x) - B_k(x) + g(x)$$

where all A_i and B_i have positive coefficients, $g(x)$ might be zero or might have positive coefficients, A_1 has a_1 terms, B_1 has b_1 terms, and so on. $g(x)$ is the rightmost block of positive terms. When p has only positive terms, g is equal to p and we have that $k = 0$. To simplify the notation, let us denote by $A_{i,j}$ to the j-th term of A_i and $B_{i,j}$ to the j-th term of B_i. For example:

$$\begin{aligned} B_{2,1}(x) &= a_{n-a_1-b_1-a_2}x^{n-a_1-b_1-a_2} \\ B_{2,b_2}(x) &= a_{n-a_1-b_1-a_2-b_2+1}x^{n-a_1-b_1-a_2-b2+1} \end{aligned}$$

Definition 3.1.1 (first-λ-breaking)**.** *Define the operation* first-λ-breaking *as follows: whenever an A_i has less terms than B_i (i.e.: whenever $a_i < b_i$), express A_i as a sum of b_i terms, by splitting* ***its last*** *term (i.e.: the term A_{i,a_i}) into $(b_i - a_i + 1)$ equal parts.*

For example: if $b_4 = 7$ and $a_4 = 5$, express $A_{4,5}(x) = \frac{A_{4,5}}{3} + \frac{A_{4,5}}{3} + \frac{A_{4,5}}{3}$. In this way, we have:

$$A_4(x) = A_{4,1}(x) + A_{4,2}(x) + A_{4,3}(x) + A_{4,4}(x) + A_{4,5}(x)$$

with 5 terms, and we will now have:

$$A_4(x) = A_{4,1}(x) + A_{4,2}(x) + A_{4,3}(x) + A_{4,4}(x) + \frac{A_{4,5}}{3} + \frac{A_{4,5}}{3} + \frac{A_{4,5}}{3}$$

with 7 terms, which is the number of terms in B_4.

Another example: if we applied the *first-λ-breaking* to the polynomial

$$p(x) = a_7x^7 + a_6x^6 - a_5x^5 - a_4x^4 - a_3x^3 + a_2x^2 - a_1x - a_0$$

we would obtain

$$p(x) = a_7x^7 + \frac{a_6x^6}{2} + \frac{a_6x^6}{2} - a_5x^5 - a_4x^4 - a_3x^3 + \frac{a_2x^2}{2} + \frac{a_2x^2}{2} - a_1x - a_0$$

The first-λ bound works as follows: call λ to the number of negative terms present in the definition of $p(x)$. Apply the operation first-λ-breaking to that expression. The number of positive terms in the resulting expression will be greater than or equal to λ. Call $N_1(x), N_2(x), \ldots, N_\lambda(x)$ to the λ negative terms and call $P_1(x), P_2(x), \ldots, P_\lambda(x)$ to the *first* λ positive terms in that resulting expression, going from left to right. Consider $x_i \in \mathbb{R}$, with $1 \leq i \leq \lambda$ to be such that $P_i(x) > N_i(x)$ for all $x > x_i$. Note that x_i always exists because P_i has higher degree than N_i. The maximum of these x_i is the bound proposed by the first-λ method.

3.1.7 Local-max [93, 15, 16]

Let $p(x)$ be a polynomial as in 3.1. Imagine traversing through the negative terms of $p(x)$ from left to right. The negative term $N(x) = -a_jx^j$ is assigned to $\frac{a_mx^m}{2^t}$, where a_mx^m is the positive term with biggest coefficient among the positive terms of degree greater than j; and t is the number of times that a_mx^m has been already used in an assignment. In other words: *traverse through the terms of p, from left to right. Assign to the negative term $N(x)$, half of the remaining fraction of the positive term with the biggest coefficient so far.* Observe that that positive term, because of the direction in which we are travelling, is always going to have degree greater than $N(x)$. Local-max bound is the induced bound in this assignment.

3.1.8 Cauchy quadratic [93]

Let $p(x)$ be a polynomial as in 3.1, Let λ_i be the number of negative coefficients in p to the right of, and including, a_i. Then, an upper bound on the values of the positive roots of $p(x)$ is given by

$$\text{ub} = \max_{a_i<0} \min_{\substack{a_j>0 \\ j>i}} \sqrt[j-1]{-\frac{a_i}{\frac{a_j}{\lambda_i}}}$$

Proof. Rewrite $p(x)$ by expressing each positive term $a_i x^i$ as a sum of λ_i equal terms. For each negative term $N_j(x) = -a_j x^j$ in the resulting expression consider set of the values at which the positive terms to the left of $N_j(x)$ start to be greater than $N_j(x)$. It is clear that all these values exist. Call x_j to the minimum of them.

When x is bigger than the maximum M of these x_j's, all the negative terms of p are lower (in absolute value) than their associated bigger order positive terms. Then, $p(x) > 0$ when x is bigger than M.

This M is, precisely, the bound suggested at the statement of the theorem. □

3.1.9 Kioustelidis quadratic [93]

Let $p(x)$ be a polynomial as in 3.1. Then, an upper bound on the values of the positive roots of $p(x)$ is given by

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

Proof. Imagine traversing through the negative terms of $p(x)$ from left to right. When visiting the negative term $N_j(x) = -a_j x^j$, consider the set of costs that would result from associating N_j to a fraction $\frac{1}{2^{w-j}}$ of each positive term $a_w x^w$ of higher degree. Call x_j to the minimum of these values. Call M to the maximum of these x_j's. It is clear that M is an upper bound for the roots of $p(x)$; and M is precisely the bound proposed in the statement. □

3.1.10 First-λ quadratic [93, 15, 12]

Let $p(x)$ be a polynomial as in 3.1. Call λ to the number of negative terms present in the definition of $p(x)$. Apply the operation first-λ-breaking, defined in page 27, to that

expression. Now, in the same way we had in the first-λ bound, we have that the number of positive terms in this resulting expression will be greater than or equal to λ.

Now imagine travelling through the negative terms of this expression, from left to right. When visiting $N_j(x) = -a_j x^j$, consider all the *remaining* (the word *remaining* will make sense in brief) positive terms located to the left of $N_j(x)$ (i.e.: with higher degree than $N_j(x)$). For each one of these positive terms, compute the value at which it starts to be greater than $|N_j(x)|$. This value exists for all the cases, since the positive terms have higher degree than $N_j(x)$. Call x_j to the minimum of them *and remove its associated positive term.* Note that this last sentence is what gives sense to the previous word *remaining.* At the beginning, it has no sense. All terms are *remaining* terms. Of course, at any point in the process a *remaining* term is a term which has not been removed before.

First-λ quadratic bound is the maximum of these x_i's.

3.1.11 Local-max quadratic [93, 15]

Let $p(x)$ be a polynomial as in 3.1. In the local-max method, we traveled through the negative terms $N_j(x)$ of $p(x)$ from left to right. We assigned $N_j(x)$ to the half of the remaining fraction of the higher-degree positive term with biggest coefficient. In the local-max quadratic the method is pretty similar, but instead of assigning $N_j(x)$ to that term, we compute what would be the costs of assigning it to the half of each one of the remaining fractions of positive terms of higher degree, and we pick the minimum. Local-max quadratic bound is the induced bound of this assignment.

3.2 A general framework

In this section we will come back to the concepts of assignments, costs and induced bounds stated briefly in previous section; we will explore it in more detail, and will define some vocabulary around it. In the same way that we were doing so far, we will be considering the problem of giving an upper bound for the biggest positive root of a polynomial $p(x)$, whose dominant term is positive. If p had negative dominant term, we would just need to compute the bound of $-p(x)$.

If we have the terms $a_n x^n$, and $a_i x^i$, with $n > i, a_n > 0, a_i < 0$, we know that at some

point x_0, the term $a_n x^n$ *starts* to be greater than the term $-a_i x^i$. That is: we know that exists x_0 such that $x > x_0 \Rightarrow a_n x^n + a_i x^i > 0$. In fact, $x_0 = \sqrt[n-i]{-a_i/a_n}$. We say that the term $a_n x^n$ *kills* the term $a_i x^i$ with *cost* $\sqrt[n-i]{-a_i/a_n}$.

Note that if we have a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

in which (1) every negative term is killed by a higher-degree positive term and (2) there is no positive term killing more than one negative term, then the maximum of the associated costs to these killings is an upper bound for the maximum positive root of p. Let us clarify this observation through some examples. Consider the polynomial:

$$p_1(x) = x^5 - 25x^4 + 200x^3 - 600x^2 + 600x - 120$$

We know that

- $x^5 > 25x^4$ when $x > 25$ (x^5 kills $-25x^4$ with cost 25);
- $200x^3 > 600x^2$ when $x > 3$ ($200x^3$ kills $-600x^2$ with cost 3);
- $600x > 120$ when $x > 0.2$ ($600x$ kills $-120x^4$ with cost 0.2).

Thus, we have: (1) all negative terms are killed by higher degree positive terms and (2) there are no positive terms killing more than one negative term. Thus, $\max(25, 3, 0.2)$ is an upper bound for the maximum root of $p_1(x)$, since all the negative terms are killed by some positive term from that point on.

Observe also that we could break positive terms in parts, and use the resulting parts separately to kill lower degree terms. For instance, if we had the polynomial

$$p_2(x) = 30x^5 - 25x^4 - 10x^3 - 200x^2 - 30$$

we could break up the term $30x^5$ into four or more parts and use four of them to kill the four terms. For example, we could break the expression $30x^5$ in many pieces, and rewrite the expression of $p_2(x)$ as follows:

$$p_2(x) = \begin{array}{cccccc} 1x^5 & +2x^5 & +3x^5 & +4x^5 & +18.5x^5 & +1.5x^5 \\ -25x^4 & -10x^3 & -200x^2 & -30 & & \end{array}$$

and we would know that

- $1x^5 > 25x^4$ when $x > 25$;
- $2x^5 > 10x^3$ when $x > 2.23$;
- $3x^5 > 200x^2$ when $x > 4.05$;
- $4x^5 > 30$ when $x > 1.49$.

Thus, we would have that $x > \max(25, 2.23, 4.05, 1.49) = \boxed{25} \implies p_2(x) > 0$. *Note that the way in which we break the* $30x^5$ *does impact in the result we obtain.* In fact, in the previous example we broke $30x^5$ into $1x^5 + 2x^5 + 3x^5 + 4x^5 + 18.5x^5 + 1.5x^5$, but we could have broken it into $10.9244x^5 + 1.90949x^5 + 16.6881x^5 + 0.477986x^5$ and we would have obtained:

$$p_3(x) = \begin{array}{cccc} 10.9244x^5 & +1.90949x^5 & +16.6881x^5 & +0.477986x^5 \\ -25x^4 & -10x^3 & -200x^2 & -30 \end{array}$$

And the killing costs, with this strategy for breaking the $30x^5$, would have been:

- $10.9244x^5 > 25x^4$ when $x > 2.28846$
- $1.90949x^5 > 10x^3$ when $x > 2.28845$
- $16.6881x^5 > 200x^2$ when $x > 2.28845$
- $0.477986x^5 > 30$ when $x > 2.28845$.

Thus, we would have $x > \max(2.28846, 2.28845, 2.28845, 2.28845) = \boxed{2.28846} \implies p_2(x) > 0$. *This bound is much better than the previous one, and the only change was the strategy we choose to break down the term* $30x^5$

Having shown these concepts, we can point out that any *strategy* of breaking positive terms and assigning positive terms to each one of the negative terms of a polynomial induces in fact a method for computing an upper bound for the maximum positive root of the polynomial. We cannot assign more than one positive term to different negatives terms, but we can split positive terms in pieces and use the pieces.

Definition 3.2.1 (Killing graph)**.** *Given a polynomial p as in 3.1, consider a undirected graph whose nodes are the terms of p after being split with any splitting strategy, and whose edges show the relation* is killed by *(or* kills*); with the associated cost being exactly the cost of the killing, defined as above. We call this a* killing graph*.*

It is clear that if all negative terms are killed by one positive term of higher degree and there are not positive terms killing more than one negative term, then the maximum of the costs in the edges is an upper bound for the positive roots of p.

Thus, any strategy of breaking positive terms and assigning to each negative term a higher-degree positive term (taking care that no positive term is assigned to two or more negative terms) induces a killing graph and vice-versa. A killing graph, in this way, is a *synonym* of a bounding method based on splitting positive terms and pairing negatives with positives.

The killing graphs associated to the previous examples we have shown are as follows:

Note. The colors in the following examples are meant just to help the easy recognition of positive and negative terms. Blue(red) terms are positive(negative), respectively.

$$p_1(x) = \overset{25}{\frown} \quad \overset{3}{\frown} \quad \overset{0.2}{\frown}$$
$$x^5 \quad -25x^4 \quad +200x^3 \quad -600x^2 \quad +600x \quad -120$$

$$p_2(x) = x^5 \quad +2x^5 \quad +3x^5 \quad +4x^5 \quad +28.5x^5 \quad +1.5x^5$$
$$\big| 25 \quad \big| 2.23 \quad \big| 4.05 \quad \big| 1.49$$
$$-25x^4 \quad -10x^3 \quad -200x^2 \quad -30$$

$$p_3(x) = 10.9244x^5 \quad +1.90949x^5 \quad +16.6881x^5 \quad +0.477986x^5$$
$$\big| 2.28846 \quad \big| 2.28845 \quad \big| 2.28845 \quad \big| 2.28845$$
$$-25x^4 \quad -10x^3 \quad -200x^2 \quad -30$$

This concept, which we have called *killing graph*, allows for easily representing a class of methods, including most of the methods mentioned in the previous section, and it also allows easily creating and representing new strategies, in a graphic way that helps to understand the underlying method by looking a graph.

Definition 3.2.2 (Splitting-and-Pairing methods)**.** *We call this class of methods as the* Splitting-and-Pairing *methods, and we can represent them by using this graph.*

As we said, **most of the current methods are, essentially, strategies to create the killing graph of the input polynomial**. Strategies for deciding which positive term we should break and in which parts, and how to assign the edges. In next subsections we show *examples and hints* intending to describe in an informal way how could those strategies be formulated for most of the current methods.

Before showing the informal descriptions of the methods, we want to recall that, as it was mentioned at the beginning of section 1, some of the methods (for example LMQ and FLQ) need to make a number of calculations proportional to the square of the number t of terms in the polynomials and some others (FL, LM, etc.) are linear in t. Linear methods LM and FL have been used in algebra systems like Mathematica and Sage; and have been the default method for root bounding in such systems.

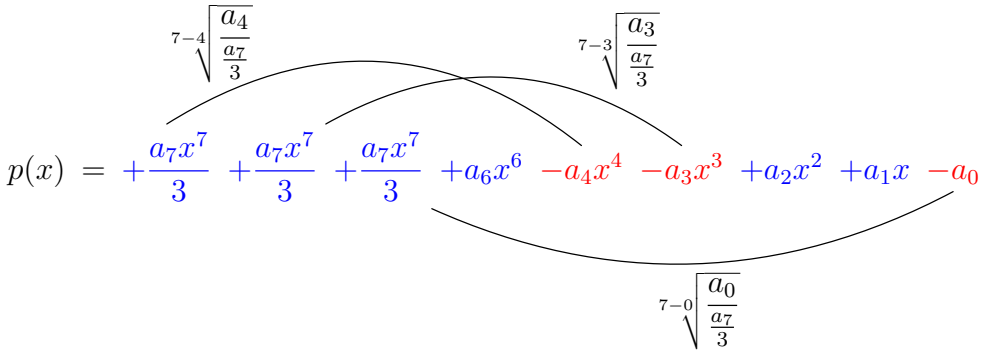
First observation. Observe that we could have, for a polynomial $p(x)$, a variant of killing graph, in which the nodes are polynomials $e_1(x), e_2(x), \ldots, e_i(x)$ instead of being terms of $p(x)$. We could only require that $p(x) - \sum_{1 \leq j \leq i} e_j(x) \geq 0$ and we could say, in a similar way, that an expression $e_1(x)$ *kills* an expression $e_2(x)$ when asymptotically $e_1(x) - e_2(x) > 0$, and that the associated cost is the largest real zero of $e_1(x) - e_2(x)$. Again, in that case, an upper bound for the roots of $p(x)$ would be the maximum cost associated to edges of the graph.

Second observation. Another simple, but crucial, fact is: if we have a killing graph in which *all the positive terms have been used*, or if we have an expressions-based killing graph (like in the previous observation) in which the sum of all the expressions is exactly $p(x)$, then *if the costs associated to the killings are all equal to some value r, then r is the maximum positive root of the polynomial $p(x)$*.

It is straightforward to prove this. Consider $p(x) = e_1(x) + e_2(x) + \cdots + e_i(x)$, $\forall j,\ e_j(x_0) = 0$, and $e_j(x) > 0,\ \forall x > x_0$, then x_0 is the maximum positive root of $\sum e_j$

3.2.1 Cauchy

Killing graph: Split the leading term into λ parts, and associate each negative term to one of these parts. In the example, $n = 7$ and $\lambda = 3$.

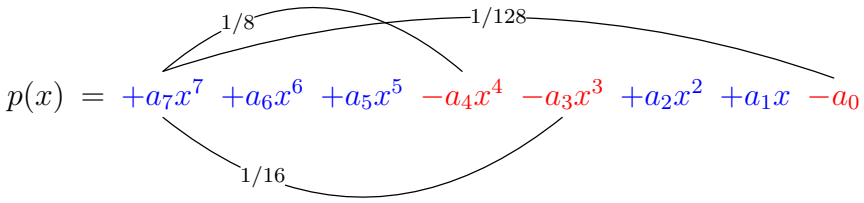


3.2.2 Lagrange

Putting the Lagrange bound in this way would be kind of *artificial*. We could in fact *force* it to fit in our framework, but this way is probably not the intuitive way in which the result has been created.

3.2.3 Kioustelidis

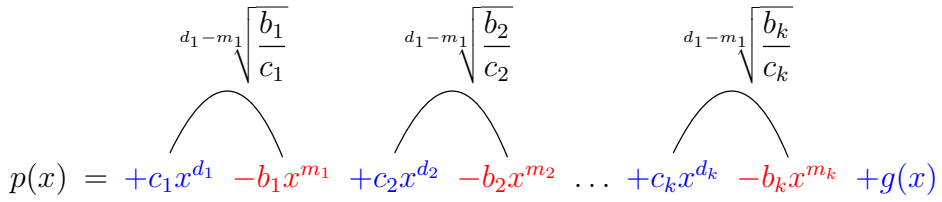
Killing graph: Kill the negative term a_kx^k using the $\frac{1}{2^{n-k}}$ part of the leading term.



Note. In this figure, for sake of clarity, we haven't followed strictly the definition of the killing graph representation. The values in the edges in here corresponds to *which fraction of the positive term is to be used in the killing* instead of the expected cost of the killing.

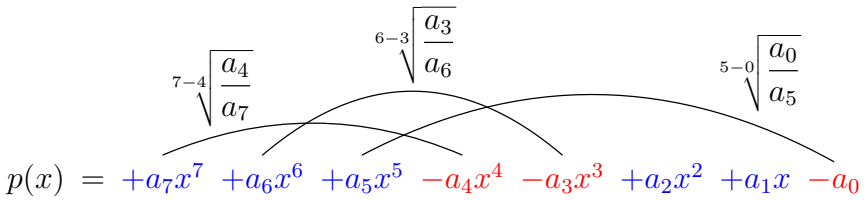
3.2.4 Stefanescu

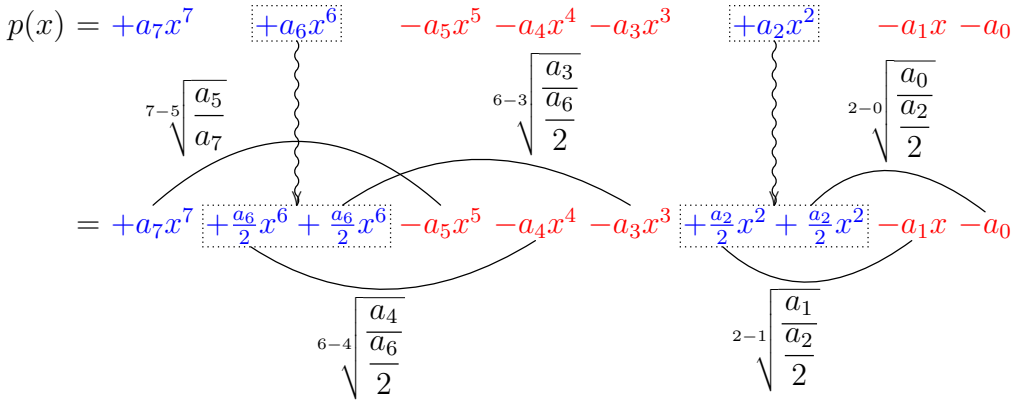
Killing graph: Kill the negative term $b_ix^{m_i}$ with the term $c_ix^{d_i}$.



3.2.5 First-λ (FL)

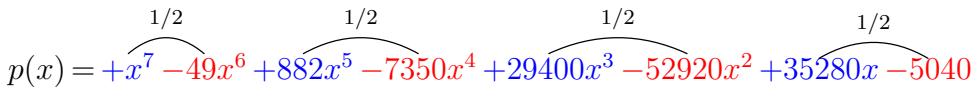
Killing graph: Apply the first-*lambda*-breaking operation. Assign the i-th negative term with the i-th positive term (in both cases, i-th when reading from left to right)

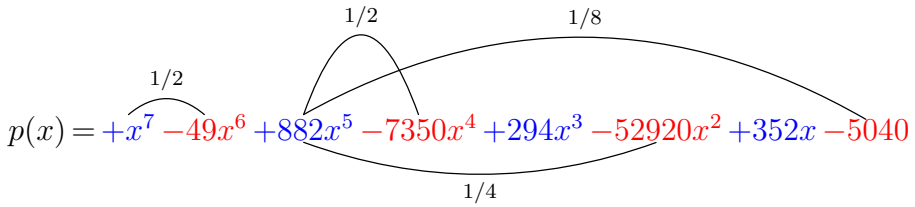




3.2.6 local-max (LM)

Killing graph: assign to $N(x)$ the half of the remaining fraction of the higher-degree positive term with biggest coefficient so far.





3.2.7 Three additional considerations

One consideration on using more involved sub-expressions

In this section we have presented a general framework which allows representing in a graphic way a class of methods that includes most of the methods mentioned at the beginning of section 1. We named the methods at this class as *Splitting-and-Pairing* methods (section 3.2.2).

On page 34, we made a short observation regarding the use of any sub-expression of $p(x)$ instead of *terms*. We could consider a variant of the killing graph in which the nodes are sub-polynomials of the input polynomial p. Of course, the problem with this approach would be the lack of a simple expression to find the cost of each edge in such a killing graph. This consideration explains a bit more this concept, which is not analyzed in this thesis. For example, if we had the polynomial

$$p(x) = +x^7 -49x^6 -882x^5 +7350x^4 +294x^3 -52920x^2 -352x -5040$$

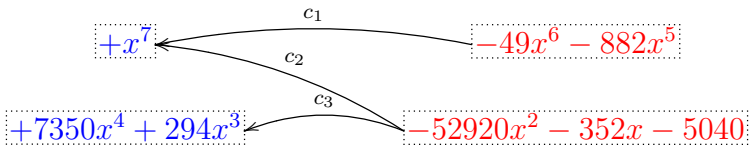
then we could try to kill *sets of terms* by using set of terms. We will propose in this example, just for the sake of simplicity, the sets of negative and positive terms to be the ones shown in the next figures. In order to assign which positive set of terms kills which negative set of terms, we could use at least two approaches:

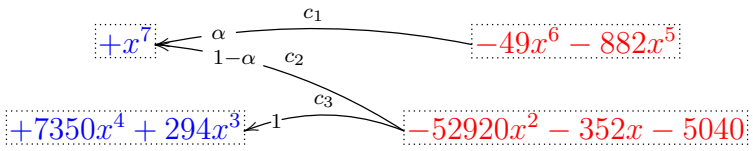
- every negative set must be killed by *exactly one* positive set, and there is no positive set killing more than one negative set.

  Figure 1 is a graph showing the relation *can be killed by*. The assignment of the killings, in this case, is trivial because there is only one possibility, since $+x^7$ cannot be used to kill more than one set of negative terms

- it is allowed to use *parts of different positive sets in order to kill negatives sets.*

For example: we could use a portion α, with $0 \leq \alpha \leq 1$, of x^7 to kill $-49x^6 - 882x^5$ and the remaining (i.e.: $1 - \alpha$) to kill $-52920x^2 - 352x - 5040$.



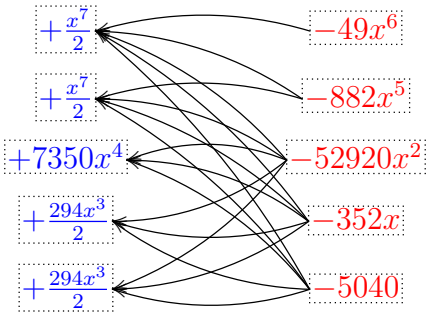


This approach (killing sets of terms with set of terms) is not going to be analyzed in the current work; and probably it would be as hard as the problem itself of finding the zeros of a given polynomial.

One consideration on computing the best assignment for a given fixed set of positive terms

If we are in the case in which one negative expression is killed by exactly one positive expression (and *expression*, right now, might refer to sets of terms or to terms, we just need to be able to compute the killing costs of the edges), we can consider the problem of computing the best possible assignment. Imagine, for the case in which expressions are terms, after having splitted the terms by any splitting strategy, a graph showing the relation *can be killed by*, with the cost of each killing associated to each edge, like in figure 3.

$$p(x) = +x^7 \quad -49x^6 - 882x^5 + 7350x^4 \quad +294x^3 \quad -52920x^2 - 352x - 5040$$
$$= +\tfrac{x^7}{2} + \tfrac{x^7}{2} - 49x^6 - 882x^5 + 7350x^4 + \tfrac{294x^3}{2} + \tfrac{294x^3}{2} - 52920x^2 - 352x - 5040$$



In this situation, we could be interested in computing *the best possible assignment of edges*. What would be that best assignment? It would be, of course, the one which minimizes the maximum cost present in the chosen edges. The cost of one assignment is the maximum cost of its edges. Then, the best assignment, is the one which *minimized the maximum*. There exists a famous problem named LBAP (Linear Bottleneck Assignment Problem). It can be stated as follows [1]:

> There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the maximum cost among the individual assignments is minimized.

So, we could assign infinite cost to the non-existent edges, we can consider that our tasks are the negative terms, and our agents are the pieces of positive terms. Thus, we could solve the problem of finding the best assignment by translating the problem into an LBAP problem. However, following an observation made at the beginning of this section, the strategy of splitting the terms is much more relevant in the resulting bounds (in the sense that it impacts much more) than using the best possible set of assignments.

In the works [39, 28] one can find approaches to solve the LBAP problem.

One consideration on computing a *lower* bound on positive roots, instead of an upper one

All the methods that we have exposed in this chapter and even the *class* of methods that we have introduced intend to compute an upper bound for the positive roots of a polynomial $p(x)$, defined as in 3.1. If the problem were to find a *lower* bound for the positive roots, we can proceed in at least two *apparently* different ways.

One way. If we replace x by $\frac{1}{x}$ in the expression 3.1, we will obtain an expression for $p\left(\frac{1}{x}\right)$. Each root α of this new expression corresponds to a root $\dfrac{1}{\alpha}$ of $p(x)$. By multiplying the new expression $p(\frac{1}{x})$ by x^n (where n is the degree of p), its positives zeros not change, since $x^n > 0$ when $x > 0$, and we recover the property of the obtained expression being a polynomial.

Thus, $\dfrac{1}{\alpha}$ is a root of $p(x) \iff \alpha$ is a root of $x^n p\left(\dfrac{1}{x}\right)$

The key observation that makes this computation very cheap is the following: $x^n p(\frac{1}{x})$ *is the polynomial* p *with the list of coefficients reversed.* This observation is at the same time straightforward and key in efficiency terms. It makes it be possible to compute the expression of $x^n p\left(\dfrac{1}{x}\right)$ in constant time[3] having $p(x)$ as input.

