Math.comput.sci. 1 (2007), 305–320

1661-8270/020305-16, *published online* October 15, 2007
DOI 10.1007/s11786-007-0022-6

Mathematics in Computer Science

A New Method for Real Root Isolation of Univariate Polynomials

Ting Zhang and Bican Xia

Abstract. A new algorithm for real root isolation of univariate polynomials is proposed, which is mainly based on exact interval arithmetic and bisection method. Although exact interval arithmetic is usually supposed to be inefficient, our algorithm is surprisingly fast because the termination condition of our algorithm is different from those of existing algorithms which are mostly based on Descartes' rule of signs or Vincent's theorem and we decrease the times of Taylor shifts in some cases. We test our algorithm on a large number of examples from the literature and report the performance.

Mathematics Subject Classification (2000). 68W30.

Keywords. Real root isolation, interval arithmetic, interval Newton operator, bisection method, Maple.

1. Introduction

Real root isolation of univariate polynomials with integer coefficients plays a significant role in many algorithms concerning computational real algebra and real algebraic geometry. In many computer algebra systems (CAS), one can find implementations of algorithms based on different principles for real root isolation. The `realroot` function in Maple and the `RealRootIntervals` function in Mathematica are such examples.

Many algorithms for real root isolation have been proposed in the literature, see, for example, [1,2,4–6,9,12], which can be classified into two kinds of methods: bisection and non-bisection methods. For example, algorithms proposed in [4–6,9, 12] are essentially based on bisection strategy, taking use of different principles to rule out intervals with no roots and to pick out intervals containing exactly one

Corresponding author: Bican Xia (xbc@math.pku.edu.cn).
This work is supported by NKBRPC-2004CB318003, NKBRPC-2005CB321902 and NSFC-60573007 in China.

root and the algorithm proposed in [1, 2] is based on Vincent's theorem, which does not bisect intervals.

The idea of bisection method is natural and easy to understand. Suppose B is the root bound of $f(x)$, a univariate polynomial with integral coefficients, and we have an effective rule $\mathcal{M}$ to determine the number of roots of $f(x)$ in an interval, then we may bisect $(-B, B)$ (or $(0, B)$) repeatedly and apply $\mathcal{M}$ each time to rule out intervals not containing roots. Sturm's theorem is an instance of such rules. Some other rules, e.g., Budan–Fourier's theorem and Descartes' rule of signs, can not be used directly to determine the exact number of roots in intervals of polynomials in general. But combined with the bisection method and some transformation techniques, the two rules are very suitable for real root isolation [6]. In fact, methods based on Descartes's rule of signs are most efficient in general [4–6,8,12]. A systematic study was initiated in [8] comparing algorithms based on different rules as well as giving some best theoretical results up to then. All the known algorithms for real root isolation based on Descartes' rule of signs and the bisection strategy are described in [12] in a unified framework, and a new algorithm REL is presented in [12] which is proved to be very efficient and can works with huge polynomials, including orthogonal polynomials of degree 1000 and more.

The CF method proposed in [1] does not perform bisection. It is directly based on Vincent's theorem and is more tricky. A comparative study of the CF method and the REL method through timings on some special polynomials is reported in [2]. But, we are confused that the timings of REL reported in [2] is inconsistent with the original ones in [12].

One main computation involved in algorithms based on Descartes' rule of signs and bisection is computing Taylor shifts. This is also true for the CF algorithm. Because computing Taylor shifts may be very costly especially for polynomials of high degrees, fast algorithms for Taylor shifts [7] are employed in the REL and CF methods. Moreover, some special technics are implemented in REL for fast Taylor shifts [11] while CF computes lower bounds of positive roots of polynomials to decrease the number of Taylor shifts [2].

We propose an algorithm which is based on interval arithmetic and bisection strategy. Our first idea is to use exact interval Newton operator to rule out intervals not containing roots quickly. Our second idea is to decrease the number of Taylor shifts. To achieve this, we perform exact interval Newton iteration instead of Taylor shifts when some conditions are satisfied (see the remark in Section 4). Although most interval algorithms and arithmetic are based on float point computation, we think it is useful to study interval arithmetic with exact calculation.

The rest of the paper is organized as follows. Section 2 reviews briefly some concepts and results in interval arithmetic. Section 3 introduces a naive algorithm, called `Nrealroot`, for real root isolation, which does not perform Taylor shifts. The weak points of `Nrealroot` is discussed and an improved algorithm

called `Trealroot` is proposed in Section 4. We report the performance of an implementation under Maple of `Trealroot` on a large number of examples from the literature and analyse the performance of our program in Section 5. Finally, we conclude in Section 6.

2. Interval arithmetic

Interval arithmetic, which is just the arithmetic operation on sets of real numbers, is easy to understand. For instance,

$$[1,2]+[3,4]=[4,6],\ [1,2]\cdot[-2,3]=[-4,6]\,,$$
$$1/[-1,2]=[-\infty,-1]\bigcup[1/2,+\infty]\,.$$

Strictly speaking, we have the following definitions [3]. Let the set of all intervals is denoted by $\mathrm{I}(\mathbb{R})$.

Definition 1. For $X=[a,b]\in\mathrm{I}(\mathbb{R})$, the *width*, the *midpoint* and the *sign* of X are defined, respectively, as $\mathrm{W}(X)=b-a$, $\mathrm{m}(X)=\frac{a+b}{2}$ and $\mathrm{sign}(X)$ is -1 if $b<0$; 1 if $a>0$ and 0 otherwise.

