IPSJ Transactions on Mathematical Modeling and Its Applications Vol.15 No.1 18–30 (Jan. 2022)

Regular Paper

Accelerating the Numerical Computation of Positive Roots of Polynomials Using Suitable Combination of Lower Bounds

Masami Takata[1,a)] Takuto Akiyama[2] Sho Araki[2] Hiroyuki Ishigami[3]
Kinji Kimura[4] Yoshimasa Nakamura[5]

Received: June 16, 2021, Revised: October 13, 2021,
Accepted: November 1, 2021

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 suitable combination of 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. Moreover, we compute another bound. First, we compute a candidate for the lower bound generated by Newton's method. Second, by using Laguerre's theorem, we check whether the candidate for the lower bound is appropriate. Numerical experiments show that the three 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. Additionally, we determine a suitable combination of those lower bounds.

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

1. Introduction

In computer algebra, the quantifier elimination is one of the important topics. The cylindrical algebraic decomposition algorithm is a tool solving for the quantifier elimination [3]. In the algorithm, only the real roots of univariate polynomial equations are required. 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], [13]. This method isolates each positive root using Descartes' rule of signs [4], 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 [10], 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 [11] and the Kioustelidis bound [9] 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 Ref. [12], we proposed a lower bound generated by Newton's method. In Ref. [8], we introduced two lower bounds: 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. In this paper, we propose a suitable combination of these lower bounds.

The remainder of this paper is organized as follows: Section 2 introduces the CF method based on Vincent's theorem. In Section 3, we introduce the bounds proposed by Akritas et al. In

[1] Nara Women's University, Nara 630–8506, Japan
[2] Kyoto University, Kyoto 606–8501, Japan
[3] Yahoo Japan Corporation, Chiyoda, Tokyo 102–8282, Japan
[4] University of Fukui, Fukui 910–8507, Japan
[5] Osaka Seikei University, Osaka 533–0007, Japan
[a)] takata@ics.nara-wu.ac.jp

Section 4, we explain the local-max2 bound and the tail-pairing first-λ bound. In Section 5, we review the lower bound generated by Newton's method. In Section 6, we reports the results of performance evaluations of these lower bounds and determine a suitable combination of these lower bounds. We end with a summary of our conclusions in Section 7.

2. Continued Fraction Method

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

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0, \quad (1)$$

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 Ref. [12].

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], [13], and isolates the real roots in $(0, \infty)$ using Theorem 1, known as Descartes' rule of signs [4].

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

$$f(x) = a_0x^n + \cdots + a_{n-1}x + a_n = 0,\ x \in \mathbb{R},\ a_i \in \mathbb{R},$$

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 $g_5(x)$.

x^3	x^2	x^1	x^0
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 [11] 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 [9]. 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:

$$f(x) = q_1(x) - q_2(x) + \cdots + q_{2m-1} - q_{2m}(x) + g_6(x), \tag{3}$$

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

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

and

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

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

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

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 Kiousteli-dis 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

$$3x^3 - 5x^2 + 4x + 7. \tag{7}$$

In this case, we can create the pair

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

However, for the polynomial

$$3x^3 - 5x^2 - 4x + 7, \tag{8}$$

we cannot create the trivial pair, since the polynomial has only one positive coefficient with the greater degree of x than the negative coefficient terms $-5x^2$ and $-4x$. 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

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

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

4. Local-max2 Bound and Tail-pairing First-λ Bound

In this section, we introduce two 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

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

```
cl ← {a_n, a_{n−1}, ··· , a_1, a_0}
if n + 1 ≤ 1 then
    return u_b_LM2 = 0
end if
j = n + 1
negativeIndices = {}
for i = n to 1 step −1 do
    if cl(i) < 0 then
        negativeIndices = negativeIndices ∪ 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_LM2 then
                    u_b_LM2 = tempub
                end if
            end for
            tempub = (2^t(−cl(negativeIndices(l))
                            /cl(j)))^{1/(j−negativeIndices(l))}
            if tempub > u_b_LM2 then
                u_b_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_LM2 then
            u_b_LM2 = tempub
        end if
    end for
    tempub = (2^t(−cl(negativeIndices(l)
                    /cl(j)))^{1/(j−negativeIndices(l))}
    if tempub > u_b_LM2 then
        u_b_LM2 = tempub
    end if
end if
return u_b_LM2
```

$\left\{\frac{10^{100}}{2^2}x^2, -10^{100}\right\}$, 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 Bound

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

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
    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
end while
return u_b_TPFL
```

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
```

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 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 $\left\{x^5, -3x^3\right\}$, $\left\{2x^4, -5x\right\}$, and $\left\{2x^2, -10^{10}\right\}$, and obtain a bound estimate of $\sqrt{10^{10}/2} = 50,000\sqrt{2}$. For the tail-pairing first-λ type-I bound, we pair the terms $\left\{x^5, -3x^3\right\}$, $\left\{2x^4, -10^{10}\right\}$, and $\left\{4x^2, -5x\right\}$, 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 $\left\{x^5, -10^{10}\right\}$, $\left\{2x^4, -3x^3\right\}$, and $\left\{4x^2, -5x\right\}$, 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. Lower Bound Generated by Newton's Method

The acceleration of the continued fraction method based on Vincent's theorem employs the origin shift, which adopts the lower bound l_b of the smallest positive root of a given polynomial equation. Thus, if the lower bound tends to the smallest positive root, then the computation time of the continued fraction method decreases.