Another way. All the methods we have exposed kill a negative term with a positive one of *higher* degree. The *good* terms were the positive ones, and the *bad* terms were the negative ones. But we could have used the same idea in a different way. We said that a cost of an edge $A(x) \to B(x)$ is a x_0 such that $B(x) > A(x)$, for $x > x_0$. This is: a point from which $B(x)$ starts to be greater than $A(x)$. We could define it in a different way. We could say that the cost of a term $A(x)$ being killed by another term $B(x)$ is a point x_0 such that *until reaching it,* $B(x)$ *is greater than* $A(x)$. Thus, the calculation is the same as the one we were doing for the other case, but now the direction of the edges is opposite and now the resulting bound will be the minimum of the costs. For example, if we wanted to compute a lower bound for the roots of the following polynomial

$$p(x) = \underbrace{-x^5 + 25x^4}_{25} \; \underbrace{-200x^3 + 600x^2}_{3} \; \underbrace{-600x + 120}_{\frac{120}{600}}$$

we would have the costs:

- $600x < 120$ when $\frac{120}{600} > x$
- $200x^3 < 600x^2$ when $3 > x$
- $5x^5 < 25x^4$ when $25 > x$

[3] A constant amount of time, which does not depend of the size of the polynomial. We can associate to each list of coefficients one variable, whose possible values are 1 and -1. When the variable is set to 1, the structure represents the sequence obtained by reading its elements from left to right. When the variable is set to -1, the structure represents the sequence obtained by reading its elements from right to left. Thus, reversing this sequence can be achieved with just *one* operation: multiplying by -1 the associated variable.

$$\Longrightarrow \text{a lower bound for the minimum positive root is } \frac{120}{600}$$

Note that, in this case, we modified the requirement that the coefficient a_n of $p(x)$ need to be positive. In this case, the coefficient that we need to be positive is a_0.

3.3 A *GAP reduction* technique

In this section we present a new technique that improves the result produced by any one of the methods introduced in the previous section; in fact by any method induced by a killing graph (i.e.: any Splitting-and-Pairing method).

Let us perform a quick analysis of one of the already seen examples. Consider, for example, the following input polynomial and a killing graph resulting from some particular strategy of computing an upper bound for the roots of it.

$$p_1(x) = \overset{25}{\frown} \quad \overset{3}{\frown} \quad \overset{0.2}{\frown}$$
$$x^5 \ -25x^4 \ +200x^3 \ -600x^2 \ +600x \ -120$$

In this particular case, with this particular method of splitting and pairing, the resulting upper bound has been 25. Observe that we are not going to be able to improve this bound by *any* method that breaks positive terms and assign edges between positives and negatives terms (i.e.: by any Splitting-and-Pairing method). This polynomial only accepts a unique assignment of edges. The only thing that we could do is to break positive terms, and we would be killing the term $-25x^4$ with *a part* of x^5 instead of using *the whole* term, so the current associated cost 25 to the killing of $-25x^4$ would become even greater, and our bound would be worst. Thus, we cannot improve the bound 25 by using *any* of the methods presented in the previous sections, and even more: we cannot improve this bound by using *any method induced by a killing graph, any Splitting-and-Pairing method.* It does not matter how clever the method is, 25 is the best we can do. However, the maximum positive root of this polynomial is 12.64, about half of that value.

Let us go into a deeper analysis of the example. $x^5 > 25x^4$ when $x > 25$, but $200x^3 > 600x^2$ much earlier; when $x = 3$. Then, we can just break the term $200x^3$ in two parts, and use one of these to *help* the x^5 in the killing of $-25x^4$. The new graph would be:

$$p_1(x) = \underbrace{\overbrace{x^5 \ -25x^4 \ +Ax^3}}_{c_1} \ +\overbrace{(200-A)x^3 \ -600x^2}^{c_2} \ \overbrace{+600x \ -120}^{0.2}$$

Since our bound is the maximum of all the associated costs to the killings, in this situation our bound would be $\max(c_1, c_2, 0.2)$. Thus, the most desirable value for A would be the value that makes $c_1 = c_2$ (if it exists), and the value that makes the distance $|c_1 - c_2|$ as smaller as possible, otherwise.

In this particular case, in which all the exponents are consecutive, we could state a quadratic expression for c_1, and a linear one for c_2, both as functions of A, and obtain a cubic expression by equating $c_1(A) = c_2(A)$. In this way, we would be able to compute the best possible A, and all the associated killing costs by solving a cubic expression.

But this is a *very* particular case, since the exponents are consecutive numbers. Let us analyze the following: what does it mean that we want $c_1 = c_2$? The fact of $c_1 = c_2$ means that c_1 and c_2 equal the maximum positive root of the fournomial[4] $x^5 - 25x^4 + Ax^3 + (200 - A)x^3 - 600x^2 = x^5 - 25x^4 + 200x^3 - 600x^2$.

The maximum positive root of this last expression is approximately 13.4418. This value, for any *fewnomial*[5] and in particular for this fournomial, can be computed used the method of Sagraloff [88, 87]. The complexity of doing this is $(O(t^3 \log(d\tau) \log(d))$, where t is the number of terms of the polynomial (4, in this example) , τ is the bitsize of its maximum coefficient and d is its degree.[6] Then, we have that a bound for the maximum positive root of $p_1(x)$ would be $\max(13.4418, 13.4418, 0.2)$, instead of our original $\max(25, 3, 0.2)$. The bound has been reduced from the initial 25 to 13.4418. It is important at this point to stress again that this improvement was not going to be feasible through any of the current methods or through any other Splitting-and-Pairing method. The best assignment of the edges, for the best strategy for splitting the positive terms, would have produced a bound equal to 25.

So, let us revisit what we have just done: we started with a killing graph, in which each negative term was associated to a bigger order positive term. In this way, we started

[4] A polynomial with exactly 4 nonzero terms.

[5] A *fewnomial* is a sparse polynomial, a polynomial whose number of nonzero terms if much smaller than its degree.

[6] We will neglect the value of τ in the present work because we will not use the bit-complexity paradigm; this assumption yields a complexity of $O(t^3 \log^2(d))$ for this method.

with a set of polynomials with two terms (i.e.: binomials), in which the term with higher degree is positive and the other term is negative. We will say from now on, and following the same meaning that we have used previously, that the *cost* of a polynomial expression is the value of its maximum positive root. Instead of computing the costs of all these binomials and considering the maximum among them as the desired upper bound, we *joined two binomials together, obtaining a fournomial*. We had that x^5 was *killing* $-25x^4$, with cost 25, and we had that $200x^3$ was *killing* $-600x^2$, with cost 3. What we did was join together *both* positive terms and use them to kill together *both* negative terms. This is: we replaced the two expressions x^5-25x^4 (cost 25) and $200x^3-600x^2$ (cost 3) with one new expression, $x^5-25x^4+200x^3-600x^2$, which happens to have cost 13.4418. Our set of costs associated to expression suffered a change. The costs 25 and 3 disappeared and the new value 13.4418 appeared. The set of expressions suffered a change too. Initially it was formed only by binomials. After we joined two elements, the set is now formed by binomials *and one fournomial*.

At this point, we observe that we could repeat the same procedure. Join together two expressions and replace its associated costs by the cost of the resulting expression. The main difference with the first step would be that right now we could need to compute the cost of a sixnomial instead of a fournomial. We could perform, then, a second step in a similar way to the first one. We choose the expression with the highest associated cost, and some other expression; and we join them together. We replace the two costs of the two expressions by the cost of the new resulting one. The main point of this explanation is to highlight that the method that we are proposing can be applied more than once, in an iterative way. At each iteration we improve the previous result; *although, as it will be shown, the method produces better bounds than even the best current approaches, in almost all cases with just* one *iteration, and with* two *iterations the strategy shown to produce the best bounds, in all the cases; so, in general, there is no need of a big number of iterations*. Going back to the explanation: we start with expressions. At the beginning, those expressions are binomials. Each expression has an associated cost and a degree (because the expressions are polynomials). We pick the expression with the biggest cost and the expression *closest* to it in terms of *difference of degree*. If there is more than one closest expression, we pick the one with higher degree. After having picked both expressions, we join together them and compute a tight upper bound for the maximum

positive root of the resulting expression[7]. The two previous costs of the expression that we picked are replaced by the new cost that we have just computed. If we started with binomial expressions, the second iteration could need to compute the maximum positive root of a 4-nomial or of a 6-nomial, 6-nomial. The third iteration could need to compute the maximum positive root of a 4-nomial, a 6-nomial or a 8-nomial and so on.

The strategy we are proposing to pick the second expression is arbitrary. In fact, choosing expressions that are *close* in terms of degree may not be the best possible choice. It has been done this way because *in general, when coefficients do not differ by many orders of magnitude, a part of an expression with much lower degree is not going to help significatively.* For example: if we had contributed (in previous example) to the killing of $-25x^4$ with a part of the term $+600x$ (which is killing the term -120 with a cost much lower than the cost of killing $-600x^2$ so it could seems to be the right choice at a first glance), the obtained new associated costs to the killings of $-25x^4$ and -120 by the terms x^5 and $600x$ would have been 24.97. Our improvement would have been almost insignificant. We would have improved from 25 to 24.97. Choosing this way we improved from 25 to 13.45.

3.4 Implementation, complexity and experiments

As mentioned, our method improves the results produced by other methods. We implemented and used it to improve the bounds of the best linear methods (LM and FL, which we also implemented).

Our implementation receives as input a set of *expressions*, each one having associated a degree and a cost. It picks the expression with the maximum cost and the expression with minimum distance to it (in terms of difference of degree), choosing the one with higher degree in case of draw.

After having picked the two expressions $e_1(x)$ and $e_2(x)$, our implementation needs to compute a bound α for the maximum positive root of $E(x) = e_1(x) + e_2(x)$. This can be done using Sagraloff's method [88]. Note that, since $E(x)$ has exactly four monomials, t^3 is constant. This implies an astonishing complexity bound for the root isolation.

[7]For this purpose, we can the Sagraloff's method, or the method proposed by Alonso García and Galligo [20], or the elementary method which will be presented in chapter 4, which is particularly efficient for fewnomials.

After that, we remove from the input set the expressions $e_1(x)$ and $e_2(x)$, and we add the expression $E(x)$, with cost α. Our method returns the new set of expressions and associated costs. In this way, performing N iterations just consists in calling N times our method. The bound, of course, is the bigger associated cost in the set of expressions.

At this point, we can perform a complexity analysis. The first step is to run one of the LM or FL methods, which have cost $O(t)$, where t is the number of terms of the input polynomial. Then, we process the killing graph of the original assignment. Since this graph has $O(t)$ nodes and edges, the complexity of this step is $O(t)$. Finally, we run N times the cost-reduction step, which reduces to one root isolation using Sagraloff's method. The complexity of these N runs of the cost-reduction step is $O(N \log^2(d))$. Fixing $N = 2$, it turns out that the complexity of our method is $O(t + \log^2(d))$.

We have compared the results obtained with our method and with all the methods listed in section II, we executed one and two iterations of our improvement technique, above the linear methods FL and LM. The results are shown in table 3.1.

The polynomial classes we used for the tests are commonly used in the bibliography [15, 14, 16, 93]. They are: Laguerre polynomials, Chebyshev polynomials (first and second kind), Wilkinson polynomials, Mignotte polynomials, polynomials with random coefficients of bitsize 20 and 1000, monic polynomials with random coefficients of bitsize 20 and 1000, polynomials which are product of random roots of bitsizes 20 and 1000.

3.5 Chapter results and discussion

The results presented in Table 3.1 show that the proposed method produces bounds of better quality, even when compared against more computationally-expensive methods. With only one iteration, our method improves the LMQ and FLQ bounds in almost all cases and, with two iterations, it gets even better. In the few cases on which our method does not produce the best bound, it produces a bound very close to the best one. Akritas suggested [14] that taking $\min(\mathrm{LM}, \mathrm{FL})$ is going to produce the best bound in almost all cases. Here, based on our results, we can extend that statement by saying that taking the best of LM and FL, and iterating once or twice our method is going to produce a bound with even better quality than the one expected when using FQL or LMQ (which are, of course, better than LM and FL; but, unlike our method, much more expensive).

Table 3.1: Bounds with different methods, for degree up to 100

Poly	**min(FL, LM)**	**min(FLQ, LMQ)**	**min(FL-1, LM-1)**	**min(FL-2, LM-2)**	**MPR**
L(5)	25	25	13.4418	**12.6408**	12.6408
L(10)	100	100	54.8938	**36.0961**	29.9207
L(100)	10000	10000	6178.13	**4458.29**	374.984
$C_I(5)$	1.11803	1.11803	**0.951057**	**0.951057**	0.951057
$C_I(10)$	1.58114	1.58114	1.11803	**0.987688**	0.987688
$C_I(100)$	5	5	3.91569	**3.31346**	0.999877
$C_{II}(5)$	1	1	**0.866025**	**0.866025**	0.866025
$C_{II}(10)$	1.5	1.5	1.06254	**0.959493**	0.959493
$C_{II}(100)$	4.97494	4.97494	3.89592	**3.29663**	0.999516
W(5)	15	15	7.80143	**5**	5
W(10)	55	55	31.3643	**20.7397**	10
W(100)	5050	5050	3134.52	**2272.55**	100
M(5)	3.68403	3.68403	**3.5441**	**3.5441**	3.5441
M(10)	1.63069	1.63069	**1.5763**	**1.5763**	1.5763
M(100)	1.04073	1.04073	**1.03618**	**1.03618**	1.03618
$RC_{20}(5)$	1.42793	1.42793	1.42782	**1.12075**	0.974423
$RC_{20}(5)$	2.13681	2.13681	1.50889	**1.47612**	1.38725
$RC_{20}(5)$	3.85868	3.85868	**2.08626**	**2.08626**	−1.21363
$RC_{20}(10)$	3.19293	3.19293	2.82809	**2.63757**	2.16249
$RC_{20}(10)$	1.79403	1.79403	1.44004	**1.35923**	0.405669
$RC_{20}(10)$	2.32846	2.32846	2.1985	**1.88458**	1.77481
$RC_{20}(100)$	3.36871	3.36871	2.89419	**2.29942**	2.1978
$RC_{20}(100)$	1.53352	1.35048	1.35048	**1.33506**	−1.23838
$RC_{20}(100)$	1.55281	**1.4714**	1.54201	1.51195	1.15025
$RC_{1000}(5)$	2.01914	2.01914	1.59626	**1.51283**	1.35064
$RC_{1000}(5)$	1.60047	1.57203	1.46953	**1.03775**	1.02639
$RC_{1000}(5)$	1.48088	1.48088	1.3023	**1.15154**	0.929611
$RC_{1000}(10)$	2.21878	2.21878	2.15324	**1.87742**	1.69825
$RC_{1000}(10)$	1.98925	1.54865	1.89367	**1.52363**	−0.423095
$RC_{1000}(10)$	0.939259	0.939259	0.851629	**0.765633**	0
$RC_{1000}(100)$	100.124	100.124	99.228	**99.2274**	99.225
$RC_{1000}(100)$	1.45431	**1.18093**	1.44102	1.40765	1.02758
$RC_{1000}(100)$	2.11528	**1.81122**	1.91284	**1.81122**	1.19917
$MRC_{20}(5)$	655713	655713	**655712**	**655712**	655712
$MRC_{20}(5)$	1.46981	1.46981	**1.26429**	**1.26429**	1.19915
$MRC_{20}(5)$	86820	86820	**43421.5**	**43421.5**	43421.5
$MRC_{20}(10)$	2.27119	2.27119	2.1993	**1.8053**	1.40511
$MRC_{20}(10)$	741396	741396	**741395**	**741395**	741395
$MRC_{20}(10)$	0.91633	0.91633	0.853212	**0.739572**	0.489456
$MRC_{20}(100)$	1.62974	**1.10295**	1.19929	1.12893	0.997674
$MRC_{20}(100)$	1.18471×10^{06}	1.18471×10^{06}	789805	**592355**	592354
$MRC_{20}(100)$	643513	643513	**643512**	**643512**	643512
$MRC_{1000}(5)$	2.03063	2.03063	1.89945	**1.3016**	1.3016
$MRC_{1000}(5)$	1.1755	1.1755	**0.881019**	**0.881019**	0.579686
$MRC_{1000}(5)$	$\mathbf{6.63368 \times 10^{300}}$	$\mathbf{6.63368 \times 10^{300}}$	$\mathbf{6.63368 \times 10^{300}}$	$\mathbf{6.63368 \times 10^{300}}$	6.63368×10^{300}
$MRC_{1000}(10)$	2.38266	2.38266	2.38266	**2.13262**	1.86223
$MRC_{1000}(10)$	1.4982×10^{301}	1.4982×10^{301}	9.988×10^{300}	$\mathbf{7.49101 \times 10^{300}}$	7.491×10^{300}
$MRC_{1000}(10)$	1.72255	1.72255	1.20123	**0.739473**	−0.422388
$MRC_{1000}(100)$	$\mathbf{8.09301 \times 10^{299}}$	$\mathbf{8.09301 \times 10^{299}}$	$\mathbf{8.09301 \times 10^{299}}$	$\mathbf{8.09301 \times 10^{299}}$	8.09301×10^{299}
$MRC_{1000}(100)$	1.35824×10^{301}	1.35824×10^{301}	$\mathbf{6.79119 \times 10^{300}}$	$\mathbf{6.79119 \times 10^{300}}$	6.79119×10^{300}
$MRC_{1000}(100)$	$\mathbf{5.73536 \times 10^{300}}$	$\mathbf{5.73536 \times 10^{300}}$	$\mathbf{5.73536 \times 10^{300}}$	$\mathbf{5.73536 \times 10^{300}}$	5.73536×10^{300}
$PoR_{20}(5)$	1.12217×10^{06}	1.12217×10^{06}	$\mathbf{1.11904 \times 10^{06}}$	$\mathbf{1.11904 \times 10^{06}}$	880746
$PoR_{20}(5)$	484348	484348	**390055**	**390055**	293174
$PoR_{20}(5)$	1.59015×10^{06}	1.59015×10^{06}	1.30163×10^{06}	$\mathbf{1.13382 \times 10^{06}}$	961467
$PoR_{20}(10)$	2.00168×10^{06}	2.00168×10^{06}	1.84352×10^{06}	$\mathbf{1.55635 \times 10^{06}}$	933051
$PoR_{20}(10)$	1.66125×10^{06}	1.66125×10^{06}	1.38495×10^{06}	$\mathbf{1.29458 \times 10^{06}}$	948190
$PoR_{20}(10)$	1.08616×10^{06}	1.11661×10^{06}	962274	**898963**	857708
$PoR_{20}(100)$	8.94407×10^{06}	6.02572×10^{06}	6.02572×10^{06}	$\mathbf{4.61586 \times 10^{06}}$	1.02909×10^{06}
$PoR_{20}(100)$	5.04341×10^{06}	4.1885×10^{06}	4.3734×10^{06}	$\mathbf{3.51671 \times 10^{06}}$	1.03893×10^{06}
$PoR_{20}(100)$	6.67879×10^{06}	6.67879×10^{06}	5.94894×10^{06}	$\mathbf{4.2101 \times 10^{06}}$	1.03908×10^{06}
$PoR_{1000}(5)$	1.04501×10^{301}	1.04501×10^{301}	9.48173×10^{300}	$\mathbf{7.8146 \times 10^{300}}$	7.8146×10^{300}
$PoR_{1000}(5)$	1.77692×10^{301}	1.77692×10^{301}	$\mathbf{1.40765 \times 10^{301}}$	$\mathbf{1.40765 \times 10^{301}}$	1.05223×10^{301}
$PoR_{1000}(10)$	2.26062×10^{301}	2.26062×10^{301}	1.88024×10^{301}	$\mathbf{1.54407 \times 10^{301}}$	1.03782×10^{301}
$PoR_{1000}(10)$	2.10389×10^{301}	2.10389×10^{301}	1.49105×10^{301}	$\mathbf{1.36694 \times 10^{301}}$	7.69627×10^{300}
$PoR_{1000}(10)$	1.92673×10^{301}	1.92673×10^{301}	1.4175×10^{301}	$\mathbf{1.36263 \times 10^{301}}$	9.0238×10^{300}
$PoR_{1000}(100)$	4.44302×10^{301}	$\mathbf{2.87869 \times 10^{301}}$	3.71558×10^{301}	3.09712×10^{301}	1.05928×10^{301}
$PoR_{1000}(100)$	9.36467×10^{301}	9.36467×10^{301}	5.8998×10^{301}	$\mathbf{4.88169 \times 10^{301}}$	1.05282×10^{301}
$PoR_{1000}(100)$	6.41865×10^{301}	6.41865×10^{301}	5.78693×10^{301}	$\mathbf{4.75131 \times 10^{301}}$	1.06241×10^{301}

Columns • min(FL, LM): best of the two bounds obtained with First-λ and Local-max methods • min(FLQ, LMQ): best of the two bounds obtained with First-λ quadratic and Local-max quadratic methods • min(FL-1, LM-1): best of the two bounds obtained with one iteration of our method, starting with First-λ and Local-max methods • min(FL-2, LM-2): best of the two bounds obtained with two iterations of our method, starting with First-λ and Local-max methods • MPR: maximum positive root (some polynomials have no positive roots and the maximum *real* root is considered
Polynomials • L: Laguerre • C_I and C_{II}: Chebyshev (first and second kind) • W: Wilkinson • M: Mignotte • RC: Random coefficients • MRC: Monic with random coefficients • PoR: Product of Roots

3.6 Chapter conclusion

In this chapter we gave a general framework which allows seeing most of the current methods to compute an upper bound for the positive roots of a given polynomial as instances of one general idea. We also introduced a method to improve the quality of the bound for the maximum positive root of a polynomial computed by existing methods. Our technique showed to produce high quality bounds, and the complexity is linear (up to logarithmic factors). We obtained this result as a first step in trying to find possible optimizations for current real root isolation algorithms (VCA, VAS).

Chapter 4

Counting and isolating the real roots of univariate polynomials

In this chapter we survey the existing methods and related results to the problem of isolating the positive roots of a univariate polynomial, and we propose an elementary and intuitive method. Although the main property of the introduced method is its simplicity and the fact that it is based only on elementary concepts, it has shown also to be quite efficient in most of the input cases. It has shown to be faster than Sturm in almost all the testcases, with a running time only improved by VAS, which is the best known algorithm, and much more complicated from a conceptual point of view.

We also introduce in this chapter an adaptation of the theorem of Fourier and two ideas, one on the mentioned adaptation and the other on the method of Sturm.

This chapter is organized as follows: sections 1 to 4 survey underlying results, are mainly expository. They present the ideas on which the following sections are strongly based. Section 1 exposes Fourier's theorem (and proof). Section 2 formulates Sturm's idea. Section 3 exposes Vincent's theorem and method, leaving the proof of Vincent's theorem for Appendix A. Section 4 exposes the three main methods derived from Vincent's theorem. Section 5 introduces an adaptation of Fourier's theorem along with two ideas that appeared during the development of this work: one concerning the adaptation of Sturm's method for the case of fewnomials and the other concerning the adaptation of our *fewnomial version* of Fourier's theorem to produce an *exact* count of roots instead of a bound of this quantity. Section 6 introduces a method for isolating roots of polynomials which, while relying on elementary ideas, shows to have a quite good performance when

compared with other approaches. Section 7 shows the implementation of the mentioned method and a comparison of performance among the suggested method, Sturm's method and VAS method.

The main contributions on this chapter are:

- A version of the Fourier theorem which is better in the case of fewnomials, in the sense that it requires less computational effort.
- A complete survey with explanations of the main approaches for the root isolation problem.
- An algorithm based on elementary ideas, which turns out to be quite efficient in most of the testcases, when compared against Sturm and VAS.

Preliminary definition.

Definition 4.0.1 (Root isolation)**.** *To* isolate *the real roots of a given polynomial p is to compute a set of disjoint intervals, each containing exactly one root of p, together containing all the real roots of p.*

4.1 The theorem of Fourier

Fourier's original work has two parts [47, 46]. In the first one he assumes that the input equation and all its derivatives do not share any root. In the second part he analyzes the case in which there are shared roots. We will formulate a version of the theorem that includes both parts, and show a slight adaptation of Fourier's proof.

Theorem 3. *Let $N(x)$ be the number of sign changes in the sequence $f(x)$, $f'(x)$, ..., $f^{(n)}(x)$, where f is a polynomial of degree n. If $a < b$ and $f(a) \neq 0$, $f(b) \neq 0$, then $N(a) \geq N(b)$, and the number of roots of f (multiplicities counted) between a and b, does not exceed $N(a) - N(b)$. Moreover, the number of roots can differ from $N(a) - N(b)$ by an even number only.*

Proof. Imagine the list of numbers $f(x)$, $f'(x)$, ..., $f^{(n)}(x)$, with x moving from a to b. The number $N(x)$ varies only if x passes through a root of the polynomial $f^{(m)}$ for some $m \leq n$ (in fact, given that $f^{(n)}(x)$ is a constant, we have that is for some $m < n$).

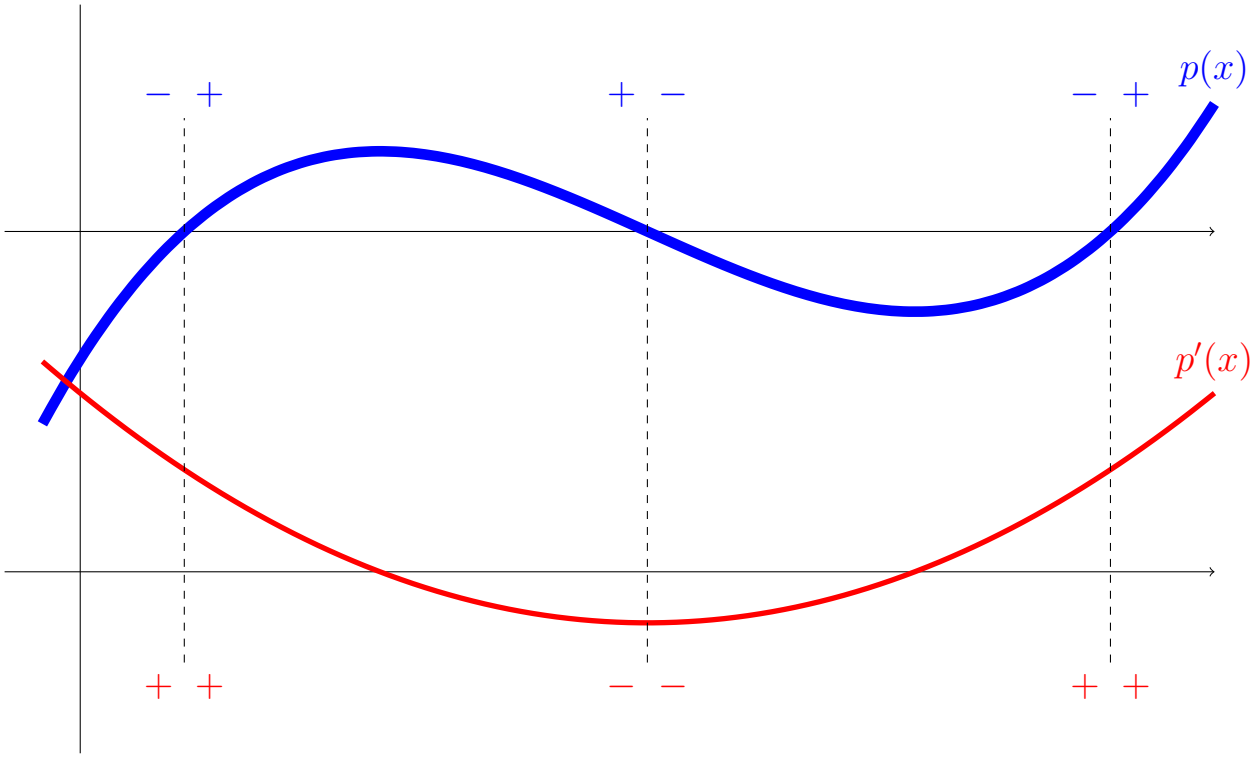


Figure 4.1: Three illustrative examples of the fact that a polynomial $p(x)$, in a neighborhood of a root α, have different sign than $p'(x)$ for $x < \alpha$ and same sign than $p'(x)$ for $x > \alpha$

In what follows, we will use the following simple **remark:** *the signs of any polynomial p and its derivative p', in a neighborhood of a root α of p, are different for $x < \alpha$ and equal for $x > \alpha$* (see figure 4.1).