Definition 2. For $X,Y\in\mathrm{I}(\mathbb{R})$ and $\diamond\in\{+,-,\cdot\}$, we define

$$X\diamond Y=\{x\diamond y|\ x\in X,\ y\in Y\}\,.$$

For $X=[a,b]\in\mathrm{I}(\mathbb{R})$, if $\mathrm{sign}(X)\neq 0$, we define

$$X^{-1}=1/X=[1/b,1/a]\,;$$

if $\mathrm{sign}(X)=0$ and $\mathrm{W}(X)\neq 0$, we define

$$X^{-1}=1/X=\begin{cases}[-\infty,1/a] & : \quad b=0\\ [1/b,+\infty] & : \quad a=0\\ [-\infty,1/a]\cup[1/b,+\infty] & : \quad a<0<b\,;\end{cases}\tag{2.1}$$

if $X=[0,0]$, X^{-1} is undefined. And Y/X is defined to be $Y\cdot X^{-1}$, where $Y/X=Y\cdot[-\infty,1/a]\cup Y\cdot[1/b,+\infty]$ if $a<0<b$.

For $a\in\mathbb{R}$, $X\in\mathrm{I}(\mathbb{R})$ *and* $\diamond\in\{+,-,\cdot,/\}$, we define $a\diamond X=[a,a]\diamond X$ and $X\diamond a=X\diamond[a,a]$.

Let f be an arithmetic expression of a polynomial in $\mathbb{R}[x_1,\ldots,x_n]$. We replace all operands of f as intervals and replace all operations of f as interval operations and denote the result by F. Then,

$$F:\mathrm{I}(\mathbb{R})^n\to\mathrm{I}(\mathbb{R})$$

is called an *interval evaluation* of f. Different expressions of one polynomial may result in different interval evaluations. For example, if we replace x by $[1,2]$, the interval evaluations of x^2-2x+1,$(x-1)^2$ and $(x-2)x+1$ are $[-2,3]$, $[0,1]$ and $[-1,1]$, respectively. Let $f(x)=\sum_{i=0}^{n}a_ix^i$ be a polynomial. We use

$$\Big(\big((a_nx+a_{n-1})x+a_{n-2}\big)x+\cdots+a_1\Big)x+a_0$$

to calculate an *interval evaluation* of $f(x)$ in our algorithm.

Let $f(x)$ be a polynomial in $\mathbb{R}[x]$ and X an interval, *the interval Newton operator* [3] is defined as

$$N(X) = \mathrm{m}(X) - \frac{f\big(\mathrm{m}(X)\big)}{F'(X)} \tag{2.2}$$

where F' is an interval evaluation of f' and $\mathrm{m}(X)$ is the midpoint of X.

The above defined Newton operator satisfies the following properties [3, 10].

1. If $x^* \in X$ is a zero of $f(x)$, $x^* \in N(X)$.
2. If $X \cap N(X) = \emptyset$, $f(x)$ does not have zeros in X.
3. If $N(X) \subset X$, $f(x)$ has zeros in X.
4. If $N(X)$ is contained in the interior of X, $f(x)$ has a unique zero in X.

3. Interval algorithm based on exact calculation

Let $f(x) \in \mathbb{Z}[x]$ be a polynomial and $X = X_0$ an initial interval, setting $X = N(X) \cap X$, repeating this *interval Newton iteration* for all resulting intervals and discarding empty sets, we will obtain a set of intervals possibly containing zeros of $f(x)$ on X_0. Naturally, one can propose an algorithm for real root isolation using interval Newton iteration and exact interval arithmetic. Let us describe the main steps of such algorithm as follows.

For a square-free polynomial $f(x) \in \mathbb{Z}[x]$ and an initial interval X_0, first, setting $X = X_0$, we compute the interval evaluation of f' on X, i.e., $F'(X)$. If the sign of $F'(X)$ is not zero, F is monotonic on X and thus it is sufficient to check the signs of $f(x)$ at the endpoints of X to determine whether or not there is a root in X.

Second, if the sign of $F'(X)$ is zero, we compute $N(X)$ by the definition of (2.2). If $N(X)$ satisfies the 2nd or 4th property listed at the end of Section 2, we are done. Otherwise, $N(X) \cap X$ can be one or two intervals, say X_1 (and X_2).

Third, letting X be X_1 and X_2, respectively, repeat the first two steps for the new X.

Obviously, the procedure terminates within a finite number of steps because $f(x)$ being square-free guarantees that the monotonicity will occur as the intervals get smaller and smaller. About the correctness of the procedure, please note first that each output interval contains only one zero and second, the 1st property of interval Newton operator ensures that we do not miss any zeros.

We will discuss on the weak point of such naive algorithm and on how to improve it later. The strongpoint of such naive algorithm is that it avoids Taylor shifts which is expensive if polynomials have high degrees. For example,

$$\begin{aligned} f(x) = {} & -10x^{93925} + 62x^{82660} - 82x^{76886} + 80x^{69549} - 44x^{68273} \\ & + 71x^{55578} - 17x^{53739} - 75x^{30731} - 10x^{22679} - 7 \end{aligned} \tag{3.1}$$

is a polynomial randomly generated by Maple. Expanding $f(x+1)$ is a challenge, so isolating real roots of $f(x)$ is a challenge to those algorithms based on Taylor shift, such as REL and CF. We use the following "pseudo-codes" to describe the above procedure more accurately.