In this paper, we review a lower bound generated by Newton's method. Note that in some polynomial equations, a bound generated by Newton's method is not suitable as the lower bound. Hence, by using Laguerre's theorem, it must be checked whether a bound generated by Newton's method is a suitable lower bound.

Newton's method is defined by the following recurrence formula:

$$x_{m+1} = x_m - \frac{f(x_m)}{f'(x_m)}. \tag{13}$$

Here, $f'(x)$ denotes the first derivative of $f(x)$. If Newton's method is adopted at the origin, then a candidate for the lower bound r is computed as follows:

$$r = 0 - \frac{f(0)}{f'(0)} = -\frac{a_n}{a_{n-1}}. \tag{14}$$

The cost for computing r is $O(1)$.

We can check whether a candidate for the lower bound r is suitable by using the following theorem.

Theorem 3 (the Laguerre theorem) For a polynomial equation

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n = 0$$

with real coefficients, let N be the number of positive roots that are larger than a positive value α. The number N is less than or equal to the number of sign changes in the following polynomials $f_k(\alpha)$:

$$f_k(\alpha) = a_0\alpha^k + a_1\alpha^{k-1} + \cdots + a_k, k = 0, 1, \ldots, n.$$

Here, $f(\alpha) \neq 0$ is assumed.

If x in Theorem 3 is replaced by $1/x$, Theorem 3 can be transformed into the following theorem.

Theorem 4 The number of positive roots of a polynomial equation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = 0,$$

in the interval $0 < x < r$ is less than or equal to the number of sign changes in the following polynomials $p_k(r)$:

$$\begin{aligned} p_0(r) &= a_0, \\ p_1(r) &= a_0 + a_1 r, \\ p_2(r) &= a_0 + a_1 r + a_2 r^2, \ldots, \\ p_n(r) &= a_0 + a_1 r + \cdots + a_n r^n. \end{aligned}$$

Here, $r > 0$ and $p_n(r) \neq 0$.

Thus, if the number of sign changes in $p_k(r), k = 0, \cdots, n$ is 0, then no positive roots in the interval $0 < x < r$ exist. In such a case, the computation cost is $O(n)$.

To get a candidate for the lower bound r, it is necessary that the signs of a_n and a_{n-1} must be opposite. Moreover, it is needed to check $p_n(r) \neq 0$.

If a candidate for the lower bound r generated by Newton's method is the lower bound of the smallest positive root, then we adopt the lower bound lb as an origin shift defined in the following equation:

$$l\_b_{\text{Newton}} = r. \tag{15}$$

If $l\_b_{\text{Newton}} > 1$, then the origin shift $x \to x + l\_b_{\text{Newton}}$ is performed.

6. Numerical Experiment

In this section, we present numerical results that evaluate the effect of the three lower bounds. Additionally, we determine a suitable combination of those lower bounds.

6.1 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(FL, LM))
- LMQ: local-max quadratic bound
- TPFL-I+LM2: (max(TPFL-I, LM2))
- TPFL-II+LM2: (max(TPFL-II, LM2))
- FL+LM+Newton: (max(FL, LM, Newton))
- LMQ+Newton: (max(LMQ, Newton))
- TPFL-I+LM2+Newton: (max(TPFL-I, LM2, Newton))
- TPFL-II+LM2+Newton: (max(TPFL-II, LM2, Newton))

Note that FL, LM, LMQ, TPFL, LM2, and Newton denote the first-λ bound, local-max bound, local-max quadratic bound, tail-pairing first-λ bound local-max2 bound, and the bound generated by Newton's method, respectively. In FL+LM, FL and LM are computed respectively, and then the maximum value is used as the bound. If no appropriate bound can be obtained, 0 bound is used. Other combinations are the same.

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

$$f(x) = \prod_{i=0}^{r}(x - x_i) \prod_{j=0}^{s}(x - \alpha_j + i\beta_j)(x - \alpha_j - i\beta_j), \tag{16}$$

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:

$$-10^9 \leq x_i, \alpha_j, \beta_j \leq 10^9. \tag{17}$$

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 Ref. [12].

The experiments were performed on an Intel Xeon E5-2695 v4 CPU with 128 GB of RAM, with GCC 10.1.1 used as the C compiler. In addition, we used GMP [5], since the CF method needs multiple-precision arithmetic to compute the coefficients in the replaced polynomial equations.

6.1.1 $\log_2$ Optimization

$\log_2$ optimization is used in various open-source software [6], [7]. Assume that we wish to calculate bounds of the following form in multiple-precision integer:

$$\left(-\frac{b}{c}\right)^{\frac{1}{d-e}}, c > 0, b < 0, d > e > 0, \tag{18}$$

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

$$\frac{1}{d-e}\left(\log_2(-b) - \log_2 c\right), \tag{19}$$

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.