Now, coming back to the proof of the theorem, consider first the case in which x passes through a root α of multiplicity r of $f(x)$. In a neighborhood of α, for $x < \alpha$, $f^{(r-1)}(x)$ must has different sign than $f^{(r)}(x)$ due to the previous remark. In the same way, $f^{(r-2)}(x)$ must has different sign than $f^{(r-1)}(x)$, $f^{(r-3)}(x)$ must has different sign than $f^{(r-2)}(x)$ and so on. Thus, the signs of the values in the list $f^{(r)}(x), f^{(r-1)}, \ldots, f(x)$, with x in a neighborhood of α, $x < \alpha$, alternates (see Fig. 4.2).

$$\begin{array}{rcccl}
\operatorname{sgn}(f^{(r-1)}(x)) & = & -\operatorname{sgn}(f^{(r)}(x)) & & \\
\operatorname{sgn}(f^{(r-2)}(x)) & = & -\operatorname{sgn}(f^{(r-1)}(x)) & = & \operatorname{sgn}(f^{(r)}(x)) \\
 & \vdots & & \vdots & \\
\operatorname{sgn}(f(x)) & = & -\operatorname{sgn}(f'(x)) & = & (-1)^r \operatorname{sgn}(f^{(r)}(x))
\end{array}$$

Figure 4.2: Signs of $f(x), f'(x), \ldots, f^{(r-1)}(x)$ in a neighborhood of α, with $x < \alpha$

We also know that in a neighborhood of α, with $x > \alpha$, the signs of the values

$f^{(r)}(x), f^{(r-1)}(x), \ldots, f(x)$ are equal. Thus, we have that when x pass through a root α of multiplicity r of f, the number of sign variations in $f(x), f'(x), f''(x), \ldots, f^{(r-1)}(x), f^{(r)}(x)$ decreases by r. This is: $N(x)$ decreases by r (see Fig. 4.3).

	$< \alpha$	α	$> \alpha$
$f(x)$	$\ldots$	0	$+$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f^{(r-2)}(x)$	$+$	0	$+$
$f^{(r-1)}(x)$	$-$	0	$+$
$f^{(r)}(x)$	$+$	$+$	$+$

r sign changes; 0 sign changes

$N(x)$ decreases by r in a neighborhood of α

Figure 4.3: $N(x)$ decreases by r in the neighborhood of the root α of multiplicity r

Consider now the case in which x passes through a root α of multiplicity r of $f^{(i)}(x)$, such that $f^{(i-1)}(\alpha) \neq 0$, $f^{(i)}(\alpha) = f^{(i+1)}(\alpha) = \cdots = f^{(i+r-1)}(\alpha) = 0$, and $f^{(i+r)}(\alpha) \neq 0$. As in the previous case we have that, in a neighborhood of α, for $x < \alpha$, $f^{(i+r-1)}(x)$ must have different sign than $f^{(i+r)}(x)$; $f^{(i+r-2)}(x)$ must has different sign than $f^{(i+r-1)}(x)$; $\ldots$; $f^{(i)}(x)$ must has different sign than $f^{(i+1)}(x)$. Thus, the signs of the values in the list $f^{(i+r)}(x), f^{(i+r-1)}(x), \ldots, f^{(i)}(x)$, with x in a neighborhood of α, $x < \alpha$, alternates (see Fig. 4.4)

$$\begin{array}{rcrcl}
\operatorname{sgn}(f^{(i+r-1)}(x)) & = & -\operatorname{sgn}(f^{(i+r)}(x)) & & \\
\operatorname{sgn}(f^{(i+r-2)}(x)) & = & -\operatorname{sgn}(f^{(i+r-1)}(x)) & = & \operatorname{sgn}(f^{(i+r)}(x)) \\
 & \vdots & & \vdots & \\
\operatorname{sgn}(f^{(i)}(x)) & = & -\operatorname{sgn}(f^{(i+1)}(x)) & = & (-1)^r \operatorname{sgn}(f^{(i+r)}(x))
\end{array}$$

Figure 4.4: Signs of $f^{(i+r-1)}(x), f^{(i+r-2)}(x), \ldots, f^{(i)}(x)$ in a neighborhood of α, with $x < \alpha$

As before, we also know that in a neighborhood of α, with $x > \alpha$, the signs of the values $f^{(i+r)}(x), f^{(i+r-1)}(x), \ldots, f^{(i)}(x)$ are equals. There are four possible situations, depending on if r is even or odd, and on if $\operatorname{sgn}(f^{(i-1)}(x)) = \operatorname{sgn}(f^{(i+r)}(x))$ or $\operatorname{sgn}(f^{(i-1)}(x)) \neq \operatorname{sgn}(f^{(i+r)}(x))$. In figure 4.5 it is easy to see that, when r is even, the number of sign variations $N(x)$ decrease by r and, when r is odd, it decreases either by $r+1$ or by $r-1$, depending on whether $\operatorname{sgn}(f^{(i-1)}(x)) = \operatorname{sgn}(f^{(i+r)}(x))$ or $\operatorname{sgn}(f^{(i-1)}(x)) \neq \operatorname{sgn}(f^{(i+r)}(x))$.

	$< \alpha$	α	$> \alpha$
$f^{(i-1)}(x)$	$+$	$+$	$+$
$f^{(i)}(x)$	$+$	0	$+$
$f^{(i+1)}(x)$	$-$	0	$+$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f^{(i+r-4)}(x)$	$+$	0	$+$
$f^{(i+r-3)}(x)$	$-$	0	$+$
$f^{(i+r-2)}(x)$	$+$	0	$+$
$f^{(i+r-1)}(x)$	$-$	0	$+$
$f^{(i+r)}(x)$	$+$	$+$	$+$
	r sc		0 sc
	$N(x)$ decreases by r		

(a) $\operatorname{sgn}(f^{(i-1)}) = \operatorname{sgn}(f^{(i+r)})$, r even

	$< \alpha$	α	$> \alpha$
$f^{(i-1)}(x)$	$-$	$-$	$-$
$f^{(i)}(x)$	$+$	0	$+$
$f^{(i+1)}(x)$	$-$	0	$+$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f^{(i+r-4)}(x)$	$+$	0	$+$
$f^{(i+r-3)}(x)$	$-$	0	$+$
$f^{(i+r-2)}(x)$	$+$	0	$+$
$f^{(i+r-1)}(x)$	$-$	0	$+$
$f^{(i+r)}(x)$	$+$	$+$	$+$
	$r+1$ sc		1 sc
	$N(x)$ decreases by r		

(b) $\operatorname{sgn}(f^{(i-1)}) \neq \operatorname{sgn}(f^{(i+r)})$, r even

	$< \alpha$	α	$> \alpha$
$f^{(i-1)}(x)$	$+$	$+$	$+$
$f^{(i)}(x)$	$-$	0	$+$
$f^{(i+1)}(x)$	$+$	0	$+$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f^{(i+r-4)}(x)$	$+$	0	$+$
$f^{(i+r-3)}(x)$	$-$	0	$+$
$f^{(i+r-2)}(x)$	$+$	0	$+$
$f^{(i+r-1)}(x)$	$-$	0	$+$
$f^{(i+r)}(x)$	$+$	$+$	$+$
	$r+1$ sc		0 sc
	$N(x)$ decreases by $r+1$		

(c) $\operatorname{sgn}(f^{(i-1)}) = \operatorname{sgn}(f^{(i+r)})$, r odd

	$< \alpha$	α	$> \alpha$
$f^{(i-1)}(x)$	$-$	$-$	$-$
$f^{(i)}(x)$	$-$	0	$+$
$f^{(i+1)}(x)$	$+$	0	$+$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f^{(i+r-4)}(x)$	$+$	0	$+$
$f^{(i+r-3)}(x)$	$-$	0	$+$
$f^{(i+r-2)}(x)$	$+$	0	$+$
$f^{(i+r-1)}(x)$	$-$	0	$+$
$f^{(i+r)}(x)$	$+$	$+$	$+$
	r sc		1 sc
	$N(x)$ decreases by $r-1$		

(d) $\operatorname{sgn}(f^{(i-1)}) \neq \operatorname{sgn}(f^{(i+r)})$, r odd

Figure 4.5: Decreasing on the amount of sign changes (sc) in the neighborhood of a root α, depending on the parity of r and on if $\operatorname{sgn}(f^{(i-1)}(x)) = \operatorname{sgn}(f^{(i+r)}(x))$, assuming that $f^{(i+r)}(x)$ is positive. If it were negative, the tables would be exactly the opposite of these.

So, in any case, $N(x)$ decreases by an even amount. Thus, we have proved the following two facts:

- *when x passes through a root of multiplicity r of f, $N(x)$ decreases by r.*
- *when x passes through a root of multiplicity r of $f^{(i)}$, with $i > 0$, $N(x)$ decreases by an even amount.*

The theorem follows from these two facts.

□

4.1.1 Descartes' rule of signs is trivially implied by Fourier's theorem

As we said in chapter 2, the original Descartes' rule of signs says:

1. A real polynomial has *no more* positive roots than alternations of signs between two consecutive coefficients.
2. A real polynomial has *no more* negative roots than permanences of signs between two consecutive coefficients.

Proof. Part (1) follows trivially from Fourier's theorem, by considering that x is moving in $[0, \infty)$.

For part (2) we need to recall the interpretation of *alternations* and *permanences* of sign explained in 2.1.1. With that interpretation, we have that the number of permanences of signs in $p(x)$ is the number of alternations in $p(-x)$. By part (1), we know that the number of positive roots of $p(-x)$ (which is the number of negative roots of p) is lower than or equal to the number of sign alternations on that expression. Now we know that the number of sign alternations in that expression is precisely the number of sign permanences on $p(x)$. Thus, we have the part (2).

□

Remark. Jacobi made a useful observation. The expression $\dfrac{x-a}{b-x}$ varies from 0 to ∞ when x varies from a to b. The expression $\dfrac{a+by}{1+y}$ varies from a to b when y varies from 0 to ∞. Then, given a polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ and an interval

(a, b), we can construct an equation $q(x)$, whose positive roots are in correspondence with the roots of $p(x)$ in the interval (a, b), by defining

$$q(x) = p\left(\frac{a + bx}{1 + x}\right)$$

We will have that

$$q(x) = a_n \frac{(a + bx)^n}{(1 + x)^n} + a_{n-1} \frac{(a + bx)^{n-1}}{(1 + x)^{n-1}} + \cdots + a_1 \frac{(a + bx)^1}{(1 + x)^1} + a_0 \frac{(a + bx)^0}{(1 + x)^0}$$

The positive roots of this expression remain the same if we multiply it by $(1 + x)^n$, since it is a positive quantity for $x > 0$. Moreover, $(1 + x)^n q(x)$ *is* a polynomial. Then, Descartes' rule of sign applies to it. Thus, in this way, Descartes' rule of sign can be applied to an interval (a, b) instead of to the complete x axis.

4.2 Sturm

Sturm proposed, in [90, 91], a theorem *strongly* based on Fourier's theorem, with a proof that mimics Fourier's proof, but with a very ingenious slight modification that makes a *big* difference between the two theorems. The list of functions on which is computed the number of sign changes used by Fourier for an input polynomial p is $p, p', p'', p''', \ldots, p^n$. As we see in the proof of Fourier's theorem, when x moves in the $x-$axis from left to right, when it passes through a root of some p^i $(i > 0)$, it is possible for the number of sign changes to decrease by 2 or to remain the same. Sturm's idea removes the case in which the number of sign changes decreases by 2. It is not possible, with Sturm's list, to have such decreases. The only moment at which the number of sign changes decreases is when x pass through a root of f. Let we see Sturm's idea in more detail.

4.2.1 Sturm theorem

Definition 4.2.1 (Sturm sequence)**.** *Given a polynomial $f(x)$, consider the polynomials $f_0(x) = f(x)$ and $f_1(x) = f'(x)$ and consider the polynomial $f_i(x)$ $(i > 1)$ to be $-1 \times$ [the remainder of the division of $f_{i-2}(x)$ by $f_{i-1}(x)$]. This is, the f_i's are such that*

$$
\begin{array}{rclcrcl}
f_0 &=& q_1 f_1 - f_2 & \Longrightarrow & f_2 &=& q_1 f_1 - f_0 \\
f_1 &=& q_2 f_2 - f_3 & \Longrightarrow & f_3 &=& q_2 f_2 - f_1 \\
 & \vdots & & & & \vdots & \\
f_{n-2} &=& q_{n-1} f_{n-1} - f_n & \Longrightarrow & f_n &=& q_{n-1} f_{n-1} - f_{n-2} \\
f_{n-1} &=& q_n f_n & & & &
\end{array}
$$

The sequence $f_0(x), f_1(x), \ldots, f_{n-1}(x), f_n(x)$ *is called the* Sturm sequence *of* f*.*

Theorem 4 (Sturm)**.** *Let* f *be a polynomial, and let* $w(x)$ *be the number of sign changes in its Sturm sequence, when evaluated at* x*. The number of roots that* f *has in the interval* (a, b)*, provided that* $f(a) \neq 0$*,* $f(b) \neq 0$ *and* $a < b$*, is equal to* $w(a) - w(b)$*.*

Proof. Let us assume, for now, that there are not shared roots between consecutive functions of the Sturm sequence of f. Imagine x moving from a to b.

Let us see what happens when x passes through a root α of some function f_i, of the Sturm sequence of f, with $i > 0$. By definition of the Sturm sequence, we have that

$$f_{i-1}(\alpha) = q_i(\alpha) f_i(\alpha) - f_{i+1}(\alpha)$$

We are under the supposition that there are not shared roots between any two consecutive functions f_t and f_{t+1} of the Sturm sequence of f. Thus, $f_{i-1}(\alpha) = -f_{i+1}(\alpha) \neq 0$ when $f_i(\alpha) = 0$ and, in a neighborhood of α, the signs of $f_{i-1}(x)$ and $f_{i+1}(x)$ are opposite. So, it does not matter if $f(x)$ changes from positive to negative or from negative to positive, $w(x)$ will not change when x pass through a root of some f_i with $i > 0$ (see figure 4.6).

	$< \alpha$	α	$> \alpha$
f_{i-1}	$+$	$+$	$+$
f_i	$+$	0	$-$
f_{i+1}	$-$	$-$	$-$
	1 variation		1 variation

	$< \alpha$	α	$> \alpha$
f_{i-1}	$+$	$+$	$+$
f_i	$-$	0	$+$
f_{i+1}	$-$	$-$	$-$
	1 variation		1 variation

Figure 4.6: The number of sign variations in the list is not affected when x pass through a root of f_i, with $i > 0$

Let us see now what happens when x pass through a root α of f_0. Given that f_1 is the derivative of f_0, in a neighborhood of α the signs of f_0 and f_1 must pass from being

different to equal, making the number of sign changes in the sequence decrease by 1. This is: making $w(x)$ to decrease by 1.

Now let us consider the case in which there exist shared roots between some pair of functions f_i and f_{i+1}. Let us make two small observations. First: the last function f_n in the Sturm sequence of f is the greatest common divisor of f and f'. The creation of the Sturm sequence follows the same steps than Euclid's algorithm for computing the greatest common divisor. It just multiply by -1 the remainder, but given that we are dealing with polynomials and not with numbers, this does not affect the fact that f_n is a common divisor of f and f'. Second: if α is root of two consecutive functions of the Sturm sequence of f, then α is root of *all* the functions of that sequence. In fact, if $f_i(\alpha) = f_{i+1}(\alpha) = 0$, by the definition of Sturm sequence, we have:

$$\begin{array}{rcl@{\qquad}rcl}
f_{i+2}(\alpha) & = & q_{i+1}(\alpha)f_{i+1}(\alpha) - f_i(\alpha) & \Longrightarrow & f_{i+2}(\alpha) & = & 0 \\
f_{i+3}(\alpha) & = & q_{i+2}(\alpha)f_{i+2}(\alpha) - f_{i+1}(\alpha) & \Longrightarrow & f_{i+3}(\alpha) & = & 0 \\
 & \vdots & & & & \vdots & \\
f_n & = & q_{n-1}(\alpha)f_{n-1}(\alpha) - f_{n-2}(\alpha) & \Longrightarrow & f_n & = & 0
\end{array}$$

and, in a similar way

$$\begin{array}{rcl@{\qquad}rcl}
f_{i-1}(\alpha) & = & q_i(\alpha)f_i(\alpha) - f_{i+1}(\alpha) & \Longrightarrow & f_{i-1}(\alpha) & = & 0 \\
f_{i-2}(\alpha) & = & q_{i-1}(\alpha)f_{i-1}(\alpha) - f_i(\alpha) & \Longrightarrow & f_{i-2}(\alpha) & = & 0 \\
 & \vdots & & & & \vdots & \\
f(\alpha) & = & q_1(\alpha)f_1(\alpha) - f_2(\alpha) & \Longrightarrow & f(\alpha) & = & 0
\end{array}$$

Thus, it cannot happen that α is root of *just two consecutive functions*. If it happens, then α is root of *all* the functions of the Sturm sequence.

Having pointed these two facts, let us imagine that the root α is a common root of all the f_i's. Consider the Sturm sequence $g_0, g_1, \ldots, g_{n'}$ of the function $\frac{f}{\gcd(f,f')}$. As we pointed, it is easy to see that that its length is equal to the length of the Sturm sequence of f and that $f_i(x) = g_i(x)\gcd(f, f')(x)$ for all i. So, whenever x is not a root of $\gcd(f, f')$ (i.e.: a root of f_n), we can obtain the Sturm sequence of f evaluated at x by multiplying each element of the Sturm sequence of g evaluated at x by the number $\gcd(f, f')(x)$. Thus, the number of sign changes in both cases is the same. Thus, the number of sign changes in the Sturm sequence of f evaluated at x is the same than the number of sign

changes in the Sturm sequence of $g = \frac{f}{\gcd(f,f')}$, which has the same roots than f, but all with multiplicity one. Thus, the decreasing in $w(x)$ behaves exactly in the same way than in the case that we have already analyzed. Then, the theorem is proved. □

4.2.2 A consequent root isolation method

Based on Sturm's theorem, one can derive in a very intuitive way an algorithm to isolate the roots of the polynomial f contained in the interval (a, b). We compute the number of sign changes of the Sturm sequence of f, when evaluated both at a and at b. If these two numbers differs by 0 or by 1, the algorithm stops. If they differs by 2 or more, we compute the two smaller isolations of the same polynomial, but in the intervals $(a, \frac{a+b}{2})$ and $(\frac{a+b}{2}, b)$.

4.3 Vincent's theorem

We stated in Chapter 2, page 16, the original formulation of Vincent's theorem, quoting literally it from the original work, in French. We also pointed out an important sentence of that formulation that has been omitted in many reformulations of the theorem, and reformulated the theorem in a way that does not require the reader to be highly familiarized with Lagrange's previous work.

Alesina and Galuzzi, in [19], show four different approaches to prove Vincent's theorem. There exist one other approach presented by Ostrowski [80], that they missed to include in their list of approaches, and they added a reference later in their *addendum* [18]. The approaches shown by Alesina and Galuzzi are: Vincent's proof, Uspensky's proof, Chen's proof, and one new approach authored by them. To avoid losing the main thread of this chapter, we left the exposition of the proof of Vincent's theorem for Appendix A, where we sketch Alesina and Galuzzi's proof.

4.3.1 Vincent's method

As we explained in the preliminaries of this chapter, a polynomial real root isolation algorithm takes as input a polynomial $p(x)$ and computes a sequence of disjoint intervals, each containing exactly one root of $p(x)$, together containing all the real roots of $p(x)$.

For now, we will focus on the problem of *counting* the number of positive roots. The adaptation of this idea to the problem of *isolating* them will be the focus of next subsections.

If we are given a polynomial *without multiple roots*

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,$$

and we want to know the number of positive roots it has, the first thing that we can think to do is: apply Descartes' rule of signs. It might help. If the number of sign variations in the list of coefficients were 0 or 1, we would know that p has 0 or 1 positive roots, respectively. Let us think in the case in which the number of sign variations is larger than 1. Notice at first that the roots of p can be greater than 1, equal to 1 or less than 1. The roots bigger than 1 can be written as $1+x$, with $x > 0$. The roots less than 1 can be written as $\frac{1}{1+x}$, with $x > 0$. Accordingly, we make two transformations to the expression of $p(x)$: $x \leftarrow x+1$ and $x \leftarrow \frac{1}{x+1}$; both of them allow pretty efficient implementations, as it will be shown in next subsections.

$$\begin{aligned} p\left(\tfrac{1}{1+x}\right) &= a_n \left(\tfrac{1}{1+x}\right)^n + a_{n-1} \left(\tfrac{1}{1+x}\right)^{n-1} + \cdots + a_0 \\ p\left(1+x\right) &= a_n \left(1+x\right)^n + a_{n-1} \left(1+x\right)^{n-1} + \cdots + a_0 \end{aligned}$$

$\frac{1}{1+x}$ varies over $(0,1)$ when x varies over $(0,\infty)$. $1+x$ varies over $(1,+\infty)$ when x varies over $(0,\infty)$. $p\left(1+x\right)$ is already a polynomial. $p\left(\frac{1}{1+x}\right)$ is an equation whose positive roots are in correspondence with the roots of p at the interval $(0,1)$, but is not a polynomial. By multiplying it by $(1+x)^n$ we preserve its positive roots and make it become a polynomial again. Through these two operations we obtain, from the expression of p, two new polynomials, namely p_1 and p_2, whose positive roots corresponds to the roots of p in $(0,1)$ and $(1,+\infty)$, respectively.

$$\begin{array}{rcccl} p_1(x) & = & (1+x)^n p\left(\frac{1}{1+x}\right) & = & (1+x)^n \left(a_n \left(\frac{1}{1+x}\right)^n + \cdots + a_0\right) \\ p_2(x) & = & p\left(1+x\right) & = & a_n \left(1+x\right)^n + a_{n-1} \left(1+x\right)^{n-1} + \cdots + a_0 \end{array}$$

After having performed this kind of *split* operation, we can apply again Descartes' rule of signs to each one of the two resulting polynomials and, if the number of sign changes in their lists of coefficients were 0 or 1, we would have that the number of roots of p in

Algorithm 1 Vincent's method for counting roots

```
1: procedure VincentCount(p)                          ▷ p is squarefree, of degree n
2:     sc ← number of sign changes in p
3:     if sc = 0 or sc = 1 then
4:         return sc
5:     p_01(x) ← (x + 1)^n p(1/(x+1))
6:     p_1∞(x) ← p(x + 1)
7:     if p(1) = 0 then
8:         return VincentCount(p_01) + VincentCount(p_1∞) + 1
9:     else
10:        return VincentCount(p_01) + VincentCount(p_1∞)
```

the intervals $(0, 1)$ and $(1, +\infty)$ is 0 or 1 respectively. If the number of sign variations in their lists of coefficients were bigger than one, we could apply the process again and again until we reach expressions with 0 or 1 sign variation. As we can see, the computation in this method is structured like a binary tree. Every time a polynomial has more than 1 sign variation in its list of coefficients, we expand it into other two polynomials. One natural question is: *is this process going to end eventually? are we always going to reach an equation in which the number of sign changes is* 0 *or* 1*?*. The answer is: *yes! we are!*. And Vincent's theorem can be used to prove this assertion. Algorithm 1 shows an implementation of this concept.

Proof that this method always terminates. Along this proof, we will consider *transformations*. A transformation of the form $x \leftarrow 1 + x$ transforms the polynomial $p(x)$ into the polynomial $p(1 + x)$. A transformation of the form $x \leftarrow \frac{1}{1+x}$ transforms the polynomial $p(x)$ into the polynomial $(1+x)^n p(\frac{1}{1+x})$, where n is the degree of p. Similarly, a transformation of the form $x \leftarrow \frac{1}{x}$ transforms the polynomial $p(x)$ into the polynomial $x^n p(\frac{1}{n})$.

Consider a sequence S of transformations of the form $x \leftarrow 1 + x$ or $x \leftarrow \frac{1}{1+x}$. We can add, after the last transformation in S, one new transformation with the form $x \leftarrow \frac{1}{x}$, obtaining a new sequence of transformations S'. The number of sign variations in the list of coefficients after the transformations given by S' is the same than after the transformations given by S.

A number a of transformations of the form $x \leftarrow 1 + x$ followed by one of the form $x \leftarrow \frac{1}{1+x}$ are equivalent to $x \leftarrow a + x$ followed by $x \leftarrow \frac{1}{1+x}$, which is equivalent to $x \leftarrow a + \frac{1}{x}$, followed by $x \leftarrow 1 + x$. In the following picture, the operator $\star$ means *followed*

by. For example, if we write $(x \leftarrow a_1 + x) \star (x \leftarrow a_2 + x)$ we mean the transformation that results by *first* substitute $x \leftarrow a_1 + x$ in the input polynomial and *then* substitute $x \leftarrow a_2 + x$ in the resulting expression.

$$\begin{aligned} & \overbrace{(x \leftarrow 1 + x) \star (x \leftarrow 1 + x) \star \cdots \star (x \leftarrow 1 + x)}^{a \text{ transformations}} \star (x \leftarrow \tfrac{1}{1+x}) \\ = \; & (x \leftarrow a + x) \star (x \leftarrow \tfrac{1}{1+x}) \\ = \; & (x \leftarrow a + \tfrac{1}{x}) \star (x \leftarrow 1 + x) \end{aligned}$$

We can rewrite S' using this rule. If S' *starts* with a transformations of the form $(x \leftarrow 1 + x)$, followed by one transformation of the form $x \leftarrow \frac{1}{1+x}$, we replace them with the two transformations $(x \leftarrow a + \frac{1}{x}) \quad (x \leftarrow 1 + x)$. We continue doing the same process, but starting at the $(x \leftarrow 1 + x)$ that we have just written. In this way we will keep writing transformations of the form $x \leftarrow a + \frac{1}{x}$ until we reach the last group of consecutive transformations of the form $x \leftarrow 1 + x$, which is followed by $x \leftarrow \frac{1}{x}$ (the transformation we added). That last term rewrites to just *one transformation with the form* $x \leftarrow w + \frac{1}{x}$*, without the need of being followed by another transformation.*

Thus, in this way we have that any sequence S' of transformations can be rewritten into a sequence with the form

$$(x \leftarrow a_1 + \frac{1}{x}),\ (x \leftarrow a_2 + \frac{1}{x}), \ldots, (x \leftarrow a_k + \frac{1}{x})$$

with $a_2, \ldots, a_k \geq 1$ and a_1 either ≥ 1 or $= 0$. If $a_1 = 0$, then $(x \leftarrow a_1 + \frac{1}{x})$ does not affect the number of sign variations in the list of coefficients of the resulting transformed expression.

By the theorem of Vincent, we have that these transformations, in sufficient number, lead to an equation with 0 or 1 sign variation.

4.3.2 *Isolating* the roots instead of *counting* them

If we wanted not only to count the roots but also to produce a list of the isolating intervals associated to them, we could note that a sequence of transformations of the form $x \leftarrow x + 1$ and $x \leftarrow \frac{1}{1+x}$ can always be expressed as a single Möbius transformation $\frac{ax+b}{cx+d}$. We can add the *current Möbius transformation* as parameter, and keep

modifying it in the recursive calls of the function. If we have a Möbius transformation with the form $M(x) = \frac{ax+b}{cx+d}$, then we have that $M(x+1) = \frac{ax+(a+b)}{cx+(c+d)}$ and that $M(\frac{1}{x+1}) = \frac{bx+(a+b)}{dx+(c+d)}$. The transformation at the beginning, is $M(x) = x = \frac{1x+0}{0x+1}$. The interval associated to a p_i, which has associated the transformation M_i, would be the interval $(\min(M_i(0), M_i(\infty)), \max(M_i(0), M_i(\infty)))$.