Algorithm: `Nrealroot`
Input: A square-free polynomial $f \in \mathbb{Z}[x]$ with $f(-1)f(0)f(1) \neq 0$.
Output: $OutL$, a list of isolating intervals for the real zeros of $f(x)$.

1. Set $OutL = [\,], n = \text{degree}(f, x)$.
2. For every interval I in SNrealroot$(f(x))$ (a sub-algorithm to be described below), add I to $OutL$.
3. For every interval I in SNrealroot$(f(-x))$, add $[-1,-1]\cdot I$ to $OutL$.
4. For every interval I in SNrealroot$(x^n f(1/x))$, add $1/I$ to $OutL$.
5. For every interval I in SNrealroot$(x^n f(-1/x))$, add $1/([-1,-1]\cdot I)$ to $OutL$.

We use a data structure $[h, u, v, r, s, k]$ in the description of the sub-algorithm, where h is a polynomial and u, v, r, s, k are integers. The zeros of $h(x)$ in (u, v) correspond to the zeros of $g(x)$ in $(r/2^k, s/2^k)$ one to one.

Sub-algorithm: `SNrealroot`
Input: A square-free $g \in \mathbb{Z}[x]$ with $g(0)g(1) \neq 0$.
Output: $OutL$, a list of isolating intervals for the zeros of $g(x)$ in $(0, 1)$.

1. Set $OutL = [\,]; L = [[g, 0, 1, 0, 1, 0]]$. Repeat the following steps until L is empty.
2. Fetch the first element of L, say $[h, u, v, r, s, k]$, and delete it in L.
3. If $h((u+v)/2) = 0$, include
$$\big[(r+s)/2^{k+1}, (r+s)/2^{k+1}\big]$$
in $OutL$ and replace $h(x)$ by $h(x)/(x-(u+v)/2)$. Include
$$\big[2^m h(x/2), 2u, u+v, 2r, r+s, k+1\big]$$
and
$$\big[2^m h(x/2), u+v, 2v, r+s, 2s, k+1\big]$$
in L as the first two elements where m is the degree of $h(x)$, then go to Step 2.
4. If $H'((u, v))$(the interval evaluation of $h'(x)$ on (u, v)) does not contain 0, then check the signs of $h(u)$ and $h(v)$. If $h(u)h(v) < 0$, then include $(r/2^k, s/2^k)$ in $OutL$.
5. If $H'((u, v))$ contains 0, apply interval Newton operator to h and (u, v), i.e., set
$$N(h, u, v) = (u+v)/2 - \frac{h\big((u+v)/2\big)}{H'(u, v)}\,.$$
   (a) If $N(h, u, v) \cap ((u+v)/2, v)$ is not empty, then include
   $$\big[2^m h(x/2), u+v, 2v, r+s, 2s, k+1\big]$$
   in L as the first element.

(b) If $N(h,u,v) \cap (u,(u+v)/2)$ is not empty, then include

$$\left[2^m h(x/2), 2u, u+v, 2r, r+s, k+1\right]$$

in L as the first element.

4. Improved algorithm

Our implementation of `Nrealroot` under Maple isolates the three real roots of $f(x)$ in (3.1) within 601 seconds on a notebook PC IBM T23 (Pentium 1.13GHz CPU, 512M Memory, Windows XP, Maple 9). However, the limitation of `Nrealroot` is obvious. For instance, it takes about 290 seconds to isolate the zeros of

$$f_{15}(x) = \sum_{i=1}^{15}(x+i)$$

and the loop count is 216904. Obviously, the number of real zeros of input polynomial is the main factor affecting the performance of `Nrealroot`.

Another main factor is that, for a polynomial $f(x)$ and an interval X, $F(X)$ (the interval evaluation of $f(x)$ on X) is often much more wider than $f(X)$. So, those intervals not containing zeros cannot be recognized quickly and thus a great number of redundant iterations occur.

We found that the interval evaluation of a polynomial $g(x)$ on $(0,1)$ has little difference to $g((0,1))$ and is easy to be computed. Note that Taylor shifts can map the zeros of one polynomial f in $(a,a+1)$ to $(0,1)$, and thus simplify the interval evaluation $F([a,a+1])$ to $F([0,1])$. Therefore, we combine the Newton iteration with Taylor shifts. As a result, the efficiency has been improved dramatically. For example, after using Taylor shifts in Step 3, Step 5(a) and Step 5(b) of `Nrealroot`, the timing and loop count for isolating the real zeros of f_{15} are $0.9s$ and 79, respectively. However, as we stated before, Taylor shifts may be very costly when the degrees of polynomials are large. So, we use some criterion to determine whether or not we need to perform Taylor shift at next step (see the Remark below).

We can further improve the efficiency of `Nrealroot` by combining some other techniques. For example, if we use Budan–Fourier's theorem to check sign changes before Step 3 of `Nrealroot`, isolating the real zeros of f_{15} just needs $0.4s$ and 45 loops. Based on the above observation, we designed an algorithm `Trealroot` which is much more efficient than `Nrealroot`.

Algorithm: `Trealroot`
Input: A square-free polynomial $f \in \mathbb{Z}[x]$ with $f(-1)f(0)f(1) \neq 0$.
Output: $OutL$, a list of isolating intervals for the real zeros of $f(x)$.

