Exact Algorithms for the Implementation of Cauchy's Rule

ALKIVIADIS G. AKRITAS

University of Kansas, Department of Computer Science, Lawrence, Kansas 66045, USA

Cauchy's little known rule for computing a lower (or upper) bound on the values of the positive roots of a polynomial equation has proven to be of great importance; namely it constitutes an indispensable and crucial part of the fastest method existing for the isolation of the real roots of an equation, a method which was recently developed by the author of this article. In this paper efficient, exact (infinite precision) algorithms, along with their computing time analysis, are presented for the implementation of this important rule.

KEY WORDS: Analysis of (exact) algorithms, polynomial real root isolation, root bounds, Cauchy's (extended) rule, Vincent's theorem.

C.R. CATEGORIES: 5.11, 3.15

1. INTRODUCTION

Recently, in Uspensky's *Theory of Equations* [7], Vincent's forgotten theorem of 1836 was discovered by the author of this paper and it formed the subject of his Ph.D. thesis [2], [5], [9]. An extended version of this important theorem is the following:

THEOREM (*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*

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

where F_k is the kth member of the fibonacci sequence

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

and

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

Then the transformation

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_m + \cfrac{1}{\xi}}}} \tag{1}$$

(which is equivalent to the series of successive transformations of the form $x = a_i + 1/\xi$, $i = 1, 2, \ldots, m$*) 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* $\bar{P}(\xi) = 0$*, which has not more than one sign variation in the sequence of its coefficients.*

The above theorem (whose proof can be found in [4]) can be used in order to isolate the real roots of a polynomial equation. The calculation of the quantities $a_1 a_2, \ldots, a_m$—for the transformations of the form (1) which lead to an equation with exactly one sign variation constitutes the polynomial real root isolation procedure. Two methods actually result, Vincent's and ours, 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; that is, $a_i \leftarrow a_i + 1$, 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. Examples of this approach can be found in Vincent's paper [9] and in Uspensky's book [7].

Our method, on the contrary, is an aesthetically pleasing interpretation of the Vincent–Uspensky–Akritas theorem. 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; that is, $a_i \leftarrow b$, which corresponds to the substitution $x \leftarrow x + b$ performed on the particular polynomial under consideration. In this discussion it is assumed that $b = \lfloor \alpha_s \rfloor$, the greatest integer $\leqq \alpha_s$, where α_s is the smallest positive root. (More details on these two methods can be found elsewhere [2], [3].)

From the preceeding paragraph we see that our method crucially depends on the ability to compute the lower bound b on the values of the positive roots of a polynomial equation. (Observe that this is equivalent to the computation of the upper bound on the values of the positive roots of an equation, because if $\tilde{\alpha} < 1/b$, where $\tilde{\alpha}$ is any positive root of $P(1/x)=0$, then $b < \alpha$ for any positive root α of $P(x)=0$). As far as we have been able to determine, the only efficient rule existing in the literature for the computation of the lower bound b on the values of the positive roots of an equation is that of Cauchy; this rule is little known and—to our knowledge—it can be found only in Obreschkoff's book ([6], pp. 50–51). Note that in our method we have made the assumption that $b = \lfloor \alpha_s \rfloor$; however, the computed bound b will be $< \lfloor \alpha_s \rfloor$ so that in general more than one application of Cauchy's rule is necessary in order to compute $\lfloor \alpha_s \rfloor$.

In the next section we present Cauchy's rule along with its extension, which provides an upper bound on the absolute values of the roots of $P(x)=0$. A discussion is also included as to how these two rules should be best implemented.

In the third and last section we first introduce the appropriate data structures for our algorithms. Subsequently, we present the algorithms, along with their computing time analysis, for the implementation of Cauchy's rule. As the title of the paper indicates, with the help of these algorithms the computations are performed exactly, i.e. without any round-off errors, since we use infinite precision integer arithmetic.

2. MATHEMATICAL BACKGROUND

In this section Cauchy's rule along with its extension is stated and proved. Both rules are stated in the way they are encountered in the literature; that is, the computed number is an upper bound. Subsequently, a discussion follows on how these two rules are best implemented.

