J Syst Sci Complex (2019) 32: 1767–1782

Logcf: An Efficient Tool for Real Root Isolation*

DAI Liyun · FAN Zhe · XIA Bican · ZHANG Hanwen

DOI: 10.1007/s11424-019-7361-7
Received: 1 November 2017 / Revised: 14 October 2018

Abstract Computing upper bounds of the positive real roots of some polynomials is a key step of those real root isolation algorithms based on continued fraction expansion and Vincent's theorem. The authors give a new algorithm for computing an upper bound of positive roots in this paper. The complexity of the algorithm is $O(n\log(u+1))$ additions and multiplications where u is the optimal upper bound satisfying Theorem 3.1 of this paper and n is the degree of the polynomial. The method together with some tricks have been implemented as a software package logcf using `C` language. Experiments on many benchmarks show that logcf is competitive with RootIntervals of Mathematica and the function realroot of Maple averagely and it is much faster than existing open source real root solvers in many test cases.

Keywords Computer algebra, continued fractions, real root isolation, univariate polynomial, vincent's theorem.

1 Introduction

Real root isolation of univariate polynomials with integer coefficients is one of the fundamental tasks in computer algebra as well as in many applications ranging from computational geometry to quantifier elimination. The problem can be stated as: Given a polynomial $p \in \mathbb{Z}[x]$, computing for each of its real roots an interval with rational endpoints containing it and being disjoint from the intervals computed for the other roots. There are three kinds of methods to isolate real roots. The first kind consists of the subdivision algorithms using counting techniques based on, e.g., the Sturm theorem or Descartes' rule of signs. This method counts the sign changes (of Sturm sequence or coefficients of some polynomials) in the considered interval and if the sign changes reach 1 or 0, the procedure returns from this interval. Otherwise it subdivides the interval and computes recursively. The symbolic implementations of this method

DAI Liyun (Corresponding author)
RISE & School of Computer and Information Science, Southwest University, Chongqing 400700*, China.*
Email: dailiyun@swu.edu.cn.
FAN Zhe · XIA Bican · ZHANG Hanwen
LMAM & School of Mathematical Science, Peking University, Beijing 100871*, China.*
*The research was supported by the National Science Foundation of China under Grant Nos. 61802318, 61732001 and 61532019.
◇ *This paper was recommended for publication by Editor LI Hongbo.*

can be found in [1–4] and the symbolic-numberic algorithms implementations can be found in [3, 5–7].

The second kind of methods take use of the continued fraction algorithms[8–10]. These methods are highly efficient and competitive[3, 11]. Especially, [11] provides a test dataset consisting of 5000 polynomials from many different settings. And its results indicate that there is no best method overall.

The third method is based on Newton-Raphson method and interval arithmetic. The search space is subdivided until it contains only a single real root and Newton's method converges. When the polynomial is sparse and has very high degree, this method will be much faster than other methods. The symbolic implementations of this kind of methods can be found in [12, 13] and the numeric implementations can be found in [14, 15].

Those methods which are based on continued fraction compute the continued fraction expansion of the real roots of a polynomial in order to compute isolating intervals for real roots. One important step is to compute the upper bounds of the positive real roots for some polynomials. There are several classic methods to compute such upper bounds, such as Cauchy's bound, Lagrange-MacLaurin's bound and Kioustelidis' bound. And there are many recent works about the upper bound of the positive roots of univariate polynomials[8, 16, 17]. Some methods for computing these bounds, such as Cauchy's bound, are of $O(n)$ complexity but the results are very coarse. Some methods, as presented in [8], are of $O(n^2)$ complexity but their bounds are sharper. The balance between the precision and efficiency for computing such upper bounds has to be taken into account.

1.1 Our Contribution

We provide a new method for computing such bounds with time complexity $O(n \log(u+1))$ where u is the optimal upper bound satisfying Theorem 3.1. Besides, compared with [8], when Algorithm 4 returns true (the upper bound is less than 1), our upper bound is at most two times that in [8]. In this way, the algorithm of isolating real roots is improved. Our method has been implemented as a software package logcf using `C` language. For many benchmarks logcf is about four times faster than the function realrootof Maple. Roughly speaking, logcf is competitive with RootIntervals of Mathematica. In some test cases logcf is faster than RootIntervals but in some other cases RootIntervals is faster than logcf and the mean time of all test cases is almost the same. But we have an interesting finding that RootIntervals may output wrong results on some input polynomials due to incorrect zero judgement. And in general logcf is also much faster than other state of the art open source exact real root isolation solvers, such as CF, ANewDsc and SLV. For those benchmarks which have only real roots, logcf is much faster than Sleeve and eigensolve which are based on numerical computation.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we review the main algorithm for real root isolation based on continued fractions. We present some theoretical results about upper bounds of positive roots in Section 3. In Section 4, we provide a new algorithm for computing an upper bound of positive roots and we also list some tricks used in logcf. In Section 5, we list the comparative experimental results of our algorithm and other software.

2 Algorithm Based on Continued Fraction

In this section, we first recall Descartes' rule of signs, which gives a bound on the number of positive real roots. Then the Vincent theorem, which ensures the termination of algorithms based on continued fractions, is presented. Finally, we review an algorithm of real root isolation based on continued fractions.

As usual, $\deg(p)$ denotes the degree of univariate polynomial p. The derivative of polynomial p with respect to the only variable is denoted by p' and $\gcd(f,g)$ means the greatest common divisor of polynomials f and g.