1. Set $OutL = [\,], n = \text{degree}(f,x)$.
2. For every interval I in `kiteflying`$(f(x))$ (a sub-algorithm to be described below), add I to $OutL$.
3. For every interval I in `kiteflying`$(f(-x))$, add $[-1,-1]\cdot I$ to $OutL$.
4. For every interval I in `kiteflying`$(x^n f(1/x))$, add $1/I$ to $OutL$.
5. For every interval I in `kiteflying`$(x^n f(-1/x))$, add $1/([-1,-1]\cdot I)$ to $OutL$.

Before we describe the sub-algorithm, we would like to give the following remark for better understanding of the idea of the algorithm.

Remark 3. In Step 3, we hope Budan–Fourier's theorem could help to decrease the number of intervals to be bisected. Similarly, lower bounds computed in Step 4 could help to decrease the time of bisection under some cases. Step 5 handles the case that the midpoint of an interval is a zero. In Step 7, we compute interval Newton iteration. However, if the width of the result interval is small (less than $(1-\frac{1}{b})\frac{v-u}{2}$) or the loop count is small (less than c), we do not perform Taylor shift as we do in `Nrealroot`. The values of the two parameters b and c are alterable and are set to be some empirical values in our algorithm.

We use a data structure $[h, u, v, r, s, k, v_1, v_2]$ in the following description, where h is a polynomial and u, v, r, s, k, v_1, v_2 are integers. The zeros of $h(x)$ in (u, v) correspond to the zeros of $g(x)$ in $(r/2^k, s/2^k)$ one to one. The number of zeros of h in (u, v) is less than or equals to $v_1 - v_2$. Also, we use $\mathrm{V}(p(x))$ to denote the sign changes of the coefficients of a polynomial $p(x)$. The two parameters b and c used in Step 7 are called the *performance parameter*, whose values are based on experience. In the following sub-algorithm, we set $b = \mathrm{degree}(g), c = iquo(b, 30)$.

Sub-algorithm: `kiteflying`
Input: A square-free polynomial $g \in \mathbb{Z}[x]$ with $g(0)g(1) \neq 0$.
Output: $OutL$, a list of isolating intervals for the real zeros of $g(x)$ in $(0, 1)$.

1. Set $v_1 = \mathrm{V}(g(x))$, $v_2 = 0$, $OutL = [\ ]$, and
$$L = [\ [g, 0, 1, 0, 1, 0, v_1, v_2]\ ]\,.$$
Repeat the following steps until L is empty.
2. Fetch the first element of L, say $[h, u, v, r, s, k, v_1, v_2]$, and delete it in L.
3. If $v_1 - v_2 = 0$ then go to Step 2.
If $v_1 - v_2 = 1$ and $h(u)h(v) < 0$, then include $(r/2^k, s/2^k)$ in $OutL$ and go to Step 2.
If $v_1 - v_2 > 1$ then go to next step.
4. If $u = 0$ and $v = 1$, calculate a lower bound a of zeros of $x^m h(1/x)$ in $(1, +\infty)$, where m is the degree of $h(x)$. If $a > 1$, then let
$$h = h(ax)\,, \quad r = ra\,, \quad s = ra + s - r\,, \quad k = k + \log_2 a\,.$$
5. If $h((u + v)/2) = 0$, then include $[(r + s)/2^{k+1}, (r + s)/2^{k+1}]$ in $OutL$ and replace $h(x)$ by $h(x)/(x - (u + v)/2)$.
Set $v_1 = \mathrm{V}(h(x))$ and $v_0 = \mathrm{V}(2^m h((x + u + v)/2))$. Include
$$\big[2^m h(x/2), 2u, u + v, 2r, r + s, k + 1, v_1, v_0\big]$$
and
$$\Big[2^m h\big((x + u + v)/2\big), 0, v - u, r + s, 2s, k + 1, v_0, v_2\Big]$$
in L as the first two elements where m is the degree of $h(x)$. Go to Step 2.
6. If $H'((u, v))$ (the interval evaluation of $h'(x)$ on (u, v)) does not contain 0, and $h(u)h(v) < 0$, then include $(r/2^k, s/2^k)$ in $OutL$. Go to Step 2.

7. If $H'((u,v))$ contains 0, apply interval Newton operator to h and (u,v), i.e., set
$$N(h,u,v) = (u+v)/2 - \frac{h\big((u+v)/2\big)}{H'(u,v)}\,.$$
Set $v_0 = v_2$.
   (a) If $N(h,u,v) \cap ((u+v)/2, v)$ is not empty then
   if the width of $N(h,u,v) \cap ((u+v)/2, v)$ is less than $(1-\frac{1}{b})\frac{v-u}{2}$
   or the loop count is less than c or $u = 0$
   then include
   $$\big[2^m h(x/2), u+v, 2v, r+s, 2s, k+1, v_1, v_2\big]$$
   in L as the first element
   else set $v_0 = \mathrm{V}(2^m h((x+u+v)/2))$ and include
   $$\Big[2^m h\big((x+u+v)/2\big), 0, v-u, r+s, 2s, k+1, v_0, v_2\Big]$$
   in L as the first element.
   (b) If $N(h,u,v) \cap (u, (u+v)/2)$ is not empty then
   if the width of $N(h,u,v) \cap (u, (u+v)/2)$ is less than $(1-\frac{1}{b})\frac{v-u}{2}$
   or the loop count is less than c
   then include
   $$\big[2^m h(x/2), 2u, u+v, 2r, r+s, k+1, v_1, v_2\big]$$
   in L as the first element
   else set $v_1 = \mathrm{V}(2^m h((x+2u)/2))$ and include
   $$\Big[2^m h\big((x+2u)/2\big), 0, v-u, 2r, r+s, k+1, v_1, v_0\Big]$$
   in L as the first element.

