Computing 24, 299—313 (1980)

Computing

The Fastest Exact Algorithms for the Isolation of the Real Roots of a Polynomial Equation

A. G. Akritas, Athens

Received March 9, 1979; revised July 22, 1979, and November 22, 1979

Abstract — Zusammenfassung

The Fastest Exact Algorithms for the Isolation of the Real Roots of a Polynomial Equation. This paper discusses a set of algorithms which, given a polynomial equation with integer coefficients and without any multiple roots, uses exact (infinite precision) integer arithmetic and the Vincent-Uspensky-Akritas theorem to compute intervals containing the real roots of the polynomial equation. Theoretical computing time bounds are developed for these algorithms which are proven to be the fastest existing; this fact is also verified by the empirical results which are included in this article.

Die schnellsten exakten Algorithmen zur Isolierung der reellen Nullstellen von Polynomen. Es werden einige Algorithmen diskutiert, die unter Verwendung exakter ganzzahliger Arithmetik und des Vincent-Uspensky-Akritas-Theorems für ein gegebenes Polynom mit ganzzahligen Koeffizienten und ohne mehrfache Wurzeln Intervalle berechnen, die die reellen Nullstellen des Polynoms enthalten. Für diese Algorithmen werden theoretische Rechenzeitschranken entwickelt, und es wird bewiesen und durch empirische Resultate belegt, daß diese Algorithmen die schnellsten unter den bisher existierenden sind.

1. Introduction

Isolation of the real roots of a polynomial equation is the process of finding real, disjoint intervals such that each contains exactly one real root, and every real root is contained in some interval. Budan and Fourier [5] presented two different — but closely related — theorems which enable us to determine the maximum possible number of real roots a polynomial has in a given interval.

Based on Fourier's proposition Sturm (1829) presented an improved theorem whose application yields the exact number of the real roots that a polynomial without multiple zeros has within an interval. Thus the isolation problem was solved and since 1830 Sturm's method is the only one widely known and used. Budan's theorem, however, is underestimated in the existing literature. It's important consequences were only recently realized by the author of this paper and appear in his Ph. D. thesis [2]. It was shown that Budan's theorem forms the basis of the Vincent-Uspensky-Akritas theorem [4] which, in turn, is the foundation of our own method for the isolation of the real roots of a polynomial equation.

0010-485X/80/0024/0299/$ 03.00

In this paper we present in detail the main algorithms needed for the implementation of our own real root isolation method. (However, only short, informal descriptions are given for the various auxilliary algorithms used.) As we will see, using our method, the real roots of a polynomial P, of degree n, with integer coefficients and without any multiple zeros, will be isolated in time

$$0(n^5 L(|P|_\infty)^3), \tag{1}$$

where $L(|P|_\infty)$ is the length, in bits, of the maximum coefficient in absolute value. This bound makes our method the fastest existing (compare for example Sturm's method which is $0(n^{13} L(|P|_\infty)^3)$ [9]). Moreover, tables with empirical results are presented in the last section and verify the superiority of our method over all others existing ([2] pp. 4—8).

In the rest of this section we introduce certain complexity notions as well as the computational model within which our algorithms will be implemented. We begin with the following:

Definition 1.1: Let f, g be real-valued functions defined on an arbitrary common domain D. Then, (i) we say that f is *dominated* by g, and write $f=0(g)$, in case there exists a positive real number c_1 and x_1 in D, such that $|f(x)| \leqq c_1 \cdot |g(x)|$ for all x in D, $x > x_1$, (ii) we say that f *dominates* g, and write $f=\Omega(g)$, in case there exists a positive real number c_2 and x_2 in D, such that $|f(x)| \geqq c_2 \cdot |g(x)|$ for all x in D, $x > x_2$, and (iii) we say that f and g are *codominant*, and write $f \sim g$, if it so happens that $f=0(g)$ and $f=\Omega(g)$.

Some properties of dominance and codominance will be used subsequently, but they follow easily from the definition.

We now introduce the appropriate date structures that are useful in designing efficiently our algorithms. Had we used the data structures inherent in any one of the existing algebraic manipulation systems, we would be system dependent and a reader unfamiliar with the peculiarities of a specific system would be unable to understand our algorithms. Moreover, our presentation will make their implementation in another system or language a trivial task.

We begin by recursively defining a *list* over an arbitrary set S to be a finite sequence $(a_1, a_2, \ldots, a_n)$, $n \geqq 0$, such that each a_i is either an element of S or a list over S; the empty list is represented by 0. When we write $A=(a_1, a_2, \ldots, a_n)$ we interpret it in two ways: (i) A is considered to be a pointer to the beginning of the list, and (ii) A represents the entire list, so that when we write $A \oplus x$, where $\oplus$ is any one of the binary operators on scalars, we mean that the operation $\oplus$ has been performed between each element of A and the scalar x.

Given the list $A=(a_1, a_2, \ldots, a_n)$, where now A is a pointer to the beginning of the list, one can define various operations on it; of interest to us are the following: length $(A)=n$; first $(A)=a_1$; last $(A)=a_n$; tail (A) resulting in $A=(a_2, a_3, \ldots, a_n)$; invert (A) resulting in $A=(a_n, \ldots, a_1)$; prefix $\bar{a}_1, \ldots, \bar{a}_k$ *to* A, $k \geqq 1$, resulting in $A=(\bar{a}_1, \ldots, \bar{a}_k, a_1, \ldots, a_n)$; advance $\bar{a}_1, \ldots, \bar{a}_k$ *in* A, $k \leqq n$, resulting in $\bar{a}_i$ pointing to a_i, $1 \leqq i \leqq k$ and $A=(a_{k+1}, \ldots, a_n)$. If $A=0$, the empty list, we define prefix a *to* A to mean $A=(a)$. In what follows, list and list elements of a list will be

denoted with capital letters of the alphabet, whereas elements of S will be denoted with small letters.

