Accelerating the Numerical Computation of Positive Roots of Polynomials using Improved Bounds

Kinji Kimura[1], **Takuto Akiyama**[2], **Hiroyuki Ishigami**[3], **Masami Takata**[4], **and Yoshimasa Nakamura**[5]
[1,2,3,5]Graduate School of Informatics, Kyoto University,
Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, JAPAN
[4]Academic Group of Information and Computer Sciences, Nara Women's University,
Kita-Uoya-Nishi-Machi, Nara 630-8506, JAPAN
[1]kkimur@amp.i.kyoto-u.ac.jp, [2]akiyama@amp.i.kyoto-u.ac.jp, [3]hishigami@amp.i.kyoto-u.ac.jp,
[4]takata@ics.nara-wu.ac.jp, [5]ynaka@i.kyoto-u.ac.jp

Abstract— *The continued fraction method for isolating the positive roots of a univariate polynomial equation is based on Vincent's theorem, which computes all of the real roots of polynomial equations. In this paper, we propose two new lower bounds which accelerate the fraction method. The two proposed bounds are derived from a theorem stated by Akritas et al., and use different pairing strategies for the coefficients of the target polynomial equations from the bounds proposed by Akritas et al. Numerical experiments show that the proposed lower bounds are more effective than existing bounds for some special polynomial equations and random polynomial equations, and are competitive with them for other special polynomial equations.*

Keywords: continued fraction method, Vincent's theorem, local-max bound, first-λ bound

1. Introduction

The real roots of univariate polynomial equations are more useful than the imaginary roots for practical applications in various engineering fields. Thus, this paper focuses on the computation of all real roots of polynomial equations. For polynomial equations without multiple roots, we can isolate each root into a specific interval. The accuracy of the isolated real roots can be easily enhanced using the bisection method.

The continued fraction (CF) method for isolating the positive roots of univariate polynomial equations is based on Vincent's theorem [2], [11]. This method isolates each positive root using Descartes' rule of signs [3], which focuses on the coefficients of the polynomial equations, and can be accelerated by an origin shift. Thus, several coefficients of a polynomial equation are transformed into nonzero coefficients, even in the case of sparse polynomial equations that have many zero coefficients. The Krawczyk method [8], which is based on numerical verification, was developed to isolate the positive roots of polynomial equations which have many zero coefficients. In this paper, we focus on the CF method for isolating the positive roots of polynomial equations which have many nonzero coefficients.

To accelerate the CF method, the choice of the origin shift is important. For the shift value, we should use a lower bound of the smallest positive root of the target polynomial. In other words, we must compute this lower bound to accelerate the CF method. We can obtain the lower bound of positive roots of a polynomial equation from the upper bound of the replaced polynomial equation corresponding to the original equation. The Cauchy bound [9] and the Kioustelidis bound [7] are well-known upper bounds of the positive roots of polynomial equations, but these bounds are known to produce overestimates in some cases. Akritas *et al.* have given a generalized theorem including the Cauchy bound and the Kioustelidis bound [1]. Using pairing strategies derived from the generalized theorem, they proposed new upper bounds called the first-λ bound, the local-max bound, and the local-max quadratic bound.

In [10], a lower bound generated by Newton's method is proposed. In this paper, we propose two lower bounds that are more effective than that based on Newton's method: the "local-max2" bound and the "tail-pairing first-λ" bound. These are derived from the local-max bound and the first-λ bound, respectively, and use a different pairing strategy from the original bound. The local-max2 bound is always better than or equal to the local-max bound. The tail-pairing first-λ bound is expected to be more suitable for the CF method than the first-λ bound.

The remainder of this paper is organized as follows: Section 2 introduces the CF method based on Vincent's theorem, and Section 3 introduces the bounds proposed by Akritas et al. Section 4 proposes the new lower bounds, before Section 5 reports the results of performance evaluations of the proposed lower bounds. We end with a summary of our conclusions in Section 6.

2. Continued fraction method

In this paper, we discuss the computation of positive roots $x \in \mathbb{R}$ that satisfy the following polynomial equation:

$$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n = 0, \quad (1)$$

where $a_i \in \mathbb{Z}$ and $a_n \neq 0$. Note that we need not consider the case $x = 0$ as one of the roots of $f(x) = 0$, since $a_n = 0$ is satisfied if any real root is equal to 0.