Cauchy's Rule Let $P(x) = x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0 = 0$ be an integral, monic polynomial equation of positive degree n, with $c_{n-k} < 0$ for at least one k, $1 \leqq k \leqq n$, and let λ be the number of its negative coefficients. Then

$$b = \max_{\substack{1 \leqq k \leqq n \\ c_{n-k} < 0}} |\lambda c_{n-k}|^{1/k}$$

is an upper bound on the values of the positive roots of $P(x)=0$.

Proof From the way b is defined we conclude that

$$b^k \geqq \lambda |c_{n-k}|$$

for each k such that $c_{n-k}<0$; for these k's the last inequality is also written as

$$b^n \geqq \lambda |c_{n-k}| b^{n-k}.$$

Summing over all the appropriate k's we obtain

$$\lambda b^n \geqq \lambda \sum_{\substack{k=1 \\ c_{n-k}<0}}^{n} |c_{n-k}| b^{n-k},$$

or

$$b^n \geqq \sum_{\substack{k=1 \\ c_{n-k}<0}}^{n} |c_{n-k}| b^{n-k}.$$

From the last inequality we conclude that, if we substitute b for x in $P(x)=0$, the first term, i.e., b^n, will be greater than or equal to the sum of the absolute values of all the negative coefficients. Therefore, $P(x)>0$ for all $x>b$. □

Remark Actually, the above proof is valid even if the coefficients of $P(x)$ are real. However, since we are interested in doing exact computations we take them to be integers.

As we mentioned before, Cauchy's rule is not widely known, and to the best of our knowledge, it appears only in Obreschkoff's book [6]. This rule can be extended so that the computed number is an upper bound on the absolute values of all the roots of $P(x)=0$.

Extended Cauchy's Rule Let $P(x)=x^n+c_{n-1}x^{n-1}+\ldots+c_1x+c_0=0$ be an integral, monic polynomial equation of positive degree n, with $c_{n-k}\neq 0$ for at least one k, $1\leqq k\leqq n$. Then if

$$B=2 \max_{1\leqq k\leqq n} |c_{n-k}|^{1/k}$$

and $P(\alpha)=0$, it follows that $|\alpha|<B$.

Proof If $P(0)=0$, then obviously $0<B$, so assume $\alpha\neq 0$ and $P(\alpha)=0$. Then clearly

$$a^n=\sum_{k=0}^{n-1}(-c_k)\alpha^k$$

from which it follows that

$$|\alpha|^n \leqq \sum_{k=0}^{n-1} |c_k| |\alpha|^k;$$

dividing through by $|\alpha|^n$ we obtain

$$1 \leqq \sum_{k=0}^{n-1} |c_k| |\alpha|^{k-n} = \sum_{k=1}^{n} |c_{n-k}| |\alpha|^{-k}$$

$$= \sum_{k=1}^{n} \left(\frac{|c_{n-k}|^{1/k}}{|\alpha|} \right)^k \leqq \sum_{k=1}^{n} \left(\frac{B}{2|\alpha|} \right)^k$$

$$= \frac{\frac{B}{2|\alpha|} - \left(\frac{B}{2|\alpha|} \right)^{n+1}}{1 - \frac{B}{2|\alpha|}} < \frac{\frac{B}{2|\alpha|}}{1 - \frac{B}{2|\alpha|}}.$$

In other words, we have

$$1 - \frac{B}{2|\alpha|} < \frac{B}{2|\alpha|}$$

from which it follows that

$$1 < \frac{B}{|\alpha|} \quad \text{or} \quad |\alpha| < B. \quad \square$$

For both rules mentioned above, when the polynomial is not monic, then obviously each c_{n-k} is divided by c_n.