The algorithms described in this article involve operations on integers, rational numbers, and polynomials.

We distinguish two types of integers — those that are represented by lists and those that are not. An integer of the first type will be represented as $A=(a_0, a_1, \ldots, a_n)$, $n \geqq 1$, where these a_i's are the coefficients of β^i's in the expression $A=\sum_{i=0}^{n} \alpha_i \beta^i$ and are all positive or negative, according to whether $A>0$ or $A<0$, respectively; $\beta=2^\mu-1$ is the largest value stored in a computer word. Each a_i is stored in a separate computer word and, except for a_n, takes up μ bits. Sign $(A)=\pm 1$ depending on whether $a_n>0$ or $a_n<0$.

A rational number R is considered to be the list $R=(N, D)$, where N and D are the numerator and denominator, respectively, and are both integers represented by lists.

An univariate polynomial $P(x)$ of degree n (and the equation $P(x)=0$) will be represented by the ordered list $P=(C_r, e_r, C_{r-1}, e_{r-1}, \ldots, C_1, e_1)$ $r \geqq 1$, where each integer coefficient C_i is $\neq 0$ and is represented by the list $C_i=(c_{i1}, c_{i2}, \ldots, c_{im_i})$, $m_i \geqq 1$; the exponents e_i are in decreasing order $e_r>e_{r-1}>\ldots>e_1$. We define sign $(P)=$ sign (C_r) and degree $(P)=e_r$. The empty list represents the polynomial $P \equiv 0$.

The language used to describe our algorithms is basically that of conventional mathematics, with the exception of the replacement operator. We use simple and compound statements, where each simple executable statement is separated from another by ";". A compound statement is a sequence of simple statements; it is enclosed in parentheses and may be preceded by the word *do*. Comments follow the step number and are enclosed in square brackets. We also use the following special statements whose meaning is obvious (see also [1] p. 35): *if* condition *then* statement *else* statement; *while* condition *do* statement; *repeat* statement *until* condition; *for* $i=1, 2, \ldots, q$ *do* statement.

Definition 1.2: By the *β-length* of an integer k we mean the number of β-digits in its representation, and we write $L_\beta(k)$. If $\lceil x \rceil$ is the *ceiling function*, the least integer greater than or equal to x, and $\lfloor x \rfloor$ is the *floor function*, the greatest integer less than or equal to x, then

$$L_\beta(k)=\begin{cases} 1 & , \quad k=0 \\ \lceil \log_\beta(|k|+1) \rceil = \lfloor \log_\beta |k| \rfloor + 1, & k \neq 0. \end{cases}$$

In what follows the subscript β will be omitted since for any other base γ we have $L_\beta \sim L_\gamma$ (if we think of L_β and L_γ as functions defined on the set of integers).

Definition 1.3: Let A be any algorithm and S the set of all valid inputs to A. t_A is the *computing time function* associated with A and defined on S. The integer $t_A(x)$, for

x in S, is the number of basic operations performed by the algorithm A when presented with the input x.

Basic operations consist of such things as single precision additions and multiplications, replacements, unconditional transfers, and subroutine calls. The reader can easily see that for two integers A, B which are represented as lists we have the following: the time to compute $A+B$ is $\sim\max(L(A), L(B))\sim L(A)+L(B)$; the time to compute $A \cdot B$ is $\sim L(A)\, L(B)$.

Computing time bounds for operations on polynomials are characteristically given as functions of the degrees and the norms of the polynomials.

Definition 1.4: Let $P(x)=\sum_{i=0}^{n} c_i x^i=0$ be a univariate polynomial equation with integer coefficients (if the coefficients are rational we first turn them into integers). The *max-norm* (or sub-infinity norm) of $P(x)$ is $|P|_\infty = \max_{0\leqq i\leqq n}(|c_i|)$, whereas the *sum-norm* (or sub-one norm) is $|P|_1=\sum_{i=0}^{n}|c_i|$.

By the definition it follows that $|P|_\infty \leqq |P|_1 \leqq (n+1)|P|_\infty$, where n is the degree of P; therefore, $L(|P|_\infty)\sim L(|P|_1)$.

Definition 1.5: Let $P(x)$ be an univariate polynomial with integer coefficients. Assume that the degree of $P(x)$ is $n\geqq 1$ and that it has k, $k\leqq n$, distinct roots $\alpha_1, \ldots, \alpha_k$. If $k\geqq 2$, we define the *minimum root separation* of P, $\Delta>0$, by

$$\Delta = \min_{1\leqq i<j\leqq k} |\alpha_i-\alpha_j|;$$

if $k=1$, then $\Delta=\infty$.

Theorem 1.1 (*Mahler*, [11]): *If $P(x)$ is an univariate polynomial with integer coefficients and of degree $n\geqq 2$, then*

$$\Delta \geqq \sqrt{3}\cdot n^{-\frac{(n+2)}{2}} \cdot |P|_1^{-(n-1)}. \tag{2}$$

2. Mathematical Background

In this section we present the propositions which are necessary for a complete understanding of our discussion; the proof of the main theorem is quite lengthy, and, since it appears in the literature, it will be omitted. Included also is a recursive description of our method for the isolation of the real roots of a polynomial equation. We begin with the main theorem.

