Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Journal of Computational and Applied Mathematics 162 (2004) 33–50

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Efficient isolation of polynomial’s real roots

Fabrice Rouillier[a,*], Paul Zimmermann[b]

[a]*LORIA/INRIA/LIP6, LIP6, BP 168, 8 rue du capitaine Scott, F-75015 Paris, France*
[b]*LORIA/INRIA, 615 rue du Jardin Botanique, F-54602 Villers-lès-Nancy Cedex, France*

Received 1 December 2001; received in revised form 17 December 2002

Abstract

This paper revisits an algorithm isolating the real roots of a univariate polynomial using Descartes’ rule of signs. It follows work of Vincent, Uspensky, Collins and Akritas, Johnson, Krandick.

Our first contribution is a generic algorithm which enables one to describe all the known algorithms based on Descartes’ rule of sign and the bisection strategy in a unified framework.

Using that framework, a new algorithm is presented, which is optimal in terms of memory usage and as fast as both Collins and Akritas’ algorithm and Krandick’s variant, independently of the input polynomial. From this new algorithm, we derive an adaptive semi-numerical version, using multi-precision interval arithmetic.

We finally show that these critical optimizations have important consequences since our new algorithm still works with huge polynomials, including orthogonal polynomials of degree 1000 and more, which were out of reach of previous methods.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Univariate polynomial; Real root; Computer algebra

1. Introduction

This paper revisits the algorithm proposed by Collins and Akritas [5], based on Descartes’ rule of signs. (This algorithm is sometimes wrongly attributed to Uspensky; we refer to [2] for historical references.)

In 1836, Vincent shows [22] that if P is a square-free polynomial, then after a finite set of transformations $x \to a_i + 1/x$ with $a_i \in \mathbb{N}^*$, the number of sign variations in the coefficients of the resulting polynomial equals 0 or 1. This theorem was then derived by Uspensky [21] who asserted that after a finite number of transformations $x \to x + 1$ and $x \to 1/(x + 1)$ the number of sign

* Corresponding author.
E-mail address: fabrice.rouillier@loria.fr (F. Rouillier).

variations in the coefficients of the resulting polynomial equals 0 or 1. Because of the one-to-one correspondance between the positive roots of $P(x+1)$ and the roots of P in $]1,+\infty[$ and of the one-to-one correspondence between the positive roots of $(x+1)^n P(1/(x+1))$—here and below n denotes $\deg(P)$—and the roots of P in $]0,1[$, Uspensky's theorem suggests a simple algorithm for isolating the positive real roots of any square-free integral polynomial.

In 1976, Collins and Akritas revisited this method [5], showing first that Uspensky's original algorithm has an exponential behavior by taking an example for which the number of substitutions $x \to 1/(x+1)$ in Uspensky's method is exponential and then proposing the first effective version of the algorithm. Vincent's theorem was then improved by Collins and Johnson [6,13], and Krandick invented a variant of Collins and Akritas' algorithm working better for polynomials with very close roots [16].

With the introduction of specific data structures [16] as suggested in [6], and optimized basic transformations for speeding up the most expensive operation ($x \to x+1$) [10], the resulting algorithm is currently one of the most powerful tools for isolating real roots of square-free integral polynomials. From our personal experience in polynomial elimination, the computation of the real roots of ill-conditioned univariate integral polynomials of high degree (1000 or more) with huge coefficients (several thousands of digits) has become a critical operation in computer algebra. Collins–Akritas algorithm is surely fast enough for solving, in a reasonable time, most of the examples we may be able to compute, but the required memory space prevents from solving such problems in practice. To take well-known examples, Mignotte polynomials of degree 500 in the case of Collins–Akritas variant, or orthogonal polynomials of degree 1000 in the case of Krandick's variant, are out of reach on a machine with 512 MB of memory.

The contributions of this article are the following. In Section 3, we propose a unified description of all methods based on Descartes' rule of sign and using the bisection strategy (this description does not generalize strategies based on continued fractions developments as proposed in [1,3]). We then deduce in Section 4 a modified algorithm storing only one polynomial at any intermediate step, which runs as fast as either the Collins–Akritas algorithm and the Krandick variant for all polynomials, and performing better than both in terms of memory usage. We then present a hybrid version using an interval-arithmetic filter, extending the results from [14], developing those described in [12], and giving full proofs and algorithms for the claims in an independent work by Collins et al. [7].

2. The basic algorithm

In this section, we recall Descartes' rule of signs, which gives a bound on the number of positive real roots of a polynomial, and the algorithm from Collins and Akritas, which isolates all real roots of a polynomial lying in $]0,1[$.

2.1. Descartes' rule of signs

Descartes' rule of sign provides a very simple method to compute a bound on the number of positive roots of any univariate polynomial. It is based on the study of the sign variations in the polynomial coefficients.

Notation 1. We denote by sign (a), the sign of an element $a \in \mathbb{R}$, as being 0 if $a = 0$, 1 if $a > 0$ and -1 if $a < 0$, and define the *number of sign changes* $V(a)$ in the list $a = (a_1, \ldots, a_k)$ of elements of $\mathbb{R} \setminus \{0\}$ by induction over k:

$$V(a_1) = 0, V(a_1, \ldots, a_k) = \begin{cases} V(a_1, \ldots, a_{k-1}) + 1 & \text{if sign } (a_{k-1}a_k) = -1, \\ V(a_1, \ldots, a_{k-1}) & \text{otherwise.} \end{cases}$$

We extend this notation to a list of elements in $\mathbb{R}$ that may contain zeroes: if b is the list obtained by removing zeroes in a, we define $V(a) = V(b)$.

Using the above notations, let us recall Descartes' rule of signs:

Theorem 2 (Descartes' rule of signs). *Let $P = \sum_{i=0}^{d} a_i x^i$ 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* pos(P) *the number of positive real roots of P counted with multiplicities, then* pos$(P) \leqslant V(P)$, *and* $V(P) - \text{pos}(P)$ *is even.*

Obviously, the bound provided by Theorem 2 gives the exact number of positive real roots when it is equal to 0 or 1.

Let us consider the following transformations and basic notations:

Definition 3. Let P be a polynomial of degree n in $\mathbb{R}[x]$, k and c non-negative integers with $c < 2^k$, and $\lambda \in \mathbb{R}$. We define

$$I_{k,c} =]\tfrac{c}{2^k}, \tfrac{c+1}{2^k}], \qquad P_{k,c} = 2^{kn} P(\tfrac{x+c}{2^k}),$$
$$R(P(x)) = x^n P(\tfrac{1}{x}), \quad H_\lambda(P(x)) = P(\lambda x), \quad T_\lambda(P(x)) = P(x + \lambda).$$

(The intervals $I_{k,c}$ are called standard intervals in [16].)