Now it might be thought that the above mentioned rules require a great amount of computation, since it seems that the calculation of kth roots is needed. However, this is not so as we will immediately see. Moreover, van der Sluis proved a theorem ([8], Theorem 2.7) from which we conclude that (the extended) Cauchy's Rule is not only nearly optimal, but also a good approximation from above can be easily obtained. This is true because instead of computing each kth root we compute the smallest integer k' such that

$$\left| \frac{c_{n-k}}{c_n} \right|^{1/k} \leqq 2^{k'} \left(\text{or} \left| \frac{c_{n-k}}{c_n} \right|^{1/k} \leqq 2^{k'} \right)$$

and then we set b (or B) $= 2^{\hat{k}+1}$, where $\hat{k}$ is the maximum of the k''s.

The computation of each k' is as follows:
Let

$$\frac{b}{c}=\left|\lambda\frac{c_{n-k}}{c_n}\right|\left(\text{or}\ \frac{b}{c}=\left|\frac{c_{n-k}}{c_n}\right|\right)$$

be a quotient, for some k, $1\leqq k\leqq n$, and suppose that

$$2^i\leqq b<2^{i+1}\quad\text{and}\quad 2^j\leqq c<2^{j+1}.$$

Then clearly

$$2^{i-j-1}<\frac{b}{c}<2^{i-j+1}. \tag{2}$$

If we set $p=i-j-1$ and $p+2=i-j+1$, then (2) becomes

$$2^p<\frac{b}{c}<2^{p+2}.$$

Taking the kth root of the last expression yields

$$2^{p/k}<\left(\frac{b}{c}\right)^{1/k}<2^{(p+2)/k} \tag{3}$$

If $p=q\cdot k+r$, $0\leqq r<k$, then clearly

$$2^q\leqq 2^{p/k};$$

moreover, $p+2=q\cdot k+r+2$ and

$$2^{(p+2)/k}=2^q\cdot 2^{(r+2)/k}\leqq 2^{q+2}, \tag{4}$$

since $(r+2)/k\leqq 2$, given $r\leqq k-1$. Combining (3) and (4) results in $k'=q+2$. We obtain a smaller value for k' if $r\leqq k-2$. In this case $(r+2)/k\leqq 1$ and

$$2^{(p+2)/k}\leqq 2^{q+1}$$

so that $k'=q+1$.

From the previous discussion we conclude that the main operations in (the extended) Cauchy's rule are:

i) the computation of $\lfloor\log_2|A|\rfloor$, the greatest integer $\leqq\log_2|A|$, and $\lceil\log_2|A|\rceil$, the smallest integer $\geqq\log_2|A|$, for any integer $A\neq 0$,

ii) the computation of 2^k for a nonnegative integer k, and
iii) the computation of $[A/2^n]$ for any integer A and n positive or negative, where $[x]=\lfloor x \rfloor$ if $x \geqq 0$ or $[x]=\lceil x \rceil$ if $x<0$.

3. COMPUTER IMPLEMENTATION OF CAUCHY'S RULE

In this section we first introduce the appropriate data structures that are useful in designing efficiently our algorithms, and then we present, informally, the auxilliary algorithms for the implementation of Cauchy's rule. A detailed algorithm for the latter follows along with its computing time analysis.

We begin by resursively 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 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, lists 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 $A=\sum_{i=0}^{n} a_i \beta^i$ 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 $P=(C_r, e_r, C_{r-1}, e_{r-1}, \ldots, C_1, e_1)$ $r \geq 1$ where each integer coefficient C_i is $\neq 0$ and is represented by the list $C_i = (c_{i1}, c_{i2}, \ldots, c_{im_i})$, $m_i \geq 1$; the exponents e_i are in decreasing order $e_r > e_{r-1} > \ldots > e_1$. We define $\text{sign}(P)=\text{sign}(C_r)$ and $\text{degree}(P)=e_r$. 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 preceeded 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 3.1 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

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

In what follows the subscript β will be omitted since for any other base γ $L_\beta \sim L_\gamma$ (if we think of L_β and L_γ as functions defined on the set of integers; $\sim$ denotes co-dominance [2]).

DEFINITION 3.2 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 $0(\max(L(A), L(B)))$ whereas the time to compute $A \cdot B$ is $0(L(A)L(B))$.