Theorem 2.1 (*Vincent-Uspensky-Akritas*): *Let $P(x)=0$ be a polynomial equation of degree $n>1$, with rational coefficients and without multiple roots, and let $\Delta>0$ be the smallest distance between any two of its roots. Let m be the smallest index such that*

$$F_{m-1}\frac{\Delta}{2}>1 \quad \text{and} \quad F_{m-1}F_m\Delta>1+\frac{1}{\varepsilon_n}, \tag{3}$$

where F_k is the k-th member of the Fibonacci sequence

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

and

$$\varepsilon_n = \left(1 + \frac{1}{n}\right)^{\frac{1}{n-1}} - 1.$$

Then the transformation

$$x = a_1 + \frac{1}{a_2} + \cdot \cdot \cdot + \frac{1}{a_m} + \frac{1}{\xi}, \tag{4}$$

(which is equivalent to the series of successive transformations of the form $x = a_i + \frac{1}{\xi}$, $i = 1, 2, \ldots, m$) presented in the form of a continued fraction with arbitrary, positive, integral elements $a_1, a_2, \ldots, a_m$, transforms the equation $P(x) = 0$ into the equation $\tilde{P}(\xi) = 0$, which has not more than one sign variation in the sequence of its coefficients.

The proof of this theorem can be found in [2]; its original form (that is, without specifying the quantity m) is due to Vincent alone and appeared in 1836 [15], [7]. The calculation of the quantities $a_1, a_2, \ldots, a_m$ — for the transformations of the form (4) which lead to an equation with exactly one sign variation — constitutes the polynomial real root isolation procedure. Two methods actually result, Vincent's and Akritas', corresponding to the two different ways in which the computation of the a_i's may be performed; the difference between these two methods can be thought of as being analogous to the difference between the integrals of Riemann and Lebesgue.

Vincent's method basically consists of computing a particular a_i by a series of unit incrementations, i.e. $a_i \leftarrow a_i + 1$, which corresponds to the substitution $x \leftarrow x + 1$. This brute force approach results in a method with an exponential behavior and hence of very little practical importance.

Akritas' method, on the contrary, is an aesthetically pleasing interpretation of Theorem 2.1; basically it consists of immediately computing a particular a_i as the lower bound b on the values of the positive roots of a polynomial, i.e. $a_i \leftarrow b$, which corresponds to the substitution $x \leftarrow x + b$ (performed on the particular polynomial under consideration). Here it is assumed that $b = \lfloor \alpha_s \rfloor$, where α_s is the smallest positive root. The computation of this bound b will be discussed in the next section.

Corollary 2.1: *Under the assumptions of Theorem* 2.1 *we have*

$$m = 0\left(n\, L(|\, P\,|_\infty) + n\, L(n)\right).$$

Proof: It follows immediately if we use either one of the inequalities (3) along with (2) and the expression $F_k = \frac{\phi^k}{\sqrt{5}}$, where $\phi = 1,618 \ldots .$ //

Remark 2.1: In most cases of interest n is small so that we can safely conclude

$$m=0\left(n\,L\left(|\,P\,|_\infty\right)\right). \tag{5}$$

The fact that the positive integral quantities $a_1, a_2, \ldots, a_m$ of Theorem 2.1 are arbitrary is of great significance and is discussed elsewhere [5]. For our purposes consider an infinite binary tree in which the root corresponds to an original polynomial equation $P(x)=0$, and each node corresponds to a transformed equation resulting from the original after a series of successive transformations of the form $x=a_i+\frac{1}{y}$. The path from each node to the right descendent corresponds to the substitution $x\leftarrow 1+x$, whereas, the path to the left descendent corresponds to the substitution $x\leftarrow\frac{1}{1+x}$. All the nodes belonging to a specific path, finite or infinite, will be considered as members of disjoint sets, which can be of three types. A set of type V_0, V_1 or V_n contains nodes corresponding to polynomials with zero, one or more than one sign variations, respectively. Sets of type V_0 or V_1 are called *terminal sets.* In the case of sets belonging to the same path, a set X is said to *precede* a set Y if and only if for all x in X and all y in Y pathlength $(x)<$ pathlength (y). In a terminal set, the node having the shortest path from the root of the tree will be called a *terminal node.* We can now state the following:

Theorem 2.2: *Let* $P(x)=c_n x^n+c_{n-1}x^{n-1}+\ldots+c_1 x+c_0=0$ *be a polynomial equation of degree* $n>1$, *with rational coefficients and without any multiple roots, which corresponds to the root of the binary tree. Suppose, moreover, that the transformation*

$$x=a_1+\cfrac{1}{a_2}+\cfrac{1}{a_3}+\cdots+\cfrac{1}{a_l}+\cfrac{1}{y} \tag{6}$$

with positive, integer elements $a_1, a_2, \ldots, a_l$, $1\leqq l\leqq m$ *(where* m *is defined in Theorem* 2.1) *transforms* $P(x)=0$ *into a new equation corresponding to a* V_0 *or* V_1*-terminal node. Then for all* k, $1\leqq k\leqq l$ *we have* $a_k=0\,(|\,P\,|_\infty)$.

Proof: As we know (6) is equivalent to a series of l successive transformations of the form $x=a_i+\frac{1}{y}$ each of which is equivalent to a general translation of length a_i followed by an inversion. In this proof $P_0(x)\equiv P(x)=0$, while $P_k(x)=0$ denotes the equation obtained from $P_0(x)=0$ after the substitutions $x\leftarrow a_1+\frac{1}{x}, \ldots,$ $x\leftarrow a_k+\frac{1}{x}$.