5. Experiments and analysis

We have no results about the complexity of our algorithm so far. Therefore, it is impossible for us to compare our algorithm with others in theory. We have implemented our algorithm `Trealroot` by Maple and done experiments on a large number of examples in the literature. We hope these data can illustrate the behavior of our algorithm.

All the examples in this paper were computed on a notebook PC IBM T23 (Pentium 1.13GHz CPU, 512M Memory, Windows XP) with Maple 9 and the timings are obtained by using the `time` function in Maple.

For randomly generated polynomials, the timings are average timings on 5 polynomials[1]. For ChebyShev polynomials (which are generated by the functions `ChebyShevT` and `ChebyShevU` in Maple), only positive roots are isolated as in the

[1]To compare with other tools, we also compute 5 randomly generated polynomials as in the literature.

literature. We take $a = 5$ for Mignotte polynomials $(x^n - 2(ax-1)^2)$. Wilkinson polynomials have the form $f_n(x) = \prod_{i=1}^{n}(x+i)$.

Here is a description of data we collected.

F1	bits of coefficients
F2	terms
F3	degree
F4	number of samples
F5	loop count
F6	Taylor shift count
F7	count when $a > 1$ at Step 4
F8	count of termination at Step 3 when $v_1 - v_2 = 0$
F9	count of termination at Step 3 when $v_1 - v_2 = 1$ and $h(u)h(v) < 0$
F10	count of termination at Step 3 when $v_1 - v_2 = 1$ and $h(u)h(v) > 0$
F11	count of termination at Step 6 when $h(u)h(v) < 0$
F12	count of termination at Step 6 when $h(u)h(v) > 0$
F13	count of termination at Step 7 when $N(h,u,v) \cap (u,v)$ is empty
F14	count at Step 7 when $N(h,u,v) \cap ((u+v)/2, v)$ is empty
F15	count at Step 7 when $N(h,u,v) \cap (u, (u+v)/2)$ is empty
F16	count of termination at Step 5
F17	average number of real zeros
F18	timing (in second)
TYPE	CT=ChebyShevT, CU=ChebyShevU, L=Laguerre, W=Wilkinson, M=Mignotte

T1-1: Randomly generated polynomials. Continued in T1-2.

ID	F1	F2	F3	F4	F5	F6	F17	F18
1	10	10	100	5	18.4	0.0	1.8	0.1
2	10	10	500	5	38.4	0.0	3.0	0.2
3	10	10	1000	5	46.4	0.0	3.4	0.3
4	10	10	2000	5	36.4	0.0	3.4	0.4
5	10	101	100	5	29.4	3.6	3.6	0.9
6	10	501	500	5	42.6	0.0	2.8	6.3
7	10	1001	1000	5	64.4	0.0	6.4	26.0
8	10	2001	2000	5	74.4	0.0	6.0	118.3
9	1000	10	100	5	18.0	1.8	2.6	0.2
10	1000	10	500	5	27.2	0.0	3.8	0.1
11	1000	10	1000	5	35.0	0.0	2.6	0.2
12	1000	10	2000	5	37.6	0.0	2.8	0.5
13	1000	101	100	5	30.6	4.0	2.8	1.0
14	1000	501	500	5	48.6	0.0	4.4	7.7
15	1000	1001	1000	5	62.6	0.0	3.2	27.6
16	1000	2001	2000	5	86.0	0.0	5.6	145.1

T2-1: Randomly generated monic polynomials. Continued in T2-2.

ID	F1	F2	F3	F4	F5	F6	F17	F18
1	10	10	100	5	18.4	2.8	4.4	0.3
2	10	10	500	5	41.2	0.0	4.2	0.2
3	10	10	1000	5	35.8	0.0	3.8	0.3
4	10	10	2000	5	53.0	0.0	3.8	0.6
5	10	101	100	5	25.2	5.8	4.4	1.1
6	10	501	500	5	49.2	0.0	4.4	6.5
7	10	1001	1000	5	58.8	0.0	4.8	22.7
8	10	2001	2000	5	70.0	0.0	5.6	102.7
9	1000	10	100	5	16.8	2.4	3.2	0.2
10	1000	10	500	5	30.2	0.0	4.4	0.2
11	1000	10	1000	5	41.4	0.0	4.2	0.3
12	1000	10	2000	5	35.6	0.0	3.2	0.5
13	1000	101	100	5	30.0	3.6	4.0	1.1
14	1000	201	200	5	44.6	1.2	4.4	2.6
15	1000	501	500	5	56.4	0.0	5.6	8.8
16	1000	1001	1000	5	64.4	0.0	5.6	30.3

T3-1: Special polynomials. Continued in T3-2.