In addition, all the polynomial equations have rational coefficients and multiple roots in the interval $[u, v]$, $(-\infty, v]$, $(u, \infty]$, or $(-\infty, \infty)$ $(u, v \in \mathbb{R})$, and can be transformed into Eq. (1) in $x \in (0, \infty)$ using certain operations. For details, see [10].

2.1 Continued fraction method

The CF method aims to compute the positive roots of a polynomial equation $f(x) = 0$. It is based on Vincent's theorem [2], [11], and isolates the real roots in $(0, \infty)$ using Theorem 1, known as Descartes' rule of signs [3].

Theorem 1 (Descartes' rule of signs): For a polynomial equation

$$f(x) = a_0 x^n + \cdots + a_{n-1}x + a_n = 0,\ x \in \mathbb{R},\ a_i \in \mathbb{R},$$

let W be the number of "changes of sign" in the list of coefficients $\{a_0,\ a_1,\ \ldots,\ a_n\}$, except for $a_i = 0$, and let N be the number of positive roots in $(0, \infty)$. Under these definitions, the following relation holds:

$$N = W - 2h,$$

where h is a non-negative integer.

Using Theorem 1, the number of positive roots of the polynomial equation $f(x) = 0$ is determined as the following conditional branch:

- Case where $W = 0$: $f(x) = 0$ does not have any positive roots in the interval $x \in (0, \infty)$.
- Case where $W = 1$: $f(x) = 0$ has only one positive root in the interval $x \in (0, \infty)$.
- Case where $W \geq 2$: the number of positive roots of $f(x) = 0$ cannot be determined.

If $W = 1$, the isolated interval should be set to $(0, u\_b]$, where u_b denotes the upper bound of the positive roots of $f(x) = 0$. Computation methods for the upper bound of the positive roots of $f(x) = 0$ are described in Section 3.

In the case that $W \geq 2$, the interval $(0, \infty)$ should first be divided into two intervals. Then, Descartes' rule of signs can be applied to each interval. In the CF method, the interval $(0, \infty)$ is divided in $(0, 1)$ and $(1, \infty)$. This division is performed by the replacement $x \to x+1$ and $x \to 1/(x+1)$. Using the replacement $x \to x + 1$, the interval $(0, \infty)$ of the replaced polynomial equation corresponds to the interval $(1, \infty)$ of the original polynomial equation. Similarly, using the replacement $x \to 1/(x + 1)$, the interval $(0, \infty)$ of the replaced polynomial equation corresponds to the interval $(0, 1)$ of the original polynomial equation. The intervals $(1, \infty)$ and $(0, 1)$ do not include the case $x = 1$. To solve for this case, we must check that a constant term of the replaced polynomial equation vanishes after either replacement. In other words, if $a_n = 0$ in the replaced polynomial equation, then $x = 1$ is a root of the original polynomial equation.

Table 1: Synthetic division for $g_5(x)$.

a_0	a_1	a_2	a_3
	a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$
a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$
	a_0	$2a_0 + a_1$	
a_0	$2a_0 + a_1$	$3a_0 + 2a_1 + a_2$	
	a_0		
a_0	$3a_0 + a_1$		

The replacements described above require the coefficients of the replaced polynomial equation to be calculated. This calculation can be performed by synthetic division. As an example, Table 1 shows the calculation of the coefficients of

$$g_5(x) = a_0(x+1)^3 + a_1(x+1)^2 + a_3(x+1) + a_4. \quad (2)$$

As can be seen in Table 1, the coefficients of x^3, x^2, x^1, and x^0 in $g_5(x)$ are a_0, $3a_0 + a_1$, $3a_0 + 2a_1 + a_2$, and $a_0 + a_1 + a_2 + a_3$, respectively. Note that the computational complexity of the synthetic division for obtaining the coefficients of the replaced polynomial equation for a replacement $x \to x + 1$ is $O(n^2)$, where n is the highest order of the polynomial equation.