In order to prove that $a_1=0\,(|\,P\,|_\infty)$ let $\alpha_1^0, \alpha_2^0, \ldots, \alpha_{P_0}^0$ be the roots of $P_0(x)=0$ with positive real part (p. R. p.) in increasing order of the values of their real parts. (This convention of listing the roots in increasing order is adopted throughout this proof.) According to (6), $P_0(x)=0$ is first transformed by the substitution $x\leftarrow a_1+x$ into the equation $\tilde{P}_1(x)=P_0(a_1+x)=0$ (which in turn yields

$P_1(x)=\tilde{P}_1\left(\frac{1}{x}\right)=0$); as a result of this translation, the real part of the roots of $P_0(x)=0$ with p. R. p. decreases by a_1 units. Since by hypothesis $P_l(x)=0$ corresponds to a V_0 or V_1-terminal node, we conclude that a_1 cannot be greater than $\lfloor \text{R. p. } (\alpha^0_{p_0})\rfloor+1$; that is, provided $l>1$

$$a_1=\lfloor \text{R. p. } (\alpha^0_{j_1})\rfloor, \tag{7}$$

for some j_1, $1\leqq j_1\leqq p_0$, whereas, if $l=1$ and $P_l(x)=0$ corresponds to a V_0-terminal node, then clearly $a_1=\lfloor \text{R. p. } (\alpha^0_{p_0})\rfloor+1$. (Obviously, if $l=1$ and $P_l(x)=0$ corresponds to a V_1-terminal node then $a_1\leqq\lfloor\alpha^0_{p_0}\rfloor$; that is $\alpha^0_{p_0}$ is necessarily real.) Moreover, ([10] p. 398) all the roots of $P_0(x)=0$ are bounded above by $b=2\max\limits_{1\leqq k\leqq n}\left|\frac{c_{n-k}}{c_n}\right|^{\frac{1}{k}}$, which in turn is $\leqq 2\,|P|_\infty$. So, for all the roots $\alpha^0_{j_1}$, $1\leqq j_1\leqq p_0$ we have

$$|\alpha^0_{j_1}|\leqq 2\,|P|_\infty. \tag{8}$$

Combining (7) and (8) we clearly see that $a_1=0(|P|_\infty)$.

The proof of this theorem will be completed if we prove the general case; that is, if we show that for any k, $1<k\leqq l$ we have $a_k=0(|P|_\infty)$. This will be accomplished with the help of the following observations: consider $\tilde{P}_1(x)=P_0(a_1+x)=0$; then due to their (deterministic) relation we have that $|\tilde{P}_1|_\infty\sim|P_0|_\infty$. Moreover, we also have $|P_1|_\infty\sim|P_0|_\infty$ since the coefficients of $P_1(x)=\tilde{P}_1\left(\frac{1}{x}\right)=0$ are obtained by simply inverting the order of the coefficients of $\tilde{P}_1(x)=0$, an operation that does not affect the maximum coefficient; i.e. $|\tilde{P}_1|_\infty=|P_1|_\infty$. Proceeding in the same way we have $|P_1|_\infty\sim|P_2|_\infty$, ... and finally we obtain

$$|P_{k-1}|_\infty\sim|P_0|_\infty. \tag{9}$$

If we now assume that $\alpha^{k-1}_1,\alpha^{k-1}_2,\ldots,\alpha^{k-1}_{p_{k-1}}$ are the roots of $P_{k-1}(x)=0$ with p. R. p., then going through similar steps as for the case $k=1$, we easily see that $a_k=\lfloor \text{R. p. } (\alpha^{k-1}_{j_k})\rfloor$, for some j_k, $1\leqq j_k\leqq p_{k-1}$, where for each root we have $|\alpha^{k-1}_{j_k}|\leqq 2\,|P_{k-1}|_\infty$; however, the last two relations combined with (9) yield $a_k=0(|P|_\infty)$, which completes the proof. //

From the preceeding discussion we can conclude that Vincent's method will behave exponentially when the quantities a_i are sufficiently big. Our method, on the contrary, is independent of the values of the a_i's and a recursive description of it follows. (As presented, our method will isolate only the positive roots of an equation; however, if we subsequently replace x by $-x$ in the original equation, the negative roots become positive, and hence they too can be isolated with our method.)

Akritas' Method: Let

$$P(x)=0 \tag{10}$$

be a polynomial equation with v sign variations in the sequence of its coefficients and without any multiple roots.

Case (i): $v=0$ *or* $v=1$. From the Cardano-Descartes rule of signs we know that $v=0$ implies that (10) has no positive roots, whereas, $v=1$ indicates that (10) has exactly one positive root, in which case $(0, \infty)$ is its isolating interval; in either case, no transformation of (10) is necessary and the method terminates.

Case (ii): $v>1$. In this case (10) has to be further investigated. We first compute the lower bound b on the values of the positive roots and then we obtain the translated equation $P_b(x)=P(b+x)=0$, which also has v sign variations provided $P(b)\neq 0$ (if $P(b)=0$, we have found an integer root of the original equation and v is decreased). $P_b(x)=0$ is now transformed by the substitutions $x\leftarrow 1+x$ and $x\leftarrow\frac{1}{1+x}$, and our method is applied again twice, once with $P_b\left(\frac{1}{1+x}\right)=0$ in place of (10) and once with $P_b(1+x)=0$.

The details of our method are made clear if the interested reader applies it on some examples ([14] pp. 129—137). It will also be observed that, when working with pencil and paper the formation of the isolating intervals is easily performed, a fact which is no longer true if we use a computer. Therefore, in order to keep track of the performed transformations, we associate with each node of the tree, besides the polynomial equation, the function which corresponds to the transformation. The root of the tree now is associated with the original polynomial equation and the function corresponding to the identity transformation $M(y)=y$. At each node, the substitutions performed on the corresponding polynomial equation are also performed on the corresponding function.