Example 2.1 Consider the polynomial

$$\begin{aligned}&p_1(x) = x^6 + 2x^5 - 4x^4 + x^3 + 10x^2 - 5x + 5,\\ &\deg(p_1) = 6, \quad p_1' = 6x^5 + 10x^4 - 16x^3 + 3x^2 + 20x - 5.\end{aligned}$$

Notation 1 (Sign variation) Let $S = \{a_0, a_1, \cdots, a_n\}$ be a finite sequence of non-zero real numbers. Define $V(S)$, the sign variation of S, as follows.

$$\begin{aligned}&V(S) = 0 \ \text{ if } |S| \le 1,\\ &V(a_0, \cdots, a_{n-1}, a_n) = \begin{cases} V(a_0, \cdots, a_{n-1}) + 1, & \text{if } a_{n-1}a_n < 0;\\ V(a_0, \cdots, a_{n-1}), & \text{otherwise.}\end{cases}\end{aligned}$$

If some elements of S are zero, remove those zero-elements to get a new sequence and define $V(S)$ to be the sign variation of this new sequence.

Theorem 2.2 (Descartes' rule of signs) *Suppose $p = \sum_{i=0}^{n} a_i x^i \in \mathbb{R}[x]$ has m positive real roots, counted with multiplicity. Set $V(p) = V(a_0, a_1, \cdots, a_n)$. Then $m \le V(p)$, and $V(p) - m$ is even.*

Theorem 2.3 (Vincent's theorem) *Let $p(x)$ be a real polynomial of degree n which has only simple roots. It is possible to determine a positive quantity δ so that for every pair of positive real numbers a and b with $|b - a| < \delta$, the coefficients sequence of every transformed polynomial of the form $p(x) = (1 + x)^n p(\frac{a+bx}{1+x})$ has exactly 0 or 1 sign variation. The second case is possible if and only if $p(x)$ has a single root within (a, b).*

Definition 2.4 We define the following transformations for a univariate polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0, n > 0$.

$$\begin{aligned}&lc(p(x)) = a_n,\\ &R(p(x)) = x^n\left(p\left(\frac{1}{x}\right)\right),\\ &H_\lambda(p(x)) = p(\lambda x),\\ &T(p(x)) = p(x+1),\\ &D(p(x)) = a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_2 x + a_1.\end{aligned}$$

$T(p)$ is also called Taylor shift one[18, 19].

Algorithm 1: loglb

Input: $p \in \mathbb{Z}[x]$, $lc(p) \neq 0$.
Output: *root_lb*, a lower bound of positive roots of p.
$p = R(p)$;;
if $lc(p) < 0$ **then** $p = -p$; *root_lb* =logup(p); `/* logup is described as Algorithm 5 */`

Vincent's theorem provides a possible method to isolate the real roots. But it needs to divide $\mathbb{R}$ into many small intervals and the width of these intervals is smaller than a given finite value. So Vincent's theorem cannot be used to isolate the real roots directly since $\mathbb{R}$ is infinite. For employing Vincent's theorem, we need to discard the intervals containing no real roots. Algorithm 1 provides a lower bound lb of positive roots for given p. Since the interval $(0, lb)$ contains no real roots, we can safely discard $(0, lb)$ when isolate the real roots.

Definition 2.5

$$intvl(a, b, c, d) = \begin{cases} \left(\min\left\{\dfrac{a}{c}, \dfrac{b}{d}\right\},\ \max\left\{\dfrac{a}{c}, \dfrac{b}{d}\right\}\right), & \text{if } cd \neq 0; \\ (0, \infty), & \text{otherwise.} \end{cases}$$

Using the above notations and definitions, an algorithm for isolating all the real roots of a nonzero univariate polynomial is described as Algorithm 2. Algorithm 3, which is a slight modification of the algorithm in [8], is presented here to make our subsequent description clearer.

Continued fractions based procedures will continue subdividing the considered interval into two subintervals and make a one to one map from (a, b) to $(0, +\infty)$ by $p(x) = (1 + x)^n p(\frac{a+bx}{1+x})$ until $V(p)$ equals 1 or 0. Informally, Algorithm 2 is employing the above map to magnify the considered interval to $(0, +\infty)$. Through Descartes' rule of signs, there is no positive real roots if $V(p) = 0$. In this case we delete the interval from the considered interval set. When $V(p) = 1$, this interval contains exact one real root by Descartes' rule of signs. In this case we throw away the interval from the considered interval set. When $V(p) > 1$, since it cannot be determined how many real roots are contained in this interval, we divide the considered interval into two subintervals. In this case we throw away the interval from the considered interval set and add two subintervals to the considered interval set. Repeating this procedure, we can divide the interval into subintervals until there is no width of the corresponding original interval greater or equal to δ in considered interval set. Then there is at most one real root in every interval in the considered interval set and then the procedure will terminate. This is also the reason why Theorem 2.3 can guarantee the termination of Algorithm 2.