6.2 Results

The appendix shows the ratio of each bound adopted in polynomials. In special polynomials, Newton bound is not adopted. This is due to the properties of polynomials. In a random polynomial, all bounds are adopted at least once. Hence, it makes sense to combine multiple bounds. In particular, for special polynomials, Newton bound is not valid. However, for random polynomials, Newton bound is more effective in some cases as is shown in **Table A·6**. Therefore, it is effective to use Newton bound in combination.

Figure 1 shows the execution time for special polynomial

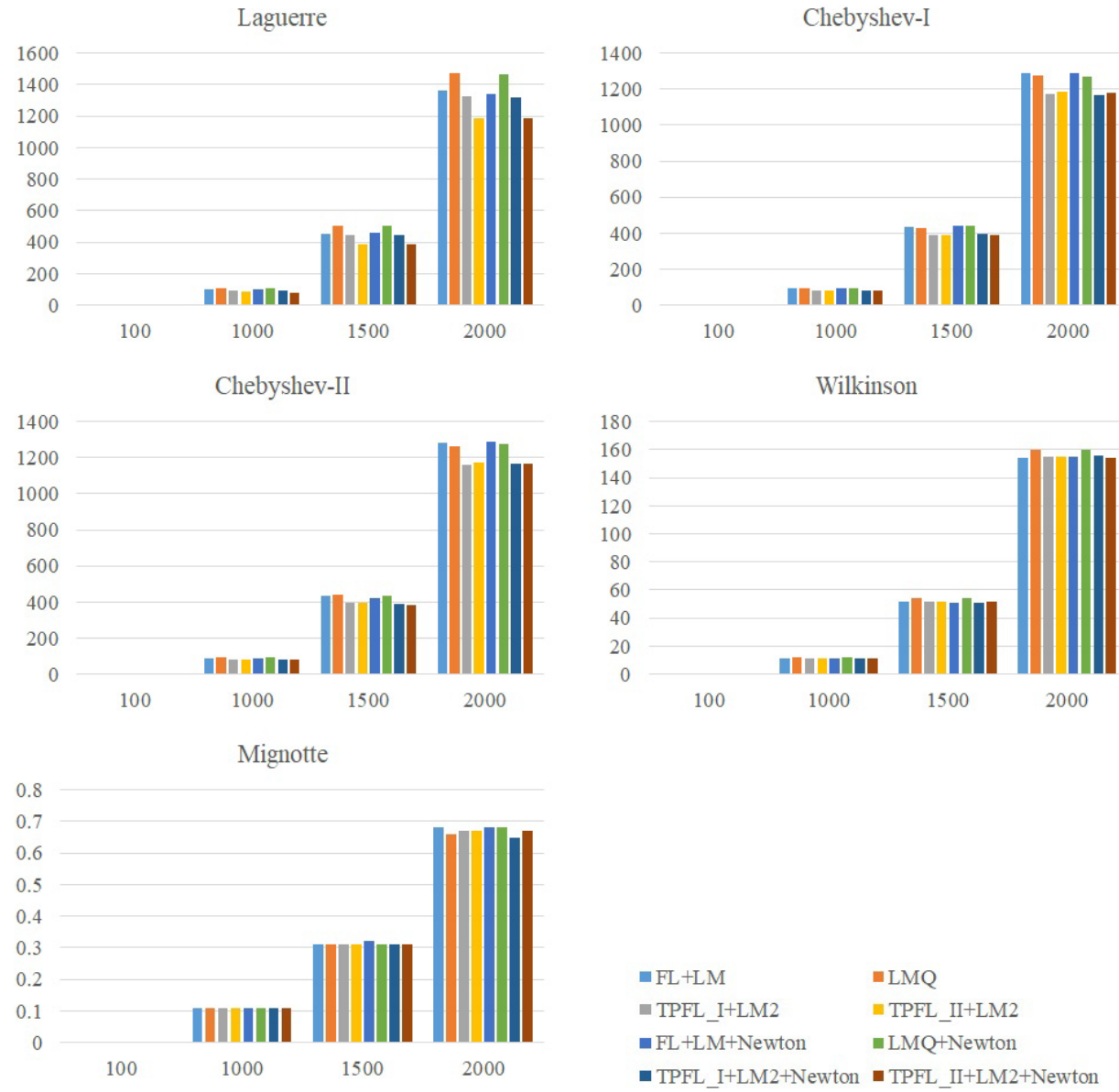


Fig. 1 Execution time for special polynomials [sec.].

equations, and **Fig. 2** shows that for random polynomial equations. Here, these vertical axes represent the computation time, and the horizontal axes correspond to degrees. Though the computational complexity of the CF method and those bounds is $O(n)$ or $O(n^2)$, the computational time becomes very long as the degree increases. Thus, the improvement of the lower bound is important. In Fig. 2, the lines drawn on the bar graph represent the maximum and minimum values. By Fig. 1 and Fig. 2, the computation times in TPFL_II+LM2 and TPFL_II+LM2+Newton are more often shorter than that in the others.