Let us see the previous example. Our input was $p(x)$. Its associated transformation would be $\frac{1x+0}{0x+1}$. We had that $\text{var}(p(x)) > 1$, so it was splitten into two polynomials:

$$\begin{aligned} p_1(x) &= (1+x)^n p\left(\tfrac{1}{1+x}\right) \\ p_2(x) &= p\,(1+x) \end{aligned}$$

The associated transformations to these two new polynomials are:

$$\begin{aligned} M_1(x) &= M(\tfrac{1}{1+x}) &&= \tfrac{0x+(1+0)}{1x+(0+1)} &&= \tfrac{1}{x+1} \\ M_2(x) &= M(1+x) &&= \tfrac{1x+(1+0)}{0x+(0+1)} &&= \tfrac{x+1}{1} \end{aligned}$$

For the step in which we splitted p_1 into

$$\begin{aligned} p_3(x) &= (1+x)^n p_1\left(\tfrac{1}{1+x}\right) \\ p_4(x) &= p_1\,(1+x) \end{aligned}$$

the correspondent transformations would be

$$\begin{aligned} M_3(x) &= M_1(\tfrac{1}{1+x}) &&= \tfrac{1x+(0+1)}{1x+(1+1)} &&= \tfrac{x+1}{x+2} \\ M_4(x) &= M_1(1+x) &&= \tfrac{0x+(0+1)}{1x+(1+1)} &&= \tfrac{1}{x+2} \end{aligned}$$

And we can continue in this way as shown in figure 4.7. The correspondent intervals produced by a process as the one exemplified in the figure would be

- $(\min(\frac{1}{2}, \frac{\infty+1}{\infty+2}), \max(\frac{1}{2}, \frac{\infty+1}{\infty+2})) = (\frac{1}{2}, 1)$
- $(\min(\frac{1}{3}, \frac{\infty+1}{2\infty+3}), \max(\frac{1}{3}, \frac{\infty+1}{2\infty+3}) = (\frac{1}{3}, \frac{1}{2})$
- $(\min(\frac{1}{3}, \frac{1}{\infty+3}), \max(\frac{1}{3}, \frac{1}{\infty+3}) = (0, \frac{1}{3})$
- $(\min(1, \infty+1), \max(1, \infty+1) = (1, \infty)$

An implementation of Vincent's method for isolating roots is shown at Algorithm 2.

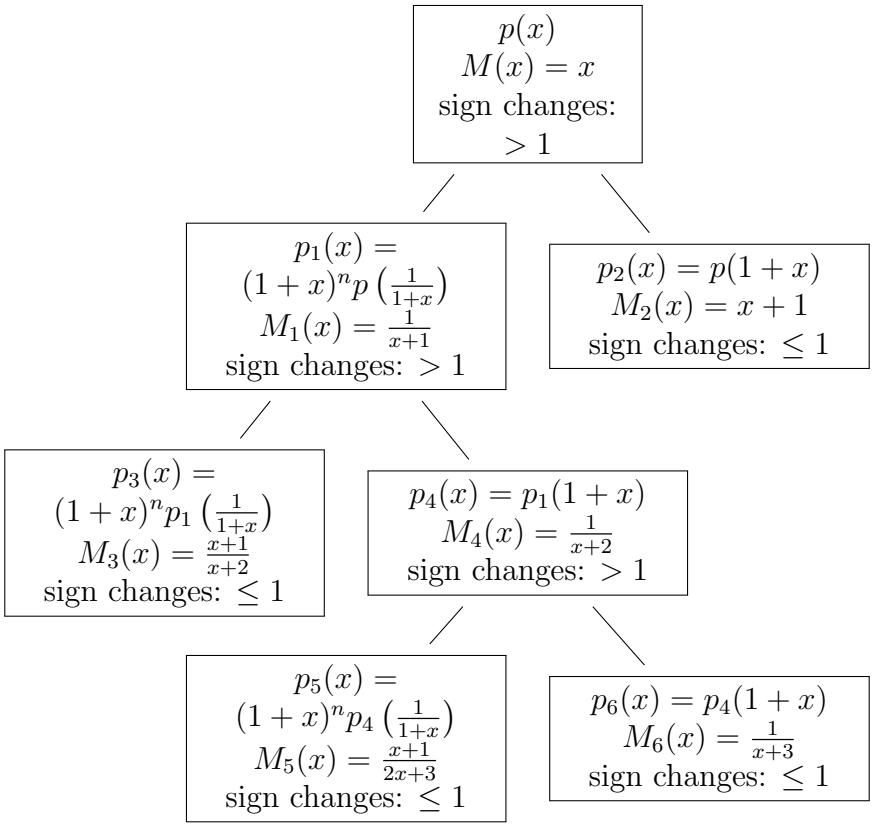


Figure 4.7: Example of Vincent's method. We start with a polynomial p, which presents more than 1 variation of signs in its list of coefficients. Every time it happens, we split into two new polynomials, by considering the *current* polynomial in the variables $\frac{1}{1+x}$ and $1+x$.

Algorithm 2 Vincent's method for isolating roots

procedure VincentIsolate($p, M(x) = \frac{ax+b}{cx+d}$)
    ▷ p is squarefree, of degree n, M is the transformation; at the beginning is $M = x$
    if sc $= 0$ **then**
        return $\emptyset$
    if sc $= 1$ **then**
        return $\{(\min(M(0), M(\infty)), \max(M(0), M(\infty)))\}$
    $p_{01}(x) \leftarrow (x+1)^n p(\frac{1}{x+1})$
    $M_{01}(x) \leftarrow M(\frac{1}{x+1})$
    $p_{1\infty}(x) \leftarrow p(x+1)$
    $M_{1\infty}(x) \leftarrow M(x+1)$
    $V_{01} \leftarrow$ VincentIsolate(p_{01}, M_{01})
    $V_{1\infty} \leftarrow$ VincentIsolate$(p_{1\infty}, M_{1\infty})$
    if $p(1) = 0$ **then**
        return $V_{01} \cup V_{1\infty} \cup \{[M(1), M(1)]\}$
    else
        return $V_{01} \cup V_{1\infty}$

4.3.3 On the implementation of $x \leftarrow \frac{1}{x+1}$ and $x \leftarrow x+1$

Algorithm 3 shows the method exposed by Knuth [60, section 4.6.4, steps H1 and H2] to compute the coefficients of $u(x + x_0)$ given the coefficients of $u(x)$, of degree n.

For example, if we have $p(x) = x^3 - 7x + 7$ and we want to compute $p(x + 1)$, the algorithm would update the values p_2, p_1, p_0 as it is shown in the following table. First, it is traversed from left to right the line with $k = 0$. Then, it is traversed from left to right the line with $k = 1$, and so on.

	p_3	p_2	p_1	p_0
	1	0	-7	7
$k = 0$		1	-6	1
$k = 1$		2	-4	
$k = 2$		3		
	1	3	-4	1

Algorithm 3 Compute the coefficients of $u(x + x_0)$ given the coefficients of $u(x)$.

procedure PolynomialShift(x_0, u) ▷ $u(x) = u_n x^n + u_{n-1} x^{n-1} + \cdots + u_0$
    for $k = 0, \ldots, n-1$ (in this order) **do**
        for $j = n-1, \ldots, k+1, k$ (in this order) **do**
            Set $u_j \leftarrow u_j + x_0 u_{j+1}$

Thus, $p(x+1) = x^3 + 3x^2 - 4x + 1$. If we want to compute $(1+x)^n p(\frac{1}{1+x})$, we can first compute $x^n p(\frac{1}{x})$ and then shift the obtained expression by 1, using this same method. The expression of $x^n p(\frac{1}{x})$ can be computed very easily by observing that its list of coefficients is the reversed list of coefficients of p. Polynomial shifting is the most time-demanding operation in the methods derived from Vincent theorem that will be shown in the next section, so it is worth to have efficient implementations of it [96, **?**].

4.4 Root isolation methods derived from Vincent's theorem

There are three main isolation methods derived from Vincent's theorem: VCA (Vincent, Collins, Akritas), VAS (Vincent, Akritas, Strzeboński) and VAG (Vincent, Alesina, Galuzzi). All these methods stand on the same basic idea. In the problem of isolating the positive roots of a given polynomial $p(x)$, it can happen that the number of sign variations at the coefficients of p is 0 or 1. In such a case, the problem is trivial: p either has 0 roots or have 1 root, contained in $(0, +\infty)$.

When the number of sign variations is larger than 1, the different methods based on Vincent's theorem propose different strategies to inspect *other polynomials* whose roots corresponds to the roots of p *in some specific interval instead of in* $(0, +\infty)$. For this purpose, these methods use *mappings*. For example, the function $\phi(x) = \frac{a+bx}{1+x}$ maps the positive x-semi axis (i.e.: the interval $(0, +\infty)$) onto the interval (a, b). The function $\phi(x) = x + 1$ maps the interval $(0, +\infty)$ into $(1, +\infty)$, the function $\phi(x) = \frac{1}{x+1}$ maps $(0, +\infty)$ onto $(0, 1)$.

Through the use of these mappings, these methods convert a problem of isolating roots in a given interval to many subproblems of isolating roots in subintervals of the given interval. The stop condition, when it is not necessary anymore to keep subdividing the problem, is in all the three cases the same: to reach a polynomial in which the Descartes' rule is conclusive. This is: reach a polynomial with 0 or 1 sign variation in its list of coefficients. The theorem of Vincent is the theoretical result on which the proofs of termination of all these three methods stand. Could we keep transforming the problem into smaller problems forever, without reaching the stop condition? No, in all these three methods, the process terminates. And it can be proved with the help of Vincent's ideas;

when not by the direct application of Vincent's theorem [32, 92, 13, 19, 17].

4.4.1 VCA

In 1976 Collins and Akritas revisited Vincent's method for root isolation [32], analyzing aspects of Uspensky's rewriting of Vincent's work, showing an example in which the computational effort required by Vincent's method is exponential on the size of the input polynomial, and then proposing the first polynomial-time version of the algorithm. In their work, Collins and Akritas show that their method has a complexity $O(n^6 L(d)^2)$, for a polynomial p of degree n and $L(d)$ being the bitsize of the sum of the absolute values of the coefficients of p.

Uspensky had pointed out that the number of substitutions $x \leftarrow \frac{1}{x+1}$ in Vincent's method is a polynomial function of n and Δ, the minimum root separation of p [92]. In fact, he pointed out that this number of substitutions is dominated by $L(n) + L(\frac{1}{\Delta})$. But, as is pointed out by Collins and Akritas, it is known that $L(\Delta^{-1}) \leq nL(d)$ [33]; so we have that the number of substitutions $x \leftarrow \frac{1}{x+1}$ in Vincent's method is $O(nL(d))$. However, as Collins and Akritas pointed out, there is another fact: the number of substitutions $x \leftarrow x + 1$ is not dominated by *any* power of $L(d)$ (the size of the input polynomial).

VCA can be compared with Vincent's original method as follows: in Vincent's original work, the intervals being mapped are $(0, 1)$ and $(1, +\infty)$. In VCA, the intervals being mapped are $(0, \frac{1}{2})$ and $(\frac{1}{2}, 1)$. VCA makes, at the beginning, before starting the computation, a transformation on the variable x of the input polynomial $p(x)$, to ensure that the calculations are going to be performed on a polynomial whose roots are all contained in the interval $(0, 1)$. Thus, due to the way in which the variable is being mapped, when we are mapping the intervals $(0, \frac{1}{2})$ and $(\frac{1}{2}, +\infty)$ to $(0, +\infty)$, we are able to ensure that all the roots of the new polynomial are in $(0, 1)$.

Before starting with the algorithm, we need to compute an upper bound B for the maximum positive root of the input polynomial $p(x)$, and make the transformation $x \leftarrow Bx$, to ensure that all the positive roots are in $(0, 1)$. As we did with Vincent's algorithm, we present *counting* (Algorithm 4) and *isolating* (Algorithm 5) versions of the method.

Algorithm 4 Vincent-Collins-Akritas for counting the positive roots of p

procedure VCACount(p) ▷ p is a squarefree polynomial, with all its positive roots in $(0,1)$.
    sc $\leftarrow$ number of sign changes in $(x+1)^{\deg p} p(\frac{1}{x+1})$
    if sc $= 0$ or sc $= 1$ **then**
        return sc
    $p_{0\frac{1}{2}}(x) \leftarrow 2^{\deg p} p(\frac{x}{2})$
    $p_{\frac{1}{2}1}(x) \leftarrow 2^{\deg p} p(\frac{x+1}{2})$
    if $p(\frac{1}{2}) = 0$ **then**
        return VCACount$(p_{0\frac{1}{2}})$ + VCACount$(p_{\frac{1}{2}1})$ + 1
    else
        return VCACount$(p_{0\frac{1}{2}})$ + VCACount$(p_{\frac{1}{2}1})$

Algorithm 5 Vincent-Collins-Akritas for isolating the positive roots p

procedure VCAIsolate($p, I = (a,b)$) ▷ p is a squarefree polynomial, with all its roots in $(0,1)$. (a,b) is the interval of the domain of the original polynomial to which maps the interval $(0,1)$ of the current polynomial. Remember that we have shrinked the variable in order to have all the roots on the interval $(0,1)$
    sc $\leftarrow$ number of sign changes in $(x+1)^{\deg p} p(\frac{1}{x+1})$
    if sc $= 0$ **then**
        return $\emptyset$
    if sc $= 1$ **then**
        return $\{I\}$
    $p_{0\frac{1}{2}}(x) \leftarrow 2^{\deg p} p(\frac{x}{2})$
    $p_{\frac{1}{2}1}(x) \leftarrow 2^{\deg p} p(\frac{x+1}{2})$
    if $p(\frac{a+b}{2}) = 0$ **then**
        return VCAIsolate$(p_{0\frac{1}{2}}, (a, \frac{a+b}{2}))$∪VCAIsolate$(p_{\frac{1}{2}1}, (\frac{a+b}{2}, b))$∪$\{[\frac{a+b}{2}, \frac{a+b}{2}]\}$
    else
        return VCAIsolate$(p_{0\frac{1}{2}}, (a, \frac{a+b}{2}))$ ∪ VCAIsolate$(p_{\frac{1}{2}1}, (\frac{a+b}{2}, b))$

4.4.2 VAS

This is the second method (after VCA) developed to handle the exponential behavior of Vincent's method. It is authored by Akritas and Strzeboński [13]. For a clear interpretation of the method, we can proceed in exactly the same way that we did for Vincent's method. We considered that the roots were of the form $1 + x$ or $\frac{1}{1+x}$. If we know a lower bound α on the roots of the polynomial, we can shift the variable x by α before making the split. This is what VAS does. In fact, depending on *how large* α is, VAS sometimes *shrinks* the variable by performing the substitution $x \leftarrow \alpha x$. Akritas and Strzeboński explained that they decide when to shrink just by putting a threshold α_0. When $\alpha > \alpha_0$, they shrink. They obtained experimentally the value of α_0 for which the method shows the best performance. That value is 16.

As we did with previous approaches, we will present two versions of this method: *root counting* (Algorithm 6) and *root isolation* (Algorithm 7).

Note that if we remove the lines $5 - 10$ of Algorithm 6, the algorithm is precisely Vincent's method.

For the root isolation version of this algorithm, we will proceed in a similar way than the used in Vincent's approach. We will keep the *transformation* that we should apply to the interval $(0, +\infty)$ to obtain the interval at the original domain. This transformation can be represented by 4 values: a, b, c, d, denoting $M(x) = \frac{ax+b}{cx+d}$. Due to the nature of the transformations that we are making, the composition of them always can be represented as a *Möbius transformation*; which is precisely what we are doing.

The facts that we are going to use are: (considering that $M(x) = \frac{ax+b}{cx+d}$)

- $M(x+1) = \frac{ax+(a+b)}{cx+(c+d)}$
- $M(\frac{1}{x+1}) = \frac{bx+(a+b)}{dx+(c+d)}$
- $M(1) = \frac{a+b}{c+d}$
- $M(x+\alpha) = \frac{ax+(a\alpha+b)}{cx+(c\alpha+d)}$
- $M(\alpha x) = \frac{a\alpha x+b}{c\alpha x+d}$

The transformation, at the beginning, is $M(x) = x = \frac{1x+0}{0x+1}$. Thus, the VAS for root isolation could be stated as shown in Algorithm 7.

Algorithm 6 Vincent-Akritas-Strzeboński for counting the positive roots of p

procedure VASCount(p) ▷ p is a squarefree polynomial.
    sc $\leftarrow$ number of sign changes in $p(x)$
    if sc $= 0$ or sc $= 1$ **then**
        return sc
    $\alpha \leftarrow$ lower bound for the positive roots of p
    if $\alpha > 16$ **then**
        $p(x) \leftarrow p(\alpha x)$
        $\alpha \leftarrow 1$
    if $\alpha \geq 1$ **then**
        $p(x) \leftarrow p(x + \alpha)$
    $p_{01}(x) \leftarrow (x+1)^{\deg p} p(\frac{1}{x+1})$
    $p_{1\infty}(x) \leftarrow p(x+1)$
    $V_{01} \leftarrow$ VASCount(p_{01})
    $V_{1\infty} \leftarrow$ VASCount($p_{1\infty}$)
    if $p(1) = 0$ **then**
        return $V_{01} + V_{1\infty} + 1$
    else
        return $V_{01} + V_{1\infty}$

Algorithm 7 Vincent-Akritas-Strzeboński for isolating the positive roots of p

procedure VASIsolate(p, M) ▷ p is a squarefree polynomial.
    sc $\leftarrow$ number of sign changes in $p(x)$
    if sc $= 0$ **then**
        return $\emptyset$
    if sc $= 1$ **then**
        return $\{(\min(M(0), M(\infty)), \max(M(0), M(\infty)))\}$
    $\alpha \leftarrow$ lower bound for the positive roots of p
    if $\alpha > 16$ **then**
        $p(x) \leftarrow p(\alpha x)$
        $M(x) \leftarrow M(\alpha x)$
        $\alpha \leftarrow 1$
    if $\alpha \geq 1$ **then**
        $p(x) \leftarrow p(x + \alpha)$
        $M(x) \leftarrow M(x + \alpha)$
    $p_{01}(x) \leftarrow (x+1)^{\deg p} p(\frac{1}{x+1})$, $M_{01}(x) \leftarrow M(\frac{1}{x+1})$
    $p_{1\infty}(x) \leftarrow p(x+1)$, $M_{1\infty}(x) \leftarrow M(x+1)$
    $V_{01} \leftarrow$ VASIsolate(p_{01}, M_{01})
    $V_{1\infty} \leftarrow$ VASIsolate($p_{1\infty}, M_{1\infty}$)
    if $p(1) = 0$ **then**
        return $V_{01} \cup V_{1\infty} \cup \{[M(1), M(1)]\}$
    else
        return $V_{01} \cup V_{1\infty}$

4.4.3 VAG

Alesina and Galuzzi proved [19, Theorem 3 (page 7)] that

$$\forall p(x), \exists \delta > 0 \text{ such that if } |a-b| < \delta,\ (1+x)^n p\left(\frac{b+ax}{1+x}\right)$$

has 0 or 1 sign variations (where n is the degree of p). It follows trivially from this theorem the simplest bisection approaches.

For the problem of isolating the real roots of a polynomial $p(x)$, we start considering an interval $(0, M)$, where M is any upper bound for the positive roots of p. We check how many sign changes has the polynomial $(1+x)^n p\left(\frac{0x+M}{1+x}\right)$. If it has 0 or 1 sign changes, then p has 0 or 1 roots, respectively. In the case it has 1, its isolating interval is $(0, M)$. In the case that there are two or more sign changes, we split the interval $(0, M)$ in two open intervals and a point: its *first half* $\left(0, \frac{0+M}{2}\right)$, its *second half* $\left(\frac{0+M}{2}, M\right)$, and its midpoint $[\frac{0+M}{2}, \frac{0+M}{2}]$. We check if the midpoint is root of p by evaluating p at it, and we repeat in the new sub-intervals the process that we have just done for $(0, M)$. The theorem of Alesina and Galuzzi ensures that this process terminates, since the width of the intervals is halved in each step.

As we did for the previous methods, we present in here two versions of the Alesina Galuzzi method. Algorithm 8 shows the implementation of root counting and Algorithm 9 shows the implementation of root isolation.

4.5 On an adaptation of Fourier's theorem

In this section we will formulate and prove a new theorem, which is similar to Fourier's theorem, but with one difference that makes it much more useful in the case of fewnomials: the list of functions, in Fourier's theorem, is composed by n functions, where n is the degree of the input polynomial. The list of functions of the theorem presented in this section is composed by T functions, where T is the number of terms present in the input polynomial.

Let $f(x)$ be a polynomial. We define the *reduction* of $f(x)$, denoted by $[f(x)]$, to be $\frac{f(x)}{x^j}$, where j is the maximum natural such that x^j divides $f(x)$. Thus, for example,

Algorithm 8 Vincent-Alesina-Galuzzi for counting the roots of p in (a, b)

procedure VAGCount$(p, (a, b))$ ▷ p is a squarefree polynomial. (a, b) is the interval in which we are counting the number of roots
    sc $\leftarrow$ number of sign changes in $(x+1)^{\deg(p)} p(\frac{a+bx}{1+x})$
    if sc $= 0 \vee$ sc $= 1$ **then**
        return sc
    $V_1 \leftarrow$ VAGCount$(p, (a, \frac{a+b}{2}))$
    $V_2 \leftarrow$ VAGCount$(p, (\frac{a+b}{2}, b))$
    if $p(\frac{a+b}{2}) = 0$ **then**
        return $V_1 + V_2 + 1$
    else
        return $V_1 + V_2$

Algorithm 9 Vincent-Alesina-Galuzzi for isolating the roots of p in (a, b)

procedure VAGIsolate$(p, (a, b))$ ▷ p is a squarefree polynomial. (a, b) is the interval in which we are counting the number of roots
    sc $\leftarrow$ number of sign changes in $(x+1)^{\deg(p)} p(\frac{a+bx}{1+x})$
    if sc $= 0$ **then**
        return $\emptyset$
    if sc $= 1$ **then**
        return $\{(a, b)\}$
    $V_1 \leftarrow$ VAGIsolate$(p, (a, \frac{a+b}{2}))$
    $V_2 \leftarrow$ VAGIsolate$(p, (\frac{a+b}{2}, b))$
    if $p(\frac{a+b}{2}) = 0$ **then**
        return $V_1 \cup V_2 \cup \{[\frac{a+b}{2}, \frac{a+b}{2}]\}$
    else
        return $V_1 \cup V_2$

$[10x^{24} + 4x^{12} - 10x^{11}] = \frac{10x^{24}+4x^{12}-10x^{11}}{x^{11}} = 10x^{13} + 4x - 10$, and $[10x^{24} + 4x^{12} - 10] = \frac{10x^{24}+4x^{12}-10}{1} = 10x^{24} + 4x^{12} - 10$. **Observe** that $[f(x)]$ has a nonzero independent term, therefore $[f(x)]'$ has one term less than $f(x)$.

Given the polynomial $f(x)$, let T be the number of nonzero terms present in f. Let us define the polynomials $l_1, l_2, \ldots, l_T$ as follows:

$$\begin{aligned}
l_1 &\coloneqq [f] && l_1 \text{ has } T \text{ terms} \\
l_2 &\coloneqq [l_1'] && l_2 \text{ has } T-1 \text{ terms} \\
&\vdots \\
l_{i+1} &\coloneqq [l_i'] && l_{i+1} \text{ has } T-i \text{ terms} \\
&\vdots \\
l_T &\coloneqq [l_{T-1}'] && l_T \text{ has 1 term}
\end{aligned}$$

We define the number of *sign variations* or *sign changes* in a sequence of numbers $S = [a_0, a_1, a_2, \ldots, a_j]$ as the number pairs (a_i', a_{i+1}') in the sequence $S' = [a_0', a_1', \ldots, a_w']$, obtained by removing all the 0's from S. For example: the number of sign variations in $[-3, 0, 0, -3]$ is 0, and the number of sign variations in $[-2, 0, 4, -5]$ is 2.

Theorem 5 (Adaptation of Fourier's theorem)**.** *Given a polynomial f, with T terms and arbitrary degree, define the list of T polynomials $[l_1, \ldots, l_T]$ as above. For any $x \in \mathbb{R}$, let $V(x)$ be the number of sign changes of the sequence $[l_1(x), \ldots, l_T(x)]$. Then the number of roots of f (counting multiplicities) between a and b, where $f(a) \neq 0$, $f(b) \neq 0$ and $0 < a < b$, does not exceed $V(a) - V(b)$. Moreover, the number of roots can differ from $V(a) - V(b)$ by an even number only.*

Proof. Let x be a point which moves along the segment $[a, b]$, from a to b. The quantity $V(x)$ varies only if x passes through a root of one or more of the polynomials $l_1, l_2, \ldots, l_{T-1}$ (we know that $l_T(x) \neq 0$ for all $x > 0$).

Consider first the case when x passes through a root α of the first k functions of the list of l_i's, with $l_{k+1}(\alpha) \neq 0$. By Lemma 1, α is a root of f with multiplicity k. It is straightforward to see that the value of $V(x)$ decreases by k (see Table 1).

Now suppose that x passes through a root α of the k functions $l_{i+1}, \ldots, l_{i+k}$, with $i \geq 1, i + k < T, l_i(\alpha) \neq 0, l_{i+k+1}(\alpha) \neq 0$. In a neighborhood of α, we have that the signs of l_{i+k+1} and l_i are constant. If l_{i+k+1} is positive, then l_{i+k} must be increasing, and

	$<\alpha$	α	$>\alpha$	$<\alpha$	α	$>\alpha$
l_1	$-$	0	$+$	$+$	0	$-$
l_2	$+$	0	$+$	$-$	0	$-$
$\cdot$	$\cdots$			$\cdots$		
l_{k-1}	$+$	0	$+$	$-$	0	$-$
l_k	$-$	0	$+$	$+$	0	$-$
l_{k+1}	$+$	$+$	$+$	$-$	$-$	$-$

	k even. $V(x)$ decreases k				k odd. $V(x)$ decreases $k \pm 1$			
	$<\alpha$	α	$>\alpha$	$<\alpha$	α	$>\alpha$	$<\alpha$	α	$>\alpha$	$<\alpha$	α	$>\alpha$
l_i	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$	$\pm$
l_{i+1}	$+$	0	$+$	$-$	0	$-$	$-$	0	$+$	$+$	0	$-$
$\cdot\cdot$	$\cdots$			$\cdots$			$\cdots$			$\cdots$		
l_{i+k-1}	$+$	0	$+$	$-$	0	$-$	$+$	0	$+$	$-$	0	$-$
l_{i+k}	$-$	0	$+$	$+$	0	$-$	$-$	0	$+$	$+$	0	$-$
l_{i+k+1}	$+$	$+$	$+$	$-$	$-$	$-$	$+$	$+$	$+$	$-$	$-$	$-$

Table 1 (left). Signs of $l_1(x), \ldots l_{k+1}(x)$, in a neighborhood of α, where $l_1(\alpha) = \cdots = l_k(\alpha) = 0 \wedge l_{k+1} \neq 0$. $V(x)$ decreases by k. **Table 2 (right).** Signs of $l_i(x), \ldots l_{i+k+1}(x)$, in a neighborhood of α, where $l_{i+1}(\alpha) = \cdots = l_{i+k}(\alpha) = 0$ and $l_{i+k+1}(\alpha) \neq 0$. $V(x)$ decreases by an even amount.

consequently l_{i+k} must be negative in $< \alpha$ and positive in $> \alpha$. Now, given that l_{i+k} is negative in $< \alpha$ and positive in $> \alpha$, we have that l_{i+k-1} must be positive in $< \alpha$ and positive in $> \alpha$, and so on. At all the steps of this reasoning we use the fact that the signs of $l'_t(x)$ and $l_{t+1}(x)$ are the same for all $x > 0$. So, continuing with this reasoning, we have that the sign of l_{i+k+1} determines the signs of all the functions $l_{i+k}, \ldots, l_{i+1}$. At this point, we can reach two different scenarios, depending on the parity of k. If k is even, the value of $V(x)$ decreases by k. If k is odd, $V(x)$ decreases by $k-1$ or by $k+1$ depending on the sign of l_i. The same (just with opposite signs) occurs when l_{i+k+1} is negative. Table 2 shows all the possibilities in a more graphical way.