ID	TYPE	F3	F4	F5	F6	F17	F18
1	CT	100	1	214	55	50	4.7
2	CT	500	1	1884	197	250	387.0
3	CT	1000	1	4014	371	500	3681.5
4	CT	1200	1	5534	415	600	7383.1
5	CU	100	1	209	51	50	4.6
6	CU	500	1	1635	200	250	373.3
7	CU	1000	1	4256	365	500	3997.9
8	CU	1200	1	6090	394	600	7834.9
9	L	100	1	302	97	100	7.7
10	L	500	1	2241	448	500	1098.3
11	L	900	1	4738	737	900	9592.0
12	L	1000	1	5450	815	1000	14822.9
13	M	100	1	238	1	4	0.6
14	M	300	1	704	0	4	9.4
15	M	400	1	936	0	4	24.7
16	M	600	1	1401	0	4	106.1
17	W	100	1	282	89	100	6.0
18	W	200	1	648	173	200	42.5
19	W	500	1	2422	430	500	1047.7
20	W	800	1	4407	647	800	6226.3
21	W	1000	1	5824	800	1000	15299.7

We collect the data on the numbers of loops and Taylor shifts because they are the most important factors for bisection methods. As in the literature, we also collect data on the terms, the degrees, the bits of coefficients, the average numbers of real roots and the timings. For our algorithm, the data on F7–F16 are also very important. A careful study and analysis on these data may help improve the algorithm. For the reason of concision, we list the data on F7–F16 in the appendix to this paper.

We try to compare our timings with those of the REL [12] and CF [2] methods but such comparison may be not suitable since the three implementations are realized by different languages[2] and the timings are obtained on different machines[3].

Compared with CF and REL.

		CF	REL	Trealroot
TYPE	F3	F18	F18	F18
ChebyShev	1000	2172	1305	3682
Laguerre	900	3790	2079	9592
Laguerre	1000	6210	3325	14823
Wilkinson	1000	256	815	15230
Mignotte	300	0.12	565	9.36
Mignotte	400	0.22	2421	24.7
Mignotte	600	0.54	>2h	106

Compared with the CF method[4].

				Trealroot	Trealroot	CF	CF
TYPE	F1	F2	F3	F17	F18	F17	F18
RANDOM	10	501	500	2.80	6.26	3.60	0.78
RANDOM	10	1001	1000	6.40	26.00	4.40	6.67
RANDOM	10	2001	2000	6.00	118.29	5.60	215.00
RANDOM	1000	501	500	4.40	7.74	3.20	0.56
RANDOM	1000	1001	1000	3.20	27.60	3.60	12.70
RANDOM	1000	2001	2000	5.60	145.09	6.00	329.00
RANDOM,MONIC	10	501	500	4.40	6.49	5.20	1.43
RANDOM,MONIC	10	1001	1000	4.80	22.68	4.80	7.12
RANDOM,MONIC	10	2001	2000	5.60	102.69	6.80	263.00
RANDOM,MONIC	1000	101	100	4.00	1.08	4.40	0.01
RANDOM,MONIC	1000	201	200	4.40	2.58	6.00	0.09
RANDOM,MONIC	1000	501	500	5.60	8.82	5.60	0.57
RANDOM,MONIC	1000	1001	1000	5.60	30.35	6.00	25.50

[2]We do not know exactly which languages REL and CF use but can guess they use C or C++ since REL may invoke MPFI package and CF is in the kernal of Mathematica.
[3]AMD Athlon 1GHz CPU and 1.5GB memory for REL, AMD Athlon 850MHz CPU and 256M memory for CF, and Pentium 1.13GHz CPU and 512M memory for `Trealroot`.
[4]We do not have corresponding data of REL.

Remark 4. The possible reason for CF to be very efficient on Wilkinson polynomials and Mignotte polynomials is that CF uses a method different from bisection.

Remark 5. We compute lower-bounds for real zeros in Step 4 of `Trealroot` as CF does [2]. Lower-bound skill suits monic polynomials which have very large real zeros as well as very small real zeros.

Now, let us analyze the performance of our program based on the above data. To simplify the description, we use T*i*-*j*-*k* to represent the row of table T*i*-*j* labeled by k, and T*i*-*j*-*k*-Fm to represent the value of Fm in T*i*-*j*-*k*.

First of all, we analyze the data on randomly generated polynomials (monic or not). Because the greatest average number of real roots (F17) is less than 6.5, the effect of F17 can be omitted.

- (F1 and F18). If the terms and the degrees are fixed, the effect of F1 (size of coefficients) on F18 (timings) is tiny. See, for example, T1-1-3-18 and T1-1-11-18 (or T2-1-3-18 and T2-1-11-18).
- (F2 and F18). If the degree and the size of coefficients are fixed, the timings (F18) increase linearly as the terms (F2) increase. More concretely, if the terms increase by k times, the timings increase almost by k times, too.
- (F3 and F18). If the terms and the size of coefficients are fixed, the effect of F3 (degree) on F18 (timings) is very little. See, for example, T1-1-1-F18, T1-1-2-F18, T1-1-3-F18 and T1-1-4-F18 (or T1-1-9-F18, T1-1-10-F18, T1-1-11-F18 and T1-1-12-F18).
- (F6 and F18). Taylor shifts are expensive when degrees of polynomials are large. If we can find a suitable way to control the number of Taylor shifts, we can get better timings. See, for example, T1-1-9 and T1-1-10 (or T2-1-9 and T2-1-10).