3. Computer Implementation of Our Own Real Root Isolation Method

In this section we present the algorithms which implement very efficiently the main operations of our method. As mentioned in the Introduction, only short, informal descriptions are given for the various auxilliary algorithms. These are subsequently incorporated into one procedure which implements our root isolation method. We begin with the following (n refers always to the degree of a polynomial):

(i) *Counting the Number of Sign Variations*

For this operation we use algorithm $v\leftarrow PCSVAR(P)$, where P is a polynomial and v the number of sign variations in the sequence of its coefficients. It is easily seen that $t_{PCSVAR}(P)=0(n\,L(|P|_\infty))$.

(ii) *Computation of the Lower Bound b*

The computation of the lower bound b on the values of the positive roots of $P(x)=0$ is equivalent to the computation — with the help of Cauchy's rule ([12] p. 50) — of the upper bound $1/b$ on the values of the positive roots of $P\left(\frac{1}{x}\right)=0$. For the latter computation we use algorithm $B\leftarrow PPRRB(P)$, where

P is a polynomial and B a rational number $B=(B_1, B_2)$; the B_i's are integers represented by lists, one of them being the list (1). We have shown [6] that $t_{PPRRB}(P)=0(n^2 L(|P|_\infty))$. (In what follows we assume that the lower bound $b=\lfloor\alpha_s\rfloor$, where α_s is the smallest positive root. This assumption is made for theoretical purposes only; in practice, in general, we have $b<\lfloor\alpha_s\rfloor$ and several applications of *PPRRB* are needed in order to compute $\lfloor\alpha_s\rfloor$.)

(*iii*) *Polynomial Translation*

For our method we need algorithms for the substitutions $x\leftarrow b+x$, $b\geqq 1$, and $x\leftarrow\frac{1}{1+x}$. The latter, however, is equivalent to the pair of transformations $x\leftarrow\frac{1}{x}$ and $x\leftarrow 1+x$, so that we actually need algorithms for the transformations $x\leftarrow b+x$, $b\geqq 1$, and $x\leftarrow\frac{1}{x}$. In order to perform the substitution $x\leftarrow b+x$, $b\geqq 1$, we employ the Ruffini-Horner method and we consider two cases: if $b>1$, we use algorithm $P'\leftarrow PTRNSL(P, B)$, where P is a polynomial, B the integer >1, and $P'(x)=P(B+x)$; if $b=1$ we use algorithm $P'\leftarrow PTRAN1(P)$. We have shown [8] that $t_{PTRNSL}(P, B)=0(n^3 L(B)^2+n^2 L(B) L(|P|_\infty))$. For the implementation of $x\leftarrow\frac{1}{x}$ we use algorithm $P'\leftarrow PINVCF(P)$, where P and P' are polynomials such that $P'(x)=P\left(\frac{1}{x}\right)=0$. It is easily seen that $t_{PINVCF}(P)\sim n$.

Obviously, by applying the above mentioned algorithms separately on the numerator and denominator of the function associated with each node of the binary tree, we can also transform the function by the previous substitutions. However, we can be more efficient by taking advantage of the fact that the numerator and denominator are both first degree (at most) polynomials. Therefore, we use algorithms $P'\leftarrow TUP(P, B)$ and $P'\leftarrow TUP1(P, F)$. With the first algorithm we obtain — from the polynomial $P(x)=c_1 x+c_0$ — the transformed polynomial $P'(x)=P(B+x)=c_1 x+(c_0+c_1 B)$. With the second algorithm we have two options: if $F=0$ then, for the same polynomial P as above, we obtain $P'(x)=P(1+x)=c_1 x+(c_0+c_1)$, whereas, if $F=1$ then $P'(x)=c_0 x+(c_0+c_1)$ which corresponds to the equation $P'(x)=P\left(\frac{1}{1+x}\right)=0$.

(*iv*) *Formation of the Isolating Interval*

The isolating interval for a real root is obtained from a function of the form

$$M(x)=\frac{N(x)}{D(x)}=\frac{a_1 x+a_0}{b_1 x+b_0}$$

where a_0, b_0, a_1, b_1 are nonnegative integers such that $a_1+b_1>0$ and $a_0 b_0>0$, if we replace x first by 0 and then by ∞. In fact, we can easily see that the isolating interval will be $\left(\frac{a_1}{b_1}, \frac{a_0}{b_0}\right)$ or $\left(\frac{a_0}{b_0}, \frac{a_1}{b_1}\right)$; ∞ is represented by -1.

Algorithm $I \leftarrow FII(N, D, F)$ is used, where if $F=1$ then I is an one-point interval, whereas, if $F=0$ we obtain one of the above mentioned intervals. For the computing time of FII we have to use results mentioned in [13] (p. 38 and p. 43).