Thus, $V(x)$ decreases by an even amount when x pass through a root of l_{i+1}, l_{i+2}, $\ldots$, l_{i+k}, and by k when x passes a root of f with multiplicity k. From these two facts, follows the theorem. □

Lemma 1. *If $\alpha > 0$ and $f, l_1, l_2, \ldots, l_{k+1}$ are defined as above, then*

$$\left.\begin{array}{rcl} l_1(\alpha) & = & 0 \\ & \vdots & \\ l_k(\alpha) & = & 0 \\ l_{k+1}(\alpha) & \neq & 0 \end{array}\right\} \iff \left\{\begin{array}{rcl} f(\alpha) & = & 0 \\ & \vdots & \\ f^{(k-1)}(\alpha) & = & 0 \\ f^{(k)}(\alpha) & \neq & 0 \end{array}\right.$$

Proof. We have

$$\begin{aligned} x^{t_1} l_1(x) &= f(x) \\ x^{t_2} l_2(x) &= l_1'(x) \\ &\vdots \\ x^{t_k} l_k(x) &= l_{k-1}'(x) \\ x^{t_{k+1}} l_{k+1}(x) &= l_k'(x) \end{aligned}$$

Let us see that $f^{(i)}$ can be expressed as a sum in which the terms are a product of three quantities: a power of x, a positive constant and one of the first $(i+1)$ elements of the list $[l_1, l_2, \dots]$. In other words, let us see that

$$f^{(i)}(x) = \sum_w a_w l_{t_w}(x) x^{j_w} \qquad \text{for some } 1 \le t_w \le i+1, \quad \text{and some } a_w > 0 \tag{4.1}$$

We proceed by induction on i.

If $i = 0$, then (4.1) holds; because $f(x) = x^{t_1} l_1(x)$.

If (4.1) holds for i, we have that

$$\begin{aligned} f^{(i+1)}(x) &= \textstyle\sum_w \left(a_w l_{t_w}'(x) x^{j_w} + a_w j_w x^{j_w - 1} l_{t_i}(x) \right) \text{ with } 1 \le t_w \le i+1 \\ &= \textstyle\sum_w \left(a_w (l_{t_w+1}(x) x^{t_w+1}) x^{j_w} + a_w j_w x^{j_w-1} l_{t_w}(x) \right) \text{ with } 1 \le t_w \le i+1 \\ &= \textstyle\sum_w \left(a_w l_{t_w+1}(x) x^{t_w+1+j_w} + a_w j_w x^{j_w-1} l_{t_w}(x) \right) \text{ with } 1 \le t_w \le i+1 \end{aligned}$$

which is a sum in which the terms are a product of a power of x, a positive constant and one of the first $(i+2)$ elements of the list $[l_1, l_2, \dots]$, so (4.1) holds for $i+1$.

From (4.1) it follows that if $l_1(\alpha) = \cdots = l_k(\alpha) = 0$, then $f(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0$, and $f^{(k)}(\alpha)$ has the form $\sum_w (a_w) \alpha^{j_w} l_{k+1}(\alpha)$, for some a_w positives. So, given that $\alpha > 0$, we have that

$$\begin{aligned} l_1(\alpha) = \cdots = l_k(\alpha) = 0, l_{k+1}(\alpha) \neq 0 \\ \implies f(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0, f^{(k)}(\alpha) \neq 0 \end{aligned} \tag{4.2}$$

Let us prove now the reciprocal of (4.2). Let us assume that $f(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0$, $f^{(k)}(\alpha) \neq 0$.

By (4.1), we have that $f^{(m)}(\alpha) = \sum_w a_w \alpha^{j_w} l_{t_w}(\alpha)$, for some $a_w > 0$, with $1 \le t_w \le m+1$. Since α^{j_w} and a_w are positive, we have that if all the l_i (with $1 \le i \le m$) vanishes

at α then the fact of $f^{(m)}$ being 0 at α would imply l_{m+1} being 0 at α. In other words:

$$f^{(m)}(\alpha) = 0, l_1(\alpha) = \cdots = l_m(\alpha) = 0 \implies l_{m+1}(\alpha) = 0 \tag{4.3}$$

So, if we have $f(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0$, we can apply (4.3) k times and iteratively conclude that $l_1(\alpha) = 0$, $l_2(\alpha) = 0$, ..., $l_k(\alpha) = 0$ as follows:

(Step 1) $f(\alpha) = 0 \implies l_1(\alpha) = 0$,

(Step 2) $f'(\alpha) = 0, l_1(\alpha) = 0 \implies l_2(\alpha) = 0$,

$\vdots$

(Step k) $f^{(k-1)}(\alpha) = 0, l_1(\alpha) = \ldots = l_{k-1}(\alpha) = 0 \implies l_k(\alpha) = 0$.

Let us continue one more line and see what happen with l_{k+1}. We know, by (4.1), that $f^{(k)}(\alpha) = \sum_w a_w \alpha^{j_w} l_{t_w}(\alpha)$, for some $a_w > 0$, with $1 \leq t_w \leq k+1$. But we have just seen that all the l_i, with $1 \leq i \leq k$, vanishes at α, so if we join together these two facts we have that $f^{(k)}(\alpha) = \sum_w a_w \alpha^{j_w} l_{k+1}(\alpha)$. Since a_w and α^{j_w} are positive, we have that $f^{(k)}(\alpha) \neq 0 \Rightarrow l_{k+1}(\alpha) \neq 0$. Thus, we have that

$$f(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0, f^k(\alpha) \neq 0 \implies l_1(\alpha) = \cdots = l_k(\alpha) = 0, l_{k+1}(\alpha) \neq 0 \tag{4.4}$$

From (4.2) and (4.4) follows the theorem. □

4.5.1 Consideration on an adaptation of Fourier's theorem to count the exact number of roots

The proposed adaptation of Fourier's theorem gives *a bound* for the number R of roots that a polynomial $p(x)$ has in the interval $(0, w)$. Let

$$\begin{aligned} L_p &= (p_1, p_2, \ldots, p_T) \\ &= ([p], [p'_1], \ldots, [p'_{T-1}]) \end{aligned}$$

be the list of polynomials that the theorem proposes to use, where T is the number of nonzero terms in $p(x)$. Let $L_p(x) = [p_1(x), p_2(x), \ldots, p_T(x)]$ be the list of numbers that

results from the evaluation of them at x. Let $\mathbf{sc}(L)$ be the number of sign changes in the list of numbers L. For example, $\mathbf{sc}(L_p(x))$ is the number of sign changes in the list of numbers $L_p(x)$. The quantity $\mathbf{sc}(L_p(0)) - \mathbf{sc}(L_p(w))$ is the precise value of R when it is 0 or 1, but in the other cases it provides a value $R + 2Q$, for some nonnegative integer Q.

Recalling the proof of Fourier's theorem, Q is the number of roots α of some p_i (with $T \geq i > 0$) such that $\text{sgn}(p_{i-1}(\alpha)) = \text{sgn}(p_{i+1}(\alpha))$.

Let us consider the sequence of functions L_{p+K}, obtained from the initial polynomial $p(x)+K$, which is the same initial polynomial $p(x)$ that we already had, plus one constant K; being K chosen so that it is big enough to ensure that $p(x) + K$ is positive in $(0, w)$. Thus, the number of roots of $p(x) + K$ in $(0, w)$ is 0. Notice that the i-th function at L_{p+K} is equal to the i-th function at L_p, for $i > 1$, since the derivative of $p(x) + K$ is equal to the derivative of $p(x)$.

We say that the *context* of a root of one polynomial p_i is given the signs of p_{i-1} and p_{i+1} at that root. Given that the i-th polynomial at L_{p+K} is equal to the i-th polynomial at L_p, for $i > 1$, all the roots of polynomials p_i at L_{p+K} and at L_p occurs *in the same context* for $i > 2$. Thus, if we were able to quantizate how are affected the contexts of the roots of p_2 when summing K to p_1, then we would be able to compute the number of roots R exactly.

4.5.2 Consideration on a possible adaptation of Sturm's method

Sturm's theorem and Sturm's method rely on Sturm's sequence, whose length is not proportional to the number of nonzero terms in a polynomial, but to its degree. The steps of the Euclidean algorithm to compute the greatest common divisor between f and f' does not preserve the fewnomial structure of f, in the cases in which f is a fewnomial. For example, if f is a fewnomial with 5 terms and degree 1000, the polynomials present at the Sturm's of f quickly become polynomials with a much-larger-than-5 number of terms.

One frustrated try that we made during the development of this thesis was related to obtain some alternative to Sturm sequence, which still presents the property that, when moving in the x axis from left to right, every time we pass through a root $x = \alpha$ of the function f_j, the functions f_{j-1} and f_{j+1} have opposite signs in a neighborhood of α (this property implied that the decreasements in the number of sign changes in the Sturm

sequence can only occur at the roots of f), *but* with the difference that its number of elements is linear in the number of terms of the input polynomial, and not its degree. For example, we would like to find a sequence that starts with the polynomial $f_0 = f$, and every some fixed number of elements of the list, the number of terms decreases. For example, perhaps f_5 has one term less than f_0, f_10 has 1 term less than f_5 and so on.

With this purpose, we realized that it is not mandatory, when creating the Sturm sequence, using the expression $f_i(x) = q_{i+1}(x)f_{i+1}(x) - f_{i+1}(x)$, to take the quantities q_{i+1} and f_{i+1} as the quotient and the ($-$remainder), respectively, of the division of $f_i(x)$ by $f_{i+1}(x)$. In fact, *any* two polynomials satisfying this expression would still fit into Sturm's proof. The only thing that we would lose is the guarantee that at every step the degree decreases.

4.6 On an elementary approach for the root isolation problem

In this section we will show a simple method to isolate the positive roots of a polynomial $p(x)$ of degree n, provided that there are not common positive roots between any derivative $p^{(i)}(x)$ and its derivative $p^{(i+1)}(x)$. Observe that this assumption is, in fact, the generic case. With the purpose of showing the method, let us first recall some elementary results that we will use in our method.

Theorem 6 (Rolle)**.** *Between two consecutive real roots a and b of a polynomial $f(x)$ there is at least one and at any rate an odd number of roots of its derivative $f'(x)$.*

Corollary 1. *Between two consecutive roots c and d of the derivative $f'(x)$ there is at most one root of $f(x)$.*

If we want to isolate the positive roots of a polynomial $p(x)$, and we know that it does not share any positive root with its derivative $p'(x)$, we can use the stated concepts as basis to a method that would work *when it is provided that the roots of the derivative are known.* Let all the distinct roots of $p'(x)$ be

$$\alpha_1 < \alpha_2 < \cdots < \alpha_r.$$

Write the list of signs of

$$p(-\infty), \quad p(\alpha_1), \quad p(\alpha_2), \quad \ldots, p(\alpha_r), \quad p(+\infty)$$

And inspect that list for variations of signs. The intervals (α_i, α_{i+1}) for which $\operatorname{sgn}(f(\alpha_i)) \neq \operatorname{sgn}(f(\alpha_{i+1}))$ isolates exactly one root of $p(x)$, and all of its roots are isolated in this way.

This method presents an asymmetry between what it produces as output and what it requires as input. This asymmetry makes it difficult to transform it into a recursive algorithm, in which the problem for a polynomial of degree n reduces to a problem for a polynomial of degree $n-1$. The asymmetry is that it produces an *isolation* of the roots of the input polynomial $p(x)$, and requires to know *exactly* the roots of $p'(x)$. We will adapt the algorithm to be able to give an isolation of the roots of $p(x)$ starting with an *isolation* of the roots of $p'(x)$. That would allow us to propose a simple recursive way of isolating roots.

Suppose that $p'(x)$ has k different roots, $\alpha_1, \alpha_2, \ldots, \alpha_k$, and that we have an isolation of them given by the intervals

$$(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$$

such that $\alpha_j \in (a_j, b_j)$. We know that $p(x)$ cannot have more than one root *between* α_j *and* α_{j+1} . If we were able to ensure that $p(x)$ has no roots in the intervals (a_j, b_j) and (a_{j+1}, b_{j+1}), then we would have that the number of roots of $p(x)$ in the interval (a_i, b_{i+1}) is the same than in (α_i, α_{i+1}).

Given that we are assuming that $p(x)$ and $p'(x)$ have not roots in common, we know that there exists $\delta > 0$ such that $p(x)$ have no roots in $(\alpha_j - \delta, \alpha_j + \delta)$ and neither in $(\alpha_{j+1} - \delta, \alpha_{j+1} + \delta)$. In fact, we could take as 2δ (i.e.: the width of the neighborhoods) the minimum separation between two roots of the polynomial which is the product of $p(x)p'(x)$. That would ensure that $p(x)$ does not change its sign in the mentioned intervals. Sagraloff [88] proved the following result.

Theorem 7. *Let f and g be polynomials of degree n or less with integer coefficients of absolute values less than 2^μ, and let*

$$L \coloneqq 128 \cdot n \cdot (\log n + \mu).$$

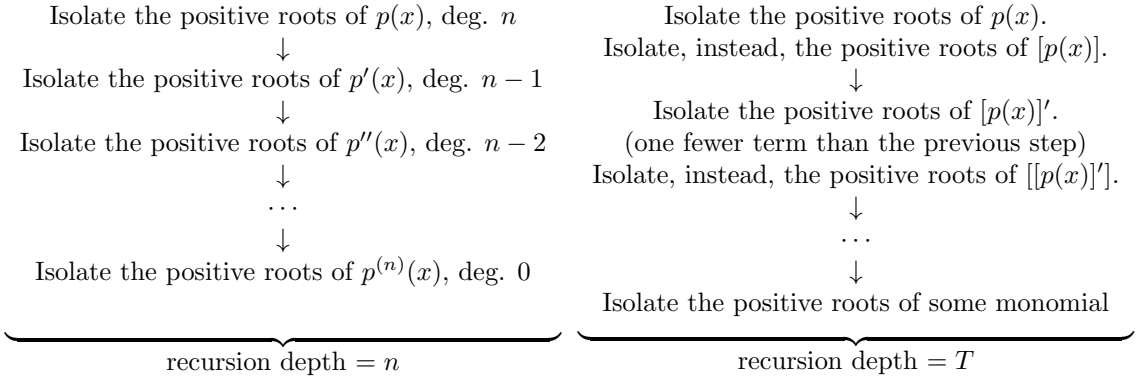


Figure 4.8: Recursive calls from the elementary method, with (right) (T calls) and without (left) (n calls) the consideration about the *reduction* operator

Then, for any two distinct roots ξ_i and ξ_j of $F \coloneqq f \cdot g$, it holds that $|\xi_i - \xi_j|^{m_i} > \frac{1}{2^L}$, where $m_i \coloneqq mult(\xi_i, F)$ denotes the multiplicity of ξ_i as a root of F. If ξ is a root of g and $f(\xi) \neq 0$, then it holds that $|f(x)| > 2^{-L/4}$ for all $x \in \mathbb{C}$ with $|x - \xi| < 2^{-L}$. Vice versa, if $f(\xi) = 0$, then $|f(x)| < 2^{-L}$ for all $x \in \mathbb{C}$ with $|x - \xi| < 2^{-L}$.

By setting the f and g of the theorem to be our p and p', and by considering μ to be largest of the bitsizes of the coefficients of $p(x)$ and $p'(x)$, we have a bound B for the root separation of $p(x)p'(x)$.

$$B = \frac{1}{2^{128 \cdot n \cdot (\log n + \mu)}}$$

Thus, if all the intervals (a_i, b_i) satisfy that $b_i - a_i < B$, we can isolate the roots of $p(x)$ by simple inspection of the signs of it when evaluated at the points a_i's. If some of the intervals (a_i, b_i) were too big, we can *refine* them until the required precision. This can be done through the highly efficient method of Abbott [3].

A note on the resulting method. With the previous observation, we are able to produce an isolation of the positive roots of a polynomial of degree n, given the isolation of the roots of its derivative. Now, let us observe something. The positive roots of a polynomial p divided by x^j, with $j \in \mathbb{N}$, are the same than the positive roots of p, since $x^j > 0$, for $x > 0$. Let us divide the input polynomial $p(x)$ by x^j, where j is the largest integer such that x^j divides $p(x)$. On page 70 we called this operation *reduction*, and denoted the resulting polynomial by $\frac{p(x)}{x^j}$ by $[p(x)]$. It is clear that $[p(x)]'$ has 1 term fewer than $[p(x)]$. With this consideration, applying the *reduction* at each step of the process, we have that the length of our recursion is not anymore as long as the degree of the input

Algorithm 10 Elementary method for isolating the real roots of a polynomial p

procedure SIMPLEISOLATION(p) ▷ it must be guaranteed that no one of the polynomials $p, p', p'', \ldots, p^{(n-1)}$ shares a positive root with its derivative.
  if p is a monomial **then**
    return an interval containing the positive root of p, or $\emptyset$ if not exist.
  while $p(0) = 0$ **do** $p(x) \leftarrow \frac{p(x)}{x}$
  $\mu \leftarrow$ largest bitsize of coefficients in p and p'.
  $L \leftarrow 128 \cdot \deg(p) \cdot (\log(\deg(p)) + \mu)$
  Intervals $\leftarrow$ SIMPLEISOLATION(p')
  Intervals $\leftarrow$ REFINEINTERVALS(Intervals, L) ▷ RefineIntervals is an auxiliary function, the recommended method for this is Abbott's
  lastX $\leftarrow -\infty$
  lastSign $\leftarrow f$(lastX)
  ints $\leftarrow \emptyset$
  for $(a_i, b_i) \in$ Intervals, from left to right **do**
    $s \leftarrow \text{sgn}(f(a_i))$
    if $s \neq$ lastSign **then**
      ints $\leftarrow$ ints $\cup\{$(lastX, b_i)$\}$
      lastX $\leftarrow b_i$
      lastSign $\leftarrow s$
  $s \leftarrow \text{sgn}(f(\infty))$
  if $s \neq$ lastSign **then**
    ints $\leftarrow$ ints $\cup\{$(lastX, ∞)$\}$
  return ints

polynomial, but as the number of terms of it. So this consideration has impact in the case of *fewnomials*.

In fact, if $p(x)$ is a polynomial of degree n and T terms, provided that we are able to isolate the positive roots of $p(x)$ from an isolation of the positive roots of $p'(x)$, the recursion could be as it is shown in figure 4.8. Algorithm 10 sketches this method.

4.7 Implementation and experiments

We have implemented the methods VAS, Sturm and the presented *elementary* approach, based on Rolle's theorem. The methods have been all tested for the set of testcases commonly used in the literature [32, 13, 14, 15, 12, 89, 93] plus an additional set of testcases composed by fewnomials of many degrees and number of terms. The classes of polynomials we used are: Laguerre polynomials, Chebyshev polynomials (first and second kind), Wilkinson polynomials, Mignotte polynomials, polynomials with random coefficients of bitsize 20 and 1000, monic polynomials with random coefficients of bitsize

20 and 1000, polynomials which are product of random roots of bitsizes 20 and 1000, fewnomials with random coefficients, of $4, 7, 10$ terms and degree $50, 100, 150, \ldots, 400$.

A note on the implementations used for the comparisons. The implementations that we have used to compare the methods are not optimized; they are pretty similar to the pseudocodes exhibited in this chapter. They have been implemented using the *rational* numbers of the GMP library. Evaluations of the polynomials, with the current implementation, are extremely expensive. But the same evaluation algorithm has been used for all the cases, and all the same *auxiliary tools* have been the same for all the methods, in order to keep the comparison as *fair* as possible.

Comparison. In table 4.9 we show a comparison of the performance of different approaches for the problem of isolating all the positive roots of a polynomial. For this problem, as we it can be seen, the performance of the simple approach is only improved (and not in *every* case) by the VAS, which is the fastest existent algorithm. In table 4.10 we show the same comparison for the problem of isolating *the minimum* positive root, with the same conclusion.

4.8 Chapter results and discussion

We have surveyed the main theoretical results related to the problem of isolating the positive roots of a univariate polynomial; we have introduced an adaptation of Fourier's theorem that requires less amount of computational effort when the input polynomial is a fewnomial. We have also analyzed the performance of an elementary algorithm based on Rolle's theorem, concluding that the performance of the simple method is quite good for most of the cases, only improved by the fastest known method, VAS.

Pol	Time(Sturm)	Time(VAS)	Time(Rolle)
L(5)	0.0004	0.0011	0.0012
L(10)	0.0009	0.0053	0.0149
L(15)	0.0024	0.0131	0.0838
L(20)	0.0049	0.0247	0.3205
L(100)	0.5753	1.0268	2.9087
L(200)	6.6050	6.2375	65.2902
$C_I(5)$	0.0007	0.0004	0.0002
$C_I(10)$	0.0009	0.0036	0.0025
$C_I(15)$	0.0014	0.0063	0.0088
$C_I(20)$	0.0043	0.0174	0.0374
$C_I(100)$	0.2104	0.6839	6.5213
$C_I(200)$	4.8121	4.3168	25.8010
$C_{II}(5)$	0.0001	0.0002	0.0001
$C_{II}(10)$	0.0006	0.0031	0.0021
$C_{II}(15)$	0.0014	0.0056	0.0091
$C_{II}(20)$	0.0037	0.0134	0.0393
$C_{II}(100)$	0.4901	0.6244	1.8060
$C_{II}(200)$	5.0063	4.3132	42.1905
W(5)	0.0003	0.0006	0.0012
W(10)	0.0011	0.0027	0.0159
W(15)	0.0027	0.0064	0.0895
W(20)	0.0057	0.0120	0.3332
W(100)	0.8479	0.4029	3.8649
W(200)	11.5530	2.2332	15.7397
M(5)	0.0008	0.0005	0.0004
M(10)	0.0006	0.0007	0.0007
M(15)	0.0009	0.0009	0.0011
M(20)	0.0013	0.0012	0.0016
M(100)	0.0425	0.0089	0.0086
M(200)	0.4315	0.0313	0.0288
$RC_{20}(5)$	0.0004	0.0009	0.0005
$RC_{20}(5)$	0.0004	0.0004	0.0009
$RC_{20}(5)$	0.0002	0.0003	0.0003
$RC_{20}(10)$	0.0015	0.0011	0.0049
$RC_{20}(10)$	0.0013	0.0008	0.0015
$RC_{20}(10)$	0.0013	0.0006	0.0044
$RC_{20}(15)$	0.0050	0.0011	0.0091
$RC_{20}(15)$	0.0049	0.0009	0.0026
$RC_{20}(15)$	0.0050	0.0008	0.0126
$RC_{20}(20)$	0.0158	0.0016	0.0088
$RC_{20}(20)$	0.0158	0.0011	0.0081
$RC_{20}(20)$	0.0163	0.0017	0.0642
$RC_{20}(100)$	interrupted	0.0107	0.0506
$RC_{20}(100)$	interrupted	0.0105	6.3683
$RC_{20}(100)$	interrupted	0.0104	26.6740
$RC_{20}(200)$	interrupted	0.0258	0.1318
$RC_{20}(200)$	interrupted	0.0364	0.4023
$RC_{20}(200)$	interrupted	0.0483	0.3945
$RC_{1000}(5)$	0.0039	0.0013	0.0018
$RC_{1000}(5)$	0.0029	0.0009	0.0018
$RC_{1000}(5)$	0.0031	0.0005	0.0015
$RC_{1000}(10)$	0.1058	0.0042	0.0969
$RC_{1000}(10)$	0.1059	0.0021	0.0035
$RC_{1000}(10)$	0.1375	0.0058	0.0095
$RC_{1000}(15)$	0.8422	0.0043	0.0337
$RC_{1000}(15)$	0.9804	0.0044	0.0233
$RC_{1000}(15)$	0.8421	0.0041	0.0051
$RC_{1000}(20)$	3.4771	0.0066	0.0730
$RC_{1000}(20)$	3.4792	0.0066	0.0766
$RC_{1000}(20)$	4.1078	0.0069	0.0612
$RC_{1000}(100)$	interrupted	0.1215	1.0416
$RC_{1000}(100)$	interrupted	0.2427	0.6822
$RC_{1000}(100)$	interrupted	0.1207	27.1066
$RC_{1000}(200)$	interrupted	0.9398	2.7383
$RC_{1000}(200)$	interrupted	4.2276	21.8314
$RC_{1000}(200)$	interrupted	0.9281	4.6379
$MRC_{20}(5)$	0.0012	0.0004	0.0015
$MRC_{20}(5)$	0.0004	0.0004	0.0012
$MRC_{20}(5)$	0.0003	0.0004	0.0004
$MRC_{20}(10)$	0.0012	0.0008	0.0008
$MRC_{20}(10)$	0.0012	0.0009	0.0009
$MRC_{20}(10)$	0.0014	0.0008	0.0046
$MRC_{20}(15)$	0.0248	0.0049	0.0750
$MRC_{20}(15)$	0.0059	0.0012	0.0635
$MRC_{20}(15)$	0.0051	0.0011	0.0159
$MRC_{20}(20)$	0.0187	0.0016	0.0978
$MRC_{20}(20)$	0.0160	0.0016	0.0090
$MRC_{20}(20)$	0.0178	0.0029	0.1967
$MRC_{20}(100)$	interrupted	0.0238	0.0728
$MRC_{20}(100)$	interrupted	0.0087	21.3785
$MRC_{20}(100)$	interrupted	0.0089	0.0657

Pol	Time(Sturm)	Time(VAS)	Time(Rolle)
$MRC_{20}(200)$	interrupted	0.0787	0.7058
$MRC_{20}(200)$	interrupted	0.0453	0.1760
$MRC_{20}(200)$	interrupted	0.0200	0.1586
$MRC_{1000}(5)$	0.0034	0.0006	0.0229
$MRC_{1000}(5)$	0.0033	0.0007	0.0134
$MRC_{1000}(5)$	0.0024	0.0002	0.0135
$MRC_{1000}(10)$	15.6917	0.0024	0.7841
$MRC_{1000}(10)$	0.1333	0.0010	0.0097
$MRC_{1000}(10)$	0.1034	0.0010	0.2587
$MRC_{1000}(15)$	1.1069	0.0012	0.0131
$MRC_{1000}(15)$	0.8325	0.0013	3.8767
$MRC_{1000}(15)$	0.8325	0.0013	0.0261
$MRC_{1000}(20)$	4.1043	0.0034	16.1248
$MRC_{1000}(20)$	3.5544	0.0009	0.0160
$MRC_{1000}(20)$	3.8617	0.0022	0.0069
$MRC_{1000}(100)$	interrupted	0.0101	0.0258
$MRC_{1000}(100)$	interrupted	0.0191	0.1702
$MRC_{1000}(100)$	interrupted	0.0181	0.0523
$MRC_{1000}(200)$	interrupted	0.0524	0.3803
$MRC_{1000}(200)$	interrupted	0.0218	0.1793
$MRC_{1000}(200)$	interrupted	0.0382	0.1284
$PoR_{20}(5)$	0.0012	0.0004	0.0009
$PoR_{20}(5)$	0.0005	0.0017	0.0020
$PoR_{20}(5)$	0.0003	0.0002	0.0003
$PoR_{20}(10)$	0.0065	0.0077	0.0206
$PoR_{20}(10)$	0.0040	0.0026	0.0134
$PoR_{20}(10)$	0.0040	0.0014	0.0121
$PoR_{20}(15)$	0.0402	0.0055	0.0661
$PoR_{20}(15)$	0.0610	0.0056	0.0843
$PoR_{20}(15)$	0.0673	0.0112	0.1669
$PoR_{20}(20)$	0.4045	0.0113	0.4704
$PoR_{20}(20)$	0.2621	0.0112	0.4429
$PoR_{20}(20)$	0.3269	0.0149	0.3112
$PoR_{20}(100)$	interrupted	0.5563	3.0863
$PoR_{20}(100)$	interrupted	0.3458	1.0399
$PoR_{20}(100)$	interrupted	0.6532	5.9421
$PoR_{20}(200)$	interrupted	3.3927	23.3433
$PoR_{20}(200)$	interrupted	4.8405	9.5568
$PoR_{20}(200)$	interrupted	3.3570	23.9424
$PoR_{1000}(5)$	0.0062	0.0014	0.0121
$PoR_{1000}(5)$	0.0055	0.0011	0.0079
$PoR_{1000}(5)$	0.0046	0.0006	0.0068
$PoR_{1000}(10)$	0.9598	0.0052	0.9810
$PoR_{1000}(10)$	1.0177	0.0051	0.9032
$PoR_{1000}(10)$	0.7815	0.0041	0.7395
$PoR_{1000}(15)$	11.6267	0.0077	7.1961
$PoR_{1000}(15)$	15.1496	0.0266	10.5409
$PoR_{1000}(15)$	20.5173	0.0134	9.8606
$PoR_{1000}(20)$	interrupted	0.0184	21.4624
$PoR_{1000}(20)$	interrupted	0.0468	0.5310
$PoR_{1000}(20)$	interrupted	0.0432	0.1456
$PoR_{1000}(100)$	interrupted	9.5364	51.6291
$PoR_{1000}(100)$	interrupted	4.3834	41.4216
$PoR_{1000}(100)$	interrupted	3.8163	34.0226
$PoR_{1000}(200)$	interrupted	35.3291	59.2604
$PoR_{1000}(200)$	interrupted	31.4577	76.3000
$PoR_{1000}(200)$	interrupted	32.0114	73.3298
$Fewnomial_4(50)$	0.0206	0.0003	0.0009
$Fewnomial_7(50)$	0.5791	0.0016	0.0014
$Fewnomial_{10}(50)$	1.1025	0.0016	0.0137
$Fewnomial_4(100)$	0.0036	0.0002	0.0003
$Fewnomial_7(100)$	28.8476	0.0031	0.0057
$Fewnomial_{10}(100)$	interrupted	0.0002	0.0004
$Fewnomial_4(150)$	0.0224	0.0004	0.0026
$Fewnomial_7(150)$	interrupted	0.0057	0.0404
$Fewnomial_{10}(150)$	interrupted	0.0058	0.0114
$Fewnomial_4(200)$	3.8274	0.0164	0.0242
$Fewnomial_7(200)$	interrupted	0.0101	0.0488
$Fewnomial_{10}(200)$	interrupted	0.0109	0.1798
$Fewnomial_4(250)$	28.4305	0.0006	0.0189
$Fewnomial_7(250)$	interrupted	0.0157	0.0061
$Fewnomial_{10}(250)$	interrupted	0.0328	0.2477
$Fewnomial_4(300)$	1.1728	0.4392	0.0262
$Fewnomial_7(300)$	interrupted	0.0222	0.1030
$Fewnomial_{10}(300)$	interrupted	0.0242	0.3339
$Fewnomial_4(350)$	interrupted	0.0005	0.0014
$Fewnomial_7(350)$	interrupted	0.0323	0.0511
$Fewnomial_{10}(350)$	interrupted	0.0360	0.0548
$Fewnomial_4(400)$	22.2213	0.0833	0.0657
$Fewnomial_7(400)$	interrupted	0.0802	0.0042
$Fewnomial_{10}(400)$	interrupted	0.0450	0.1971

Figure 4.9: Running times of the methods of Sturm, VAS and our elementary Rolle-based approach, for the problem of isolating *all* the positive roots of the input polynomial. **Polynomials** • L: Laguerre • C_I and C_{II}: Chebyshev (first and second kind) • W: Wilkinson • M: Mignotte • RC: Random coefficients • MRC: Monic with random coefficients • PoR: Product of Roots • $Fewnomial_i(d)$: fewnomial with i terms and degree d, coefficients chosen at random.