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

DEFINITION 3.3 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

$$|P|_\infty = \max_{0 \leqq i \leqq n} (|c_i|).$$

We now present sort, informal descriptions for the various auxilliary algorithms, which perform the operations (i), (ii), and (iii) mentioned at the end of the last section.

i) Computation of the floor and ceiling functions of the logarithm.

For this operation we use algorithm IFCLOG(A, m, n). The input is A, a nonzero integer represented by the list $A=(a_1, a_2, \ldots, a_l)$, $l \geqq 1$; the outputs are $m=\lfloor \log_2 |A| \rfloor$ and $n=\lceil \log_2 |A| \rceil$. It is easily seen that $t_{\text{IFCLOG}}(A, m, n) = 0(L(A))$.

ii) Computation of 2^k

Here we consider two different algorithms. The computation of 2^k for a nonnegative integer k is performed with the help of algorithm $A \leftarrow$ IP2(k). The output is $A=2^k$, which is represented by the list $A=(a_1, a_2, \ldots, a_n)$ $n \geqq 1$. Obviously, $t_{\text{IP2}}(k)=0(k+1)$. Since it is sometimes desirable to obtain 2^k as a rational number (and, thus, include the case when $k<0$) we make use of the algorithm $R \leftarrow$ RNP2(k). The output is the rational number $R = 2^k$; in the case where $k \geqq 0$, R is represented by the list $R=(A, B)$, where A and B are both integers represented by the lists $A=(a_1, a_2, \ldots, a_n)$, $n \geqq 1$ and $B=(1)$. If $k<0$, then obviously $R=(B, A)$. Clearly, $t_{\text{RNP2}}(k)=0(|k|+1)$.

iii) Computation of $[A/2^n]$.

Algorithm $B \leftarrow$ ITRUNC(A, n) is used in this case. The inputs are the integers A and n. A is represented by the list $A=(a_1, a_2, \ldots, a_l)$, $l \geqq 1$, whereas n is a simple integer, which can be positive, negative or equal to zero. The output is the integer B, which is $[A/2^n]$, the integer part of the division in case $n>0$, and $B=A \cdot 2^n$ in case $n<0$. In both cases B is represented by the list $B=(b_1, b_2, \ldots, b_m)$, $m \geqq 1$. It can be shown that for $A \neq 0$ we have: $t_{\text{ITRUNC}}(A, n)=0(L(A))$ if $n>0$, and $t_{\text{ITRUNC}}(A, n)=0(L(A)+|n|)$, if $n<0$.

Using the previously defined algorithms we now implement Cauchy's rule.

Algorithm 3.1 (Cauchy's Rule). $B \leftarrow$CAUCHY(P)

Specifications. The input $P \neq 0$ is an univariate polynomial 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 integer coefficient C_i is represented by the list $C_i=(c_{i1}, c_{i2}, \ldots, c_{im_i})$, $m_i \geqq 1$. The output B is a positive upper root bound for P; B is a binary rational number of the form $B=(B_1, B_2)$, where the B_i's are integers represented by lists (one of them being the list (1)). If $P(x)=\sum_{i=0}^{n} c_i x^i$, $c_n \neq 0$, then B is the smallest power of 2 such that

$$\left|\lambda \frac{c_{n-k}}{c_n}\right|^{1/k} \leqq B,$$

for $1 \leqq k \leqq n$, and $c_{n-k}<0$. If $c_{n-k}=0$ for all the appropriate k's, then $B=1$.