Algorithm 2: main

Input: A non-zero polynomial $p(x) \in \mathbb{Z}[x]$.
Output: I, a set of real root isolating intervals of $p(x)$.
$I = \emptyset$; ;
if deg(p) equals to 0 **then**
  return I;
end
$p = \frac{p}{\gcd(p,p')}$; `/* square free */`
if $p(0)$ equals to 0 **then**
  I.add([0, 0]); `/* add` $[0,0]$ `to set` I `*/`
  $p = D(p)$; `/* derivative of` p `*/`
end
I.addAll(cf(p)); ;
`/* add all the positive root intervals to set` I `*/`
`/* cf is described as Algorithm 3 */`
$p = p(-x)$;;
I.addAll(-cf(p));

Algorithm 3: cf

.
Input: A squarefree polynomial $p \in \mathbb{Z}[x] \setminus \{0\}$.
Output: *roots*, a list of isolating intervals of positive roots of p.
$roots = \emptyset$; $s = V(p)$;
$intstack = \emptyset$; $intstack$.add($\{1, 0, 0, 1, p, s\}$); **while** $intstack \neq \emptyset$ **do**
  $\{a, b, c, d, p, s\} = intstack$.pop();`/* pop the first element */`
  $\lambda = loglb(p)$; **if** $\lambda \geq 1$ **then** $\{a, c, p\} = \{\lambda a, \lambda c, H_\lambda(p)\}$;
  $\{b, d, p\} = \{a + b, c + d, T(p)\}$; **if** $p(0) == 0$ **then** $roots$.add($[\frac{b}{d}, \frac{b}{d}]$); $p = \frac{p}{x}$;
  $s = V(p)$; **if** $s == 0$ **then** continue; **else if** $s == 1$ **then** $roots$.add($intvl(a, b, c, d)$);
  continue; $\{p_1, a_1, b_1, c_1, d_1, r\} = \{T(p), a, a + b, c, c + d, 0\}$
  if $p_1(0) == 0$ **then** $roots$.add($[\frac{b_1}{d_1}, \frac{b_1}{d_1}]$); $p_1 = \frac{p_1}{x}$; $r = 1$; $s_1 = V(p_1)$;
  $\{s_2, a_2, b_2, c_2, d_2\} = \{s - s_1 - r, b, a + b, d, c + d\}$; **if** $s_2 > 1$ **then**
  $p_2 = (x + 1)^{\deg(p)} T(p)$; **if** $p_2(0) == 0$ **then** $p_2 = \frac{p_2}{x}$; $s_2 = V(p_2)$; **if** $s_1 == 1$ **then**
  $roots$.add($intvl(a_1, b_1, c_1, d_1)$); **else if** $s_1 > 1$ **then**
  $intstack$.add($\{a_1, b_1, c_1, d_1, p_1, s_1\}$);
  if $s_2 == 1$ **then** $roots$.add($intvl(a_2, b_2, c_2, d_2)$); **else if** $s_2 > 1$ **then**
  $intstack$.add($\{a_2, b_2, c_2, d_2, p_2, s_2\}$).
end

3 A New Upper Bound of Positive Real Roots for Polynomials

One key ingredient of continued fractions based methods is the computation of the positive real roots' upper bounds. We give in Theorem 3.1 a new characteristic of such upper bounds of univariate polynomials.

Theorem 3.1 *Suppose $p = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ $(a_n > 0)$ is a univariate polynomial in x with real coefficients. Then a nonnegative number u is an upper bound of positive roots of p if u satisfies that $\sum_{i=j}^{n} a_i u^{i-j} \geq 0$ for $j = 0, 1, \cdots, n$.*

Proof If $n = 0$, then p is a nonzero constant and any positive number is its upper bound of positive roots.

Otherwise, if $b > u$, we claim that $\sum_{i=j}^{n} a_i b^{i-j} > \sum_{i=j}^{n} a_i u^{i-j}$ for any $j = 0, 1, \cdots, n-1$.

When $j = n-1$, $\sum_{i=n-1}^{n} a_i b^{i-n+1} - \sum_{i=n-1}^{n} a_i u^{i-n+1} = a_n(b-u) > 0$. Thus, the claim holds.

Assume the claim holds when $j = k$. When $j = k-1$, $\sum_{i=k-1}^{n} a_i b^{i-k+1} = \left(\sum_{i=k}^{n} a_i b^{i-k}\right) b + a_{k-1}$. By assumption, $\sum_{i=k}^{n} a_i b^{i-k} > \sum_{i=k}^{n} a_i u^{i-k} \geq 0$. Since $b > u \geq 0$, $\left(\sum_{i=k}^{n} a_i b^{i-k}\right) b > \left(\sum_{i=k}^{n} a_i u^{i-k}\right) u$ and $\sum_{i=k-1}^{n} a_i b^{i-k+1} > \sum_{i=k-1}^{n} a_i u^{i-k+1}$. So $\sum_{i=j}^{n} a_i b^{i-j} > \sum_{i=j}^{n} a_i u^{i-j}$ for any $j = 0, 1, \cdots, n-1$.

By the above claim, $p(b) = \sum_{i=0}^{n} a_i b^i > 0$ when $b > u$. Because b is arbitrarily chosen, u is an upper bound of the positive roots of p.

For Example 2.1,

- $\sum_{i=6}^{6} a_i x^{i-j}$ is 1.
- $\sum_{i=5}^{6} a_i x^{i-j}$ is $x + 2$ and x is positive for $x \geq 0$.
- $\sum_{i=4}^{6} a_i x^{i-j}$ is $x^2 + 2x - 4$ and its two real roots are $-1 - \sqrt{5}$ and $-1 + \sqrt{5}$.
- $\sum_{i=3}^{6} a_i x^{i-j}$ is $x^3 + 2x^2 - 4x + 1$ and its largest real root is 1.
- $\sum_{i=2}^{6} a_i x^{i-j}$ is $x^4 + 2x^3 - 4x^2 + x + 10$ and its largest real root is less than 1.
- $\sum_{i=1}^{6} a_i x^{i-j}$ is $x^5 + 2x^4 - 4x^3 + x^2 + 10x - 5$ and its largest real root is less than 1.
- $\sum_{i=0}^{6} a_i x^{i-j}$ is $x^6 + 2x^5 - 4x^4 + x^3 + 10x^2 - 5x + 5$ and its largest real root is less than 1.