Pol	Time(Sturm)	Time(VAS)	Time(Rolle)
L(5)	0.0003	0.0002	0.0012
L(10)	0.0008	0.0009	0.0149
L(15)	0.0017	0.0021	0.0853
L(20)	0.0028	0.0028	0.3235
L(100)	0.1027	0.0466	0.2480
L(200)	0.6530	0.1934	1.9600
$C_I(5)$	0.0007	0.0004	0.0002
$C_I(10)$	0.0005	0.0012	0.0025
$C_I(15)$	0.0010	0.0017	0.0089
$C_I(20)$	0.0020	0.0040	0.0383
$C_I(100)$	0.0353	0.0725	0.7084
$C_I(200)$	0.4567	0.2854	1.9235
$C_{II}(5)$	0.0001	0.0002	0.0001
$C_{II}(10)$	0.0005	0.0013	0.0021
$C_{II}(15)$	0.0010	0.0017	0.0092
$C_{II}(20)$	0.0021	0.0038	0.0402
$C_{II}(100)$	0.0800	0.0708	0.2688
$C_{II}(200)$	0.4797	0.2819	0.9230
W(5)	0.0002	0.0002	0.0012
W(10)	0.0007	0.0004	0.0162
W(15)	0.0014	0.0007	0.0905
W(20)	0.0026	0.0009	0.3343
W(100)	0.1045	0.0048	0.0400
W(200)	0.8086	0.0113	0.1090
M(5)	0.0007	0.0005	0.0004
M(10)	0.0006	0.0007	0.0007
M(15)	0.0009	0.0008	0.0011
M(20)	0.0013	0.0011	0.0016
M(100)	0.0427	0.0073	0.0086
M(200)	0.4306	0.0246	0.0287
$RC_{20}(5)$	0.0004	0.0009	0.0005
$RC_{20}(5)$	0.0004	0.0004	0.0009
$RC_{20}(5)$	0.0002	0.0003	0.0003
$RC_{20}(10)$	0.0015	0.0011	0.0049
$RC_{20}(10)$	0.0013	0.0008	0.0015
$RC_{20}(10)$	0.0013	0.0006	0.0045
$RC_{20}(15)$	0.0050	0.0010	0.0091
$RC_{20}(15)$	0.0049	0.0008	0.0026
$RC_{20}(15)$	0.0050	0.0008	0.0125
$RC_{20}(20)$	0.0158	0.0015	0.0088
$RC_{20}(20)$	0.0158	0.0009	0.0081
$RC_{20}(20)$	0.0165	0.0017	0.0641
$RC_{20}(100)$	interrupted	0.0087	0.0932
$RC_{20}(100)$	interrupted	0.0084	6.3644
$RC_{20}(100)$	interrupted	0.0081	26.7029
$RC_{20}(200)$	interrupted	0.0193	0.2055
$RC_{20}(200)$	interrupted	0.0239	0.1447
$RC_{20}(200)$	interrupted	0.0298	0.0535
$RC_{1000}(5)$	0.0038	0.0006	0.0018
$RC_{1000}(5)$	0.0029	0.0005	0.0018
$RC_{1000}(5)$	0.0030	0.0004	0.0015
$RC_{1000}(10)$	0.1063	0.0015	0.0978
$RC_{1000}(10)$	0.1068	0.0008	0.0035
$RC_{1000}(10)$	0.1393	0.0007	0.0096
$RC_{1000}(15)$	0.8485	0.0014	0.0342
$RC_{1000}(15)$	0.9864	0.0014	0.0234
$RC_{1000}(15)$	0.8483	0.0012	0.0051
$RC_{1000}(20)$	3.4929	0.0015	0.0737
$RC_{1000}(20)$	3.4937	0.0015	0.0775
$RC_{1000}(20)$	4.1251	0.0018	0.0617
$RC_{1000}(100)$	interrupted	0.0082	0.0664
$RC_{1000}(100)$	interrupted	0.0173	0.1289
$RC_{1000}(100)$	interrupted	0.0083	27.1548
$RC_{1000}(200)$	interrupted	0.0182	0.1375
$RC_{1000}(200)$	interrupted	0.1280	0.6162
$RC_{1000}(200)$	interrupted	0.0332	0.3091
$MRC_{20}(5)$	0.0012	0.0004	0.0015
$MRC_{20}(5)$	0.0004	0.0004	0.0012
$MRC_{20}(5)$	0.0003	0.0004	0.0004
$MRC_{20}(10)$	0.0012	0.0008	0.0008
$MRC_{20}(10)$	0.0012	0.0009	0.0009
$MRC_{20}(10)$	0.0014	0.0009	0.0046
$MRC_{20}(15)$	0.0252	0.0034	0.0760
$MRC_{20}(15)$	0.0061	0.0011	0.0638
$MRC_{20}(15)$	0.0052	0.0011	0.0160
$MRC_{20}(20)$	0.0189	0.0016	0.0982
$MRC_{20}(20)$	0.0161	0.0016	0.0091
$MRC_{20}(20)$	0.0179	0.0029	0.1983
$MRC_{20}(100)$	interrupted	0.0228	0.0364
$MRC_{20}(100)$	interrupted	0.0081	21.3858
$MRC_{20}(100)$	interrupted	0.0089	0.0564

Pol	Time(Sturm)	Time(VAS)	Time(Rolle)
$MRC_{20}(200)$	interrupted	0.0303	0.2094
$MRC_{20}(200)$	interrupted	0.0169	0.1178
$MRC_{20}(200)$	interrupted	0.0180	0.1223
$MRC_{1000}(5)$	0.0033	0.0006	0.0229
$MRC_{1000}(5)$	0.0033	0.0007	0.0134
$MRC_{1000}(5)$	0.0024	0.0002	0.0135
$MRC_{1000}(10)$	15.7168	0.0014	0.7882
$MRC_{1000}(10)$	0.1334	0.0010	0.0100
$MRC_{1000}(10)$	0.1047	0.0010	0.2603
$MRC_{1000}(15)$	1.1132	0.0011	0.0131
$MRC_{1000}(15)$	0.8359	0.0014	3.8840
$MRC_{1000}(15)$	0.8321	0.0013	0.0260
$MRC_{1000}(20)$	4.1064	0.0033	17.7283
$MRC_{1000}(20)$	3.4553	0.0009	0.0159
$MRC_{1000}(20)$	3.4575	0.0017	0.0065
$MRC_{1000}(100)$	interrupted	0.0093	0.0237
$MRC_{1000}(100)$	interrupted	0.0174	0.1190
$MRC_{1000}(100)$	interrupted	0.0173	0.1498
$MRC_{1000}(200)$	interrupted	0.0345	0.2228
$MRC_{1000}(200)$	interrupted	0.0188	0.0901
$MRC_{1000}(200)$	interrupted	0.0176	0.1072
$PoR_{20}(5)$	0.0012	0.0004	0.0009
$PoR_{20}(5)$	0.0004	0.0008	0.0020
$PoR_{20}(5)$	0.0003	0.0002	0.0003
$PoR_{20}(10)$	0.0037	0.0007	0.0209
$PoR_{20}(10)$	0.0036	0.0013	0.0135
$PoR_{20}(10)$	0.0029	0.0010	0.0121
$PoR_{20}(15)$	0.0275	0.0010	0.0662
$PoR_{20}(15)$	0.0390	0.0013	0.0845
$PoR_{20}(15)$	0.0361	0.0038	0.1674
$PoR_{20}(20)$	0.1585	0.0018	0.4729
$PoR_{20}(20)$	0.1446	0.0027	0.4445
$PoR_{20}(20)$	0.2433	0.0021	0.3137
$PoR_{20}(100)$	interrupted	0.0115	0.0415
$PoR_{20}(100)$	interrupted	0.0114	0.0724
$PoR_{20}(100)$	interrupted	0.0115	0.0350
$PoR_{20}(200)$	interrupted	0.0791	0.5890
$PoR_{20}(200)$	interrupted	0.0484	0.2411
$PoR_{20}(200)$	interrupted	0.0338	0.3822
$PoR_{1000}(5)$	0.0082	0.0014	0.0121
$PoR_{1000}(5)$	0.0046	0.0006	0.0079
$PoR_{1000}(5)$	0.0046	0.0007	0.0068
$PoR_{1000}(10)$	0.6636	0.0016	0.9810
$PoR_{1000}(10)$	0.7225	0.0017	0.9039
$PoR_{1000}(10)$	0.5446	0.0016	0.7374
$PoR_{1000}(15)$	8.0876	0.0038	7.2101
$PoR_{1000}(15)$	9.3840	0.0053	10.8276
$PoR_{1000}(15)$	10.7521	0.0036	9.8444
$PoR_{1000}(20)$	interrupted	0.0052	21.4443
$PoR_{1000}(20)$	interrupted	0.0034	0.0090
$PoR_{1000}(20)$	interrupted	0.0034	0.0248
$PoR_{1000}(100)$	interrupted	0.0907	0.2807
$PoR_{1000}(100)$	interrupted	0.0922	0.8729
$PoR_{1000}(100)$	interrupted	0.0917	0.4970
$PoR_{1000}(200)$	interrupted	1.0650	2.4857
$PoR_{1000}(200)$	interrupted	0.8327	7.2031
$PoR_{1000}(200)$	interrupted	0.6698	6.6873
$Fewnomial_4(50)$	0.0206	0.0003	0.0010
$Fewnomial_7(50)$	0.5875	0.0012	0.0014
$Fewnomial_{10}(50)$	1.1076	0.0012	0.0138
$Fewnomial_4(100)$	0.0001	0.0001	0.0003
$Fewnomial_7(100)$	28.8697	0.0018	0.0057
$Fewnomial_{10}(100)$	0.0001	0.0001	0.0004
$Fewnomial_4(150)$	0.0223	0.0004	0.0026
$Fewnomial_7(150)$	interrupted	0.0033	0.0402
$Fewnomial_{10}(150)$	interrupted	0.0032	0.0114
$Fewnomial_4(200)$	3.8269	0.0107	0.0243
$Fewnomial_7(200)$	interrupted	0.0051	0.0484
$Fewnomial_{10}(200)$	interrupted	0.0057	0.2217
$Fewnomial_4(250)$	28.9472	0.0005	0.0190
$Fewnomial_7(250)$	interrupted	0.0073	0.0061
$Fewnomial_{10}(250)$	interrupted	0.0150	0.4377
$Fewnomial_4(300)$	1.2803	0.0154	0.0284
$Fewnomial_7(300)$	interrupted	0.0059	0.1027
$Fewnomial_{10}(300)$	interrupted	0.0108	0.3335
$Fewnomial_4(350)$	0.0003	0.0004	0.0014
$Fewnomial_7(350)$	interrupted	0.0078	0.0504
$Fewnomial_{10}(350)$	interrupted	0.0140	0.0547
$Fewnomial_4(400)$	22.0639	0.0361	0.0649
$Fewnomial_7(400)$	interrupted	0.0500	0.0042
$Fewnomial_{10}(400)$	interrupted	0.0100	0.1950

Figure 4.10: Running times of the methods of Sturm, VAS and our elementary Rolle-based approach, for the problem of isolating *the minimum* positive root of the input polynomial. **Polynomials** • L: Laguerre • C_I and C_{II}: Chebyshev (first and second kind) • W: Wilkinson • M: Mignotte • RC: Random coefficients • MRC: Monic with random coefficients • PoR: Product of Roots • $\text{Fewnomial}_i(d)$: fewnomial with i terms and degree d, coefficients chosen at random.

Chapter 5

One application: higher-order Quantized State Systems

In the present chapter we present an application of the methods shown in Chapter 4, in the context of numerical integration methods. The present case study has been developed in collaboration with Dr. Federico Bergero from CIFASIS-CONICET, Rosario, Argentina.

Contributions in this chapter This chapter introduces the application of root isolation techniques to the Quatized State Systems method of integrating systems of Ordinary Differential Equations [22, 68]. Currently there are only QSS methods of order[1] less than or equal to 4 [64, 65, 67]. We make an observation which, combined with the introduction of the techniques presented in Chapter 4 (with small adaptations), allows the generalization to QSS of any order, and we have obtained experimental results showing that higher order methods do require a considerably fewer number of iterations than current existing approaches.

5.1 The problems

The *simulation* of a system allows to answer questions about it. In this chapter, we will consider simulation of processes modeled by an Ordinary Differential Equations (ODE) of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \tag{5.1}$$

[1] See the note on page 90 on the two different meanings of the word *order* in the current context.

where $\mathbf{x}(t)$ is the state vector with initial states values $\mathbf{x}(t = t_0) = \mathbf{x}_0$.

One example of such model is as follows. Imagine that we want to study a stone in free fall, in the vacuum. Of course, we could propose a lot of different models for this, depending mainly on *the kind of questions* that we want to be able to answer. Let us consider the model in which the *state* of the process is described by two quantities: the *position* of the stone, and its *velocity*. Let us call $x_1(t)$ to the first (i.e.: $x_1(t)$ is the distance travelled by the stone at the moment t) and $x_2(t)$ to the second (i.e.: $x_2(t)$ is the velocity of the stone at the moment t). We will use in our model the same laws that the classical mechanics model proposes; so we have:

$$\begin{cases} \frac{d}{dt}x_1(t) &= f_1(x_1(t), x_2(t), t) &= x_2(t) \\ \frac{d}{dt}x_2(t) &= f_2(x_1(t), x_2(t), t) &= 9.8 \end{cases}$$

In this particular example, we have an unusual situation that makes computations easier: one of the *state variables* is precisely the derivative of the other.

To *integrate* a system is to compute the state variables, which are the functions $x_i(t)$.

Most of the times it is too complicated to obtain analytical solutions for the state variables, and it is necessary to obtain numerical approximations. To *simulate* a system is to run a program which updates the system state following some given strategy (for example: every some fixed period of time).

In our problem, suppose that we want to know the distance that the stone travelled at $t = 10$, and assume that our initial condition was $x_1(0) = 0$ and $x_2(0) = 0$ (i.e.: the stone did not travel any distance before starting, and its velocity was 0 at that moment). Of course, this particularly simple system can be easily integrated and we will obtain that $x_1(t) = x_1(0) + x_2(0)t + \frac{1}{2}9.8t^2$, so we have that $x_1(10) = 490$. But if we do not have such an analytical solution, we could *simulate* the system up to time $t = 10$ and check the value of x_1 at that moment. There are different techniques to simulate a system. In this chapter we will focus on a method named Quantized State System [29].

5.2 Quantized State Systems

The Quantized State System numerical integration methods [29] are a family of algorithms to approximately solve a system of Ordinary Differential Equations (ODE) with

the form of (5.1).

Given the ODE of Eq. (5.1), the first order Quantized State System method (QSS1) [68] approximates it by

$$\dot{\widetilde{\mathbf{x}}}(t) = \mathbf{f}(\mathbf{q}(t), t) \tag{5.2}$$

and it considers $\widetilde{\mathbf{x}}(t_0) = \mathbf{x}(t_0)$. Just to clarify the abbreviations, let us expand them. The ODE of Eq. (5.1) is:

$$\begin{cases} \frac{d}{dt}x_1(t) = f_1(x_1(t), x_2(t), \ldots, x_m(t), t) \\ \frac{d}{dt}x_2(t) = f_2(x_1(t), x_2(t), \ldots, x_m(t), t) \\ \vdots \\ \frac{d}{dt}x_m(t) = f_m(x_1(t), x_2(t), \ldots, x_m(t), t) \\ x_1(t_0), x_2(t_0), \ldots, x_m(t_0) \text{ are given} \end{cases}$$

The QSS1, instead of computing $x_1(t), x_2(t), \ldots, x_m(t)$, computes *another* set of functions, named $\widetilde{x}_1(t), \widetilde{x}_2(t), \ldots, \widetilde{x}_m(t)$, where $\widetilde{x}_i(t)$ is an approximation of $x_i(t)$. Thus, the abbreviation of Eq. (5.2) means:

$$\begin{cases} \frac{d}{dt}\widetilde{x}_1(t) = f_1(q_1(t), q_2(t), \ldots, q_m(t), t) \\ \frac{d}{dt}\widetilde{x}_2(t) = f_2(q_1(t), q_2(t), \ldots, q_m(t), t) \\ \vdots \\ \frac{d}{dt}\widetilde{x}_m(t) = f_m(q_1(t), q_2(t), \ldots, q_m(t), t) \\ \widetilde{x}_1(t_0) = x_1(t_0) \\ \widetilde{x}_2(t_0) = x_2(t_0) \\ \vdots \\ \widetilde{x}_m(t_0) = x_m(t_0) \end{cases}$$

Here, $\mathbf{q}$ is the *quantized state vector*. Its entries are component-wise related with those of the state vector $\widetilde{\mathbf{x}}$ by the following *hysteretic quantization function*:

$$q_j(t) = \begin{cases} \widetilde{x}_j(t), \text{ if } |\widetilde{x}_j(t) - q_j(t^-)| \geq \Delta Q_j \\ q_j(t^-), \text{ otherwise} \end{cases} \tag{5.3}$$

where ΔQ_j is called *quantum* and $q_j(t^-)$ denotes the left-sided limit of q_j at time t.

Equation (5.3) says that the quantized state variable $q_j(t)$ only changes when its

difference with the state variable $\widetilde{x}_j(t)$ becomes greater than or equal to the quantum ΔQ_j. When this condition is reached, the quantized state variable is reset to the value of its associated state variable, i.e., $q_j(t) = \widetilde{x}_j(t)$.

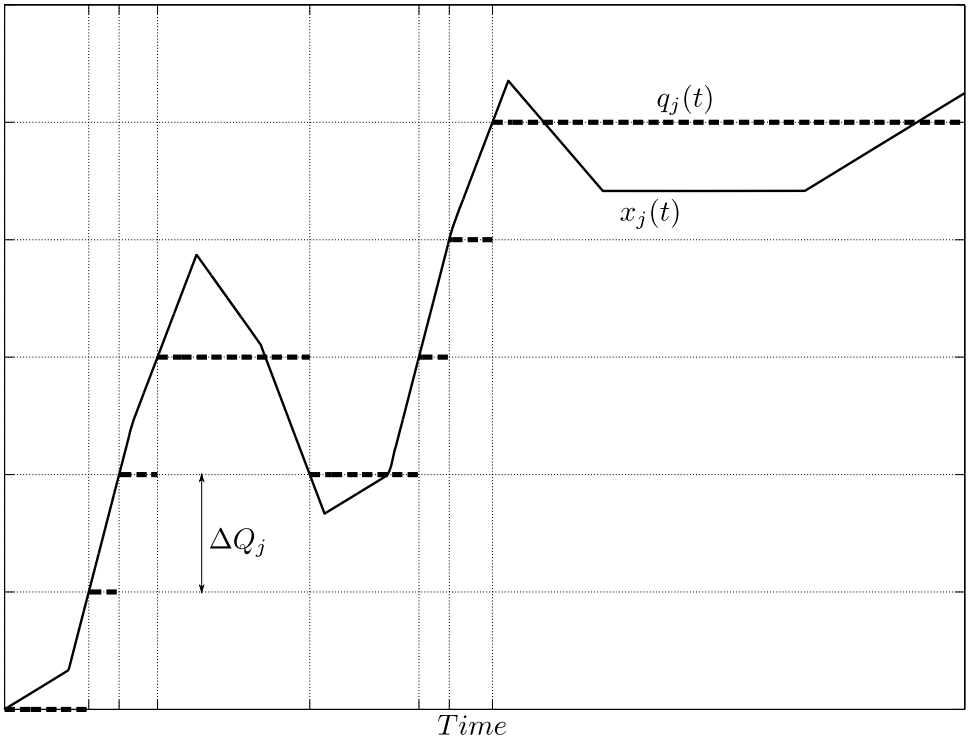
Since the quantized state trajectories $q_j(t)$ are piecewise constant, the state derivatives $\dot{\widetilde{x}}_j(t)$ also follow piecewise constant trajectories and, consequently, the states $\widetilde{x}_j(t)$ follow piecewise linear trajectories.

Let us explain in a more concise way. The QSS1 method, for the problem of computing the functions $x_1(t), x_2(t), \ldots, x_m(t)$ of the problem (5.1), computes the functions $\widetilde{x}_1(t), \widetilde{x}_2(t), \ldots, \widetilde{x}_m(t)$ of problem (5.2). The initial conditions of both problems are equal.

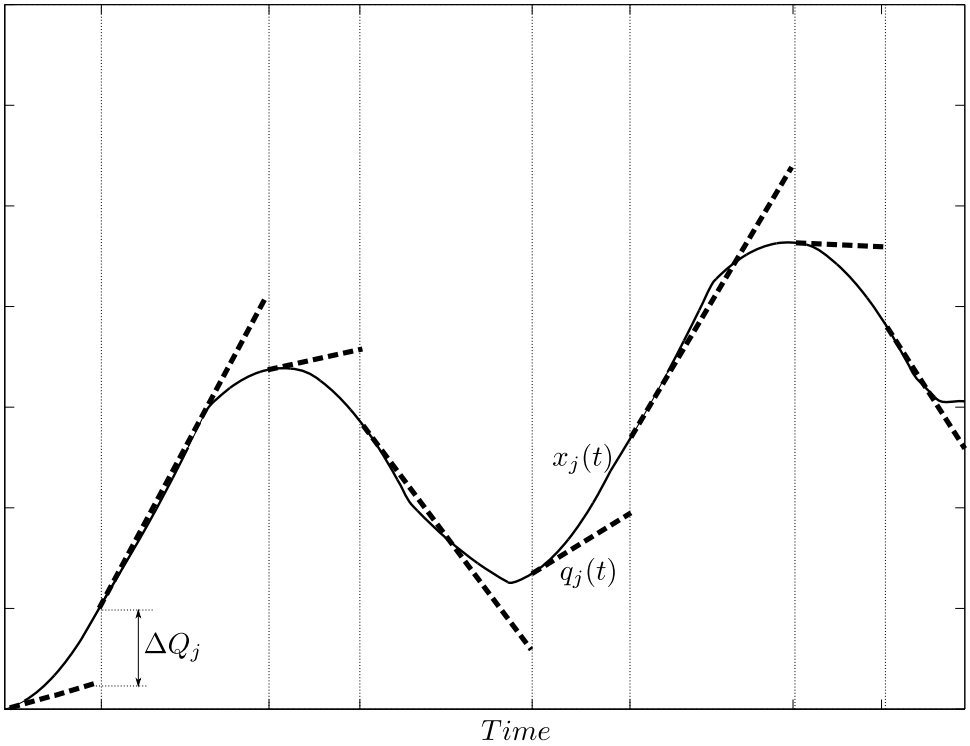
QSS1 proceeds as follows: we define a threshold for each component of $\widetilde{\mathbf{x}}$, and we call it $\Delta Q = (\Delta Q_1, \Delta Q_2, \ldots, \Delta Q_m)$. The values for $\widetilde{x}_1(t_0), \widetilde{x}_2(t_0), \ldots, \widetilde{x}_m(t_0)$ are the initial condition. Initially, QSS starts are $t = t_0$. QSS advances in time through discrete steps; not necessary equal in width. Let us see how QSS proceeds to *advance* the time when it is at $t = t_w$. Assume for now the existence of a *black box* which computes the *next time* t, *with* $t > t_w$, *for which we know that it must happen that* $|\widetilde{x}_j(t) - q_j(t^-)| \geq \Delta Q_j$, *for some* j. QSS will set the new *current* time t_{w+1} to be t. At this point t, QSS updates the state of the system. It updates the value of $\widetilde{x}_j$, by considering that its derivative have remained constant (and equal to q_j) since the last time it was updated. After having updated $\widetilde{x}_j$, it needs to update the values of all the $\dot{\widetilde{x}}_1(x), \dot{\widetilde{x}}_1(x), \ldots, \dot{\widetilde{x}}_m(x)$, since they depend also on the variable $\widetilde{x}_j$, which has been just updated.

Notice that the way in which QSS proceeds does not quantizate *the time*, but *the state variables*. This is the main difference between QSS and other approaches, which makes QSS more suitable for a variety of situations.

The *black box* mentioned in the previous paragraph computes the first time $t > t_w$ for which some of the conditions $|\widetilde{x}_j(t) - q_j(t^-)| \geq \Delta Q_j$ holds. Given that $\widetilde{x}_j(t)$ is a linear function and $q_j(t^-)$ is a constant, it is very simple to compute, for each j, the time t_j at which we know that the condition $|\widetilde{x}_j(t) - q_j(t^-)| \geq \Delta Q_j$ starts to hold. We can take the minimum of these times and that could be a way of implementing such a black box for QSS1.