A modern version of the algorithm has been invented by Collins and Akritas [5]. They introduce a bisection strategy and show that if transformations $H_{1/2}$ and $H_{1/2}T_1$ are applied iteratively on any square-free polynomial P, one obtain, after a finite number of transformations, a polynomial Q such that $V(T_1R(Q))$ gives 0 or 1. Such a process provides a simple method for isolating all the positive real roots in $]0,1[$ of P with intervals in the form $I_{k,c}$. The method can obviously be generalized to isolate all the real roots of P in any interval and so, using well-known bounds on roots locations, to isolate all the roots of P.

The proof of the termination and the complexity computation of their algorithm is based on the two following theorems:

Theorem 4 (Vincent [22]). *Let $P \in \mathbb{R}[X]$ be a square-free real polynomial. If P has exactly one real root in the interval* $]0,1[$ *and all of the nonreal roots are outside the two disks of radius* 1 *centered at* $(0,0)$ *and* $(1,0)$ *in the complex plane, then* $V(T_1R(P)) = 1$.

Theorem 5 (Collins and Johnson [6]). *Let $P \in \mathbb{R}[X]$ be a square-free real polynomial. If P does not have any nonreal roots in the disk of radius* $1/2$ *centered at* $(1/2,0)$, *then* $V(T_1R(P)) = 0$.

Remark. In [18], Mehlhorn improves Theorem 4, by showing that it still holds with the two disks of radius $\frac{1}{\sqrt{3}}$ centered in $(\frac{1}{2}, \frac{\pm 1}{2\sqrt{3}})$, which are contained in those defined by Collins and Johnson.

3. A unified framework

Let DesBound a function taking as input a polynomial P, and applying Descartes' rule of signs on $T_1(R(P(x)))=(x+1)^n P(1/(1+x))$. One can easily show that every polynomial computed in the algorithm is in the form $P_{k,c}$. Hence, isolating the real roots in $]0,1[$ of a square-free polynomial $P \in \mathbb{R}[x]$ using such a strategy consists first in computing all the pairs (k,c) such that $\mathtt{DesBound}(P_{k,c}) > 1$, and then to deduce the pairs (k,c) such that $\mathtt{DesBound}(P_{k,c})$ gives 0 or 1, taking care of the points $c/2^k$ that are roots of P.

Definition 6. Let P be a univariate polynomial of degree n, k and c two nonnegative integers with $c < 2^k$. We define:

- $\mathtt{Internal}(P) = \{(k,c) : P_{k,c}(0) \neq 0, \mathtt{DesBound}(P_{k,c}) > 1\}$,
- $\mathtt{Tree}(P) = \{(0,0)\} \cup \{(k,c) : (k-1, \lfloor c/2 \rfloor) \in \mathtt{Internal}(P)\}$,
- $\mathtt{Exact}(P) = \{(k,c) \in \mathtt{Tree}(P) : P_{k,c}(0) = 0\}$,
- $\mathtt{Isol}(P) = \{(k,c) \in \mathtt{Tree}(P) : P_{k,c}(0) \neq 0, \mathtt{DesBound}(P_{k,c}) = 1\}$,
- $\mathtt{Lost}(P) = \{(k,c) \in \mathtt{Tree}(P) : P_{k,c}(0) \neq 0, \mathtt{DesBound}(P_{k,c}) = 0\}$.

From Theorems 2, 4 and 5, it follows:

- If P is square-free, then $\mathtt{Internal}(P)$ is finite,
- $\mathtt{Tree}(P) = \mathtt{Internal}(P) \cup \mathtt{Exact}(P) \cup \mathtt{Isol}(P) \cup \mathtt{Lost}(P)$,
- α is a root of P in $]0,1[$ if and only if: $\alpha \in I_{k,c}$ for some $(k,c) \in \mathtt{Isol}(P)$, or $\alpha = c/2^k$ for some $(k,c) \in \mathtt{Exact}(P)$.

Thus, whatever the strategy used, isolating the real roots of P using Descartes' rule and a bisection strategy consists in computing $\mathtt{Tree}(P)$. The differences between the implementations are:

(1) the method for computing the polynomials $P_{k,c}$,
(2) the order of traversal of the tree: depth-first for Collins and Akritas' algorithm, breadth-first for Krandick's algorithm (but the considered nodes, which are exactly those of $\mathtt{Tree}(P)$, do not depend on the order),
(3) the way the nodes of $\mathtt{Tree}(P)$ are stored. In Collins and Akritas' algorithm, a node is represented both by a pair (k,c) and the corresponding polynomial $P_{k,c}$, but we may store (k,c) only and compute $P_{k,c}$ on demand (see Section 3.1.3).

The computation time of the resulting implementation depends mostly on item (1), while the memory usage depends on items (2) and (3).

According to item (2), we need to fix an order of traversal of $\mathtt{Tree}(P)$. Such an order can obviously be deduced from a total ordering over the pairs (k,c) from $\mathbb{N}^2$, and will be a parameter of our generic algorithm, together with the following functions:

- $\mathtt{initTree}(P)$ initializes at level $k=0$ the set T representing the working subset of $\mathtt{Tree}(P)$;

- $(k,c,Q) \leftarrow$ `getNode`$(T,<)$ takes the lowest node (k,c) of T with respect to the ordering $<$, removes that node from T, and sets Q to $P_{k,c}$;
- $(E,T) \leftarrow$ `addSucc`$((k,c),Q,E,T)$ adds to T the successor nodes of (k,c), say $(k+1,2c)$ and $(k+1,2c+1)$, taking care of the case where $Q(1/2)=0$; in this case, the node $(k+1,2c+1)$ is added to the set E.

Different implementations of these functions will lead to different strategies for computing or storing the final or intermediate results. Thus, we consider them as parameters of the generic algorithm:

GenDescartes

Input: A square-free polynomial P with all its positive roots in $]0,1[$, three functions `initTree`, `getNode` and `addSucc` and an ordering $<$ over $\mathbb{N}^2$.

Output: The lists $E=$ `Exact`(P) and $I=$ `Isol`(P).