Hence, $\sum_{i=j}^{6} a_i u^{i-j} \geq 0, j = 0, 1, \cdots, 6$ when $u = -1 + \sqrt{5}$. By Theorem 3.1, $-1 + \sqrt{5}$ is one upper bound of positive roots of p_1.

The following theorem was given by Akritas, et al. in [8], which computes positive root upper bounds of univariate polynomials.

Theorem 3.2 (see [8]) *Let $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ $(a_n > 0)$ be a polynomial with real coefficients and let $d(p)$ and $t(p)$ denote the degree and the number of its terms, respectively.*

Moreover, assume that $p(x)$ can be written as

$$p(x) = q_1(x) - q_2(x) + q_3(x) - q_4(x) + \cdots + q_{2m-1}(x) - q_{2m}(x) + g(x), \tag{1}$$

where all the coefficients of polynomials $q_i(x)$ $(i = 1, 2, \cdots, 2m)$ *and* $g(x)$ *are positive. In addition, assume that for* $i = 1, 2, \cdots, m$ *we have*

$$q_{2i-1}(x) = c_{2i-1,1}x^{e_{2i-1,1}} + \cdots + c_{2i-1,t_{2i-1}}x^{e_{2i-1,t_{2i-1}}}$$

and

$$q_{2i}(x) = b_{2i,1}x^{e_{2i,1}} + \cdots + b_{2i,t_{2i}}x^{e_{2i,t_{2i}}},$$

where $e_{2i-1,1} = d(q_{2i-1})$, $e_{2i,1} = d(q_{2i})$, $t_{2i-1} = t(q_{2i-1})$, *and* $t_{2i} = t(q_{2i})$ *and the exponent of each term in* $q_{2i-1}(x)$ *is greater than the exponent of each term in* $q_{2i}(x)$. *If for all indices* $i = 1, 2, \cdots, m$, *we have*

$$t(q_{2i-1}) \geq t(q_{2i}),$$

then an upper bound of the values of the positive roots of $p(x)$ *is given by*

$$up = \max_{i=1,2,\cdots,m} \left\{ \max_{j=1,2,\cdots,t_{2i}} \left\{ \left(\frac{b_{2i,j}}{c_{2i-1,j}} \right)^{\frac{1}{e_{2i-1,j} - e_{2i,j}}} \right\} \right\} \tag{2}$$

for any permutation of the positive coefficients $c_{2i-1,j}$, $j = 1, 2, \cdots, t_{2i-1}$. *Otherwise, for each of the indices* i *for which we have*

$$t_{2i-1} < t_{2i},$$

we break up one of the coefficients of $q_{2i-1}(x)$ *into* $t_{2i} - t_{2i-1} + 1$ *parts, so that now* $t(q_{2i}) = t(q_{2i-1})$ *and apply the same formula* (2) *given above.*

For Example 2.1 we have

$$q_1 = x^6 + 2x^5, \quad -q_2 = -4x^4, \quad q_3 = x^3 + 10x^2,$$
$$-q_4 = -5x, \quad q_5 = 5.$$

A direct application of Theorem 3.2 pairs the terms $\{x^6, -4x^4\}$ of $q_1(x)$ and $q_2(x)$, and ignores the last term of $q_1(x)$. For $q_3(x)$ and $q_4(x)$, application of Theorem 3.2 pairs the terms $\{x^3, -5x\}$ of them, and ignores the last term of $q_3(x)$. The resulting upper bound is $\sqrt{5}$. As $\sqrt{5} > -1+\sqrt{5}$, for Example 2.1, the upper bound given by Theorem 3.2 is greater than the upper bound given by Theorem 3.1. For the upper bound, the value is better if the estimated upper bound is smaller. So in this case, Theorem 3.1 is better than Theorem 3.2. In general, we shall show in Theorem 3.3 that the optimal bound given by Theorem 3.1 is better than that given by Theorem 3.2.

Theorem 3.3 *Let* $p(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_0$ $(a_n > 0)$ *be a polynomial with real coefficients. If* u *is an upper bound of positive roots of* p *obtained by Theorem* 3.2, *then* $\sum_{i=j}^{n} a_iu^{i-j} \geq 0$ *for* $j = 0, 1, \cdots, n$.

Proof For every $a_i < 0$, by Theorem 3.2, there exist $c_{i_1}x^{e_{i_1}}$ and $b_{i_2}x^{e_{i_2}}$, respectively, such that $e_{i_1} > e_{i_2}$, $b_{i_2}x^{e_{i_2}}$ is the term $-a_ix^i$ and $c_{i_1}x^{e_{i_1}}$ is either a whole or a part (broken up by Theorem 3.2) of a positive term of p. Since u satisfies the equation (2), $c_{i_1}u^{e_{i_1}} \geq b_{i_2}u^{e_{i_2}}$. So $p(x)$ can be written as $p(x) = \sum(c_{i_1}x^{e_{i_1}} - b_{i_2}x^{e_{i_2}}) + g(x)$, where $c_{i_1} > 0, b_{i_2} > 0, e_{i_1} > e_{i_2}, e_{i+1_1} \geq e_{i_1}$

and all the coefficients of polynomial $g(x)$ are positive. So for every $a_j > 0$, the sum of coefficient c_{i_1} where $e_{i_1} = j$ and the terms of $c_{i_1}x^{e_{i_1}}$ has a corresponding $b_{i_2}x^{e_{i_2}}$ is less or equal than a_j. In other words, $\left(\sum_{a_i<0,e_{i_1}=j} c_{i_1}\right) \leq a_j$ for every $a_j > 0$. So, if $u \geq 0$, then

$$\sum_{i=k}^{n} a_i u^i \geq \sum_{i=k,a_i<0,a_ix^i=-b_{i_2}x^{e_{i_2}}}^{n} (c_{i_1}u^{e_{i_1}} - b_{i_2}u^{e_{i_2}})$$

for any $k = 0, 1, \cdots, n$. Since $c_{i_1}u^{e_{i_1}} \geq b_{i_2}u^{e_{i_2}}$, $\sum_{i=k}^{n} a_i u^i \geq 0$ for $k = 0, 1, \cdots, n$. Since $u \geq 0$, $\sum_{i=k}^{n} a_i u^{i-k} \geq 0$ for any $k = 0, 1, \cdots, n$. ∎