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 1.15, and for the Chebyshev-I and -II polynomials it is about 1.08 and 1.08 times, respectively. TPFL-II+LM2 is

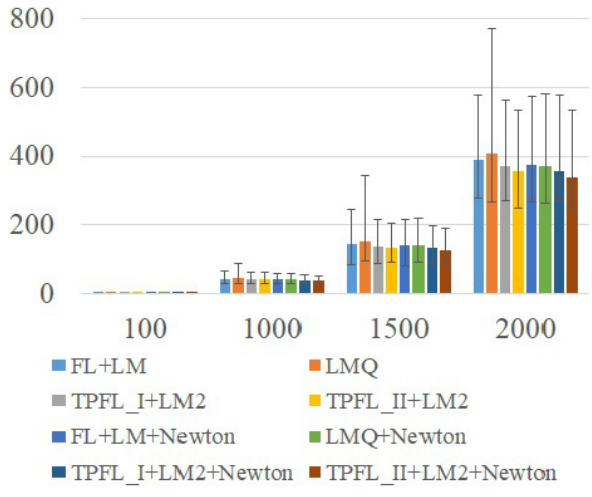


Fig. 2 Execution time for random polynomials [sec.].

Table 2 Execution time for special polynomials.

Polynomial Class	Degree	Time [sec.]							
		FL +LM	LMQ	TPFL-I +LM2	TPFL-II +LM2	FL +LM +Newton	LMQ +Newton	TPFL-I +LM2 +Newton	TPFL-II +LM2 +Newton
Laguerre	100	0.03	0.04	0.03	0.03	0.04	0.04	0.03	0.03
Laguerre	1,000	96.68	107.51	93.88	81.57	97.13	107.77	93.97	81.08
Laguerre	1,500	453.84	501.13	442.54	385.46	457.76	501.06	442.34	388.71
Laguerre	2,000	1,364.65	1,474.83	1,328.50	1,187.06	1,341.79	1,464.29	1,316.48	1,185.34
Chebyshev-I	100	0.04	0.04	0.03	0.03	0.04	0.04	0.03	0.03
Chebyshev-I	1,000	91.56	92.15	81.78	81.70	91.34	91.43	80.22	80.72
Chebyshev-I	1,500	436.02	429.64	387.35	389.72	439.02	438.68	395.56	388.59
Chebyshev-I	2,000	1,287.97	1,279.02	1,173.40	1,184.46	1,291.24	1,271.36	1,170.00	1,176.89
Chebyshev-II	100	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
Chebyshev-II	1,000	90.35	91.53	79.04	80.36	89.56	91.23	80.10	79.03
Chebyshev-II	1,500	431.71	442.45	396.36	394.67	423.09	432.00	388.06	385.69
Chebyshev-II	2,000	1,284.83	1,261.85	1,160.02	1,172.12	1,290.67	1,275.41	1,167.55	1,166.38
Wilkinson	100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Wilkinson	1,000	11.12	12.00	11.14	11.11	11.05	11.98	11.07	11.02
Wilkinson	1,500	51.45	54.56	51.46	51.42	51.00	54.07	50.93	51.48
Wilkinson	2,000	154.42	160.06	154.92	154.68	155.15	159.98	155.80	154.11
Mignotte	100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mignotte	1,000	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
Mignotte	1,500	0.31	0.31	0.31	0.31	0.32	0.31	0.31	0.31
Mignotte	2,000	0.68	0.66	0.67	0.67	0.68	0.68	0.65	0.67

Table 3 Execution time for random polynomials.

Parameters	Degree	Time [sec.]							
		FL +LM	LMQ	TPFL-I +LM2	TPFL-II +LM2	FL +LM +Newton	LMQ +Newton	TPFL-I +LM2 +Newton	TPFL-II +LM2 +Newton
Ave.									
$s = 40$ $r = 20$	100	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
$s = 490$ $r = 20$	1,000	43.24	46.32	41.22	40.64	41.77	41.86	39.96	38.11
$s = 740$ $r = 20$	1,500	144.31	152.38	137.67	132.12	141.95	142.45	134.08	127.29
$s = 990$ $r = 20$	2,000	389.94	407.25	370.77	357.20	376.79	371.99	358.18	336.99
Max.									
$s = 40$ $r = 20$	100	0.06	0.06	0.05	0.05	0.05	0.06	0.05	0.05
$s = 490$ $r = 20$	1,000	65.17	87.62	61.53	60.61	58.75	57.20	55.41	52.00
$s = 740$ $r = 20$	1,500	244.77	344.02	214.41	205.77	215.31	220.02	197.31	190.44
$s = 990$ $r = 20$	2,000	579.54	772.44	565.13	533.85	574.59	583.78	579.29	536.14
Min.									
$s = 40$ $r = 20$	100	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
$s = 490$ $r = 20$	1,000	30.49	29.75	28.75	27.80	28.73	29.41	27.90	27.94
$s = 740$ $r = 20$	1,500	85.45	94.66	86.37	90.65	81.79	93.31	86.84	84.17
$s = 990$ $r = 20$	2,000	278.80	265.84	271.74	249.39	267.18	262.67	266.51	249.64

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. Additionally, we can confirm that the lower bound generated by Newton's method is effective for random polynomial equations. In Table 3, the maximum and minimum values in TPFL-II+LM2 is almost equal to the time in TPFL-II+LM2+Newton. It is because the Newton bound is not so employed in these cases. In the case of the maximum values, namely, when the input polynomials such as special polynomials in Table 2 are difficult to solve, the Newton bound is light but may *not* be sharp (or, tight) and effective and then other bounds may be employed. In the case of the minimum values, the input polynomials are easy to solve. In the case, Newton bound is also not employed because other bounds work effectively. Therefore, we propose TPFL-II+LM2+Newton as a suitable combination of the lower bounds for accelerating the numerical computation of positive roots of polynomials.