Auxiliary function: `DesBound`(P) using Descartes' rule of sign for $T_1R(P)$.

```
begin
  $E \leftarrow \emptyset$,   $I \leftarrow \emptyset$,   $T \leftarrow$ initTree($P$)
  while $T \neq \emptyset$ do
    $(k,c,Q) \leftarrow$ getNode($T,<$)
    $s \leftarrow$ DesBound($Q$)
    if $s=1$ then $I \leftarrow I \cup \{(k,c)\}$
    if $s>1$ then $(E,T) \leftarrow$ addSucc($(k,c),Q,E,T$)
end
```

3.1. Known and new strategies as specializations of the generic algorithm

In this section, it is shown that both Collins and Akritas' algorithm [5] and Krandick's strategy [15] can be described in terms of the generic algorithm, using the *same* functions `initTree`, `getNode` and `addSucc`, the only difference being the ordering used. We then deduce a new and simple algorithm which is optimal in terms of memory usage. In the rest of the paper, $P \in \mathbb{R}[x]$ is a square-free polynomial with all its roots in $]0,1[$.

3.1.1. Collins and Akritas' algorithm

Collins and Akritas' algorithm consists in computing recursively the polynomials $P_{k,c}$ for $(k,c) \in$ `Tree`(P) using a depth-first strategy. By introducing the following ordering over $\mathbb{N}^2$:

$$(k,c) <_{\text{back}} (k',c') \quad \Leftrightarrow \quad \left(\frac{c}{2^k} < \frac{c'}{2^{k'}}\right) \quad \text{or} \quad \left(\left(\frac{c}{2^k} = \frac{c'}{2^{k'}}\right) \text{ and } (k < k')\right),$$

we can easily show that this consists in computing the polynomials $P_{k,c}$ in increasing order with respect to $<_{\text{back}}$.

With Collins and Akritas' strategy, the polynomial $P_{k,c}$ is needed for testing the nodes $(k+1,2c)$ and $(k+1,2c+1)$ that require the computation of $P_{k+1,2c}$ and $P_{k+1,2c+1}$, so that we have to represent `Tree`(P) by a list of elements of the form $(k,c,P_{k,c})$.

We extend canonically the ordering used by taking

$$(k, c, Q) <_{\text{back}} (k', c', Q') \iff (k, c) <_{\text{back}} (k', c').$$

Thus, the specialization

`GenDescartes(initTreeClassic,getNodeClassic,addSuccClassic,` $<_{\text{back}}$`)`

implements Collins and Akritas' algorithm with the following set of functions:

```
    initTreeClassic
begin
    $T \leftarrow \{(0, 0, P)\}$
end
```

Since all polynomials $P_{k,c}$ are stored in the set T, the function `getNode` is trivial:

```
        getNodeClassic
begin
    $(k, c, Q) \leftarrow \min_{<_{\text{back}}}(T)$
    remove $(k, c, Q)$ from $T$
end
```

```
                    addSuccClassic
begin
    $L \leftarrow 2^n H_{1/2}(Q)$
    $R \leftarrow T_1(L)$
    if $R(0) = 0$ then $E \leftarrow E \cup \{(k+1, 2c+1)\}$
    $T \leftarrow T \cup \{(k+1, 2c, L), (k+1, 2c+1, R)\}$
end
```

3.1.2. *Krandick's algorithm*

In [15], Krandick proposed to improve Collins and Akritas' algorithm by computing the polynomials level by level instead of using a depth-first strategy. According to our definitions, Krandick's algorithm is defined as

`GenDescartes(initTreeClassic,getNodeClassic,addSuccClassic,` $<_{\text{lev}}$`)`

where the ordering $<_{\text{lev}}$ over $\mathbb{N}^2$ can be defined as follows:

$$(k, c) <_{\text{lev}} (k', c') \quad \Leftrightarrow \quad (k < k') \quad \text{or} \quad ((k = k') \text{ and } (c < c')).$$

In both situations we can remark that `Tree`(P) is never fully stored. As noted by Krandick [15], in Collins and Akritas' algorithm, the number of stored elements of `Tree`(P) is $O(k_{\max})$ where $1/2^{k_{\max}}$ denotes the minimal interval length in the final result, while in Krandick's algorithm it is $O(n)$ where n denotes the degree of the initial polynomial.

3.1.3. A simple constant-memory algorithm

All the polynomials needed during the traversal of Tree(P) can be deduced from the initial polynomial P. We can thus represent Tree(P) simply by a list of pairs (k, c) and customize the functions initTree, getNode and addSucc as follows to get a naïve but constant-memory algorithm:[1]

```
initTreeConst
begin
    $T \leftarrow \{(0,0)\}$
end
```

The polynomials $P_{k,c}$ are not stored in the representation of Tree(P), but are computed at demand by the following function:

```
getNodeConst
begin
    $(k,c) \leftarrow \min_<(T)$
    remove $(k,c)$ from $T$
    $Q \leftarrow 2^{nk} T_c H_{1/2^k}(P)$
end
```

Owing to the simple structure of T, the addSucc function is very short:

```
addSuccConst
begin
    if $Q(1/2) = 0$ then $E \leftarrow E \cup \{(k+1, 2c+1)\}$
    $T \leftarrow T \cup \{(k+1, 2c), (k+1, 2c+1)\}$
end
```

Whatever the ordering $<$ used, the algorithm

GenDescartes(initTreeConst,getNodeConst,addSuccConst, $<$)

is very efficient in terms of memory usage, due to the way the nodes of Tree(P) are stored (only one pair of integers for each node). Only two polynomials need to be stored: the initial polynomial P and the current one $Q = P_{k,c}$.

4. An efficient memory-optimal algorithm

The full study of the complexity of classical versions of the algorithm (Collins–Akritas/Krandick) can be found in [15]. With the "big O" notation, it is easy to see that all the algorithms above have the same time-complexity bound.

[1] We mean by "constant-memory" an algorithm which stores a constant number of degree-n polynomials at any time.

The main difference between all these versions is the way chosen for the computation of $P_{k,c}$. In Collins–Akritas and Krandick's variants, $P_{k,c}$ is computed from $P_{k-1,\lfloor c/2\rfloor}$, using the relation $P_{k,c}=H_{1/2}P_{k-1,\lfloor c/2\rfloor}$, or from $P_{k,c-1}$ using the relation $P_{k,c}=T_1P_{k,c-1}$. Since the transformation $H_{1/2}$ has linear arithmetic cost, the most expensive operation is the Taylor shift T_1. This Taylor shift can be computed within $O(n^2)$ additions if P is a polynomial of degree n, or using the fast algorithms described in [10].