Description

1. [Sign $(P)=1$?] $\hat{P} \leftarrow P$; *if* sign $(\hat{P})=-1$, *then* $\hat{P} \leftarrow -\hat{P}$.
2. [Initialize for λ.] $\lambda \leftarrow 0$; $P' \leftarrow \hat{P}$; advance C, e *in* P'.
3. [Count negative terms.] *while* $P' \neq 0$ *do* (advance C, e *in* P'; $s \leftarrow$sign (C); *if* $s=-1$ *then* $\lambda \leftarrow \lambda+1$).
4. [Initialize for B.] $\hat{k} \leftarrow 0$; *if* $\lambda=0$ *then go to* 6; $P' \leftarrow \hat{P}$; advance C, e *in* P'; *if* $P'=0$ *then go to* 6; IFCLOG (C, j, j'); $t \leftarrow 0$.
5. [Process negative terms.] *repeat* (advance C_1, e_1 *in* P'; *if* sign $(C_1)=1$ *then do* (*if* $P'=0$ *then go to* 6 *else go to* 5); $k \leftarrow e-e_1$; $C_2 \leftarrow \lambda \cdot C_1$; IFCLOG$(C_2, i, i')$; $p \leftarrow i-j-1$; $q \leftarrow p/k$; $r \leftarrow p-k \cdot q$; *if* $r<0$ *then do* ($r \leftarrow r+k$; $q \leftarrow q-1$); $k' \leftarrow q+1$; *if* $r=k-1$ *then do* ($C_2' \leftarrow |C_2|$; $C' \leftarrow$ITRUNC$(C, -k' \cdot k)$; *if* $C_2'>C'$ *then* $k' \leftarrow k'+1$); *if* $t=0$ *or* $k'>\hat{k}$ *then* $\hat{k} \leftarrow k'$; $t \leftarrow 1$) *until* $P'=0$.
6. [Finish up.] $B \leftarrow$RNP2$(\hat{k})$; *return*.

THEOREM 3.1 *Let* $P(x)=\sum_{i=0}^{n} c_i x^i$ *be an integral, univariate polynomial of degree* $n>0$, *with* $c_n \neq 0$ *and* $c_{n-k}<0$ *for at least one* k, $1 \leqq k \leqq n$. *Then* $t_{\text{CAUCHY}}(P)=O(n^2 L(|P|_\infty))$.

Proof It is easily seen that steps 1 through 3 are executed in time $O(nL(|P|_\infty))$. As for the other steps, recalling the computing times for IFCLOG and ITRUNC, we see that one execution of step 5 (which computes k' such that $|\lambda c_{n-k}/c_n|^{1/k} \leqq 2^{k'}$) is done in time $\leqq c[k+L(|c_{n-k}|)] \sim c[k+L(|P|_\infty)]$, which dominates the computing times of steps 4 and 6. Step 5, moreover, is executed at most n times, for $k=1, 2, \ldots, n$, so that

$$t_{\text{CAUCHY}}(P) \leqq c\left[\sum_{k=1}^{n} (k + L(|P|_\infty))\right]$$

$$= c\left[(n+1)\frac{n}{2} + n \cdot L(|P|_\infty)\right]$$

$$\sim c[n^2 + nL(|P|_\infty)]$$

$$\sim cn^2 L(|P|_\infty),$$

which proves the theorem. □

The implementation and analysis of the Extended Cauchy's Rule is quite similar.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1976.

[2] A. G. Akritas, *Vincent's Theorem in Algebraic Manipulation*. Ph.D. Thesis, Operations Research Program, North Carolina State University, Raleigh, 1978.

[3] A. G. Akritas, A New Method for Polynomial Real Root Isolation. *Proc. of the 16th Annual Southeast Regional ACM Conference, Atlanta, Georgia*, 39–43 (1978).

[4] A. G. Akritas, A Correction on a Theorem by Uspensky. *Bulletin of the Greek Mathematical Society* 19 (1978), 278–285.

[5] A. G. Akritas and S. D. Danielopoulos: On the Forgotten Theorem of Mr. Vincent. *Historia Mathematica* 5 (1978), 427–435.

[6] N. Obreschkoff, *Verteilung und Berechnung der Nullstellen reeller Polynome*. VEB Deutscher Verlag der Wissenschaften, Berlin 1963.

[7] J. V. Uspensky, *Theory of Equations*. McGraw-Hill, New York, 1948.

[8] A. van der Sluis, Upperbounds for Roots of Polynomials. *Numerische Mathematik* 15 (1970), 250–262.

[9] A. J. H. Vincent, Sur la Résolution des Équations Numériques. *Journal de Mathématique Pures et Appliquées* 1 (1836), 341–372.