4 A New Algorithm of Computing Upper Bounds

Theorem 3.1 is a good theoretical result, it can easily guarantee a value is an upper bound for a given polynomial. However it is difficult to directly use it for computing the optimal upper bound. So, in this section we will provide a new algorithm for computing upper bounds. The correctness of this new algorithm is guaranteed by the theoretical results in Section 3. And this algorithm can easily output an upper bound which is at most two times greater than the optimal result of Theorem 3.1.

Algorithm 4 is based on Theorem 3.1 to check whether 1 is an upper bound for a given polynomial.

We explain the correctness of Algorithm 4 as follow. When the algorithm reaches line 6, we have

$$\underbrace{a_n}_{+}, \underbrace{\underbrace{a_{n-1}}_{+_0}}_{i}, \underbrace{\cdots}_{+_0,\cdots,+_0}, \underbrace{a_{start+1}}_{+}, \underbrace{\underbrace{a_{start}}_{-}}_{j}, \underbrace{a_{start-1}}_{*}, \underbrace{\cdots}_{*,\cdots,*}, \underbrace{a_{lastNeg+1}}_{*}, \underbrace{a_{lastNeg}}_{-}, \underbrace{a_{lastNeg-1}}_{+_0}, \underbrace{\cdots}_{+_0,\cdots,+_0},$$

where "+" means the corresponding a_k is positive, "−" means the corresponding a_k is negative, "$*$" means the sign of the corresponding a_k is unknown and "$+_0$" means the corresponding a_k is nonnegative. The loop between lines 4 and 4 shifts i and j right until one of them reaches value $lastNeg-1$. If j reaches value $lastNeg-1$ at line 14, it is easy to check that $i \geq j \wedge cfSum \geq 0$ always holds in this iteration. In other words, $\sum_{k=l}^{n} a_k \geq 0$ for $l = lastNeg, \cdots, n$. As $a_k > 0$ for $k = 0, 1, \cdots, lastNeg - 1$, $\sum_{k=l}^{n} a_k \geq 0$ for $l = 0, 1, \cdots, n$. By Theorem 3.1, 1 is an upper bound of the positive real roots of p.

In Example 2.1, at the beginning, $cfSum = 1$ and the values of i, j, $lastNeg$ are as follows:

$$1, \underbrace{2}_{i=5}, \underbrace{-4}_{j=4}, 1, 10, \underbrace{-5}_{lastNeg=1}, 5.$$

After executing the main loop of Algorithm 4 between line 4 and line 4, the values will become $cfSum = -3$,

$$1, \underbrace{2}_{i=5}, -4, \underbrace{1}_{j=3}, 10, \underbrace{-5}_{lastNeg=1}, 5.$$

After the second iteration, the values will become $cfSum = -1$,

$$1, 2, \underbrace{-4}_{i=4}, \underbrace{1}_{j=3}, 10, \underbrace{-5}_{lastNeg=1}, 5.$$

Hence, during the third iteration, the algorithm will return at line 4. This means 1 is not an upper bound of positive real roots of p_1.

Algorithm 4: lessOne

Input: $p = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \in \mathbb{Z}[x], a_n > 0,\ \exists a_i, a_n a_i < 0.$
Output: true: the positive root bound of p must be less than 1;
false: cannot determine whether the bound is less than 1.

```
start = n − 1; lastNeg = 0;
while a_lastNeg ≥ 0 do
    lastNeg = lastNeg + 1;
end
while a_start ≥ 0 do
    start = start − 1;
end
cfSum = a_n; i = n − 1; j = start;  while i ≥ lastNeg − 1and j ≥ lastNeg − 1  do
    if cfSum < 0 then
        while i > jand a_i ≤ 0 do
            i = i − 1;
        end
        if i == j then  return false;  cfSum = cfSum + a_i; i = i − 1;
    end
    else
        if j == lastNeg − 1 then  return  true;   while j ≥ lastNeg and a_j ≥ 0 do
            j = j − 1;
        end
        cfSum = cfSum + a_j; j = j − 1;
    end
end
return true.
```

It is difficult to compute the optimal value which satisfies Theorem 3.1. Hence, we provide Algorithm 5 based on the idea of dichotomic search to find a value which is close enough to the optimal value. The correctness of Algorithm 5 is directly based on Theorem 3.1. Theorem 3.1 needs a value u satisfying that $\sum_{i=j}^{n} a_i u^{i-j} \geq 0$ for $j = 0, 1, \cdots, n$. Since the resulting λ of Algorithm 5 satisfies that $\sum_{i=j}^{n} a_i \lambda^{i-j} \geq 0, j = 0, 1, \cdots, n$, λ is an upper bound for the given polynomial. Moreover, by Theorem 4.1, λ is at most two times greater than the optimal upper bound which satisfies Theorem 3.1.