2.2 Acceleration using a lower bound

The CF method requires many replacement operations $x \to x + 1$ and $x \to 1/(x + 1)$. If the positive roots are much larger than 1, then the execution time increases, since we must repeat many replacement operations $x \to x + 1$. Thus, to decrease the execution time, the lower bound of the smallest positive root of a polynomial equation should be used as a shift.

The procedure for computing the lower bound l_b of $f(x) = 0$ is as follows:

1) Replace x with $1/x$ in $f(x)$.
2) Compute u_b, the upper bound of the positive roots of the replaced polynomial equation.
3) Obtain l_b as $l\_b = 1/u\_b$.

However, the replacement $x \to x + l\_b$ should not always be adopted, since l_b is not sufficiently large to reduce the execution time if $l\_b \leq 1$. Thus, the replacement $x \to x+l\_b$ is only adopted in $f(x)$ if $l\_b > 1$.

3. Computation of the upper bound of positive roots

The Cauchy rule [9] is a well-known idea for computing the upper bound of the positive roots of $f(x) = 0$. The Kioustelidis bound is related to the Cauchy rule [7]. However, both of these bounds are known to overestimate the upper bound in some cases.

To overcome this problem, Akritas et al. derived the following generalized theorem for computing the upper bound of the positive roots of $f(x) = 0$.

Theorem 2 (Akritas, 2006): Let $f(x)$ be a polynomial with real coefficients, and assume $a_0 > 0$. Let $d(f)$ and $t(f)$ denote its degree and number of terms, respectively. In addition, assume that $f(x)$ can be reshaped as follows:

$$f(x) = q_1(x) - q_2(x) + \cdots - q_{2m}(x) + g_6(x), \quad (3)$$

where the polynomials $q_i(x)$, $i = 1, \ldots, 2m$, and $g_6(x)$ have only positive coefficients. Moreover, assume that, for $i = 1, 2, \ldots, m$, we obtain

$$\begin{aligned} q_{2i-1}(x) = c_{2i-1,1}x^{e_{2i-1,1}} + \cdots \\ + c_{2i-1,t(q_{2i-1})}x^{e_{2i-1,t(q_{2i-1})}} \end{aligned} \quad (4)$$

and

$$q_{2i}(x) = b_{2i,1}x^{e_{2i,1}} + \cdots + b_{2i,t(q_{2i})}x^{e_{2i,t(q_{2i})}} \quad (5)$$

where $e_{2i-1,1} = d(q_{2i-1})$ and $e_{2i,1} = d(q_{2i})$, and the exponent of each term in $q_{2i-1}(x)$ is greater than the exponent of each term in $q_{2i}(x)$. If $t(q_{2i-1}) \geq t(q_{2i})$ for all indices $i = 1, 2, \cdots, m$, then the upper bound of the positive roots of $f(x) = 0$ is defined by

$$\begin{aligned} u\_b = \max_{i=1,2,\ldots,m} \Bigg\{ \left(\frac{b_{2i,1}}{c_{2i-1,1}} \right)^{\frac{1}{e_{2i-1,1} - e_{2i,1}}}, \ldots, \\ \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}} \right)^{\frac{1}{e_{2i-1,t(q_{2i})} - e_{2i,t(q_{2i})}}} \Bigg\}, \end{aligned} \quad (6)$$

for any permutation of the positive coefficients $c_{2i-1,j}$, $j = 1, 2, \cdots, t(q_{2i-1})$. Otherwise, for each of the indices i for which we obtain $t(q_{2i-1}) < t(q_{2i})$, we break up one of the coefficients of $q_{2i-1}(x)$ into $t(q_{2i}) - t(q_{2i-1}) + 1$ parts, so that $t(q_{2i}) = t(q_{2i-1})$. We can then apply the formula defined in Eq. (6).

Note that the ideas underlying both the Cauchy and Kiouste-lidis bounds are included in this theorem.

The sharpness of the upper bound is dependent on pairing coefficients from the non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$ for $1 \leq l < i$.