Second, because we use interval Newton iteration as the main steps in our algorithm, it is nature that the main factor affecting the efficiency of our algorithm is the number of real zeros of the input polynomial. One can see this point from the data in T3-1. To get more information on the relation between the number of real roots (F17) and the timing (F18), we isolate the real zeros of polynomials in the following form [8] by our program,

$$\prod_{i=1}^{r}(a_i x + b_i) \prod_{j=1}^{s} \left(c_j^2 x^2 - 2d_j c_j x + d_j^2 + e_j^2\right), \tag{5.1}$$

where a_i,b_i,c_j,d_j and e_j are random integers between $-2^{10}+1$ and $2^{10}-1$. The data on such polynomials of degree 1000 is reported in Table T4.

More or less surprisingly, the program is very fast on such polynomials. To compare the data in Table T4 with the data in [8] is not suitable because that work was carried out more than 10 years ago under a different environment[5] and only the data on the case $r+s=20$ was reported there. From Table T4, one

[5]SPARCstation 1+, with 64M memory and rated at 15.8 mips.

can see that the timing increases linearly as the number of real zeros increases, as showed in Figure 1.

T4: Polynomials with varying number of real roots.

r	s	F3	F4	F5	F6	F17	F18
0	500	1000	1	193	30	0	590
10	495	1000	1	275	42	10	789
50	475	1000	1	622	77	50	1456
100	450	1000	1	620	113	99	1886
200	400	1000	1	1415	203	200	3610
400	300	1000	1	3081	335	400	6518
800	100	1000	1	6975	624	798	12997

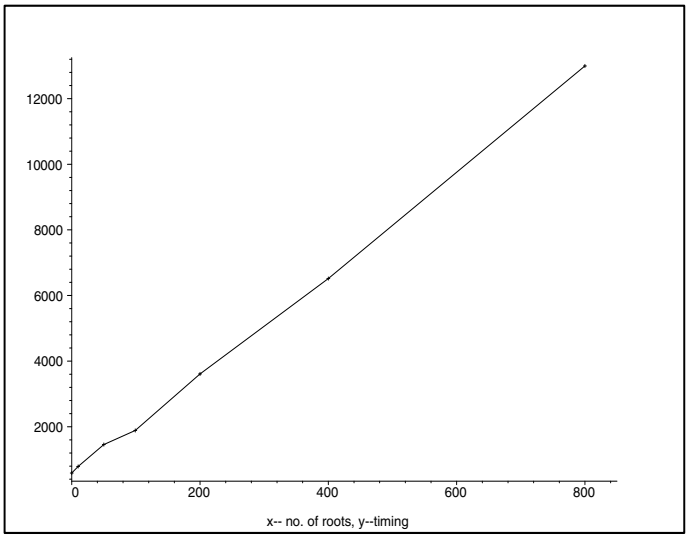

FIGURE 1

6. Conclusion

We propose a new algorithm, called `Trealroot`, for real root isolation, which is based on interval arithmetic and bisection. Generally speaking, for randomly generated polynomials, the current implementation of `Trealroot` is slower than the CF method for polynomials of degrees less than 1500 but is faster for those with higher degrees. Usually, polynomials generated randomly have few real roots as showed in Tables T1-1 and T2-1. To study the behavior of our method on random polynomials with many real roots, we apply `Trealroot` to polynomials defined by (5.1). From the data reported in Table T4, one can see that the timing

increases linearly as the number of real roots increases. Moreover, `Trealroot` is super efficient for sparse polynomials with large degrees and few real zeros. For example, our program isolates the three real roots of the polynomial defined by (3.1) within 350 seconds. Based on the data collected, we believe that the performance of `Trealroot` can be further improved by a new implementation.

Acknowledgements

The authors would like to thank the referees for their valuable comments and suggestions on an earlier version of this paper, which help improve the presentation of this paper greatly.

References

[1] A. G. Akritas, A. V. Bocharov, A. W. Strzeboński, *Implementation of real root isolation algorithms in Mathematica.* In: Abstracts of the International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (Interval'94), St. Petersburg, Russia, March 7–10, 23–27, 1994.

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

[3] G. Alefeld, J. Herzberger, *Introduction to interval computations.* Academic Press, New York, 1983.

[4] G. E. Collins, A. G. Akritas, *Polynomial real root isolation using Descartes' rule of signs.* In: Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computations, Yorktown Heights, N.Y., 272–275, 1976.

[5] G. E. Collins, J. R. Johnson, *Quantifier elimination and the sign variation method for real root isolation.* In: Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ACM Press, 264–271, 1989.

[6] G. E. Collins, R. Loos, *Real zeros of polynomials.* In: Computer Algebra: Symbolic and Algebraic Computation (B. Buchberger, G. E. Collins, R. Loos, eds.), Springer-Verlag, Wien, New York, 83–94, 1983.

[7] J. von zur Gathen, J. Gerhard, *Fast algorithms for Taylor shifts and certain difference equations.* In: Proceedings of ISSAC'97, Maui, Hawaii, 40–47, 1997.

[8] J. R. Johnson, *Algorithms for polynomial real root isolation.* Technical Report OSU-CISRC-8/91-TR21, Ohio State University, 1991. Also in: Quantifier Elimination and Cylindrical Algebraic Decomposition, (B. F. Caviness and J. R. Johnson, eds.), pp. 269–299, Springer 1998.