Theorem 4.1 *Let* $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ $(V(p) > 0)$ *be a polynomial with real coefficients. Let* u *denote the output of Algorithm* 5 *and* u_1 *denote the optimal upper bound of* p *satisfying Theorem* 3.1. *When* u *is less than or equal to* 1, $u < 2u_1$.

Proof In Algorithm 5, if $\frac{1}{2^{base}} \geq u_1$, then $\sum_{i=j}^{n} a_i \left(\frac{1}{2^{base}}\right)^{i-j} \geq 0$ for $j = 0, 1, \cdots, n$ by the proof of Theorem 3.1. Thus the loop does not terminate at this step. So when Algorithm 5 returns, *base* must satisfy $\frac{1}{2^{base}} < u_1$. Therefore, the output $u = \frac{1}{2^{base-1}}$ and $u < 2u_1$. Obviously, this algorithm will terminate.

Furthermore, $\sum_{i=j}^{n} a_i \left(\frac{1}{2^{base-1}}\right)^{i-j} \geq 0$ for $j = 0, 1, \cdots, n-1$ by Theorem 3.1. Hence, $u = \frac{1}{2^{base-1}}$ is an upper bound of p. ∎

Corollary 4.2 *Let* $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ $(V(p) > 0)$ *be a polynomial with real coefficients. Set* u *to be the optimal upper bound of positive roots of* p *satisfying Theorem* 3.1. *Then Algorithm* 5 *costs at most* $O(n \log(u + 1))$ *additions and multiplications.*

5 Experiments

5.1 Environment

We first explain the implementation and computation environment. We compare logcf with SLV, ANewDsc, Mathematica's† RootIntervals and Maple's‡ realroot on a 64-bit Intel(R) Core(TM) i7 CPU-4710Q @ 2.50GHz with 8GB RAM memory and Windows 7. In this environment logcf was compiled by visual studio 2013.

5.2 Tricks

Variable substitution If $p(x) \in \mathbb{Z}[x]$ and $p(x) = p_1(x^k)$ $(k > 1)$, then substitute $y = x^k$ in p. Obviously, $\deg(p_1, y) = \frac{\deg(p,x)}{k}$. We first isolate the real roots of p_1 then obtain the real roots of p. Using this trick, we can greatly reduce the running time of *ChebyshevT* and *ChebyshevU* when each term of the polynomials is of even degree. The same trick was also taken into account in [20].

Incomplete termination check If $p(x) \in \mathbb{Z}[x]$ and $V(p) = 2$, we may try to check whether the sign of $p(1)$ is the same as the sign of the leading coefficient of p. If they are not the same, then p has one positive root in $(0, 1)$ and the other one in $(1, +\infty)$. So, we can terminate this subtree. Since the whole logcf procedure is a tree and logcf spends more than 90 percent of the total time on computing $T(p)$, this trick may improve the efficiency of the algorithm greatly.

5.3 Benchmarks

5.3.1 W_n

Wilkinson polynomials: $W_n = \Pi_{i=1}^{n}(x - i)$.

5.3.2 mW_n

Modified Wilkinson polynomials: $mW_n = W_n - 1$.

If $n > 10$, mW_n has n simple real roots but most of them are irrational.

†11.1 version

‡Maple(TM) 2017,Windows(R) (64-bit)

Algorithm 5: logup

Input: $p = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 \in \mathbb{Z}[x], a_n > 0,\ \exists a_i, a_na_i < 0.$

Output: an upper bound of the positive roots of p.

$start = n-1$; $lastNeg = 0$; $base = 1$;

if $\neg$*lessOne(p)* **then** return 2; `/* 2 is a special value which leads Algorithm 3 to run line 3 branch. */`

while $a_{lastNeg} \geq 0$ **do** $lastNeg = lastNeg + 1$; **while** $a_{start} \geq 0$ **do** $start = start - 1$;

$i = n$; **while** $i == n$ **do**

  $i = n-1$; $j = start$; $cfSum = a_n$;;

  while $i \geq lastNeg - 1$ *and* $j \geq lastNeg - 1$ **do**

    if $cfSum < 0$ **then**

      while $i > j$ *and* $a_i \leq 0$ **do**

        $i = i - 1$;

      end

      if $i == j$ **then** break; $cfSum = cfSum + a_i2^{(n-i)base}$; `/* As in symbolic computation division is slower than multiplication, we check wether` $2^np(x)$ `is greater than 0 instead of` $p(x)$`. */`

      $i = i - 1$;

    end

    else

      if $j == lastNeg - 1$ **then** $j = lastNeg - 2$;`/*` $lastNeg - 2$ `is a special value which leads program to run line 5 branch. */`

      break;

      while $j \geq lastNeg$ *and* $a_j \geq 0$ **do**

        $j = j - 1$;

      end

      $cfSum = cfSum + a_j2^{(n-j)base}$; $j = j - 1$;

    end

  end

  if $j == lastNeg - 2$ **then**

    $base = base + 1$; $i = n$;

  end

end

return $\frac{1}{2^{base-1}}$.

5.3.3 WP_n

Wilkinson-like polynomials: WP_n is polynomial which convert $\Pi_{i=1}^{n}(x-\frac{i}{n-i})$ to polynomial with integer coefficients.

5.3.4 IW_n

The distance between W_n's two nearest real roots is 1 and the distance between mW_n's two nearest real roots is nearly 1. We construct new polynomials $IW_n = \Pi_{i=1}^{n}(ix-1)$, which have a completely different nearest distance.

5.3.5 mIW_n

We modify IW_n into $mIW_n = IW_n - 1$ for the same purpose as we construct mW_n. Most real roots of mIM_n become irrational.