For example, consider the polynomial

$$3x^3 - 5x^2 + 4x + 7. \quad (7)$$

In this case, we can create the pair

$$\left\{3x^3, -5x^2\right\}.$$

However, for the polynomial

$$3x^3 - 5x^2 - 4x + 7, \quad (8)$$

we cannot create the trivial pair, since the polynomial has only one positive coefficient. In this case, since

$$3x^3 = \frac{3}{2}x^3 + \frac{3}{2}x^3 = x^3 + 2x^3,$$

we can create the pair as

$$\left\{\frac{3}{2}x^3, -5x^2\right\}, \left\{\frac{3}{2}x^3, -4x\right\} \text{ or } \left\{x^3, -5x^2\right\}, \left\{2x^3, -4x\right\}.$$

Using Theorem 2, Akritas et al. proposed the "local-max" bound and the "first-λ" bound as follows:

Definition 1 ("local-max"): For a polynomial equation $f(x) = 0$ given by Eq. (1), the coefficient $-a_k$ of the term $-a_k x^{n-k}$ in $f(x) = 0$ is paired with the coefficient $a_m/2^t$ of the term $a_m x^{n-m}$, where a_m is the largest positive coefficient with $0 \leq m < k$ and t denotes the number of times the coefficient a_m has been used.

Definition 2 ("first-λ"): For a polynomial equation $f(x)$ given by Eq. (3) with λ negative coefficients, we first consider all cases for which $t(q_{2i}) > t(q_{2i-1})$ by breaking up the last coefficient $c_{2i-1,t(q_{2i})}$ of $q_{2i-1}(x)$ into $t(q_{2i}) - t(q_{2i-1}) + 1$ equal parts. We then pair each of the first λ positive coefficients of $f(x)$, encountered as we move in non-increasing order of exponents, with the first unmatched negative coefficient.

Note that the computational complexity of these bounds is $O(n)$.

Akritas et al. also proposed the "local-max quadratic" bound as follows:

Definition 3 ("local-max quadratic"): For a polynomial equation $f(x)$ given by Eq. (1), each negative coefficient $a_i < 0$ is "paired" with each of the preceding positive coefficients a_j divided by 2^{t_j}. That is, each positive coefficient a_j is "broken up" into unequal parts, as for the locally maximum coefficient in the local max bound. t_j is initially set to 1, and is incremented each time the positive coefficient a_j is used, and the minimum is taken over all j. Subsequently, the maximum is taken over all i.

From Definition 3, the local-max quadratic bound is computed as

$$u\_b_{\text{LMQ}} = \max_{a_i<0} \min_{a_j>0:j>i} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{t_j}}}}. \quad (9)$$

Note that the computational complexity of this bound is $O(n^2)$.

4. New upper bounds

In this section, we propose two new upper bounds for the positive roots of a polynomial equation. The first is the "local-max2" bound, and the second is the "tail-pairing first-λ" bound.

4.1 Local-max2 bound

The local-max2 bound is derived from the local-max bound. To compute the local-max bound, the largest positive coefficient a_m is broken up into unequal parts $a_m/2^t (t = 1, \cdots, s+1)$. For the local-max2 bound, we first break up the largest positive coefficient a_m into unequal parts $a_m/2^t (t = 1, \cdots, s)$. Then, since

$$a_m - \left(\frac{a_m}{2} + \cdots + \frac{a_m}{2^s}\right) = \frac{a_m}{2^s}, \tag{10}$$

we use $a_m/2^s$, which is the remaining part of a_m, as the last pair. It is obvious that the local-max2 bound is better than or equal to the local-max bound for all polynomials.

Algorithm 1 describes the implementation of the local-max2 bound. As for the local-max bound, the complexity of computing the local-max2 bound is $O(n)$.

For example, consider the polynomial

$$x^3 + 10^{100}x^2 - x - 10^{100}. \tag{11}$$

For the local-max bound, we pair the terms $\left\{\frac{10^{100}}{2}x^2, -x\right\}$ and $\left\{\frac{10^{100}}{2^2}x^2, -10^{100}\right\}$, and obtain a bound estimate of 2. For the local-max2 bound, we pair the terms $\left\{\frac{10^{100}}{2}x^2, -x\right\}$ and $\left\{\frac{10^{100}}{2}x^2, -10^{100}\right\}$, and obtain a bound estimate of $\sqrt{2}$. As a result, the upper bound of the local-max2 bound is better than that of the local-max bound for the polynomial (11).