In the simple constant-memory algorithm of Section 3.1.3, each node requires the computation of $P_{k,c}$ using the formula $P_{k,c}=2^{nk}T_cH_{1/2^k}(P)$.

The translation T_c by an integer c can be written in the following way: $P(x+c)=P(c(x/c+1))=H_{1/c}T_1H_c(P)$. However, the simple constant-memory algorithm will compute one Taylor shift for each node, whereas Collins–Akritas and Krandick's variants perform one shift for each right son only. Hence, the resulting method may be up to two times slower.

On the other hand, Collins–Akritas and Krandick's variants require the storage of several polynomials $P_{k,c}$. This number is proportional to the width $O(n)$ of the tree for Krandick's variant, and proportional to the height $O(n\log(d))$ of the tree for Collins–Akritas, where d is the Euclidian norm of P. In contrast, the simple constant-memory variant stores only one polynomial.

In this section, we propose a new transformation that speeds up the simple constant-memory algorithm. The idea is, at each step in the algorithm, to compute the next polynomial from the preceeding one. In other words, one has to study the operations needed to compute $P_{k',c'}$ from $P_{k,c}$.

Proposition 7. *Let $P\in K[x]$. Let (k,c) and (k',c') be two pairs of integers such that $k,k'\geqslant 0, 0\leqslant c<2^k, 0\le c'<2^{k'}$. Then using the notations of Definition* 3, *we have with $d=k-k'$:*

$$P_{k',c'}=2^{-nd}H_{2^d}T_{2^dc'-c}(P_{k,c}).$$

Proof. $P_{k,c}:=2^{nk}T_cH_{1/2^k}(P)$, which gives $P=2^{-nk}H_{2^k}T_{-c}(P_{k,c})$. Identifying with the same equality for $P_{k',c'}$, we obtain $P_{k',c'}=2^{-nd}T_{c'}H_{2^d}T_{-c}(P_{k,c})$. The result then follows from $T_aH_b=H_bT_{ab}$. □

These formulae provide a new function getNodeRel for computing $P_{k,c}$ in algorithm GenDescartes, taking advantage of the previous computations:

```
                    getNodeRel
begin
  let $(k,c)$ be the previous node, $Q=P_{k,c}$ the corresponding polynomial
  $(k',c')\leftarrow \min_<(T)$
  remove $(k',c')$ from $T$
  $Q\leftarrow 2^{n(k'-k)}H_{2^{k-k'}}T_{2^{k-k'}c'-c}(Q)$
end
```

For any ordering $<$, the algorithm

GenDescartes(initTreeConst,getNodeRel,addSuccConst, $<$)

is optimal in terms of memory usage, since only one polynomial has to be stored if we replace $P_{k,c}$ in place by $P_{k',c'}$.

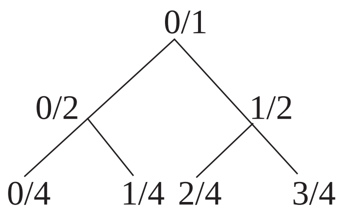

Fig. 1. Successor nodes $c/2^k$ in depth-first traversal: starting from 0/1, the node 0/2 follows with $H_{1/2}$, then 0/4 using $H_{1/2}$ again, then 1/4 using T_1, then 1/2 with H_2T_1, followed by 2/4 using $H_{1/2}$, and finally 3/4 using T_1.

We can optimize the computation time by choosing at each step in the algorithm the node that induces a translation by the smallest integer (by extending the ordering). But the following proposition shows that if we choose $<_{\text{back}}$ as ordering, then the resulting algorithm is optimal in terms of computation time.

Proposition 8. *In the algorithm*

`GenDescartes(initTreeConst,getNodeRel,addSuccConst,` $<_{\text{back}}$`),`

all translations involved in the procedure `getNodeRel` *are either* T_0 *or* T_1. *In other words, if* (k,c) *and* (k',c') *are two consecutive nodes in the ordered list—with respect to* $<_{\text{back}}$— $[(k_i,c_i)]_i$ *that represents* `Tree(`P`)`, *then the expression* $2^{k-k'}c'-c$ *equals either* 0 *or* 1.

Proof. This proposition comes from a general property of binary trees (see Fig. 1). In a depth-first traversal of such a tree, the successor of a node n is:

- its left son if n is an internal node;
- otherwise the right brother of n's nearest parent which is itself a left son.

In the first case we have $k'=k+1$ and $c'=2c$, therefore $2^{k-k'}c'-c=0$; in the second case, the nearest parent which is itself a left son is of the form $k''=k-i$, $c''=(c+1-2^i)/2^i$. The successor of (k,c) is the right brother of (k'',c'') i.e., $k'=k''=k-i$, $c'=c''+1=(c+1)/2^i$. We then have $2^{k-k'}c'-c=2^i(c+1)/2^i-c=1$. □

The algorithm

`GenDescartes(initTreeConst,getNodeRel,addSuccConst,` $<_{\text{back}}$`)`

computes exactly the same polynomials as in Collins and Akritas' or Krandick's methods, performing the same number of homothetic transformations of the form $H_{1/2^k}$ and the same number of translations of the form T_1 (one for each right son in `Tree(`P`)`). Moreover, the representation of `Tree(`P`)` requires only the storage of the studied intervals and of the current polynomial $P_{k,c}$.

Let us study more precisely the transformations involved in the algorithm

`GenDescartes(initTreeConst,getNodeRel,addSuccConst,` $<_{\text{back}}$`),` which differ from those involved in Collins–Akritas method or Krandick's variant when two successive nodes $(k,c) <_{\text{back}} (k',c')$ satisfy $k > k'$. (This happens when (k,c) corresponds to a right leaf of $\texttt{Tree}(P)$.) Even if it does not affect the theoretical complexity, this transformation may induce, in practice, an overhead in our algorithm, because of the sizes of the involved coefficients.

In such a situation, necessarily c and c' are odd, since both nodes are right sons, so that in Collins–Akritas or Krandick's variants, $P_{k',c'}$ is obtained using the identity $P_{k',c'} = T_1(P_{k',c'-1})$, while, in our algorithm, it is obtained from the formula of Proposition 7, which consists in computing $P_{k',c'} = 2^{-nh}H_{2^h}T_1(P_{k,c})$ with $h = k - k'$.