5.3.6 T_n

ChebyshevT polynomials: $T_0 = 1, T_1 = x, T_{n+1} = 2xT_n - T_{n-1}$. T_n has n simple real roots.

5.3.7 U_n

ChebyshevU polynomials: $U_0 = 1, U_1 = 2x, U_{n+1} = 2xU_n - U_{n-1}$. U_n has n simple real roots.

5.3.8 L_n

Laguerre polynomials: $L_0 = 1$,$L_1 = 1 - x$,$L_{n+1}(x) = \frac{(2n+1-x)L_n(x)-nL_{n-1}(x)}{(n+1)}$. Obviously, $n!L_n$ is a polynomial with integer coefficients.

5.3.9 LP_n

Legendre polynomials of the first kind. $LP_1 = 1$ and $LP_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n$.

5.3.10 M_n

Mignotte polynomials: $x^n - 2(5x-1)^2$. If n is odd, M_n has three simple real roots. If n is even, it has four simple real roots.

5.3.11 MR_n

Mignotte rational center polynomials: $x^n - ((2^7-1)x-1)^2$.

5.3.12 H_n

Hermite polynomials: $H_0 = 1, H_1 = 2x, H_n = 2xH_{n-1} - 2(n-1)H_{n-2}$. H_n has n simple real roots.

5.3.13 MI_n

Mignotte irrational center polynomials: $x^{129} - ((2^{\frac{1}{4}n}-1)x^2-1)^2$.

5.3.14 $R(n,b,r)$

Randomly generated polynomials: $R(n,b,r)$=$a_nx^n + \cdots + a_1x + a_0$ with $|a_i| \leq b, Pr[a_i \geq 0] = \frac{1}{2}$ and $Pr[a_i \neq 0] = 1-r$, where Pr means probability. For each setting (n,b,r), we generate randomly five instances and compute the mean of five running times. The degree of random cases are between 10 to 1500, the coefficients belong to $[-17951, 17951]$ and the value of r belongs to $\{0.1, 0.2, \cdots, 0.9\}$.

5.4 Results

We convert all input polynomials to squarefree before isolating their real roots. And the converting time is not included in the following timings.

As a built-in Maple function, realroot is compared with our tool logcf. The Maple we use has a version number 2017. For almost all benchmarks, our software logcf can be four times faster than realroot. The comparative data can be found in Figure 1 and Table 1. In many cases, logcf is much faster than realroot, the mean speedup is more than 4 and the largest speedup is more than 900. A possible reason may be that realroot is based on Descartes's rule of signs and bisection method while logcf is based on continuous fractions representation and Vincent's theorem.

As a built-in Mathematica symbol, RootIntervals is compared with our tool logcf. The Mathematica we use has a version number 11.1. logcf is as good as RootIntervals, in other words, in some test cases logcf is faster than RootIntervals but in some other cases RootIntervals is faster than logcf and the mean time of all test cases is almost the same. The reason may be that logcf and RootIntervals both are based on continuous fractions representation and Vincent's theorem. In the comparison we find a bug of RootIntervals. In Table 2 when $n = 1024, 1448, 2096$, RootIntervals outputs a double root. A possible reason for this may be that RootIntervals uses PossibleZeroQ to check whether an expression is zero or not but PossibleZeroQ cannot guarantee its outputs on those examples.

Table 1 shows that logcf is much faster than other solvers on W_n. The reason is that Algorithm 4 firstly check whether 1 is an upper bound and the distance of any two consecutive real roots of W_n is just 1, which guarantees that the equation of line 12 in Algorithm 3 always holds.

We also consider open software, such as ANewDsc[4] and SLV[2] which seem to be the fastest open software available for exact real root isolation. Many experiments about state of the art open software for isolating real roots have been done in [2, 4, 11], which indicate that ANewDsc and SLV are the fastest in many cases. In Figure 1 and Table 1, logcf is about 4 times faster than ANewDsc and ANewDsc is better on polynomials MR_n and MI_n which have two very close real roots. Figure 2 plots the compared result between logcf and SLV. As the figure shows, logcf is 7 times faster than SLV on W_n, T_n, U_n, H_n, WP_n averagely and a little slower than SLV on MI_n polynomials.

For randomly generated polynomials, we consider different settings of (n, b, r) as shown in Figure 3. In almost every randomly generated benchmark, logcf is faster than other solvers. And we can also find that degree is the main factor affecting the running time.