4.2 Tail-pairing first-λ

The tail-pairing first-λ bound is derived from the first-λ bound. As for the first-λ bound, if there are more negative than positive coefficients, we first break up the last positive coefficient into several parts. In addition, we pair positive coefficients with unpaired tail negative coefficients when the number of positive coefficients is greater than that of negative coefficients.

Although it is not always better than or equal to the first-λ bound, we expect the tail-pairing first-λ bound to be better for the total number of shifts. The CF method performs the replacement $x \to x + l\_b$ many times. Thus, pairing negative coefficients of low degree with positive coefficients is an important task. In the tail-pairing first-λ bound, we pair high-degree coefficients with low-degree coefficients whenever possible.

There are two strategies for computing the tail-pairing first-λ bound. In the first strategy, we initially pair negative coefficients in the corresponding list, and then pair the tail

Algorithm 1 Implementation of the "local-max2" bound.

$cl \leftarrow \{a_n, a_{n-1}, \cdots, a_1, a_0\}$
if $n+1 \le 1$ **then**
  return $u\_b_{\mathrm{LM2}} = 0$
end if
$j = n+1$
$negativeIndices = \{\}$
for $i = n$ to 1 step -1 **do**
  if $cl(i) < 0$ **then**
    $negativeIndices = negativeIndices \cup i$
  else if $cl(i) > cl(j)$ **then**
    if $count(negativeIndices) > 0$ **then**
      $t = 0$
      $l = count(negativeIndices)$
      for $k = 1$ to $l-1$ **do**
        $t{+}{+}$
        $tempub = (2^t(-cl(negativeIndices(k))/cl(j)))^{1/(j-negativeIndices(k))}$
        if $tempub > u\_b_{\mathrm{LM2}}$ **then**
          $u\_b_{\mathrm{LM2}} = tempub$
        end if
      end for
      $tempub = (2^t(-cl(negativeIndices(l))/cl(j)))^{1/(j-negativeIndices(l))}$
      if $tempub > u\_b_{\mathrm{LM2}}$ **then**
        $u\_b_{\mathrm{LM2}} = tempub$
      end if
    end if
    $j = i$
    $negativeIndices = \{\}$
  end if
end for
if $count(negativeIndices) > 0$ **then**
  $t = 0$
  $l = count(negativeIndices)$
  for $k = 1$ to $l-1$ **do**
    $t{+}{+}$
    $tempub = (2^t(-cl(negativeIndices(k))/cl(j)))^{1/(j-negativeIndices(k))}$
    if $tempub > u\_b_{\mathrm{LM2}}$ **then**
      $u\_b_{\mathrm{LM2}} = tempub$
    end if
  end for
  $tempub = (2^t(-cl(negativeIndices(l)/cl(j)))^{1/(j-negativeIndices(l))}$
  if $tempub > u\_b_{\mathrm{LM2}}$ **then**
    $u\_b_{\mathrm{LM2}} = tempub$
  end if
end if
return u_b_{LM2}

negative coefficients. We call this the "tail-pairing first-λ type-I bound". The second strategy pairs the tail negative coefficients first, and then pairs the negative coefficients in the corresponding list. We call this the "tail-pairing first-λ type-II bound". Algorithm 2 describes the computation of the tail-pairing first-λ type-I bound. As for the first-λ bound, the computational complexity of both tail-pairing first-λ bounds is $O(n)$.

For example, consider the polynomial

$$x^5 + 2x^4 - 3x^3 + 4x^2 - 5x - 10^{10}. \tag{12}$$

For the first-λ bound, we pair the terms $\{x^5, -3x^3\}$, $\{2x^4, -5x\}$, and $\{2x^2, -10^{10}\}$, and obtain a bound estimate of $\sqrt{10^{10}/2} = 50000\sqrt{2}$. For the tail-pairing first-λ type-I bound, we pair the terms $\{x^5, -3x^3\}$, $\{2x^4, -10^{10}\}$, and $\{4x^2, -5x\}$, and find a bound estimate of $\sqrt[4]{10^{10}/2} = 100\sqrt[4]{50}$. For the tail-pairing first-λ type-II bound, we pair the terms $\{x^5, -10^{10}\}$, $\{2x^4, -3x^3\}$, and $\{4x^2, -5x\}$, which gives a bound estimate of $\sqrt[5]{10^{10}} = 100$. Thus, the tail-pairing first-λ bounds are better than the first-λ bound, and the tail-pairing first-λ type-II bound is better than the type-I bound for this polynomial.