A t–test is presented for the null hypothesis that the experi-

Table 4 t–test.

Degree	p–value
100	0.044934725
1,000	$1.40089*10^{-20}$
1,500	$2.43597*10^{-20}$
2,000	$1.52971*10^{-22}$

mental results obtained from the two different methods are equal. The target methods are LMQ, which is an existing method, and TPFL-II+LM2+Newton, which is a proposed method combining all the methods. **Table 4** shows the p–values for the 100 test polynomial equations. By Table 4, since the p–value is smaller than 0.05, the null hypothesis is rejected and a significant difference is confirmed.

7. Conclusions

In this study, we have proposed a suitable combination of the lower bounds based on the local-max bound, the first-λ bound, and the lower bound generated by Newton's method 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. The reason is as follows: in the CF method, the coefficients of the lower order terms tend to be large because the replacement $x \rightarrow x + l\_b$ is needed many times in the CF method. The tail-pairing first-λ bound takes care of the lower order (tail) terms firstly. The feature makes tail-pairing first-λ bound work more effectively than first-λ bound. Here, l_b is generally an integer larger than 2. Thus, the above can be confirmed by considering the relationship between the order and the coefficients in the binomial expansion. The numerical results show that the average execution time of the CF method with both the local-max2 bound, the tail-pairing first-λ bound, and the lower bound generated by Newton's method 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] Akritas, A., Strzeboński, A. and Vigklas, P.: Implementations of a new theorem for computing bounds for positive roots of polynomials, *Computing*, Vol.78, pp.355–367 (2006).

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

[3] Collins, G.: Quantifier Elimination for the Elementary Theory of Real Closed Fields by Cylindrical Algebraic Decomposition, *Lect Notes Comput. Sci.*, Vol.33, pp.134–183 (1975).

[4] Collins, G. and Akritas, A.: Polynomial real root isolation using Descartes' rule of signs, *SYMSAC '76, Proc. 3rd ACM Symposium on Symbolic and Algebraic Computation*, pp.272–275, ACM (1976).

[5] The GNU MP Bignum Library (2013) (online), available from ⟨http://gmplib.org/⟩.

[6] Sage (2013) (online), available from ⟨http://sagemath.org/⟩.

[7] SymPy (2013) (online), available from ⟨http://sympy.org/en/index.html⟩.

[8] Kimura, K., Akiyama, T., Ishigami, H., Takata, M. and Nakamura, Y.: Accelerating the Numerical Computation of Positive Roots of Polynomials using Improved Bounds, *Proc. 2014 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA'14*), Vol.II, pp.201–207 (2014).

[9] Kioustelidis, B.: Bounds for positive roots of polynomials, *J. Comput. Appl. Math.*, Vol.16, No.2, pp.241–244 (1986).

[10] Moore, R.E.: *Interval Analysis*, Prentice Hall, Englewood Cliffs, N.J. (1966).

[11] Obreschkoff, N.: *Verteilung und Berechnung der Nullstellen reeller Polynome*, VEB Deutscher Verlag der Wissenschaften (1963).

[12] Takata, M., Akiyama, T., Araki, S., Kimura, K. and Nakamura, Y.: Improved Computation of Bounds for Positive Roots of Polynomials, *Proc. 2013 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA'13*), Vol.II, pp.168–174 (2013).

[13] Vincent, A.J.H.: Sur la resolution des équations numériques, *J. Math. Pures Appl.*, Vol.1, pp.341–372 (1836).

Appendix

Table A·1 Ratio of each bound adopted in a special polynomial (Laguerre).

Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.510309	-	0.0257732	-	0.463918	-	-	-
	LMQ	0.482759	0.517241	-	-	-	-	-	-
	TPFL-I+LM2	0.475	-	-	-	-	0.08	-	0.445
	TPFL-II+LM2	0.552486	-	-	-	-	-	0.0497238	0.39779
	FL+LM+Newton	0.510309	-	0.0257732	0	0.463918	-	-	-
	LMQ+Newton	0.482759	0.517241	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.475	-	-	0	-	0.08	-	0.445
	TPFL-II+LM2+Newton	0.552486	-	-	0	-	-	0.0497238	0.39779
1,000	FL+LM	0.540261	-	0.023395	-	0.436344	-	-	-
	LMQ	0.484666	0.515334	-	-	-	-	-	-
	TPFL-I+LM2	0.53148	-	-	-	-	0.0839478	-	0.384572
	TPFL-II+LM2	0.667354	-	-	-	-	-	0.0756014	0.257045
	FL+LM+Newton	0.540261	-	0.023395	0	0.436344	-	-	-
	LMQ+Newton	0.484666	0.515334	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.53148	-	-	0	-	0.0839478	-	0.384572
	TPFL-II+LM2+Newton	0.667354	-	-	0	-	-	0.0756014	0.257045
1,500	FL+LM	0.528397	-	0.0176132	-	0.45399	-	-	-
	LMQ	0.481642	0.518358	-	-	-	-	-	-
	TPFL-I+LM2	0.508561	-	-	-	-	0.0757741	-	0.415665
	TPFL-II+LM2	0.647006	-	-	-	-	-	0.0661305	0.286863
	FL+LM+Newton	0.528397	-	0.0176132	0	0.45399	-	-	-
	LMQ+Newton	0.481642	0.518358	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.508561	-	-	0	-	0.0757741	-	0.415665
	TPFL-II+LM2+Newton	0.647006	-	-	0	-	-	0.0661305	0.286863
2,000	FL+LM	0.537649	-	0.0181473	-	0.444204	-	-	-
	LMQ	0.484521	0.515479	-	-	-	-	-	-
	TPFL-I+LM2	0.5201	-	-	-	-	0.0817854	-	0.398115
	TPFL-II+LM2	0.619202	-	-	-	-	-	0.0687642	0.312034
	FL+LM+Newton	0.537649	-	0.0181473	0	0.444204	-	-	-
	LMQ+Newton	0.484521	0.515479	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.5201	-	-	0	-	0.0817854	-	0.398115
	TPFL-II+LM2+Newton	0.619202	-	-	0	-	-	0.0687642	0.312034