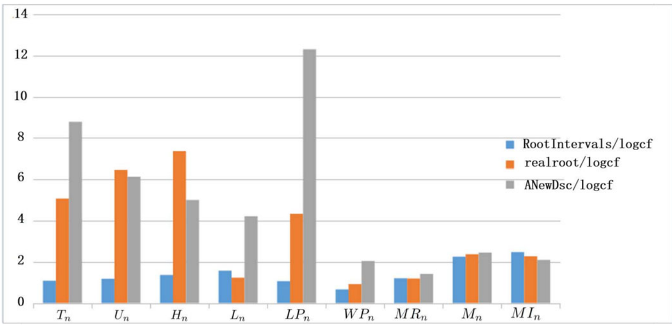

Figure 1 Mean running time compared with RootIntervals, realroot and ANewDsc

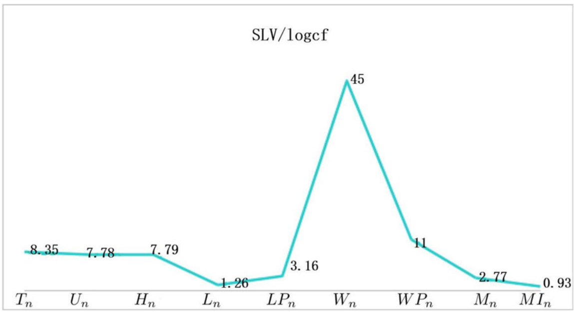

Figure 2 Comparison of logcf and SLV on benchmarks

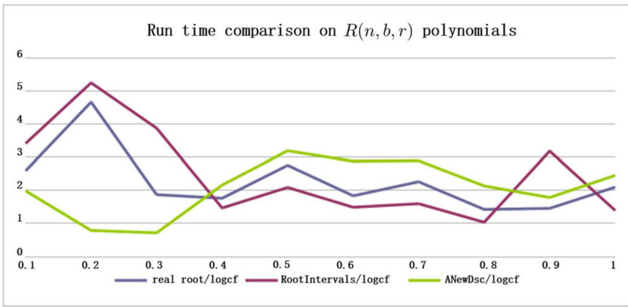

Figure 3 Comparison on random benchmarks $R(n,b,r)$ with different settings

Table 1 Comparison on three special benchmarks

Case	$\frac{\text{RootIntervals}}{\text{logcf}}$	$\frac{\text{realroot}}{\text{logcf}}$	$\frac{\text{ANewDsc}}{\text{logcf}}$
W_n	8.7	18.04	202
MR_n	0.70	980	0.174
MI_n	0.58	1.89	0.0029

Table 2 Comparison on MR_n

Degree	logcf	realroot	RootIntervals	ANewDsc
128	0.016	0.734	0.015	0.02
181	0.018	11.154	0.031	0.038
256	0.034	94.849	0.046	0.063
362	0.083	>600	0.094	0.11
512	0.20	>600	0.219	0.20
724	0.55	>600	0.344	0.42
1024	1.46	>600	error	0.79
1448	6.48	>600	error	1.69
2096	24.19	>600	error	3.16

More test results can be found at https://github.com/djuanbei/logcf/blob/master/testresult.xls.

In our experiments when Algorithm 5 is used for computing upper bounds, $T(p)$ takes more than ninety percent of running time§. We have considered methods in [18] for computing $T(p)$, but finally we chose the classical method (Horner's method) for its simplicity. In future work we will use Divide & Conquer method which is the fastest in [18]. We think this will further improve the performance of our tool.

References

[1] Collins G and Akritas A, Polynomial real roots isolation using descartes' rule of signs, *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, New York, 1976.

[2] Kobel A, Rouillier F, and Sagraloff M, Computing real roots of real polynomials ... and now for real, *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, Waterloo, 2016, 303–310.

[3] Rouillier F and Zimmermann P, Efficient isolation of polynomials' real roots, *Comput. Math. Appl.*, 2004, **162**(4): 33–50.

[4] Tsigaridas E, Slv: A software for real root isolation, *ACM Commun. Comput. Algebra*, 2016 **50**(3): 117–120.

[5] Eigenwillig A, Real root isolation for exact and approximate polynomials using Descartes' rule of signs, PhD thesis, Saarland University, 2008.

[6] Eigenwillig A, Kettner L, Krandick W, et al., A descartes algorithm for polynomials with bitstream coefficients, *CASC*, Springer, 2005, 138–149.

[7] Mehlhorn K and Sagraloff M, A deterministic algorithm for isolating real roots of a real polynomial, *J. Symb. Comput.* 2011, **46**: 70–90.

§The result of GNU gprof.

[8] Akritas A, Strzebonski A, and Vigklas P, Improving the performance of the continued fractions new bounds of positive roots, *Nonlinear Analysis: Modelling and Control*, 2008, **13**(3): 365–279.

[9] Sharma V, Complexity of real root isolation using continued fractions, *Theor. Comput. Sci.*, 2008, **409**(2): 292–310.

[10] Tsigaridas E P and Emiris I Z, On the complexity of real root isolation using continued fractions, *Theor. Comput. Sci.*, 2008, **392**: 158–173.

[11] Hemmer M, Tsigaridas E, Zafeirakopoulos Z, et al., Experimental evaluation and cross-benchmarking of univariate real solvers, *Proceedings of the* 2009 *Conference on Symbolic Numeric Computation*, ACM, Kyoto, 2009, 45–54.

[12] Xia B and Zhang T, Real solution isolation using interval arithmetic, *Comput. Math. Appl.*, 2006, **52**: 853–860.

[13] Zhang T and Xia B, A new method for real root isolation of univariate polynomials, *Mathematics in Computer Science*, 2007, **1**(2): 305–320.

[14] Hammer R, C++ Toolbox for Verified Scientific Computing - Theory, Algorithms and Programs: Basic Numerical Problems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[15] Rump S, INTLAB - Interval Laboratory. *Developments in Reliable Computing*, Ed. by Csendes T, Kluwer Academic Publishers, Dordrecht, 1999, 77–104. http://www.ti3.tu-harburg.de/rump/.

[16] Ştefănescu D, New bounds for the positive roots of polynomials, *J. Universal Comput. Sci.* 2005, **11**(12): 2132–2141.

[17] Hong H, Bounds for absolute positiveness of multivariate polynomials, *J. Symb. Comput.*, 1998, **25**(5): 571–585.

[18] Gerhard J, Modular algorithms in symbolic summation and symbolic integration, *Lecture Notes in Computer Science*, Springer Press, 2004.

[19] Johnson J R, Krandick W, and Ruslanov A D, Architecture-aware classical taylor shift by 1, *ISSAC*, ACM, Beijing, 2005, 200–207.

[20] Johnson J R, Krandick W, Richardson D, et al., High-performance implementations of the descartes method, *ISSAC* ACM, Genoa, 2006, 154–161.