We now propose an optimization, using the fact that since $P_{k',c'}$ has integer coefficients, all the coefficients of $H_{2^h}T_1(P_{k,c})$ are necessarily multiples of 2^{nh}.

```
                         HTranslate(P, h)
Input: $P$ a polynomial of degree $n$, $P_i$ its coefficient of degree $i$, $h$ a positive integer
Output: the coefficients $P_i'$ of $2^{-nh}H_{2^h}T_1(P)$, known to be integers

1. $\mu \leftarrow h + \lceil \frac{n+1}{2^h} \rceil$
2. for $i := 0$ to $n$ do $P_i^{(0)} \leftarrow \lfloor P_i 2^{\mu - h(n-i)} \rceil$
3. for $i := 1$ to $n$ do
     for $j := n-i$ to $n-1$ do $P_j^{(i)} \leftarrow P_j^{(i-1)} + \lfloor P_{j+1}^{(i-1)} 2^{-h} \rceil$
4. for $i := 0$ to $n$ do $P_i' \leftarrow \lfloor P_i^{(n)} 2^{-\mu} \rceil$
```

Proposition 9. *If $H_{2^h}T_1(P)$ has integral coefficients, then* **HTranslate**(P,h) *correctly computes them.*

Proof. Let $\tilde{P}_j^{(i)}$ be the coefficients computed by **HTranslate** when replacing the rounding to nearest $\lfloor \cdot \rceil$ by the identity function at steps $1-3$. After step 2 we have $\tilde{P}^{(0)} = 2^{\mu-nh}P(2^hx)$, then step 3 transforms $Q(x)$ into $Q(x+2^{-h})$, giving $2^{\mu-nh}P(2^h(x+2^{-h}))$, and step 4 divides by 2^μ, giving $2^{-nh}P(2^hx+1)$.

For $i = 0 \ldots n$, let $e_i = \max_{0 \leqslant j \leqslant n} |\tilde{P}_j^{(i)} - P_j^{(i)}|$. It suffices to show that $e_n < 2^{\mu-1}$. Obviously, $e_0 \leqslant \frac{1}{2}$ and for $1 \leqslant i \leqslant n$, $e_i < e_{i-1} + e_{i-1}2^{-h} + \frac{1}{2}$. It follows that $e_n < 2^{h-1}(1+2^{-h})^{n+1}$, and the given value of μ works.

4.1. Theoretical analysis of the jumps overhead

As said above, using the HTranslate procedure induces an overhead with respect to Collins–Akritas method or Krandick's variant for each right leaf (k,c). In such a case, Collins–Akritas method will compute $P_{k,c}$ from $P_{k,c-1}$ using a simple translation, whereas HTranslate will compute $P_{k,c}$ from $P_{k+h,2^hc-1}$.

Assume all coefficients of $P_{k,c}$ have binary size at most λ. Collins–Akritas method computes those from $P_{k,c-1}$ with T_{-1} using a method similar to HTranslate, with $h = \mu = 0$; after step i the binary size is at most $\lambda + i$, thus coefficients of $P_{k,c-1}$ have binary size is at most $\lambda + n$, and the binary cost of Collins–Akritas method is $O(n^2(\lambda + 2n))$ since it involves $O(n^2)$ additions of numbers of binary size at most $\lambda + 2n$.

The coefficient of degree i from $P_{k+h,2^h c}$ has binary size at most $\lambda + h(n-i)$ since it is obtained from that of $P_{k,c}$ by $x \to x/2^h$. Therefore, the coefficient of degree i from $P_{k+h,2^h c-1}$ has binary size at most $\lambda + n + h(n-i)$ since it is obtained by T_{-1} after at most n additions or subtractions involving the coefficients of degree i or higher from $P_{k+h,2^h c}$. Therefore, the coefficients $P_i^{(0)}$ in HTranslate have binary size at most $\lambda + n + h + \lceil (n+1)/2^h \rceil$, and the maximal final size is $\lambda + 2n + h + \lceil (n+1)/2^h \rceil$. The overhead with respect to Collins–Akritas method is therefore $O(n^2(h + \lceil (n+1)/2^h \rceil))$ per jump of height h.

A detailed average-case analysis of the contribution of the h and $1/2^h$ terms can be found in [17]. It shows that the number of jumps, and also the jump distance, will usually be very small.

5. Using intervals and floating-point arithmetic

In [14], the authors made experiments using floating-point arithmetic showing that such a strategy is efficient in practice and may run correctly in most cases. In [7], the authors present a floating-point version and claim that it always terminates, but do not give any proof to support this claim. The original contribution of this section is to give a proof of that claim (Lemma 11).

Notation 10. Given any polynomial $P = \sum_{i=0}^n a_i X^i$ with integral coefficients, we denote by $P_\varepsilon = \sum_{i=0}^n [l_i, r_i] X^i$ an approximation of P using intervals with floating-point bounds of fixed precision such that:

- $a_i \in [l_i, r_i]$,
- $\|r_i - l_i\| < \varepsilon$

and we denote by $w([l_i, r_i]) = \|r_i - l_i\|$ the width of the interval $[l_i, r_i]$ For any interval $[l, r], l \leqslant r$, we define:

$$\operatorname{sign}([l,r]) = \begin{cases} 1 & \text{if } l > 0, \\ 0 & \text{if } l = r = 0, \\ -1 & \text{if } r < 0, \\ ? & \text{otherwise.} \end{cases}$$

In [14], the authors recall that methods based on Descartes's rule of signs need only to compare the signs of the coefficients of the computed polynomials. With our notations, this means that we are only interested in computing the signs of the coefficients of the polynomials $P_{k,c}$. Obviously, the authors pointed out that, due to the use of interval arithmetic, these signs may not be computable when zero appears in at least one coefficient so that the algorithm may not isolate all the roots correctly.

Even if some particular bad cases may be solved (for example, the three consecutive signs $1, ?, -1$ lead to a variation equal to 1 whatever the exact sign corresponding to ?), the problem of possible multiple roots still remains like in [14]: *When we embed $A(X)$ into an interval polynomial that also contain a polynomial with multiple roots, the interval method might run forever. To prevent this, one can abort the procedure when the search tree reaches a certain height.*

We now describe a version of the algorithm that takes as input a polynomial with intervals as coefficients. We will prove that it ends in every situation, whatever the considered polynomial, without setting bounds on the search tree.