Table A·2 Ratio of each bound adopted in a special polynomial (Chebyshev-I).

Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.525773	-	0.0103093	-	0.463918	-	-	-
	LMQ	0.531915	0.468085	-	-	-	-	-	-
	TPFL-I+LM2	0.547619	-	-	-	-	0.0119048	-	0.440476
	TPFL-II+LM2	0.547619	-	-	-	-	-	0.0119048	0.440476
	FL+LM+Newton	0.525773	-	0.0103093	0	0.463918	-	-	-
	LMQ+Newton	0.531915	0.468085	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.547619	-	-	0	-	0.0119048	-	0.440476
	TPFL-II+LM2+Newton	0.547619	-	-	0	-	-	0.0119048	0.440476
1,000	FL+LM	0.618119	-	0.0137615	-	0.368119	-	-	-
	LMQ	0.597387	0.402613	-	-	-	-	-	-
	TPFL-I+LM2	0.597132	-	-	-	-	0	-	0.402868
	TPFL-II+LM2	0.597132	-	-	-	-	-	0	0.402868
	FL+LM+Newton	0.618119	-	0.0137615	0	0.368119	-	-	-
	LMQ+Newton	0.597387	0.402613	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.597132	-	-	0	-	0	-	0.402868
	TPFL-II+LM2+Newton	0.597132	-	-	0	-	-	0	0.402868
1,500	FL+LM	0.645849	-	0.0114242	-	0.342727	-	-	-
	LMQ	0.615262	0.384738	-	-	-	-	-	-
	TPFL-I+LM2	0.606034	-	-	-	-	0	-	0.393966
	TPFL-II+LM2	0.606034	-	-	-	-	-	0	0.393966
	FL+LM+Newton	0.645849	-	0.0114242	0	0.342727	-	-	-
	LMQ+Newton	0.615262	0.384738	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.606034	-	-	0	-	0	-	0.393966
	TPFL-II+LM2+Newton	0.606034	-	-	0	-	-	0	0.393966
2,000	FL+LM	0.661394	-	0.016403	-	0.322203	-	-	-
	LMQ	0.624319	0.375681	-	-	-	-	-	-
	TPFL-I+LM2	0.608442	-	-	-	-	0	-	0.391558
	TPFL-II+LM2	0.608442	-	-	-	-	-	0	0.391558
	FL+LM+Newton	0.661394	-	0.016403	0	0.322203	-	-	-
	LMQ+Newton	0.624319	0.375681	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.608442	-	-	0	-	0	-	0.391558
	TPFL-II+LM2+Newton	0.608442	-	-	0	-	-	0	0.391558

Table A·3 Ratio of each bound adopted in a special polynomial (Chebyshev-II).

Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.537634	-	0.0215054	-	0.44086	-	-	-
	LMQ	0.54023	0.45977	-	-	-	-	-	-
	TPFL-I+LM2	0.529412	-	-	-	-	0.0235294	-	0.447059
	TPFL-II+LM2	0.529412	-	-	-	-	-	0.0235294	0.447059
	FL+LM+Newton	0.537634	-	0.0215054	0	0.44086	-	-	-
	LMQ+Newton	0.54023	0.45977	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.529412	-	-	0	-	0.0235294	-	0.447059
	TPFL-II+LM2+Newton	0.529412	-	-	0	-	-	0.0235294	0.447059
1,000	FL+LM	0.626424	-	0.00797267	-	0.365604	-	-	-
	LMQ	0.598109	0.401891	-	-	-	-	-	-
	TPFL-I+LM2	0.606299	-	-	-	-	0	-	0.393701
	TPFL-II+LM2	0.606299	-	-	-	-	-	0	0.393701
	FL+LM+Newton	0.626424	-	0.00797267	0	0.365604	-	-	-
	LMQ+Newton	0.598109	0.401891	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.606299	-	-	0	-	0	-	0.393701
	TPFL-II+LM2+Newton	0.606299	-	-	0	-	-	0	0.393701
1,500	FL+LM	0.657165	-	0.0138675	-	0.328968	-	-	-
	LMQ	0.609968	0.390032	-	-	-	-	-	-
	TPFL-I+LM2	0.608621	-	-	-	-	0	-	0.391379
	TPFL-II+LM2	0.608621	-	-	-	-	-	0	0.391379
	FL+LM+Newton	0.657165	-	0.0138675	0	0.328968	-	-	-
	LMQ+Newton	0.609968	0.390032	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.608621	-	-	0	-	0	-	0.391379
	TPFL-II+LM2+Newton	0.608621	-	-	0	-	-	0	0.391379
2,000	FL+LM	0.664512	-	0.0152761	-	0.320212	-	-	-
	LMQ	0.626521	0.373479	-	-	-	-	-	-
	TPFL-I+LM2	0.616393	-	-	-	-	0	-	0.383607
	TPFL-II+LM2	0.616393	-	-	-	-	-	0	0.383607
	FL+LM+Newton	0.664512	-	0.0152761	0	0.320212	-	-	-
	LMQ+Newton	0.626521	0.373479	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.616393	-	-	0	-	0	-	0.383607
	TPFL-II+LM2+Newton	0.616393	-	-	0	-	-	0	0.383607