(v) *Isolation of the Real Roots*

In what follows we incorporate all the preceding algorithms into one procedure which implements our own method for the isolation of the real roots. We first present the algorithm which isolates just the positive roots. This algorithm consists of a loop, in which a list of quadruplets of the form (P, N, D, v) is examined; P is an univariate polynomial with integer coefficients and without multiple roots, which presents v sign variations, $v>1$, whereas, N and D are the numerator and denominator, respectively, of the associated function. Initially, the list contains a single quadruplet, which corresponds to the input data; P is the given polynomial (whose roots we want to isolate) with $v>1$ sign variations, while $N=x$ and $D=1$. Obviously, if the input data is such that $v=0$ or $v=1$, we do not have to enter the loop, since from the Cardano-Descartes rule of signs, we can immediately determine the number of positive roots of P. The first quadruplet (P, N, D, v) is removed from the list, and the lower bound b on the values of the positive roots of P is computed; $b=\lfloor \text{R. p. } (\alpha_s) \rfloor$, where α_s is the root of P with the smallest positive real part. (P, N, D, v) is transformed by the substitution $x \leftarrow b+x$ into the new quadruplet (P_b, N_b, D_b, v_b), and then it is discarded. $P_b(0)=0$ implies that b is a root of $P(x)=0$, in which case we obtain an isolating interval and save it for output. (P_b, N_b, D_b, v_b) is first transformed by the substitution $x \leftarrow 1+x$ into $(P_{b1}, N_{b1}, D_{b1}, v_{b1})$, where $0 \leqq v_{b1} < v_b$, and as before, a test is made to see whether $P_{b1}(0)=0$. We distinguish three cases, according to whether $v_{b1}=0$, $v_{b1}=1$ or $1<v_{b1}<v_b$: (a) $v_{b1}=0$. This implies that P_{b1} has no positive roots, and hence the new quadruplet is discarded. (b) $v_{b1}=1$. Here P_{b1} has exactly one positive root; its isolating interval is first obtained from N_{b1} and D_{b1} (and is saved for output), and then the new quadruplet is discarded. (c) $1<v_{b1}<v_b$. In this case P_{b1} has to be further investigated, so $(P_{b1}, N_{b1}, D_{b1}, v_{b1})$ is inserted at the head of the list.

In all three cases, (P_b, N_b, D_b, v_b) is transformed once more by the substitution $x \leftarrow \dfrac{1}{1+x}$, and then it is discarded; the resulting quadruplet $(P_{b2}, N_{b2}, D_{b2}, v_{b2})$ is subject to the same treatment as the one obtained by the substitution $x \leftarrow 1+x$, and the whole process is repeated. The loop terminates when the list of quadruplets becomes empty.

Formally, this procedure is described by algorithm

$R \leftarrow AMPRRI(P, N, D)$: *A*kritas' *M*ethod, *P*ositive *R*eal *R*oot *I*solation.

Specifications: The inputs are P, N and D. $P \neq 0$ is an univariate polynomial with integer coefficients and without multiple roots, represented by the list $P=(C_r, e_r, C_{r-1}, e_{r-1}, \ldots, C_1, e_1)$, $r \geqq 1$, where each coefficient C_i is represented by a list; moreover, zero is not a root $(P(0) \neq 0)$. N is the special polynomial x, while D is the constant polynomial 1; they are the numerator and denominator,

respectively, of the function associated with the root of the binary tree, and are both represented in the same way as P. The output is R, a list of the isolating intervals — not necessarily ordered — of the positive roots of P.

Description:

1. [Initialize.] $R \leftarrow 0$; $Q \leftarrow 0$; $F \leftarrow 0$; $v \leftarrow PCSVAR(P)$.
2. [No sign variations.] *if* $v=0$ *then return.*
3. [One sign variation.] *if* $v=1$ *then do* ($I \leftarrow 0$; prefix 0, ∞ *to* I; prefix I *to* R; *return*).
4. [Continue initialization.] $P_0 \leftarrow P$; $N_0 \leftarrow N$; $D_0 \leftarrow D$; *go to* 7.
5. [Test for completion.] *if* $Q=0$ *then return.*
6. [Obtain first quadruplet.] $Q_0 \leftarrow$ first (Q); tail (Q); advance P_0, N_0, D_0, v *in* Q_0.
7. [Compute positive lower root bound b.] $\bar{P} \leftarrow PINVCF(P_0)$; $B' \leftarrow PPRRB(\bar{P})$; advance A, B *in* B'; *if* $B<A$ *then go to* 9.
8. [$x \leftarrow b+x$.] *if* $B>A$ *then do* ($P'_0 \leftarrow PTRNSL(P_0, B)$; $N'_0 \leftarrow TUP(N_0, B)$; $D'_0 \leftarrow TUP(D_0, B)$) *else do* ($P'_0 \leftarrow PTRAN1(P_0)$; $N'_0 \leftarrow TUP1(N_0, 0)$; $D'_0 \leftarrow TUP1(D_0, 0)$); $P_0 \leftarrow P'_0$; $N_0 \leftarrow N'_0$; $D_0 \leftarrow D'_0$; $s \leftarrow P_0(0)$; *if* $s \neq 0$ *then go to* 9; $P_0 \leftarrow P_0/x$; $I \leftarrow FII(N_0, D_0, 1)$; prefix I *to* R.
9. [$x \leftarrow 1+x$.] $\tilde{v} \leftarrow v$; $\hat{P} \leftarrow PTRAN1(P_0)$; $\hat{N} \leftarrow TUP1(N_0, 0)$; $\hat{D} \leftarrow TUP1(D_0, 0)$; $s \leftarrow \hat{P}(0)$; *if* $s \neq 0$ *then go to* 11 *else do* ($\hat{P} \leftarrow \hat{P}/x$; $I \leftarrow FII(\hat{N}, \hat{D}, 1)$; prefix I *to* R; *go to* 11).
10. $\left[x \leftarrow \dfrac{1}{1+x}\,?\right]$ *if* $\tilde{v}=v$ *then do* ($F \leftarrow 0$; *go to* 5); $\bar{P} \leftarrow PINVCF(P_0)$; *if* sign $(\bar{P})=-1$ *then* $\bar{P} \leftarrow -\bar{P}$; $\hat{P} \leftarrow PTRAN1(\bar{P})$; $\hat{N} \leftarrow TUP1(N_0, 1)$; $\hat{D} \leftarrow TUP1(D_0, 1)$.
11. [Change flag.] $F \leftarrow F+1$; $v \leftarrow PCSVAR(\hat{P})$.
12. [No sign variation.] *if* $v=0$ *then go to* 15.
13. [One sign variation.] *if* $v=1$ *then do* ($I \leftarrow FII(\hat{N}, \hat{D}, 0)$; prefix I *to* R; *go to* 15).
14. [More than one sign variation.] $\hat{Q} \leftarrow 0$; prefix $\hat{P}, \hat{N}, \hat{D}, v$ *to* $\hat{Q}$; prefix $\hat{Q}$ *to* Q.
15. [Go back to loop.] *if* $F=1$ *then go to* 10 *else do* ($F \leftarrow 0$; *go to* 5).