[9] J. R. Johnson, W. Krandick, *Polynomial real root isolation using approximate arithmetic.* In: Proceedings of ISSAC'97, Maui, Hawaii, 225–232, 1997.

[10] G. Mayer, *Epsilon-inflation in verification algorithms.* J. of Computational and Applied Mathematics **60** (1995), 147–169.

[11] F. Rouillier, personal communication, 2006.

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

Appendix

T1-2: Randomly generated polynomials.

ID	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
1	3.6	0.0	0.0	0.0	1.0	7.8	1.8	0.2	1.0	0.0
2	3.8	0.0	0.0	0.0	2.0	15.6	2.0	2.2	1.0	0.0
3	3.4	0.0	0.2	0.2	2.2	18.4	2.6	2.0	1.2	0.0
4	3.6	0.0	0.0	0.0	2.4	15.2	1.6	0.8	1.2	0.0
5	5.8	0.2	0.6	0.0	3.0	9.6	2.2	1.6	0.6	0.0
6	4.0	0.0	0.0	0.0	2.8	13.6	5.0	2.2	1.6	0.0
7	3.8	0.0	0.0	0.0	6.4	19.8	5.6	3.4	1.4	0.0
8	3.6	0.0	0.0	0.0	6.0	23.4	7.6	3.6	0.8	0.0
9	4.8	0.4	0.2	0.0	1.4	7.0	1.4	1.0	0.2	0.0
10	3.4	0.0	0.2	0.2	2.6	10.8	0.8	1.0	1.0	0.0
11	3.6	0.0	0.0	0.0	1.6	15.2	1.0	2.2	1.2	0.0
12	3.4	0.0	0.0	0.0	1.8	17.0	1.0	1.2	0.8	0.0
13	6.0	0.6	0.2	0.0	2.6	8.4	3.0	2.8	2.2	0.0
14	3.8	0.0	0.0	0.0	4.4	14.2	6.2	1.4	1.6	0.0
15	3.8	0.0	0.0	0.0	3.2	18.8	9.4	2.0	1.8	0.0
16	3.4	0.0	0.0	0.0	5.6	24.6	11.8	3.2	2.8	0.0

T2-2: Randomly generated monic polynomials.

ID	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
1	4.6	0.2	1.2	0.4	2.2	5.8	1.0	0.4	0.4	0.0
2	3.8	0.0	0.0	0.0	3.2	16.0	2.2	1.6	0.8	0.0
3	3.8	0.0	0.0	0.0	2.8	14.8	1.6	1.0	0.4	0.0
4	3.8	0.0	0.0	0.0	2.8	21.8	2.4	2.0	1.0	0.0
5	7.8	0.4	0.4	0.0	4.0	8.2	0.6	1.4	0.6	0.0
6	3.8	0.0	0.0	0.0	4.4	13.8	6.8	1.0	2.2	0.0
7	4.0	0.0	0.0	0.0	4.8	17.0	7.4	1.8	2.6	0.0
8	3.6	0.0	0.0	0.0	5.6	18.6	11.2	1.6	1.6	0.0
9	4.0	1.0	0.6	0.0	1.8	6.0	0.4	0.6	0.6	0.0
10	3.2	0.0	0.0	0.0	3.4	11.0	1.4	1.6	1.0	0.0
11	3.6	0.0	0.2	0.2	3.0	15.4	2.0	2.4	1.4	0.0
12	3.8	0.0	0.0	0.0	2.2	15.2	1.8	0.8	0.4	0.0
13	6.2	0.2	0.4	0.0	3.6	7.2	4.0	2.0	1.2	0.0
14	4.8	0.0	0.4	0.0	4.0	14.2	3.6	2.4	1.8	0.0
15	3.8	0.0	0.0	0.0	5.6	15.6	7.0	2.6	1.4	0.0
16	4.0	0.0	0.0	0.0	5.6	16.6	9.4	3.4	1.8	0.0

T3-2: Special polynomials.

ID	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
1	8	11	31	0	19	22	13	17	7	0
2	59	21	84	2	166	183	355	152	112	0
3	124	31	154	2	346	366	821	340	236	0
4	160	35	141	2	459	495	1225	449	373	0
5	11	9	30	0	20	16	20	15	6	0
6	58	23	94	0	155	154	276	134	99	1
7	129	29	153	2	347	389	901	346	270	0
8	156	27	123	4	477	578	1371	501	431	0
9	16	10	74	0	26	14	17	15	9	0
10	102	34	292	4	208	151	320	137	90	0
11	196	39	463	1	437	398	761	326	218	0
12	216	43	508	4	492	472	883	379	271	0
13	2	1	1	1	3	114	1	0	0	0
14	1	0	1	1	3	348	1	0	0	0
15	1	0	1	1	3	464	1	0	0	0
16	1	0	1	1	3	697	0	0	1	0
17	8	8	74	0	19	18	16	8	8	6
18	26	9	136	2	56	34	66	30	16	7
19	90	34	281	1	210	162	397	157	99	8
20	164	38	404	4	386	337	783	295	212	9
21	212	53	471	7	519	493	1009	423	301	9

Ting Zhang
Accounting Centre of China Aviation
Beijing 100028
China
e-mail: ztfriend@tom.com

Bican Xia
LMAM & School of Mathematical Sciences
Peking University
Beijing 100871
China
e-mail: xbc@math.pku.edu.cn

Received: December 1, 2006.
Accepted: July 27, 2007.