Table A·4 Ratio of each bound adopted in a special polynomial (Wilkinson).

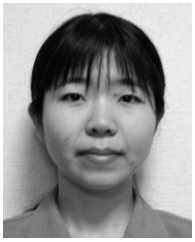
Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.989899	-	0	-	0.010101	-	-	-
	LMQ	1	0	-	-	-	-	-	-
	TPFL-I+LM2	0.989899	-	-	-	-	0	-	0.010101
	TPFL-II+LM2	0.989899	-	-	-	-	-	0	0.010101
	FL+LM+Newton	0.989899	-	0	0	0.010101	-	-	-
	LMQ+Newton	1	0	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.989899	-	-	0	-	0	-	0.010101
	TPFL-II+LM2+Newton	0.989899	-	-	0	-	-	0	0.010101
1,000	FL+LM	0.998999	-	0	-	0.001001	-	-	-
	LMQ	1	0	-	-	-	-	-	-
	TPFL-I+LM2	0.998999	-	-	-	-	0	-	0.001001
	TPFL-II+LM2	0.998999	-	-	-	-	-	0	0.001001
	FL+LM+Newton	0.998999	-	0	0	0.001001	-	-	-
	LMQ+Newton	1	0	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.998999	-	-	0	-	0	-	0.001001
	TPFL-II+LM2+Newton	0.998999	-	-	0	-	-	0	0.001001
1,500	FL+LM	0.999333	-	0	-	0.000667111	-	-	-
	LMQ	1	0	-	-	-	-	-	-
	TPFL-I+LM2	0.999333	-	-	-	-	0	-	0.000667111
	TPFL-II+LM2	0.999333	-	-	-	-	-	0	0.000667111
	FL+LM+Newton	0.999333	-	0	0	0.000667111	-	-	-
	LMQ+Newton	1	0	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.999333	-	-	0	-	0	-	0.000667111
	TPFL-II+LM2+Newton	0.999333	-	-	0	-	-	0	0.000667111
2,000	FL+LM	0.9995	-	0	-	0.00050025	-	-	-
	LMQ	1	0	-	-	-	-	-	-
	TPFL-I+LM2	0.9995	-	-	-	-	0	-	0.00050025
	TPFL-II+LM2	0.9995	-	-	-	-	-	0	0.00050025
	FL+LM+Newton	0.9995	-	0	0	0.00050025	-	-	-
	LMQ+Newton	1	0	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.9995	-	-	0	-	0	-	0.00050025
	TPFL-II+LM2+Newton	0.9995	-	-	0	-	-	0	0.00050025

Table A·5 Ratio of each bound adopted in a special polynomial (Mignotte).

Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.4	-	0	-	0.6	-	-	-
LMQ	0.4	0.6	-	-	-	-	-	-	
	TPFL-I+LM2	0.2	-	-	-	-	0	-	0.8
	TPFL-II+LM2	0.2	-	-	-	-	-	0	0.8
	FL+LM+Newton	0.4	-	0	0	0.6	-	-	-
	LMQ+Newton	0.4	0.6	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.2	-	-	0	-	0	-	0.8
	TPFL-II+LM2+Newton	0.2	-	-	0	-	-	0	0.8
1,000	FL+LM	0.4	-	0	-	0.6	-	-	-
	LMQ	0.4	0.6	-	-	-	-	-	-
	TPFL-I+LM2	0.2	-	-	-	-	0	-	0.8
	TPFL-II+LM2	0.2	-	-	-	-	-	0	0.8
	FL+LM+Newton	0.4	-	0	0	0.6	-	-	-
	LMQ+Newton	0.4	0.6	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.2	-	-	0	-	0	-	0.8
	TPFL-II+LM2+Newton	0.2	-	-	0	-	-	0	0.8
1,500	FL+LM	0.4	-	0	-	0.6	-	-	-
	LMQ	0.4	0.6	-	-	-	-	-	-
	TPFL-I+LM2	0.2	-	-	-	-	0	-	0.8
	TPFL-II+LM2	0.2	-	-	-	-	-	0	0.8
	FL+LM+Newton	0.4	-	0	0	0.6	-	-	-
	LMQ+Newton	0.4	0.6	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.2	-	-	0	-	0	-	0.8
	TPFL-II+LM2+Newton	0.2	-	-	0	-	-	0	0.8
2,000	FL+LM	0.4	-	0	-	0.6	-	-	-
	LMQ	0.4	0.6	-	-	-	-	-	-
	TPFL-I+LM2	0.2	-	-	-	-	0	-	0.8
	TPFL-II+LM2	0.2	-	-	-	-	-	0	0.8
	FL+LM+Newton	0.4	-	0	0	0.6	-	-	-
	LMQ+Newton	0.4	0.6	-	0	-	-	-	-
	TPFL-I+LM2+Newton	0.2	-	-	0	-	0	-	0.8
	TPFL-II+LM2+Newton	0.2	-	-	0	-	-	0	0.8