5. Numerical experiment

In this section, we present numerical results that evaluate the effect of the proposed bounds.

5.1 Contents of the numerical experiment

To evaluate the effect of the proposed bounds, we implement the CF method with the following bounds:

- FL+LM: $(\max(\text{FL}, \text{LM}))$
- LMQ: local-max quadratic bound
- TPFL-I+LM2: $(\max(\text{TPFL-I}, \text{LM2}))$
- TPFL-II+LM2: $(\max(\text{TPFL-II}, \text{LM2}))$

Note that FL, LM, TPFL, and LM2 denote the first-λ bound, local-max bound, tail-pairing first-λ bound, and local-max2 bound, respectively.

As test polynomial equations, the following were used:

- Laguerre: $L_0(x) = 1$, $L_1(x) = 1 - x$, and $L_{n+1}(x) = \frac{1}{n+1}((2n + 1 - x)L_n(x) - nL_{n-1}(x))$
- Chebyshev-I: $T_0(x) = 1$, $T_1(x) = x$, and $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$
- Chebyshev-II: $U_0(x) = 1$, $U_1(x) = 2x$, and $U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$
- Wilkinson: $W_n(x) = \prod_{i=1}^{n}(x - i)$
- Mignotte: $M_n(x) = x^n - 2(5x - 1)^2$
- Randomized polynomial

The randomized polynomials are defined as

$$f(x) = \prod_{i=0}^{r}(x - x_i) \prod_{j=0}^{s}(x - \alpha_j + i\beta_j)(x - \alpha_j - i\beta_j), \tag{13}$$

Algorithm 2 Implementation of the "tail-pairing first-λ" bound.

```
cl ← {a_n, a_{n−1} ··· , a_1, a_0}
λ ← the number of negative elements of cl
if n + 1 ≤ 1 then
    return u_b_TPFL = 0
end if
posStartIndex = n + 1
negTailIndex = 1
while negTailIndex ≤ n + 1
      and cl(negTailIndex) ≥ 0 do
    negTailIndex + +
end while
while λ > 0 do
    while posStartIndex >= 0
     and cl(posStartIndex) ≤ 0 do
        posStartIndex − −
    end while
    posEndIndex = posStartIndex + 1
    while posEndIndex >= 0
     and cl(posEndIndex) ≥ 0 do
        posEndIndex − −
    end while
    negHeadStartIndex = negHeadEndIndex
    negHeadStartIndex = posEndIndex
    while negHeadEndIndex >= 0
     and negHeadEndIndex ≤ negTailIndex
     and cl(negHeadEndIndex) ≤ 0 do
        negHeadEndIndex − −
    end while
    posCount = posEndIndex − posStartIndex
    negHeadCount = negHeadEndIndex
     −negHeadStartIndex
    j = posStartIndex
    call Algorithm 3
    call Algorithm 4
end while
return u_b_TPFL
```

where $x_i,\ \alpha_j,\ \beta_j \in \mathbb{R}$. Note that the parameters x_i, α_j, and β_j were randomly set in the following range:

$$-10^9 \le x_i, \alpha_j, \beta_j \le 10^9. \tag{14}$$

The parameter s was set to 40, 490, 740, or 990, and r was set to 20. We then generated 100 test polynomial equations for each combination of parameters. All polynomials were preprocessed to have integer coefficients using the method introduced in [10].

The experiments were performed on an Intel Core i7 3770K CPU with 32 GB of RAM, with GCC 4.6.3 used as the C compiler. In addition, we used GMP [4], since the CF method needs multiple-precision arithmetic to compute the coefficients in the replaced polynomial equations.