Let DesBoundInterv be a function returning $V(T_1R(P))$ when its computation is possible and setting a global flag usp_{error} to true otherwise, and let us modify slightly our main function:

```
DesInterv
Input: a polynomial $P$ with intervals as coefficients, three functions initTree, getNode and
addSucc and an ordering $<$ over $\mathbb{N}^2$.
Output: $E = \mathtt{Exact}(P)$, $I = \mathtt{Isol}(P)$, and $\mathtt{Error}(P)$
Auxiliary function: DesBoundInterv($P$).
  $E \leftarrow \emptyset$,   $I \leftarrow \emptyset$,   $\mathtt{Error}(P) = \emptyset$,  $T \leftarrow \mathtt{initTree}(P)$
  while $T \neq \emptyset$ do
    $(k, c, Q) \leftarrow \mathtt{getNode}(T, <)$
    $s \leftarrow \mathtt{DesBoundInterv}(Q)$
    if usp_error then $\mathtt{Error}(P) \leftarrow \mathtt{Error}(P) \cup \{(k, c)\}$
    else
      if $s = 1$ then $I \leftarrow I \cup \{(k, c)\}$
      if $s > 1$ then $(E, T) \leftarrow \mathtt{addSucc}((k, c), Q, E, T)$
```

Lemma 11. *Let* $P = \sum_{i=0}^{n} [l_i, r_i]X^i$ *be an interval polynomial. If* $w([l_i, r_i]) > 0$ *for some* i, *then the following algorithm always terminates*:

$$\mathtt{DesInterv(initTreeConst,getNodeRel,addSuccConst,} <_{\text{back}}).$$

Proof. Let us denote by

$$\mathtt{Internal}_{\text{exact}}(P) = \left\{ \begin{array}{l} (k, c), k \geqslant 0, 0 \leqslant c < 2^k, P_{k,c}(0) \neq 0, \\ \mathtt{DesBoundInterv}(P_{k,c}) > 1 \text{ and } usp_{\text{error}} = \text{false} \end{array} \right\}.$$

The algorithm terminates if and only if $\#\mathtt{Internal}_{\text{exact}}(P_\varepsilon) < \infty$. Let $Q = \sum_{i=0}^{n} b_i X^i$ with $b_i \in [l_i, r_i]$, $\forall i = 0, \ldots, n$. If Q is square-free, then we have obviously $\#\mathtt{Internal}_{\text{exact}}(P_\varepsilon) \subset \mathtt{Internal}(Q)$. Since Q is square-free, then $\#\mathtt{Internal}(Q) < \infty$ and then $\#\mathtt{Internal}_{\text{exact}}(P_\varepsilon) < \infty$. Therefore, if $\exists Q = \sum_{i=0}^{n} b_i X^i$ square-free with $b_i \in [l_i, r_i]$, $\forall i = 0 \ldots n$, then the algorithm DesInterv always terminates.

Let us now assume that there exists a nonempty list of indices $I \subset [0, \ldots, n]$ such that $\forall i \in I$, $w([l_i, r_i]) > 0$. Consider the polynomial $Q = \sum_{i=0\, i \notin I}^{n} b_i X^i + \sum_{i \in I} Z_i X^i$ where $Z_i, i \in I$ are new indeterminates. The algebraic set defined by the values of z_i, $i \in I$ such that $\sum_{i=0\ i \notin I}^{n} b_i X^i + \sum_{i \in I} z_i X^i$ is not square-free has dimension at most $d - 1$ if $d = \#I$, so that $\prod_{i \in I} [l_i, r_i]$ contains at least a point $(z_i)_{i \in I}$ such that $\sum_{i=0\ i \notin I}^{n} b_i X^i + \sum_{i \in I} z_i X^i$ is square-free. □

Consider $P = \sum_{i=0}^{n} [l_i, r_i]X^i$. According to Lemma 11, the algorithm DesInterv will terminate if $\exists i$ with $w(l_i, r_i) > 0$. Suppose that $w(l_i, r_i) = 0$ for all i: in such a case, $l_i = r_i$ for all i, so that P can be considered as having rational coefficients and so integral coefficients. In such a case, one can compute explicitly its square-free part $\bar{P}$ and study the real roots of $\bar{P}$ instead of those of P. Thus, using the function

```
                    ReduceIfCan(P)
Input: $P = \sum_{i=0}^{n} [l_i, r_i]X^i$
Output: $Q = \sum_{i=0}^{n} [l'_i, r'_i]X^i$ with the same roots as $P$ and such that
$\exists(b_0, \ldots, b_n) \in \prod_{i=0}^{n} [l'_i, r'_i]$ such that $\sum_{i=0}^{n} b_i X^i$ is square-free.
  if $w([l_i, r_i]) = 0$ for all $i \in [0, \ldots, n]$ then return $(\frac{P}{gcd(P,P')})$ else return$(P)$
```

we have the following result:

Proposition 12. *Let P be a polynomial with intervals as coefficients. The function* `DesInterv`*, when applied on* `ReduceIfCan`(P)*, always terminates.*

According to Proposition 12, the algorithm `DesInterv` applied on `ReduceIfCan`(P) will always terminate, returning a list `Exact`$\cup$`Isol` of intervals that contain exactly one real root of P, and a list `Error` of intervals for which no decision was possible. In the case of a nonempty `Error` list, one can increase the precision and run again the algorithm taking as input $P_{k,c}$ for each $(k, c) \in$ `Error`.

The following algorithm describes a full strategy for isolating all the real roots in $]0, 1[$ of any square-free polynomial with integral coefficients. Note that we run `DesInterv` up to a maximum precision `maxPrec`. (We could remove this limitation, since when the precision grows, at some point Q will be represented in an exact way.)

```
                    DescartesHyb(P)
Parameters: initTree, getNode, addSucc, an order $<$ over $\mathbb{N}^2$, floatPrec and maxPrec
Input: $P = \sum_{i=0}^{n} a_i X^i \in \mathbb{Z}[X]$
Output: Isol$(P)$, Exact$(P)$

  $todo \leftarrow \{($floatPrec$, ]0, 1[)\}$
  Isol$(P) \leftarrow \emptyset$, Exact$(P) \leftarrow \emptyset$
  while $todo \neq \emptyset$
    (currentPrec$, ]\frac{c}{2^k}, \frac{c+1}{2^k}]) \leftarrow \min_{<}(todo)$
    remove (currentPrec$, ]\frac{c}{2^k}, \frac{c+1}{2^k}])$ from $todo$
    floatPrec $\leftarrow$ currentPrec
    $Q \leftarrow ReduceIfCan(convertInterval(P,$floatPrec$))$
    if floatPrec $<$ maxPrec then
      $\{l_{\text{isol}}, l_{\text{exact}}, l_{\text{error}}\} \leftarrow$ DesInterv$(T_c H_{1/2^k}(Q))$
    else
      $\{l_{\text{isol}}, l_{\text{exact}}\} \leftarrow$ GenDescartes$(T_c H_{1/2^k}(P))$
      $l_{\text{error}} \leftarrow \emptyset$
    Exact$(P) \leftarrow$ Exact$(P) \cup \{H_{2^k} T_{-c}(I),\ I \in l_{\text{exact}}\}$
    Isol$(P) \leftarrow$ Isol$(P) \cup \{H_{2^k} T_{-c}(I),\ I \in l_{\text{isol}}\}$
    $todo \leftarrow \{(increase($floatPrec$), H_{2^k} T_{-c}(I)),\ I \in l_{\text{error}}\}$
```

6. Experiments

We choose examples coming from various sources: the first four classes are well-known ill-conditioned polynomials. The last class comes from polynomial elimination problems occurring in various applications (chemistry, robotics, ...), or from classical benchmarks. All examples are polynomials of high degree with huge integer coefficients.