Table A·6 Ratio of each bound adopted in random polynomials.

Degree	Method	ratio							
		0 bound	LMQ	LM	Newton	FL	TPFL-I	TPFL-II	LM2
100	FL+LM	0.462123	-	0.022208	-	0.525669	-	-	-
	LMQ	0.411793	0.598207	-	-	-	-	-	-
	TPFL-I+LM2	0.377454	-	-	-	-	0.066387	-	0.566158
	TPFL-II+LM2	0.360135	-	-	-	-	-	0.090284	0.559581
	FL+LM+Newton	0.463192	-	0.022431	0.001204	0.523173	-	-	-
	LMQ+Newton	0.41255	0.596237	-	0.001214	-	-	-	-
	TPFL-I+LM2+Newton	0.377069	-	-	0.001154	-	0.066071	-	0.565705
	TPFL-II+LM2+Newton	0.360689	-	-	0.001369	-	-	0.091499	0.556443
1,000	FL+LM	0.480478	-	0.027954	-	0.501568	-	-	-
	LMQ	0.465412	0.544588	-	-	-	-	-	-
	TPFL-I+LM2	0.447046	-	-	-	-	0.069961	-	0.492993
	TPFL-II+LM2	0.462829	-	-	-	-	-	0.104902	0.442269
	FL+LM+Newton	0.497423	-	0.024113	0.014243	0.47422	-	-	-
	LMQ+Newton	0.480076	0.51092	-	0.019004	-	-	-	-
	TPFL-I+LM2+Newton	0.457372	-	-	0.016016	-	0.047278	-	0.489333
	TPFL-II+LM2+Newton	0.486612	-	-	0.01888	-	-	0.082564	0.421944
1,500	FL+LM	0.501306	-	0.024251	-	0.484442	-	-	-
	LMQ	0.462036	0.547964	-	-	-	-	-	-
	TPFL-I+LM2	0.456905	-	-	-	-	0.052323	-	0.500772
	TPFL-II+LM2	0.461266	-	-	-	-	-	0.10205	0.446684
	FL+LM+Newton	0.511473	-	0.024046	0.009817	0.464664	-	-	-
	LMQ+Newton	0.475204	0.520537	-	0.014259	-	-	-	-
	TPFL-I+LM2+Newton	0.461589	-	-	0.011905	-	0.03305	-	0.503457
	TPFL-II+LM2+Newton	0.47086	-	-	0.015639	-	-	0.086116	0.437385
2,000	FL+LM	0.490812	-	0.027281	-	0.491907	-	-	-
	LMQ 0.466728	0.543272	-	-	-	-	-	-	
	TPFL-I+LM2	0.45005	-	-	-	-	0.060527	-	0.499423
	TPFL-II+LM2	0.463252	-	-	-	-	-	0.110232	0.436516
	FL+LM+Newton	0.504082	-	0.027923	0.013191	0.464804	-	-	-
	LMQ+Newton	0.490855	0.500833	-	0.018312	-	-	-	-
	TPFL-I+LM2+Newton	0.456051	-	-	0.014843	-	0.0382	-	0.500906
	TPFL-II+LM2+Newton	0.482208	-	-	0.019193	-	-	0.094456	0.414142



Masami Takata received her Ph.D. degree from Nara Women's University in 2004. She has been a lecturer of the Research Group of Information and Communication Technology for Life at Nara Women's University. Her research interests include numerical algebra and parallel algorithms.

Takuto Akiyama received his B.E. and M.I. degrees from Kyoto University in 2012 and 2014.

Sho Araki received his Ph.D. degree from Kyoto University in 2021. His research interests include parallel algorithms for eigenvalue and singular value decomposition with high accuracy.

Hiroyuki Ishigami received his Ph.D. degree from Kyoto University in 2016. His research interests include parallel algorithms for eigenvalue and singular value decomposition. He is an IPSJ member.

Kinji Kimura received his Ph.D. degree from Kobe University in 2004. He became an assistant professor at Kyoto University in 2006, an assistant professor at Niigata University in 2007, a lecturer at Kyoto University in 2008, and associate professor at Fukui University in 2019.

Yoshimasa Nakamura is a vice-president and professor of Osaka Seikei University and is a professor emeritus of Kyoto University from 2021. His subject is to design new numerical algorithms for singular value decomposition by using discrete-time integrable systems. He is a member of JSIAM and MSJ.