Algorithm 3 Subroutine 1 of the "tail-pairing first-λ" bound.

```
if negHeadCount > 0 then
  i = negHeadStartIndex
  while negHeadCount > 0 do
    if posCount == 1
      and negHeadCount > posCount then
      k = negHeadCount − posCount + 1
      for v = 1 to k do
        tempub = (−cl(i)/(cl(j)/k))^{1/(j−i)}
        if tempub > u_b_TPFL then
          u_b_TPFL = tempub
        end if
        negHeadCount − −
        λ − −
        i − −
        while i ≥ 0 and cl(i) == 0 do
          i − −
        end while
      end for
    else
      tempub = (−cl(i)/(cl(j))^{1/(j−i)}
      if tempub > u_b_TPFL then
        u_b_TPFL = tempub
      end if
      negHeadCount − −
      λ − −
      i − −
      while i ≥ 0 and cl(i) == 0 do
        i − −
      end while
    end if
    posCount − −
    if posCount > 0 then
      j − −
      while cl(j) == 0 do
        j − −
      end while
    end if
  end while
end if
```

Algorithm 4 Subroutine 2 of the "tail-pairing first-λ" bound.

```
while posCount > 0
  and negHeadEndIndex < negTailIndex do
  i = negTailIndex
  tempub = (−cl(i)/(cl(j))^{1/(j−i)}
  if tempub > u_b_TPFL then
    u_b_TPFL = tempub
  end if
  posCount − −
  λ − −
  if λ == 0 then
    break
  end if
  negTailIndex + +
  while negTailIndex >= 0
   and cl(negTailIndex) ≥ 0 do
    negTailIndex + +
  end while
  if posCount > 0 then
    j − −
    while cl(j) == 0 do
      j − −
    end while
  end if
end while
```

5.1.1 $\log_2$ optimization

$\log_2$ optimization is used in various open-source software [5] [6]. Assume that we wish to calculate bounds of the following form in multiple-precision integer:

$$\left(-\frac{b}{c}\right)^{\frac{1}{d-e}}, c > 0, b < 0, d > e > 0, \tag{15}$$

using division and root functions. It takes a considerable amount of time to calculate the bounds, and the execution time for each function depends on the bit-length of the arguments. Here, we can use $\log_2$ to find the bounds

$$\frac{1}{d-e}\left(\log_2(-b) - \log_2 c\right), \tag{16}$$

and the execution time of $\log_2$ for multiple-precision integer does not depend on the bit-length of the argument. Therefore, we can avoid division and root functions in multiple-precision integer by comparing $\log_2$ values. The bounds computed with $\log_2$ can be worse than those given by the division and root functions. However, this method saves a lot of time in computing the bounds, and is fast in terms of total execution time.

5.2 Results

Table 2 lists the execution time for special polynomial equations, and Table 3 lists that for random polynomial equations. Our proposed bounds are more effective than FL+LM and LMQ for the Laguerre polynomial and the Chebyshev polynomial, and are competitive with FL+LM for the Wilkinson and Mignotte polynomials. The maximum speed-up for the Laguerre polynomial is about 1.19, and for the Chebyshev-I and -II polynomials it is about 1.12 and 1.14 times, respectively. TPFL-II+LM2 is more effective than TPFL-I+LM2 for some special polynomial equations. We can see this tendency for random polynomial equations: both TPFL-I+LM2 and TPFL-II+LM2 are more effective than FL+LM and LMQ. We can also see that TPFL-II+LM2 is more effective than TPFL-I+LM2.

Table 2: Execution time for special polynomials.

Polynomial Class	Degree	Time (s)			
		FL +LM	LMQ	TPFL-I +LM2	TPFL-II +LM2
Laguerre	100	0.01	0.01	0.01	0.01
Laguerre	1000	43.51	48.20	41.77	36.57
Laguerre	1500	221.10	242.69	217.21	189.34
Laguerre	2000	704.95	755.48	683.57	617.01
Chebyshev-I	100	0.01	0.01	0.01	0.01
Chebyshev-I	1000	40.22	41.11	36.30	36.48
Chebyshev-I	1500	206.87	210.86	184.45	185.61
Chebyshev-I	2000	650.85	638.67	590.36	590.36
Chebyshev-II	100	0.01	0.01	0.01	0.01
Chebyshev-II	1000	40.48	40.88	35.74	35.56
Chebyshev-II	1500	203.53	210.73	182.73	182.67
Chebyshev-II	2000	652.94	636.42	599.48	579.28
Wilkinson	100	0.00	0.00	0.00	0.00
Wilkinson	1000	4.53	4.92	4.52	4.54
Wilkinson	1500	22.45	23.82	22.46	22.46
Wilkinson	2000	70.46	73.97	70.59	70.60
Mignotte	100	0.00	0.00	0.00	0.00
Mignotte	1000	0.04	0.04	0.04	0.04
Mignotte	1500	0.12	0.12	0.12	0.12
Mignotte	2000	0.27	0.27	0.27	0.27