We used the framework described in Section 3 for implementing Collins–Akritas' (COL), Krandick's (KRA) and our version (REL), so the same arithmetic layer (GMP [11]), Taylor shifts, and memory management are used for all implementations. Our semi-numerical variant (HYB) uses the MPFI interval arithmetic library [19], based on the MPFR library [20], which uses basic functions from the GMP library.

The strategy used for increasing the precision in the HYB strategy is the following: start with a precision of 53 bits and double the precision when needed. If the precision exceeds 1024 bits, then the exact (REL) algorithm is used.

All tests were made on the computers of the MEDICIS[2] resource center using a 1.5 GB, 1 GHz AMD Athlon processor.

We measure the computation time (T) in seconds, the memory used to store the computation tree (M) in MB. The binary size of the computation tree was limited to 256 MB.

We do not give the amount of memory used by the HYB algorithm since it is always less than the value obtained with the REL algorithm.

6.1. Computation times

In Fig. 2, the first three classes have a large number of real roots (equal to the degree of the polynomial) so that the amount of memory used by algorithm KRA is large. The time difference between REL and COL on these examples is mainly due to the memory management since COL uses up to 20 times more memory than REL. The HYB algorithm is only slightly more efficient than exact strategies on these examples since most of the roots require a large working precision.

Mignotte polynomials ($x^n - 2(5x-1)^2$) have only four real roots but two of them are very close so that the amount of memory used by algorithm COL is large. On these examples, one may consider that the computation times of both REL and KRA are equivalent. On such examples, the HYB strategy is efficient compared with the others.

On examples coming from various applications (last class), the memory usage is not so critical. However, the HYB strategy seems to be very efficient.

Other efficient root finder programs are `MPSolve` from Bini and Fiorentino [4], and `eigensolve` from Fortune [9,8]. Both programs give a floating-point approximation of *all* complex roots of a univariate polynomial. Since Fortune already compared `eigensolve` to `MPSolve`, we just compared our timings to `eigensolve`. For Chebyshev and Laguerre polynomials of degree 500, our program is 2–3 times faster than `eigensolve.v1.0`. This speedup goes to more than 5 for the degree-500 Wilkinson polynomial. However, `eigensolve` is 100 times faster for Mignotte's polynomials: our algorithm does not use the fact that those polynomials are sparse.

[2] http://www.medicis.polytechnique.fr

	Deg.	Real	HYB	REL		COL		KRA	
		Roots	T.	T.	M.	T.	M.	T.	M.
Chebyshev	900	900	695	764	1.0	901	3.1	771	68.7
Chebyshev	1000	1000	1183	1305	1.2	1555	4.2	1340	110.6
Laguerre	900	900	2116	2079	1.7	3119	31.5	2950	193.1
Laguerre	1000	1000	3055	3325	2.1	4414	40.2	?	> 256
Wilkinson	900	900	542	526	1.8	677	29.7	821	218
Wilkinson	1000	1000	840	815	2.2	1050	36.8	?	> 256
Mignotte	300	4	33	565	2.0	1130	176.5	566	7.8
Mignotte	400	4	122	2421	4.7	?	> 256	2393	18.5
Mignotte	600	4	428	>2h	?	?	> 256	>2h	?
P2	1366	50	335	2113	9.7	2172	141.1	2108	26.0
fau2	244	18	8	24	0.4	44	15.5	30	3.0
cyclic7	924	56	42	189	1.0	199	7.7	200	10.5
katsura9	512	120	27	49	0.4	50	2.8	58	7.9
virasoro	256	224	10	8	0.2	15	2.3	25	5.6

Fig. 2. Comparison of the different algorithms.

6.2. *Detailed experiments with the hybrid algorithm*

Figs. 3 and 4 display how many roots can be computed at the different steps of `DescartesHyb`. At each step corresponds a precision for the floating-point arithmetic. We set `maxPrec` to 1024 bits and increase the precision at each step by doubling it.

Column *Deduced* displays the number of roots deduced without computations (even polynomials for example) while column *ES* gives the maximum bit-size of integers appearing in algorithm REL. One can remark that on the first three classes of examples, a great precision is needed to obtain all the roots of the polynomials. These examples are known to be difficult, since they have a large number of roots, or very close roots.

On the second figure, with polynomials coming from an algebraic elimination on systems of algebraic equations, one can see that the use of doubles is sufficient to obtain some roots and anyway a small precision (relatively to the size of the input) is enough to get all the solutions.

7. Conclusion

This paper gave a unified framework, with which all known algorithms for isolating polynomial's real roots using Descartes' rule of sign and a bisection strategy can be described in a generic way. From this framework, we first derived a very simple algorithm, which performs at most twice as much translations—which are the most expensive operations—as previous algorithms, and needs to store only two polynomials at a time. Then we deduce a new exact algorithm that is both optimal in space and time, in terms of arithmetic complexity: it needs to store only one polynomial at a