Theorem 3.1: *Let* $P \neq 0$ *be an univariate polunomial of degree* $n \geqq 0$, *with integer coefficients and without multiple roots. If* $n=0$ or $n=1$, *then* $t_{AMPRRI}(P, x, 1) \sim L(|P|_\infty)$, *so assume* $n>1$ *and let* p *be the number of roots of* P *with positive real part. Then* $t_{AMPRRI}(P, x, 1) = 0(p\, n^4\, L(|P|_\infty)^3)$.

Proof: Clearly, the computing time of this algorithm is dominated by the computing time of step 8, where the substitution $x \leftarrow b+x$ is performed (recall that we have made the assumption $b = \lfloor R.p.(\alpha_s) \rfloor$). Consequently, in order to prove

the theorem, we have to compute, first, an upper bound on the number of executions of step 8, and then a time bound for a single, typical execution of it.

The upper bound on the number of executions of step 8 is determined with the help of Remark 2.1 and Theorem 2.2. From this theorem we conclude that for each root with positive real part, a transformation of the form (6) has to be performed, which, in the binary tree, corresponds to a V_0 or V_1-terminal node. As we recall, it the nature of our method to perform a transformation of the form (6) by executing l general translations $x \leftarrow a_i + x$, since each a_i is computed immediately as the lower bound b on the values of the positive roots of $P_{i-1}(x)=0$ (P_k was defined in the proof of Theorem 2.2). Therefore, for each root with p. R. p., step 8 is executed at most m times, where m is defined in (5).

In order to estimate the computing time bound for a single, typical execution of step 8, it will suffice to do so for the substitution $x \leftarrow a_{k+1} + x$ applied on the polynomial P_k, where $a_{k+1} \leqq c \mid P \mid_\infty$, for some constant c. From [8] we know that $\tilde{P}_{k+1}(x) = P_k(a_{k+1} + x)$, $a_{k+1} > 1$, is obtained from $P_k(x)$ in time

$$0\left(n^3 L(a_{k+1})^2 + n^2 L(a_{k+1}) L(\mid P_k \mid_\infty)\right) \tag{11}$$

and that, if $v = n+1$,

$$\mid P_{k+1} \mid_\infty < 2^v a_{k+1}^v \mid P_k \mid_\infty \leqq c\, 2^{(k+1)v} \mid P \mid_\infty^{(k+1)v+1}, \tag{12}$$

where the right inequality in (12) is obtained by using the result of Theorem 2.2, that is, all the a_i's are $\leqq c \mid P \mid_\infty$. Due also to (12), we can easily see that (11) reduces to

$$0\left(n^3 L(\mid P \mid_\infty)^2\right), \tag{13}$$

which is the time bound for a single, typical execution of step 8. Combining, therefore, (5) and (13) we see that the total computing time for each root with p. R. p. is $0\left(n^4 L(\mid P \mid_\infty)^3\right)$. The theorem now follows from the fact that there are p such roots. //

In order to isolate the negative roots we have to substitute x by $-x$ in the original equation. Algorithm $P' \leftarrow PSNV(P)$ is used, where $P'(x) = P(-x) = 0$; it is easily seen that $t_{PSNV}(P) = 0\left(n L(\mid P \mid_\infty)\right)$.

We now incorporate the two previously described algorithms into one procedure, which isolates all the real roots of a polynomial, implementing thus our own method.

$R \leftarrow AKRITAS(P)$: *AKRITAS*' method.

Specifications: The input P is an univariate polynomial with integer coefficients and without multiple roots, represented by the list $P = (C_r, e_r, C_{r-1}, e_{r-1}, \ldots, C_1, e_1)$, $r \geqq 1$, where each coefficient C_i is represented by a list. R is a list of the form $R = (R_1, Z, R_2)$; $Z = 0$ implies that $P(0) = 0$, whereas, otherwise $P(0) \neq 0$. R_1 and R_2 are the lists of the isolating intervals of the "negative" and positive roots, respectively.

Description:

1. [Initialize.] $R \leftarrow 0$; *if* $P=0$ *then return*; $P_0 \leftarrow P$; $N_0 \leftarrow 0$; $D_0 \leftarrow 0$; $F \leftarrow 0$.
2. [Form the polynomials N_0 and D_0.] prefix $1, 1$ *to* N_0; prefix $1, 0$ *to* D_0.
3. [Is 0 a root?] $s \leftarrow P_0(0)$; *if* $s=0$ *then do* $(P_0 \leftarrow P_0/x; F \leftarrow 1)$.
4. [Isolate the positive roots.] $R_2 \leftarrow AMPRRI(P_0, N_0, D_0)$; prefix R_2 *to* R; *if* $F=1$ *then* prefix 0 *to* R.
5. [Isolate the negative roots.] $P'_0 \leftarrow PSNV(P_0)$; *if* $P'_0=P_0$ *then do* $(R_1 \leftarrow R_2$; *go to* 6) *else* $R_1 \leftarrow AMPRRI(P'_0, N_0, D_0)$.
6. [Finish up.] prefix R_1 *to* R; *return*.