Table 3: Execution time for random polynomials.

Parameters	Degree	Time (s), Avg (Min/Max)	
		FL+LM	LMQ
$s = 40$ $r = 20$	100	0.015(0.01/0.02)	0.0188(0.01/0.03)
$s = 490$ $r = 20$	1000	29.046(19.15/43.61)	30.161(17.47/49.39)
$s = 740$ $r = 20$	1500	135.59(94.78/203.07)	139.06(92.1/211.72)
$s = 990$ $r = 20$	2000	415.37(296.62/645.55)	425.47(270.36/835.35)

Parameters	Degree	Time (s), Avg (Min/Max)	
		TPFL-I+LM2	TPFL-II+LM2
$s = 40$ $r = 20$	100	0.0145(0.01/0.02)	0.0127(0.01/0.02)
$s = 490$ $r = 20$	1000	27.325(19.05/38.39)	26.88(17.22/39.38)
$s = 740$ $r = 20$	1500	128.07(91.69/179.71)	123.84(86.17/176.16)
$s = 990$ $r = 20$	2000	384.11(266.41/617.17)	368.36(271.71/603.31)

6. Conclusions

In this study, we have proposed new lower bounds based on the local-max bound and the first-λ bound for accelerating the CF method. The local-max2 bound is sharper than or equal to the local-max bound. The tail-pairing first-λ bound is expected to be more suitable for the CF method than the first-λ bound, because of the need to replace $x \to x + l\_b$ many times in the CF method. The numerical results show that the average execution time of the CF method with both the local-max2 bound and the tail-pairing first-λ bound is faster than or nearly equal to that with the local-max bound, first-λ bound, and local-max quadratic bound for all polynomial equations.

References

[1] A. Akritas, A. Strzeboński, P. Vigklas, "Implementations of a new theorem for computing bounds for positive roots of polynomials", *C*omputing, 78, pp. 355–367, 2006.

[2] A. Akritas, A. Strzeboński, P. Vigklas, "Improving the performance of the continued fractions method using new bounds of positive roots", *N*onlinear Analysis: Modelling and Control, 13, pp. 265–279, 2008.

[3] G. Collins, A. Akritas, "Polynomial real root isolation using Descartes' rule of signs", *S*YMSAC '76, Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, Yorktown Heights, NY, USA, ACM, pp. 272–275, 1976.

[4] (2013) The GNU MP Bignum Library. [Online]. Available: http://gmplib.org/

[5] (2013) Sage. [Online]. Available: http://sagemath.org/

[6] (2013) SymPy. [Online]. Available: http://sympy.org/en/index.html

[7] B. Kioustelidis, "Bounds for positive roots of polynomials", *J*. Comput. Appl. Math., 16(2), pp. 241–244, 1986.

[8] R.E. Moore, "Interval Analysis", Prentice Hall, Englewood Cliffs, N.J., 1966.

[9] N. Obreschkoff, "Verteilung und Berechnung der Nullstellen reeller Polynome", Berlin: VEB Deutscher Verlag der Wissenschaften 1963.

[10] Masami Takata, Takuto Akiyama, Sho Araki, Kinji Kimura, Yoshimasa Nakamura, "Improved Computation of Bounds for Positive Roots of Polynomials", In Proceedings of the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13), Vol.II, pp. 168–174, 2013.

[11] A.J.H. Vincent, "Sur la resolution des équations numériques", J. Math. Pures Appl. 1, pp. 341–372, 1836. 2002.