(a) Example of one trajectory of one state variable $\widetilde{x}_j$, of the system used by QSS1 to approximate the input problem. Note that the trajectory is piecewise-linear. In solid black line: the trajectory of the state variable $\widetilde{x}_j$. In dotted lines: the trajectory of the quantized state variable q_j.



(b) Example of one trajectory of one state variable $\widetilde{x}_j$, of the system used by QSS2 to approximate the input problem. Note that the trajectory is piecewise-quadratic. In solid black line: the trajectory of the state variable $\widetilde{x}_j$. In dotted lines: the trajectory of the quantized state variable q_j.

5.2.1 QSS2, QSS3, QSS4

The differences between QSS1 and higher-degrees methods QSS2, QSS3 and QSS4 is that the quantized variables are not piecewise-constant, but piecewise-{linear, quadratic, cubic} polynomials. We can imagine in the following way: in QSS1, the derivatives of the state variables, in the approximated system, are piecewise-constants. Thus, the state variables in that system are approximated through piecewise-linear trajectories. In QSS2, the derivatives of the state variables are piecewise-linear and consequently the system that we want to integrate is approximated through a piecewise-quadratic system. Similarly, in QSS3 and QSS4 the derivatives of state variables are piecewise-quadratic and piecewise-cubic, respectively; and the systems are approximated with piecewise-cubic and piecewise-quartic systems, respectively.

The idea behind all the four methods is the same: it changes the degree of the pieces used in the approximation of the system being integrated. The advantage of increasing the degree of such functions is that the number of pieces needed decreases, since the precision of the approximation increases and consequently, the time steps taken are larger.

One observation concerning all the four methods We might want to compute the value of some state variable $\widetilde{x}_j$ at some *specific* time t. For example: in our previous example of the stone in free fall, we might want to know the distance travelled by the stone after π units of time. In this cases, as soon as our QSS method arrives into a time greater than or equal to t, we will need to adjust the required value to the time t, by considering the last value of it and the value of its derivative since its last update.

On two meanings of the word *order* In the present context, when speaking about a method, we could say *a system* of order n, or *a method* of order n. The two concepts are different things. The order of *a QSS method* is the degree of the polynomials which are the pieces of the approximated system. For example, in QSS1, the state variables of the approximated system are piecewise-linear. Thus, the degrees of these polynomials are 1 and the degree of the method is 1. Similarly, the degrees of QSS2, QSS3 and QSS4 methods are 2, 3 and 4, respectively. The order of *a system* like (5.1) is the number of state variables. In our example of the stone in free fall, we had two state variables; so it was of order two. The system (5.1) has order m, since it has m state variables (as can be

seen in the expansion of it, in page 87). In this chapter we present a way to generalize the QSS method and allow it to have order greater than 4. Any one of the QSS versions, even QSS1, can be applied to solve, for example, systems of order 1000. The main advantage of higher degrees approaches is, as we will mention in next sections, the gain in precision and consequently a reduction in the number of required simulation steps.

Higher orders

The limitation that prevented the generalization of QSS to *any order* comes from the mentioned *black box* of the previous explanation. For systems whose computed approximations are piecewise-linear, we need to find the smallest roots of polynomials of degree 1, and that is an easy task. Similarly, for systems whose computed approximations are piecewise-{degree d}, we need to find the smallest roots of polynomials of degree d. The current implementations of these QSS methods relies on the fact that polynomials of degrees less than 5 have analytic expressions for its roots.

Federico Bergero pointed out that in fact the black box is not using the algebraic form of the roots of these polynomials, they just need a numerical (as tight as possible, lower bound) for the value of the minimum positive root of them. After this observation, we can introduce adaptations of the methods exposed in Chapter 4, for computing such values. These adaptations allow the introduction of QSS methods of fifth and higher order, for which we were not able to find in any reference in the literature.

5.3 Implementation of higher order QSS methods

As we mentioned, the observation that leads to the work presented in this chapter is the following fact: current QSS implementations of orders 1, 2, 3 and 4 rely on the existence of a way to compute the minimum positive root of a polynomial whose degree is equal to the order of the method. But they do not use at any moment the *algebraic expressions* of these roots. In fact, they only make use of its *approximated numerical value*. This observation made by Federico Bergero allows us to introduce methods of root isolation and, in this way, be able to run simulations of systems approximated with polynomial of order greater than 4.

Algorithm 11 VAS adapted for isolating the minimum positive root of p

```
1: procedure VASIsolateMinimum(p, M)                ▷ p is a squarefree polynomial.
2:     sc ← number of sign changes in p(x)
3:     if sc = 0 then
4:         return ∅
5:     if sc = 1 then
6:         return {(min(M(0), M(∞)), max(M(0), M(∞)))}
7:     α ← lower bound on the positive roots of p
8:     if α > 16 then
9:         p(x) ← p(αx)
10:        M(x) ← M(αx)
11:        α ← 1
12:    if α ≥ 1 then
13:        p(x) ← p(x + α)
14:        M(x) ← M(x + α)
15:    p01(x) ← (x + 1)^(deg p) p(1/(x+1)), M01(x) ← M(1/(x+1))
16:    p1∞(x) ← p(x + 1), M1∞(x) ← M(x + 1)
17:    if M01((0, +∞)) is to the left of M1∞((0, +∞)) then
18:        if VASIsolateMinimum(p01, M01) ≠ ∅ then
19:            return VASIsolateMinimum(p01, M01)
20:    else
21:        if VASIsolateMinimum(p1∞, M1∞) ≠ ∅ then
22:            return VASIsolateMinimum(p1∞, M1∞)
23:    if p(1) = 0 then
24:        return [M(1), M(1)]
25:    if M01((0, +∞)) is to the left of M1∞((0, +∞)) then
26:        if VASIsolateMinimum(p1∞, M1∞) ≠ ∅ then
27:            return VASIsolateMinimum(p1∞, M1∞)
28:    else
29:        if VASIsolateMinimum(p01, M01) ≠ ∅ then
30:            return VASIsolateMinimum(p01, M01)
31:    return ∅
```

The algorithms showed in Chapter 4 for root isolation can be adapted to isolate *just the minimum* positive root of the input polynomial, instead of all of them. Algorithm 11 shows the adaptation of the VAS method.

5.3.1 Implementation in PowerDEVS

PowerDEVS [22] is general purpose Discrete Event System simulation tool that implements all the QSS methods. Models are described graphically by dragging and dropping and connecting blocks. The behavior of each block is described as a C++ class. The QSS methods are implemented as an Integrator block as show in Figure 5.2. This block runs the QSS algorithm previously described and uses an external function

```
double minposroot(double coeff[], int order);
```

to find the minimum positive root of a polynomial. This function returns the analytical value for the first positive root (if any) thus it only works with polynomials up to fourth order.

We have included in this block the possibility to use the numerical root finding algorithms developed in this Thesis by including three extra functions:

```
double minposroot_vas(double coeff[], int order);
double minposroot_sturm(double coeff[], int order);
double minposroot_rolle(double coeff[], int order);
```

which implement the three different root bounding methods.

Also the static PowerDEVS blocks (like the Adder in Figure 5.2) were adapted to support higher order approximation.

The changes can be viewed online on the PowerDEVS website [2].

5.4 Examples

In this section we will test the prototype implementation analyzing the quality and performance of the proposed methods.

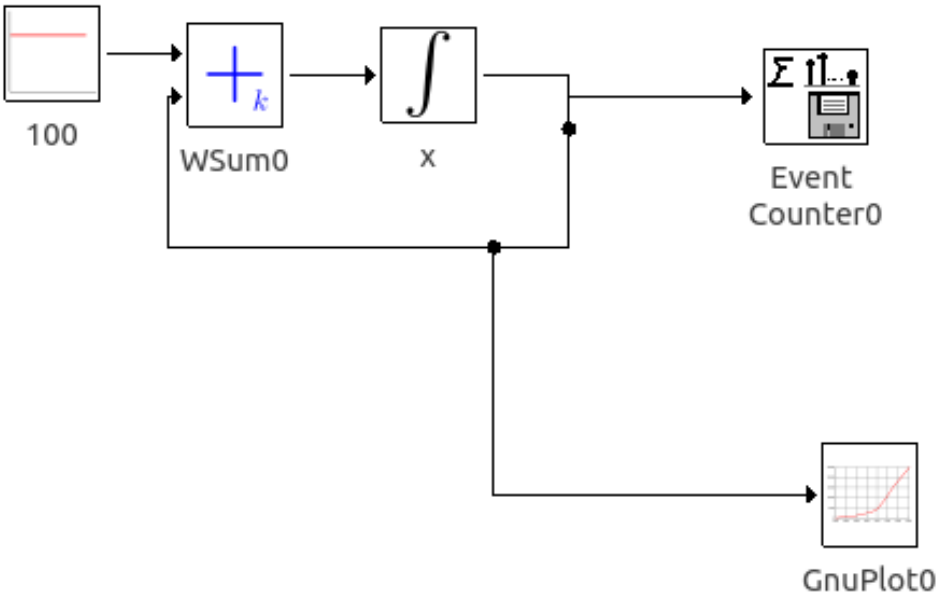


Figure 5.2: PowerDEVS model of Example I

Example I: A first order system

The first system we will study is a first order differential equation given by:

$$\dot{x}(t) = 100 - \frac{x(t)}{1000} \tag{5.4}$$

This simple problem has an analytical solution, i.e. an exponential like the one shown in Figure 5.3.

Example II: An oscillatory system

On the second example we will focus on a second-order oscillatory system given by the following ODE equations:

$$\begin{aligned} \dot{x_1}(t) &= x_2(t) \\ \dot{x_2}(t) &= -x_1(t) \end{aligned}$$

with initial conditions $x_1(0) = 0$ and $x_2(0) = 1$.

The analytical solution of these equations (shown in Fig 5.4) are a pair of sine and cosine functions. This kind of oscillatory system requires very high numerical precision since all numerical integration methods tend to provide a damped solution instead of a

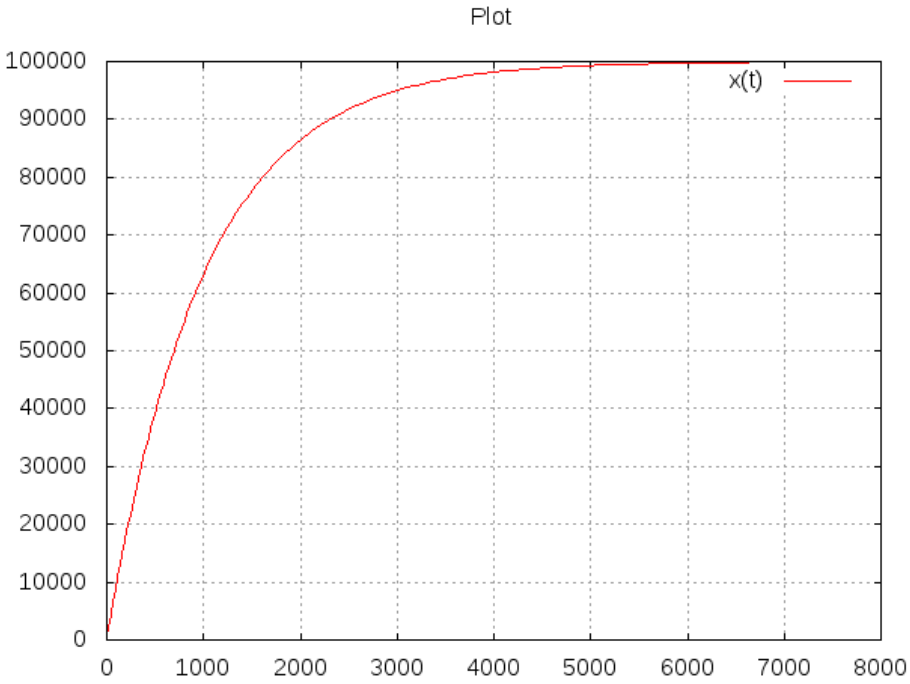


Figure 5.3: State Trajectories for Example I

purely harmonic one. Thus we will study the quality of the solution obtained with each method and the computational cost associated with it for a given tolerance of 1×10^{-10}.

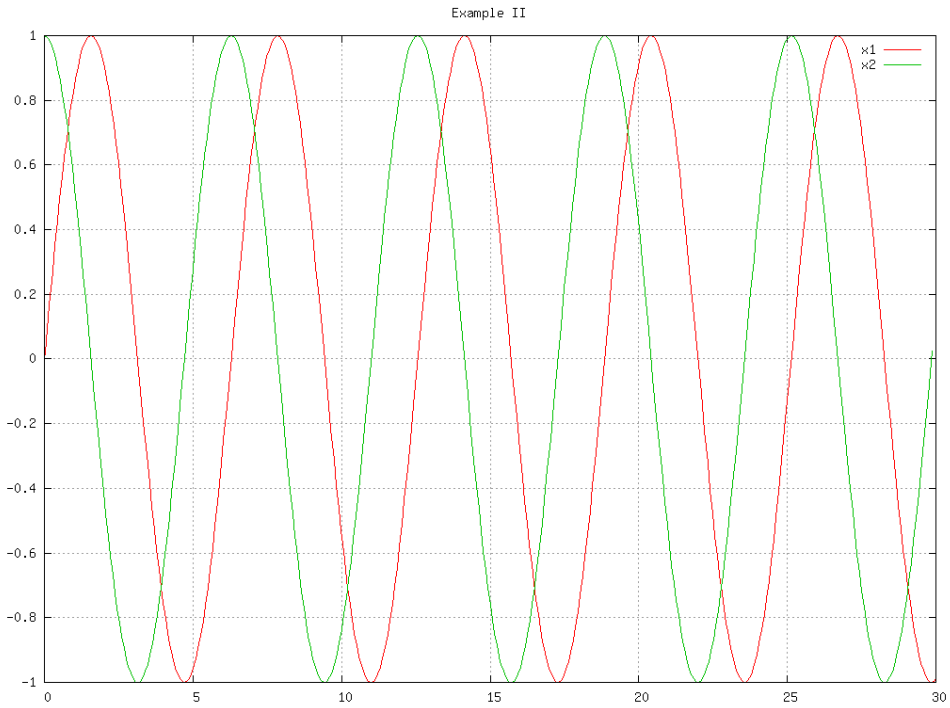


Figure 5.4: State Trajectories for Example II

Table 5.1 shows the number of steps and maximum error (compared to the analytical solution) for QSS4 both analytical and numerical and for QSS5 up to QSS8 using the numerical root finding algorithm presented in this Thesis. There we see that number of steps for a given tolerance decreases for higher order method. Also the solution given by the higher order QSS methods is more accurate.

Table 5.1: Simulation steps and error for Example II.

	Steps	Max. Error
QSS4 Analytical	7104	2.4×10^{-6}
QSS4 Numerical	7104	2.4×10^{-6}
QSS5	1996	3.0×10^{-7}
QSS6	848	5.6×10^{-8}
QSS7	462	1.0×10^{-8}
QSS8	279	2.90×10^{-9}

5.4.1 Performance Analysis

We will now compare the performance of both root finding approaches, the analytical one and the numerical one. This can only be done for fourth order (or lower) QSS method.

As shown before, using higher-order methods will require less simulation steps, thus fewer calls to the root-finding functions.

Therefore we expect that, at some point, the extra cost in the numerical root finding function will be compensated by incurring in fewer calls to it, leading a faster simulation.

Table 5.2 shows the timing consumed by the different root finding algorithms for QSS of different orders, namely, the analytical one, and the three numerical ones, VAS, Sturm and Rolle for a given tolerance of 1×10^{-10}.

We see that from the three numerical algorithms VAS is the fastest one. Also we see that using VAS with higher-order methods (sixth and higher) we get a comparable performance to that of QSS3 and QSS4 since the later needs 1200 times more calls to the root finding functions. These facts again show the feasibility of using numerical root finding algorithms to implement higher-order QSS methods, enabling further research in this topic, which is significative in the QSS area.

Table 5.2: CPU time and number of calls for Example I.

	CPU Time (ms)	Calls
QSS3 Analytical	802	307246
QSS4 Analytical	58.2	17566
QSS4 Sturm	6450	17566
QSS4 Rolle	11300	17566
QSS4 VAS	5970	17566
QSS5 VAS	1430	3046
QSS6 VAS	583	896
QSS7 VAS	316	378
QSS8 VAS	225	248
QSS9 VAS	157	256

5.5 Chapter conclusions and discussion

We have presented a way to generalize a technique of numerical integration QSS to orders equal to or greater than 5. We have experimentally proved that:

- the number of simulation steps required by such methods is lower than the number required by lower degrees approximations.
- the precision of numerical root finding algorithms is comparable with the analytic one.
- while the computational cost of Sturm, Rolle and VAS is higher than the analytic one, their cost on higher-order QSS methods is compensated by requiring fewer simulation steps than low-order methods.

Conclusions and Future Work

In this work we have surveyed most of the main results of a central problem in symbolic computation, which is the ancient problem of computing the roots of a univariate polynomial. Although classic, this problem is still subject of much research, due to its wide spectrum of applications.

Ancient approaches are still used, and many of them are far from being deprecated. Budan-Fourier's, Sturm's and Vincent's theorem were formulated almost two centuries ago, and all of them are the main ideas behind state-of-the-art approaches in the area.

The problem has been evolving for centuries and, nowadays, it has many subproblems; all of them with active research in course. In this thesis we have focused on the subproblems known as *root bounding* and *root isolation*.

In Chapter 3 we presented a framework which allows understanding the current methods to compute an upper bound for the positive roots of a given polynomial as instances of it. Then we introduced an iterative technique to improve the bounds obtained through the application of any of those methods. This technique showed to produce tighter bounds than existing methods, without introducing significant extra computational effort. This result impacts on the performance of the VAS algorithm, whose performance depends strongly on the quality of the bounds it computes on every execution of its main loop.

In Chapter 4 we surveyed the existent root isolation algorithms, and underlying theorems. We proposed an adaptation of Fourier's theorem, which requires much less computational effort in the case of fewnomials, and an elementary method which shows quite good performance, only enhaced by the VAS method.

Along with these two results, we also exposed two ideas that we find promising, although he have not been able to obtain concrete results from them. One is related to the idea of Sturm, and the other is related to Fourier's theorem.

Chapter 5 shows an application of the presented methods into the area of simulation

of physical systems. The results presented in this chapter showed that the approaches of Chapter 4 permitted to push forward the order of the integration solvers of the family QSS. This fact lead to the inclusion of the methods of Chapter 4 into the software PowerDEVS [22, 2].

Future work

The implementations of the algorithms presented throughout this work were not optimized. Most of them have been implemented in order to work with rationals of arbitrary precision; this is good from the point of view of the precision, but uses to be *very* slow. Thus the adaptation of the presented algorithms to work with usual floating point numbers, producing certified results (for instance: we must be able to ensure that the approximated result is always lower than or equal to the exact result) is one of the paths to be explored in future.

Moreover, it would be of great interest to study the bit-complexity [99] of the introduced algorithms. This problem was not considered in this thesis, for the complexity analysis is a complementary point of view of the algorithms. The bit-complexity model considers, along with the usual complexity parameters as the input and output, the size in bits of the operands at each step of the computation. The cost of each operation becomes thus a function of the sizes in bits of their operands. This model of computation contrasts with the usual Real-RAM model, on which the cost of all operations is unitary. The bit-complexity model approximates better the practical running-time of the algorithms, despite usually incurring in overestimations.

For the application shown in Chapter 5, there are many paths to explore. The application opened the door for a new family of algorithms, and there are many situations in the field of simulations that present particular conditions which could be exploited in the algorithms we are considering. For example, it happens quite often that simulations need to compute the minimum positive roots of two polynomials p and q, which only differ by a constant. We did not explore this particular situation; when considered, it could lead to significant improvements in the performance of the simulators. Another of the paths to explore in future is to understand more deeply the behavior of simulators and to detect particular conditions that could be exploited in order to accelerate the underlying method which we are using as an auxiliary tool.

Appendix A

We sketch in this Appendix the proof of Vincent's theorem given by Alesina and Galuzzi [17, 18]. Let us recall the statement of the theorem of Vincent given in Chapter 2.

Theorem A.1 (Vincent)**.** *Let $f(x)$ be a polynomial of degree n with rational coefficients and without multiple roots. The sequence of successive variable changes*

$$x \leftarrow a_1 + \frac{1}{x}, \quad x \leftarrow a_2 + \frac{1}{x}, \quad x \leftarrow a_3 + \frac{1}{x}, \quad \ldots x \leftarrow a_h + \frac{1}{x}$$

can be seen as one only change of the form

$$x \leftarrow a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots + \cfrac{1}{a_h + \cfrac{1}{x}}}}$$

which is,

$$x \leftarrow \frac{Ax + B}{Cx + D}$$

for some nonnegative integers A, B, C, D. For h sufficiently large, the polynomial

$$(Cx + D)^n f\left(\frac{Ax + B}{Cx + D}\right)$$

has either 0 or 1 sign variations in its list of coefficients. Moreover, in the first case f has no roots in the interval whose endpoints are $\frac{B}{D}$ and $\frac{A}{C}$, while in the second case it has exactly 1 root in it.

Alesina and Galuzzi, following an observation of Vincent, who on his part was following an observation of Lagrange, pointed that we can set $a \leftarrow \frac{B}{D}$, $b \leftarrow \frac{A}{C}$ and making the

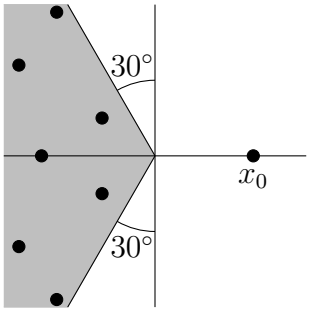


Figure 5.5: The shaded region (i.e.: $S_{\sqrt{3}}$) correspond to the points (α, β) such that $\beta^2 \leq 3\alpha^2$. Lemma A.1 states that if all the roots of a polynomial $p(x)$ lie in the shaded region, except x_0 (which is real and positive), then $p(x)$ has exactly one sign variation in its list of coefficients.

substitution $x \leftarrow \frac{D}{C}x$, we are reduced to studying the variations of the polynomial

$$\phi(x) = (1+x)^n f\left(\frac{a+bx}{1+x}\right)$$

which are the same than in the previous, more complicated, statement of the theorem.

The proof of the Vincent's theorem that we will present will make use of the following Lemma [17, page 247].

Lemma A.1. *If a real polynomial has one positive simple root x_0, and all other (possibly multiple) roots lie in the sector:*

$$S_{\sqrt{3}} = \left\{ x = -\alpha + i\beta \mid \alpha > 0 \text{ and } |\beta| \leq \sqrt{3}|\alpha| \right\}$$

(as shown in figure 5.5) then the sequence of its coefficients has exactly one sign variation.

Proof. Along this proof, we will sometimes call *number of variations* or *number of sign changes* of some polynomial to the number of sign changes (or sign variations) in the list of its coefficients. Let

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \tag{A.1}$$

be a polynomial with one positive root x_0 and all the other roots in $S_{\sqrt{3}}$. We will assume $a_n > 0$, and similar reasoning to the ones exposed in here apply for $a_n < 0$.

In the region $S_{\sqrt{3}}$ there are two *kind* of roots: the real roots, with the form $x = -\alpha$, being α a real positive number; and the *complex* roots, which come in pairs of the form

$x = -\alpha \pm i\beta$, being $\alpha > 0$ and $0 < \beta \leq \sqrt{3}\alpha$. Claim 1: *If $\alpha > 0$ and the polynomial $f(x)$ has exactly one sign variation, then the polynomial $(x - (-\alpha)) \times f(x) = (x + \alpha) \times f(x)$ has exactly one sign variation.* Claim 2: *If $\alpha > 0$ and $0 < \beta \leq \sqrt{3}\alpha$ and the polynomial $f(x)$ has exactly one sign variation, then the polynomial $(x - (-\alpha + i\beta)) \times (x - (-\alpha - i\beta)) \times f(x) = [x^2 + 2\alpha x + (\alpha^2 + \beta^2)]f(x)$ has exactly one sign variation.*

Before proceeding with the proofs of the claims, let us see how the Lemma follows almost trivially from them. We can express $f(x)$ as the product of its roots, in the form:

$$f(x) = a_n \times (x - x_0) \times \overbrace{(x - (-\alpha_0)) \times (x - (-\alpha_1)) \times \cdots \times (x - (-\alpha_k))}^{-\alpha_0, -\alpha_1, \ldots, -\alpha_k \text{ real roots}} \underbrace{\times (x - (-\alpha_{k+1} - i\beta_{k+1})) \times (x - (-\alpha_{k+1} + i\beta_{k+1})) \times \ldots}_{(-\alpha_j - i\beta_j, -\alpha_j + i\beta_j) \text{ pairs of complex roots}} \overbrace{\times (x - (-\alpha_w - i\beta_w)) \times (x - (-\alpha_w + i\beta_w))}$$

We can start with the polynomial $a_n(x - x_0) = a_n x - a_n x_0$, which has exactly one sign variation, and apply iteratively the exposed claims until obtain the stated Lemma, as follows:

$$\begin{aligned}
& a_n(x - x_0) \text{ has 1 sign change} \\
\text{Thus, } & a_n(x - x_0) \times (x - (-\alpha_0)) \text{ has 1 sign change} \\
\text{Thus, } & a_n(x - x_0) \times (x - (-\alpha_0)) \times (x - (-\alpha_1)) \text{ has 1 sign change} \\
& \vdots \\
\text{Thus, } & f(x) = a_n \times (x - x_0) \times (x - (-\alpha_0)) \times (x - (-\alpha_1)) \times \cdots \times (x - (-\alpha_k)) \\
& \times (x - (-\alpha_{k+1} - i\beta_{k+1})) \times (x - (-\alpha_{k+1} + i\beta_{k+1})) \\
& \times \cdots \times (x - (-\alpha_w - i\beta_w)) \times (x - (-\alpha_w + i\beta_w)) \text{ has 1 sign change}
\end{aligned}$$

Thus, in this way the Lemma follows from the stated claims. Let us prove them, in order to complete the proof of the Lemma.

Proof of Claim 1. The degree of $(x + \alpha) \times f(x)$ is $n + 1$. The fact that $f(x)$ has exactly one sign variation, together with the assumption $a_n > 0$, implies that exist i, j such that $i > j$ and • $a_n, a_{n-1}, \ldots a_{i+1} \geq 0$ • $a_i > 0$ • $a_{i-1}, a_{i-2}, \ldots, a_{j+1} = 0$ • $a_j < 0$ • $a_{j-1}, a_{j-2}, \ldots, a_0 \leq 0$, as shown in the following figure (braces annotations refers to

the signs of the coefficients involved):

$$f(x) = \overbrace{a_n x^n + a_{n-1}x^{n-1} + \cdots + a_{i+1}x^{i+1}}^{\geq 0} \underbrace{+a_i x^i}_{>0} \underbrace{+a_{i-1}x^{i-1} + \cdots + a_{j+1}x^{j+1}}_{=0} \underbrace{+a_j x^j}_{<0} \underbrace{+a_{j-1}x^{j-1} + \cdots + a_0}_{\leq 0}$$

Let

$$(x+\alpha) \times f(x) = \sum_{k=0}^{n+1} d_k x^k.$$

It is easy to see that $d_k = a_{k-1} + \alpha a_k$, defining $a_{-1} = a_{n+1} = 0$. Let us analyze the sign of the d_k's. If $i = j+1$, the only unpredictable sign is the sign of d_{j+1}, and whatever it is, the polynomial $f(x)(x+\alpha)$ has one sign variation. If $i \geq j+2$, the signs of all d_k's is determined and $f(x)(x+\alpha)$ has exactly one sign variation.