Theorem 3.2: *Let* $P \neq 0$ *be an univariate polynomial of degree* $n \geqq 0$, *with integer coefficients and without multiple roots. If* $n=0$ *or* $n=1$, *then* $t_{AKRITAS}(P) \sim L(|P|_\infty)$, *whereas, if* $n>1$, *then* $t_{AKRITAS}(P)=0(n^5 L(|P|_\infty)^3)$.

Proof: The proof follows immediately from the fact that *AMPRRI* was used twice, once for the roots with positive and once for the roots with negative real part, the sum of which is n. //

This concludes the presentation of our method. In summary we can state that the major assets of this method are [3]:

(i) it is the only method with polynomial computing time bound, which isolates the real roots of an equation by continued fraction approximation (instead of by bisection), and

(ii) it has the best theoretical computing time bound achieved thus far.

The empirical results, which are presented in the next section confirm the fact that the theoretically predicted excellent behavior is commensurate in practice.

4. Empirical Results and Conclusions

In this section we present several tables showing the observed computing times for certain classes of polynomials for the methods of Sturm and Akritas. We compare these two methods because we think that only these can be considered classical; we have shown [5] that they are based on two very old and related theorems. All times are in seconds and were obtained by using the SAC-1 computer algebra system on the IBM S/370 Model 165 computer located at the Triangle Universities Computation Center, where a subroutine is available which reads the computer clock. (For a survey of computer algebra systems see *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation* ed. S. R. Petrick, March 1971, available from ACM.) Some of the polynomials used in these tests are the same as those used by other real root isolation methods ([2] pp. 4—8), so that the reader can easily verify the superiority of our method by a simple comparison of the ratios of the times of the various methods to the corresponding times obtained with Sturm's method.

Table 1. *Chebyshev polynomials*

Degree	Sturm	Akritas
5	.63	.19
10	3.80	1.53
15	9.76	4.73
20	22.84	16.25
25	45.48	33.33

It is a known fact that Sturm's method is particularly fast in the case of Chebyshev polynomials; however, from the above table we observe that this is not the case. This should be attributed to the fact that our method does not have to isolate all the roots of an equation, whenever its positive and negative roots appear as symmetrical pairs with respect to the origin — and this is the case with Chebyshev polynomials This indicates that the implementation of Sturm's method [9] could be modified.

Table 2. *Polynomials with randomly generated coefficients*

Degree	Sturm	Akritas
5	2.05	.26
10	33.28	.46
15	156.40	.94
20	524.42	2.36

All the coefficients of the polynomials in Table 2 were nonzero, each ten decimal digits long and randomly generated. In this case we observe that Sturm's method is completely "out of the race".

Table 3. *Polynomials with randomly generated roots*

Degree	Sturm	Akritas
5	.73	.71
10	22.50	23.22
15	151.42	96.35
20	>600	288.49

The randomly generated roots for Table 3 were all in the interval $(0, 10^3)$, i.e. they were all positive. Notice that the roots were uniformly distributed and, hence, none of the methods were favored. Each polynomial was obtained by forming the product of the linear factors $x-\alpha_j, j=1, 2, \ldots, n$ and $n=5, 10, 15, 20$.

Acknowledgements

It is a pleasure to thank Professor Stylianos D. Danielopoulos, currently at the University of Ioannina, for all the help and encouragement during the preparation of my Ph. D. thesis. Appreciation is also extended to the referee for his constructive criticisms and comments which improved this paper.

References

[1] Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The design and analysis of computer algorithms. Reading, Mass.: Addison-Wesley 1976.
[2] Akritas, A. G.: Vincent's theorem in algebraic manipulation. Ph. D. thesis, Operations Research Program, North Carolina State University, Raleigh, N. C., 1978.
[3] Akritas, A. G.: A new method for polynomial real root isolation. Proc. of the 16th Annual Southeast Regional ACM Conference, Atlanta, Georgia, 39—43 (1978). This paper received the First Prize in the student paper competition.
[4] Akritas, A. G.: A correction on a theorem by Uspensky. Bulletin of the Greek Mathematical Society *19*, 278—285 (1978).
[5] Akritas, A. G.: The two different versions of the Budan-Fourier theorem and their consequences, in: 5th volume of lectures given at the General Mathematical Seminar of the University of Patras, 127—146 (1979). (In Greek.)
[6] Akritas, A. G.: Exact algorithms for the implementation of Cauchy's rule. (Submitted for publication.)
[7] Akritas, A. G., Danielopoulos, S. D.: On the forgotten theorem of Mr. Vincent. Historia Mathematica *5*, 427—435 (1978).
[8] Akritas, A. G., Danielopoulos, S. D.: On the complexity of algorithms for the translation of polynomials. Computing *24*, 51—60 (1980).
[9] Heindel, L. E.: Integer arithmetic algorithms for polynomial real zero determination. Journal of the ACM *18*, 533—548 (1971).
[10] Knuth, D. E.: The art of computer programming, Vol. II: Seminumerical algorithms. Reading, Mass.: Addison-Wesley 1969.
[11] Mahler, K.: An inequality for the discriminant of a polynomial. Michigan Mathematical Journal *11*, 257—262 (1964).
[12] Obreschkoff, N.: Verteilung und Berechnung der Nullstellen reeller Polynome. Berlin: VEB Deutscher Verlag der Wissenschaften 1963.
[13] Rubald, C. M.: Algorithms for polynomials over a real algebraic number field. Ph. D. thesis, Computer Science Department, University of Wisconsin, Madison, 1974.
[14] Uspensky, J. V.: Theory of equations. New York: McGraw-Hill 1948.
[15] Vincent, A. J. H.: Sur la résolution des équations numériques. Journal de Mathématiques Pures et Appliquées *1*, 341—372 (1836).

Dr. A. G. Akritas
Unit of Applied Mathematics II
University of Athens
Athens 621
Greece