Name	Deg	53	107	215	431	863	Exact	Deduced	ES
Chebyshev	100	0	0	50	0	0	0	50	960
Chebyshev	200	0	0	0	100	0	0	100	2240
Chebyshev	300	0	0	0	150	0	0	150	3648
Chebyshev	400	0	0	0	0	200	0	200	5248
Chebyshev	500	0	0	0	0	250	0	250	7040
Chebyshev	600	0	0	0	0	300	0	300	8448
Chebyshev	700	0	0	0	0	0	350	350	10560
Laguerre	100	0	0	100	0	0	0	0	1536
Laguerre	200	0	0	2	198	0	0	0	3456
Laguerre	300	0	0	0	31	269	0	0	5440
Laguerre	400	0	0	0	7	393	0	0	7648
Laguerre	500	0	0	0	0	500	0	0	9568
Laguerre	600	0	0	0	0	70	530	0	12064
Laguerre	700	0	0	0	0	45	655	0	14048
Laguerre	800	0	0	0	0	15	785	0	16864
Laguerre	900	0	0	0	0	3	897	0	18944
Laguerre	1000	0	0	0	0	0	1000	0	21056
Wilkinson	100	0	0	0	100	0	0	0	1600
Wilkinson	200	0	0	0	2	198	0	0	3776
Wilkinson	300	0	0	0	0	300	0	0	5632
Wilkinson	400	0	0	0	0	8	392	0	8288
Wilkinson	500	0	0	0	0	0	500	0	10336
Mignotte	100	1	0	0	2	0	0	1	11616
Mignotte	200	1	0	0	2	0	0	1	46400
Mignotte	300	1	0	0	2	0	0	1	104352
Mignotte	400	1	0	0	0	2	0	1	185536
Mignotte	500	1	0	0	0	2	0	1	?
Mignotte	600	1	0	0	0	2	0	1	?

Fig. 3. Detailed analysis of the hybrid algorithm (I).

time, and performs exactly the same number of translations as both Collins/Akritas' and Krandick's algorithms.

In the second part of the paper, we consider polynomials with interval coefficients. We exhibit an algorithm that always terminates for interval polynomials, where previous algorithms required a bound on the tree height. When used with floating-point intervals, this yields a new hybrid algorithm, which decreases the average bit-size—and thus the cost—of arithmetic operations.

Name	Deg	53	107	215	431	863	Exact	Deduced	ES
P2	1366	1	7	9	8	25	0	0	46464
fau2	244	0	2	8	8	0	0	0	17792
cyclic7	924	56	0	0	0	0	0	0	10208
katsura9-1	512	3	53	63	0	0	0	1	8896
katsura9-2	512	0	0	0	0	0	120	0	9248
virasoro	256	22	26	0	84	50	41	1	5664

Fig. 4. Detailed analysis of the hybrid algorithm (II).

Finally, extensive experiments on different polynomial classes show that the new exact algorithm practically behaves better than both Collins/Akritas' and Krandick's algorithms, both in terms of time and memory usage. In turn, the new hybrid algorithm never behaves much worse than the new exact one, and in some cases behaves much better, especially those coming from real applications.

Some open questions still remain. Is it possible to perform a Taylor shift of a degree-n polynomial with l-bit coefficients in less than $\Theta(n^2 l)$ time? Is it possible to obtain a tight bound, depending on the input polynomial, for the precision needed in the hybrid algorithm?

Acknowledgements

The authors would like to thank the anonymous referees for their careful reading of the paper, and their very useful comments, and Guillaume Hanrot for providing an independent and efficient implementation of our algorithms.

References

[1] A.G. Akritas, An implementation of Vincent's Theorem, Numer. Math. 36 (1980) 53–62.

[2] A.G. Akritas, There is no "Uspensky's method", in: Proceedings of the 1986 SYMSAC, ACM, 1986, New York, pp. 88–90.

[3] A.G. Akritas, A. Bocharov, A. Strzebonski, Implementation of real roots isolation algorithms in Mathematica, in: Abstracts of the International Conference on Interval and Computer Algebraic Methods in Science and Engineering, 1994, pp. 23–27.

[4] D. Bini, G. Fiorentino, Numerical computation of polynomial roots: Mpsolve—version 2.0. http://www.dm.unipi.it/pages/bini/public_html/papers/mps2.html, 1998, 21pp.

[5] G. Collins, A. Akritas, Polynomial real roots isolation using Descartes' rule of signs, in: SYMSAC, 1976, pp. 272–275.

[6] G. Collins, J. Johnson, Quantifier elimination and the sign variation method for real roots isolation, in: ACM-SYGSAM ISSAC, 1989, pp. 264–271.

[7] G.E. Collins, J. Johnson, W. Krandick, Interval arithmetic in cylindrical algebraic decomposition, J. Symbolic Comput. 34 (2002) 145–157.

[8] S. Fortune, An iterated eigenvalue algorithm for approximating roots of univariate polynomials, 2000, 23p. http://cm.bell-labs.com/who/sjf/ieaarup.ps.gz.

[9] S. Fortune, Polynomial root finding using iterated eigenvalue computation, in: B. Mourrain (Ed.), Proceedings of ISSAC'01, University of Western Ontario, Ontario Research Centre for Computer Algebra, ACM Press, 2001, pp. 121–128.
[10] J.V.Z. Gathen, J. Gerhard, Fast algorithms for Taylor shifts and certain difference equations, in: Proceedings of International Symposium On Symbolic and Algebraic Computations, 1997, pp. 40–47.
[11] GMP. http://www.swox.com/gmp.
[12] G. Hanrot, F. Rouillier, P. Zimmermann, Well-defined semantics for floating-point computations: the MPFR library, 2000, ISSAC'2000 and MuPAD Workshop 2000 poster sessions, abstract available at http://www.mupad.de/mw2000/anno/abstracts_p/index_e.shtml.
[13] J. Johnson, Algorithms for polynomial real root isolation, Technical Report OSU-CISRC-8/91-TR21, Ohio State University, Department of Computer and Information Science, 1991.
[14] J. Johnson, W. Krandick, Polynomial real roots isolation using approximate arithmetic, in: W. Küchlin (Ed.), Proceedings of International Symposium On Symbolic and Algebraic Computations, ACM Press, New York, 1997.
[15] W. Krandick, Isolierung reeller Nullstellen von Polynomen, in: J. Herzberger (Ed.), Wissenschaftliches Rechnen, Akademie Verlag, Berlin, 1995, pp. 105–154.
[16] W. Krandick, A data structure for approximation, Technical Report MS 96-021, University of Edinburgh, Department of Mathematics and Statistics, 1996.
[17] W. Krandick, Trees and jumps and real roots, J. Comput. Appl. Math. 162 (2004) 51–55.
[18] K. Mehlhorn, Private communication, October 2001, 6pp.
[19] MPFI. http://www.ens-lyon.fr/∼nrevol/nr_software.html.
[20] MPFR. http://www.loria.fr/projets/mpfr/.
[21] J. Uspensky, Theory of Equations, McGraw-Hill Book Company, New York, 1948.
[22] A.J.H. Vincent, Sur la résolution des équations numériques, J. Math. Pures Appl. Ser. 1, 1 (1836) 341–372.