Proof of Claim 2. The degree of $[x^2 + 2\alpha x + (\alpha^2 + \beta^2)] \times f(x)$ is $n+2$. Consider i and j as in the previous claim. Let

$$[x^2 + 2\alpha x + (\alpha^2 + \beta^2)] \times f(x) = \sum_{k=0}^{n+2} d_k x^k.$$

It is easy to see that $d_k = a_{k-2} + 2\alpha a_{k-1} + (\alpha^2 + \beta^2)a_k$, defining $a_{n+2} = a_{n+1} = a_{-1} = a_{-2} = 0$. Let us analyze the sign of the d_k's. The d_k's with $k \leq j$ are ≤ 0. The sign of d_k's, with $n+2 \geq k \geq i+1$ are ≥ 0. Thus, the only remaining d_k's for which we don't know the sign are $d_{i+1}, d_i, d_{i-1}, \ldots, d_{j+1}$. In the case in which $i \geq j+3$ (i.e.: at least two zero terms between a_i and a_j at the expression of $f(x)$), it is very easy to see that the sequence of d_k's presents exactly one sign variation. The same occurs in the case in which $i = j+2$ (i.e.: one zero term between a_i and a_j), where the only element of the d_k's for which we cannot infer the sign is d_i; and both if it is positive or if it is negative, the sequence of d_k's presents exactly one sign variation.

So the only non-trivial case is when $i = j+1$. In this case, the two d_k's for which we cannot easily know the sign are d_{i+1} and d_i We will show that if $d_{i+1} < 0$, then $d_i \leq 0$

and, consequently, we will have that the sequence of d_k's have exactly one variation.

$$\begin{aligned} d_{i+1} &= \overbrace{(\alpha^2+\beta^2)a_{i+1}}^{\geq 0} + \overbrace{2\alpha a_i}^{>0} + \overbrace{a_{i-1}}^{=a_j<0} \\ d_i &= \underbrace{(\alpha^2+\beta^2)a_i}_{>0} + \underbrace{2\alpha a_{i-1}}_{<0} + \underbrace{a_{i-2}}_{\leq 0} \end{aligned}$$

If $d_{i+1} < 0$, then $2\alpha a_i + a_{i-1} < 0$. Due to the fact that $\beta^2 \leq 3\alpha^2$ and to the fact that $a_{i-2} \leq 0$, we have:

$$d_i \leq (\alpha^2 + 3\alpha^2)a_i + 2\alpha a_{i-1} = 2\alpha(2\alpha a_i + a_{i-1}) \leq 0$$

□

The transformation $T(z) = \frac{z-a}{b-z}$

We have already used the transformation $T(x) = \frac{x-a}{b-x}$ to map the interval (a, b) into the interval $(0, +\infty)$. Let us briefly analyze what happens when we consider the transformation T in all the complex plane. Let $T : \mathbb{C} \to \mathbb{C}$ be defined as $T(z) = \dfrac{z-a}{b-z}$ where $a, b \in \mathbb{R}^+$. Let us highlight two facts that follows trivially.

Fact 1. T takes the circle C that have the segment $\overline{ab}$ as diameter into the imaginary axis, its exterior points into the left half-plane $\mathrm{Re}(z) < 0$, and its interior points into the right half-plane $\mathrm{Re}(z) > 0$ (see figure 5.6).

Fact 2. Let c^+ be the point in the perpendicular bisector of $\overline{ab}$, such that $\widehat{bac^+} = 30°$, and let c^- be the point in the same perpendicular bisector, such that $\widehat{c^-ab} = 30°$. Let C^+ be the circle centered at c^+, which passes through the points a and b, and let C^- the the circle centered at c^-, which passes through a and b, as shown in figure 5.7. The transformation T takes the exterior points of C^+ into the half-plane to the left of the

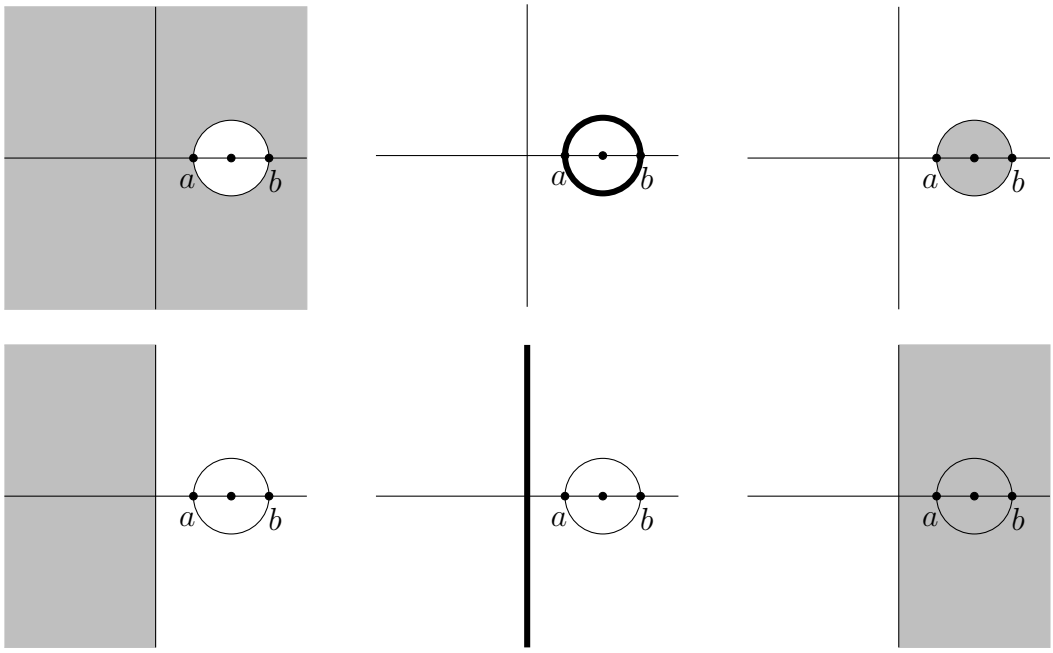


Figure 5.6: The transformation T takes the circle C, which has the segment $\overline{ab}$ as diameter, into the imaginary axis (middle column); its exterior points into the left half-plane (left column); and its interior points into the right half-plane (right column)

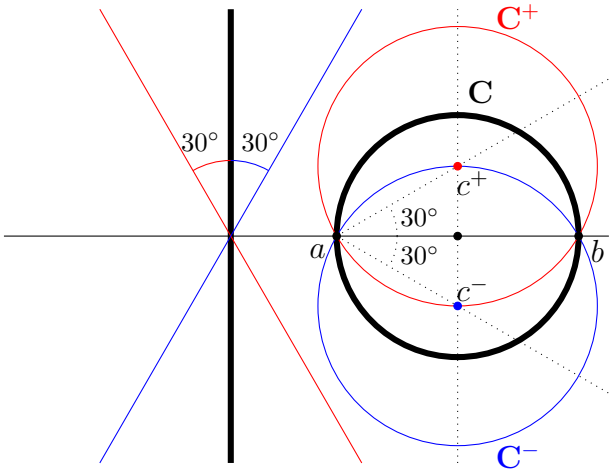


Figure 5.7: The transformation T takes the red circle into the red line, its exterior points into to left side of the red line and its interior points to the right of it. The same occurs with the black and blue circles.

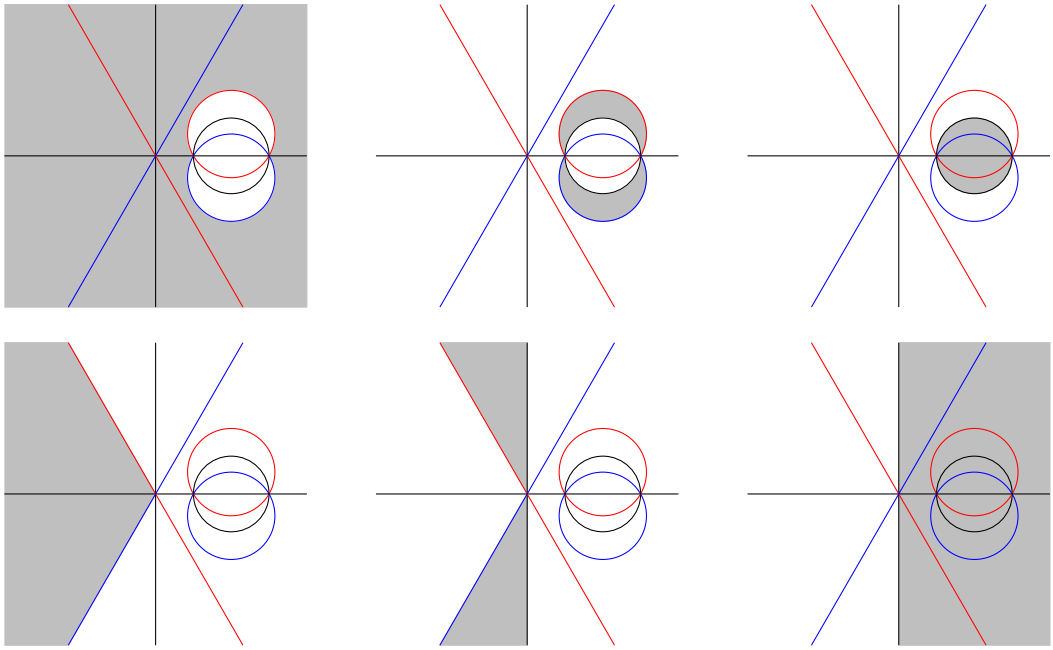


Figure 5.8: T takes the exterior points of $C^+ \cup C^-$ into $S_{\sqrt{3}}$ (left column), $(C^+ \cup C^-) - C$ into $\{z|\mathrm{Re}(z) < 0\} - S_{\sqrt{3}}$ (middle column), and C into $\{z|\mathrm{Re}(z) > 0\}$ (right column).

line L that passes through the origin and forms an angle of 30° with the imaginary axis, the points on C^+ into L, and the points on the interior of C^+ into the half-plane to the right of L. The same happens for C^- and the line which forms an angle of $-30°$ with the imaginary axis, as shown in the Figure 5.7. It easily follows that the sector $S_{\sqrt{3}}$ defined in A.1 is the image of the exterior points of $C^+ \cup C^-$ (see Figure 5.8).

Now that we have pointed out these two simple facts about the transformation $T(z) = \dfrac{z-a}{b-z}$, let us see one more fact that follows easily from them. If it is given a polynomial $f(z)$, without multiple roots, let us pick the points a and b of the previous explanation such that the distance between them is lower than $\sqrt{3} \times \frac{\Delta}{2}$, where Δ is the minimum distance between pairs of different roots of f. By doing this, we are sure that the circle C (id est: the circle whose diameter is $\overline{ab}$) (see Figure 5.7) cannot have two roots; so consequently it can have either one real root or no roots at all (since in the case that it has one complex root, it must has its conjugated; ant the distance between the two of them would be lower than the minimum distance between roots, so that is impossible). Moreover: if the circle C contains a real roots, there are no roots *at* C^+ *and at* C^-,

since we picked the distance between a and b to be smaller than $\sqrt{3} \times \frac{\Delta}{2}$. In fact: the maximum possible distance between a point in the circle C^+ and a point in the segment $\overline{ab}$ is $2 \times \frac{|b-a|}{2 \times \cos 30^\circ} = \frac{2}{\sqrt{3}} \times |b-a| < \frac{2}{\sqrt{3}} \times \sqrt{3} \times \frac{\Delta}{2} = \Delta$. Thus, we have that the following two cases are possible:

- the segment $\overline{ab}$ contains a root of f and consequently there are no roots of f inside $C^+ \cup C^-$.
- there are no roots of f inside the circle C

In the first case, we have that the transformation T applied takes all the roots of f (except the one contained in $\overline{ab}$ into the region $S_{\sqrt{3}}$. In the second case, we have that T takes all the roots the left half-plane; in the first case, due to Lemma A.1, the resulting transformed polynomial have exactly 1 sign variation. In the second case it follows trivially that it has 0 sign variations. The inverse of $T(z) = \dfrac{z-a}{b-z}$ is $S(z) = \dfrac{a+bz}{1+z}$. Thus, the polynomial $(1+z)^n f(\dfrac{a+bz}{1+z})$ have 0 sign variations in the case in which $\overline{ab}$ contains no root, and 1 sign variations in the case in which it contains 1 root.

From this observations, we can conclude Vincent's theorem. Following Lagrange's observation, Vincent's theorem can be stated in the following equivalent way:

Theorem A.2 (Vincent)**.** *Let $f(x)$ be a real polynomial of degree n without multiple roots. It exist $\delta \in \mathbb{R}_+$ so that $\forall a, b \in \mathbb{R}$ with $|b-a| < \delta$, the polynomial*

$$\phi(z) = (1+z)^n f\left(\frac{a+bz}{1+z}\right)$$

has exactly 0 or 1 sign variations. In the first case, f has 0 roots in the interval whose endpoints are a and b. In the second case, it has 1 root in it.

And this is, in fact, the result that we have just proved.

Bibliography

[1] Linear bottleneck assignment problem. `https://en.wikipedia.org/wiki/Linear_bottleneck_assignment_problem`. Retrieved Feb. 16, 2016.

[2] PowerDEVS site. `http://sourceforge.net/projects/powerdevs/`. Retrieved Feb. 16, 2016.

[3] Abbott, J. Quadratic interval refinement for real roots. *ACM Communications in Computer Algebra 48*, 1/2 (2014), 3–12.

[4] Abel, N. H. Démonstration de l'impossibilité de la résolution algébrique des équations générales qui passent le quatrieme degré. *Journal für die reine und angewandte Mathematik 1* (1826), 65–96.

[5] Abel, N. H. *Oeuvres completes*, vol. 1. Grøndahl & søn, 1881.

[6] Akritas, A., Bocharov, A., and Strzebonski, A. Implementation of real root isolation algorithms in mathematica. In *International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (Interval'94)* (1994), pp. 23–27.

[7] Akritas, A., and Strzeboński, A. On the various bisection methods derived from Vincent's theorem. *Serdica Journal of Computing 2*, 1 (2008), 89–104.

[8] Akritas, A., and Vigklas, P. Counting the number of real roots in an interval with Vincent's theorem. *Bull. Math. Soc. Sci. Math. Roumanie 53*, 3 (2010), 201–211.

[9] Akritas, A. G. There is no "Uspensky's method". In *Proceedings of the Fifth ACM Symposium on Symbolic and Algebraic Computation* (New York, NY, USA, 1986), SYMSAC '86, ACM, pp. 88–90.

[10] Akritas, A. G. There is no "Descartes' method". *Computer Algebra in Education* (2008), 19–35.

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

[12] Akritas, A. G., Argyris, A., and Strzebońki, A. W. FLQ, the fastest quadratic complexity bound on the values of positive roots of polynomials. *Serdica Journal of Computing 2*, 2 (2008).

[13] Akritas, A. G., and Strzebońki, A. W. A comparative study of two real root isolation methods. *Nonlinear Analysis: Modelling and control 10*, 4 (2005), 297–304.

[14] Akritas, A. G., Strzebońki, A. W., and Vigklas, P. S. Implementations of a new theorem for computing bounds for positive roots of polynomials. *Computing 78*, 4 (dec 2006), 355–367.

[15] Akritas, A. G., Strzebońki, A. W., and Vigklas, P. S. Improving the performance of the continued fractions method using new bounds of positive roots. *Nonlinear Analysis: Modelling and Control 13*, 3 (2008), 265–279.

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

[17] Alesina, A., and Galuzzi, M. A new proof of Vincent's theorem. *L'Enseignement Mathematique 44* (1998), 219–256.

[18] Alesina, A., and Galuzzi, M. Addendum to the paper a new proof of Vincent's theorem. *L'Enseignement Mathematique 45* (1999), 379–380.

[19] Alesina, A., and Galuzzi, M. Vincent's theorem from a modern point of view. *Rendiconti del circolo matematico di Palermo*, 64 (2000), 179–191.

[20] Alonso García, M. E., and Galligo, A. A root isolation algorithm for sparse univariate polynomials. In *Proceedings of ISSAC '12* (New York, NY, USA, 2012), ACM, pp. 35–42.

[21] Bensimhoun, M. Historical account and ultra-simple proofs of Descartes's rule of signs, de Gua, Fourier, and Budan's rule. *arXiv preprint arXiv:1309.6664* (2013).

[22] Bergero, F., and Kofman, E. PowerDEVS: a tool for hybrid system modeling and real-time simulation. *Simulation 87*, 1-2 (2011), 113–132.

[23] Biagioli, E., Peñaranda, L., and Oliveira, R. I. New method for bounding the roots of a univariate polynomial. In *28 Conference on Graphics, Patterns and Images (SIBGRAPI)* (2015), pp. 35–42. Available at http://urlib.net/8JMKD3MGPBW34M/3JS24KL.

[24] Boulier, F. Systèmes polynomiaux: que signifie "résoudre". Université Lille 1. Available online at http://www.lifl.fr/ boulier/polycopies/resoudre.pdf, 2010.

[25] Budan, F. F. D. *Nouvelle méthode pour la résolution des équations numériques d'un degré quelconque.* Chez Courcier, 1800.

[26] Bugeaud, Y., and Mignotte, M. On the distance between roots of integer polynomials. *Proceedings of the Edinburgh Mathematical Society (Series 2) 47*, 3 (2004), 553–556.

[27] Bugeaud, Y., and Mignotte, M. Polynomial root separation. *International Journal of Number Theory 6*, 3 (2010), 587–602.

[28] Carpaneto, G., and Toth, P. Algorithm for the solution of the bottleneck assignment problem. *Computing 27*, 2 (1981), 179–187.

[29] Cellier, F. E., and Kofman, E. *Continuous System Simulation.* Springer-Verlag, New York, 2006.

[30] Collins, G. E. Continued fraction real root isolation using the Hong root bound. *Journal of Symbolic Computation 72* (2014), 21–54.

[31] Collins, G. E. Krandick's proof of Lagrange's real root bound claim. *Journal of Symbolic Computation 70* (2015), 106–111.

[32] Collins, G. E., and Akritas, A. G. Polynomial real root isolation using Descarte's rule of signs. In *Proceedings of the Third ACM Symposium on Symbolic*

and Algebraic Computation (New York, NY, USA, 1976), SYMSAC '76, ACM, pp. 272–275.

[33] Collins, G. E., and Horowitz, E. The minimum root separation of a polynomial. *Mathematics of Computation 28*, 126 (1974), 589–597.

[34] Collins, G. E., and Loos, R. Polynomial real root isolation by differentiation. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation* (1976), ACM, pp. 15–25.

[35] De Figueiredo, L. H., and Stolfi, J. Affine arithmetic: concepts and applications. *Numerical Algorithms 37*, 1-4 (2004), 147–158.

[36] de Gua de Malves, J.-P. Recherche du nombre des racines réelles ou imaginaires, réelles positives ou réelles negatives, qui peuvent se trouver dans les équations de tous les degrés. *Memoires de l'Académie Royale de Sciences* (1741), 435–494.

[37] Decker, T., and Krandick, W. Parallel real root isolation using the Descartes method. In *High Performance Computing–HiPC'99*. Springer, 1999, pp. 261–268.

[38] Dedieu, J.-P. Estimations for the separation number of a polynomial system. *Journal of Symbolic Computation 24*, 6 (1997), 683–693.

[39] Derigs, D.-M. U., and Zimmermann, U. An augmenting path method for solving linear bottleneck assignment problems. *Computing 19*, 4 (1978), 285–295.

[40] Descartes, R., et al. *Lettres de Mr. Descartes*. Chez Charles Angot, 1725.

[41] Edelman, A., and Kostlan, E. How many zeros of a random polynomial are real? *Bulletin of the American Mathematical Society 32*, 1 (1995), 1–37.

[42] Eigenwillig, A. *Real root isolation for exact and approximate polynomials using Descartes' rule of signs*. PhD thesis, Universität des Saarlandes Saarbrücken, Saarland, 2008.

[43] Eigenwillig, A., Sharma, V., and Yap, C. K. Almost tight recursion tree bounds for the Descartes method. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation* (2006), ACM, pp. 71–78.

[44] Emiris, I., and Tsigaridas, E. A note on the complexity of univariate root isolation. Research Report RR-6043, INRIA, 2006. Available at https://hal.inria.fr/inria-00116985.

[45] Emiris, I. Z., Galligo, A., and Tsigaridas, E. P. Random polynomials and expected complexity of bisection methods for real solving. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation* (2010), ACM, pp. 235–242.

[46] Fourier, J.-B. J. Seconde partie de la note relative aux limites des racines. *Bulletin des Sciences par la Société Philomatique de Paris* (1820), 181–187.

[47] Fourier, J.-B. J. Usage do théoréme de Descartes dans le recherche des limites de racines. *Bulletin des Sciences par la Société Philomatique de Paris* (1820), 156–165.

[48] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (June 2007).

[49] Fujii, M., and Kubo, F. Buzano's inequality and bounds for roots of algebraic equations. *Proceedings of the American Mathematical Society 117*, 2 (1993), 359–361.

[50] Galligo, A. Improved budan-fourier count for root finding. Preprint. Available at https://hal.inria.fr/hal-00653762, Dec 2011.

[51] Geddes, K., Czapor, S., and Labahn, G. *Algorithms for Computer Algebra*. Springer US, 1992.

[52] Heindel, L. *Algorithms for exact polynomial root calculation*. PhD thesis, University of Wisconsin, 1970.

[53] Heindel, L. E. Integer arithmetic algorithms for polynomial real zero determination. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation* (1971), ACM, pp. 415–426.

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

[55] Ibragimov, I. A., and Maslova, N. On the expected number of real zeros of random polynomials I. Coefficients with zero means. *Theory of Probability & Its Applications 16*, 2 (1971), 228–248.

[56] Johnson, J. R., Krandick, W., Lynch, K., Richardson, D. G., and Ruslanov, A. D. High-performance implementations of the Descartes method. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation* (2006), ACM, pp. 154–161.

[57] Kerber, M., and Sagraloff, M. Root refinement for real polynomials. *arXiv preprint arXiv:1104.1362* (2011).

[58] Khovanskiĭ, A. *Fewnomials*. Translations of mathematical monographs. American Mathematical Society, 1991.

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

[60] Knuth, D. E. *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming*. Addison Wesley Longman Publishing Co., Inc., 1981.

[61] Knuth, D. E. *Fundamental algorithms*, 3rd ed., vol. 1 of *The Art of Computer Programming*. Addison Wesley Longman Publishing Co., Inc., 1997.

[62] Knuth, D. E. *Sorting and Searching*, 2nd ed., vol. 3 of *The Art of Computer Programming*. Addison Wesley Longman Publishing Co., Inc., 1998.

[63] Kobel, A., and Sagraloff, M. Fast approximate polynomial multipoint evaluation and applications. *arXiv preprint arXiv:1304.8069* (2013).

[64] Kofman, E. A Second Order Approximation for DEVS Simulation of Continuous Systems. *Simulation: Transactions of the Society for Modeling and Simulation International 78*, 2 (2002), 76–89.

[65] Kofman, E. Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing 25*, 5 (2004), 1771–1797.

[66] Kofman, E. A third order discrete event simulation method for continuous system simulation. *Latin American Applied Research 36*, 2 (2006), 101–108.

[67] Kofman, E. Relative error control in quantization based integration. *Latin American Applied Research 39*, 3 (2009), 231–238.

[68] Kofman, E., and Junco, S. Quantized state systems. a DEVS approach for continuous system simulation. *Transactions of SCS 18*, 3 (2001), 123–132.

[69] Krandick, W. A data structure for approximation. Tech. rep., Department of Mathematics and Statistics, the University of Edinburgh, 1996.

[70] Krandick, W., and Mehlhorn, K. New bounds for the Descartes method. *Journal of Symbolic Computation 41*, 1 (2006), 49–66.

[71] Lagrange, J.-L. Réflexions sur la résolution algébrique des équations. *Memoires de l'Academie de Berlin* (1770), 205–421.

[72] Lagrange, J.-L. *De la résolution des équations numériques de tous les degrés*. Chez Duprat, 1798.

[73] Lagrange, J.-L. *Traité de la résolution des équations numériques de tous les degrés: avec des notes sur plusieurs points de la théorie des équations algébriques*. Bachelier, 1826.

[74] Laubenbacher, R., McGrath, G., and Pengelley, D. Lagrange and the solution of numerical equations. *Historia Mathematica 28*, 3 (2001), 220–231.

[75] Lenstra Jr, H. W. On the factorization of lacunary polynomials. *Number theory in progress 1* (1999), 277–291.

[76] Mahler, K., et al. An inequality for the discriminant of a polynomial. *The Michigan Mathematical Journal 11*, 3 (1964), 257–262.

[77] Mehlhorn, K., and Sagraloff, M. A deterministic algorithm for isolating real roots of a real polynomial. *Journal of Symbolic Computation 46*, 1 (2011), 70–90.

[78] Mehlhorn, K., Sagraloff, M., and Wang, P. From approximate factorization to root isolation with application to cylindrical algebraic decomposition. *Journal of Symbolic Computation 66* (2015), 34–69.

[79] Mignotte, M. On the distance between the roots of a polynomial. *Applicable Algebra in Engineering, Communication and Computing 6*, 6 (1995), 327–332.

[80] Ostrowski, A. Note on Vincent's theorem. *Annals of Mathematics* (1950), 702–707.

[81] Peñaranda, L. *Géométrie algorithmique non linéaire et courbes algébriques planaires*. PhD thesis, Université Nancy 2, 2010.

[82] Prasolov, V. V. *Polynomials*. Algorithms and Computation in Mathematics. Springer Berlin Heidelberg, 2009.

[83] Reynolds, G. S. Investigation of different methods of fast polynomial evaluation. Master's thesis, The University of Edinburgh, Edinburgh, Scotland, UK, 2010.

[84] Rojas, J. M., and Ye, Y. On solving fewnomials over intervals in fewnomial time. *arXiv preprint math/0106225* (2001).

[85] Rouillier, F., and Zimmermann, P. Efficient isolation of polynomial's real roots. *Journal of Computational and Applied Mathematics 162* (2004), 33–50.

[86] Rump, S. M. Polynomial minimum root separation. *Mathematics of Computation 33*, 145 (1979), 327–336.

[87] Sagraloff, M. When Newton meets Descartes: A simple and fast algorithm to isolate the real roots of a polynomial. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation* (2012), ACM, pp. 297–304.

[88] Sagraloff, M. A near-optimal algorithm for computing real roots of sparse polynomials. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation* (New York, NY, USA, 2014), ISSAC '14, ACM, pp. 359–366.

[89] Ştefănescu, D. New bounds for positive roots of polynomials. *Journal of Universal Computer Science 11*, 12 (dec 2005), 2125–2131.

[90] Sturm, C. F. Analyse d'un mémoire sur la résolution des équations numériques. *Bulletin des Sciences mathematiques de Férussac XI* (1829), 419–425.

[91] Sturm, C. F. Mémoire sur la résolution des équations numériques. *Mémoires divers présentés par des savants étrangers VI* (1835), 273–318.

[92] Uspensky, J. *Theory of equations.* McGraw-Hill Book Co., 1948.

[93] Vigklas, P. S. *Upper bounds on the value of the positive roots of polynomials.* PhD thesis, University of Thessaly, 2010.

[94] Vincent, A. J. H. Note sur la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées 1* (1836), 341–372.

[95] Vincent, A. J. H. Addition à una précédente note relative à la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées 3* (1838), 235–243.

[96] Von Zur Gathen, J., and Gerhard, J. Fast algorithms for Taylor shifts and certain difference equations. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation* (1997), ACM, pp. 40–47.

[97] Wang, P. S., and Trager, B. M. New algorithms for polynomial square-free decomposition over the integers. *SIAM Journal on Computing 8*, 3 (1979), 300–305.

[98] Yap, C. K. *Fundamental Problems of Algorithmic Algebra.* Oxford University Press, dec 1999.

[99] Yap, C. K., and Dubé, T. The exact computation paradigm. In *Computing in Euclidean Geometry*, D.-Z. Du and F. Hwang, Eds., 2nd ed. World Scientific Press, 1995, pp. 452–486.

[100] Yun, D. Y. On square-free decomposition algorithms. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation* (1976), ACM, pp. 26–35.

[101] Zeigler, B., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation - Second Edition.* Academic Press, 2000.