Root-Squaring for Root-Finding*

Pedro Soto[1][0000−0002−7120−7362], Soo Go[2], and Victor Pan[3]

[1] The Graduate Center, CUNY New York, NY, USA psoto@gradcenter.cuny.edu
https://pedrojuansoto.github.io/
[2] The Graduate Center, CUNY New York, NY, USA sgo@gradcenter.cuny.edu
[3] Lehman College, CUNY New York, NY, USA victor.pan@lehman.cuny.edu

Abstract. We revisit the classical root-squaring formula of Dandelin-Lobachevsky-Gräffe for polynomials and find new interesting applications to root-finding.

Keywords: symbolic-numeric computing · root finding · polynomial algorithms · computer algebra.

1 Introduction

We revisit the famous method for finding roots that was independently discovered by Dandelin, Lobachevsky, and Gräffe and make further progress by applying this identity to the problem of approximating the root radius of a polynomial. (See [**?**] for a history of this problem.) This is a key step in many subdivision based root-finding algorithms.

One of the novelties of our algorithm is the Rational Root Tree Algorithm (Alg. 1), which allows one to compute the angles of the iterated 2^{nd} roots in Eq. 3 with exact precision over the rationals, thus greatly reducing the numerical stability of Eq. 3. This improvement is due to the fact the most numerically unstable steps in Eq. 3 is the computation of the roots $x^{\frac{1}{2^m}}$, as shown in Sec. 5 as well as Thm. 3.

Another advantage of our algorithm is that it assumes the black box polynomial model. That is, our algorithm works under the stronger assumption that one only computes $p(x)$ via some oracle. We give both theoretical guarantees in Sec. 5 and empirical evidence in Sec. 6 that our algorithm performs well.

2 Related Works

The main method for root-finding by root-squaring was independently discovered by Dandelin, Lobachevsky, and Gräffe in the 19^{th} century over the course of 10 years (See [**?**]). Two of the first works to consider algorithms based on the formulae for modern computers were [**?**] and [**?**]; the later of which gave explicit pseudocode for the a root-finding algorithm that uses Eq. 1. The work [**?**] also

* Supported by organization x.

considers root-finding using Eq. 1 and further state that the DLG method becomes very unstable for $\ell > 2$; by contrast, as we will see in Sec. 6, our algorithm performs reasonably well for ℓ as large as 12. The same authors went on to design a variation of the DLG root-finding algorithm that makes use of different renormalizations and other preprocessing transformations for DLG root finding in [**?**], and the work also proves convergence results for their DLG iterative algorithm.

The authors of [**?**] consider DLG based algorithms for the solutions of fractional-order polynomials, *i.e.,* generalized polynomials that have rational exponents. In [**?**] the authors use apply the DLG formulae to root counting. The authors of [**?**] apply the DLG formulae to solving polynomials over finite fields. The authors of [**?**] apply DLG iterations to the benchmark problem (*i.e.*, the isolsation of the roots of a polynomials, See Sec. 3) to improve upon the record bound in [**?**] which at the time was unbroken for 14 years.

This short survey illustrates that the application of the DLG to root-finding, while more than a century old, is still an interesting research topic. One of the novelties and improvements of our algorithms is to consider the identities given by Eq. 2, instead of the usual Eq.1, to approximate the root radius. Furthermore, we show that, contrary to intuition, our algorithm is numerically stable and well-behaved when taking the limit of Eq. 2 to 0.

3 Background and Motivation

We set up some basic notation and background: we discuss Dandelin, Lobachevsky, and Gräffe's Formulae in Sec. 3.1, properties of the extremal root radii in Sec. 3.2, and well-known estimates for extremal root radii in Sec. 3.3. The reverse of a polynomial $p := p_0 + p_1 x^1 + ... + p_d x^d$ is defined as $p_{\text{rev}} := p_d + p_{d-1} x^1 + ... + p_0 x^d$ and for a polynomial $p(x) = \prod_{i \in [d]} (x - \rho_i)$ where $|\rho_d| \geq ... \geq |\rho_1|$ we define $r_i(c, p) = |\rho_i - c|$ as the root radii p about c. We are particular interested in the case where $c = 0$ since we can always reduce to this case (see Sec. 3.2).

3.1 Dandelin, Lobachevsky, and Gräffe's Formulae

The common procedure for root radii approximation based on the classical technique of recursive root-squaring is to first make an input polynomial $p(x)$ monic by scaling it and/or the variable x and then apply k DLG (that is, Dandelin's aka Lobachevsky's or Gräffe's) root-squaring iterations (cf. [**?**]),

$$p_0(x) = \frac{1}{p_d} p(x),\ p_{i+1}(x) = (-1)^d p_i(\sqrt{x}) p_i(-\sqrt{x}),\ i = 0, 1, \ldots . \ell \qquad (1)$$

for a fixed positive integer ℓ (see Remark 1 below). The ith iteration squares the roots of $p_i(x)$ and consequently the root radii from the origin, as well as the isolation of the unit disc $D(0, 1)$. Then one approximates the ratio ρ_+/ρ_-, the new scaled ratio of the root radii, for the polynomial $p_\ell(x)$ within a factor of γ and readily recovers the approximation of this ratio for $p_0(x)$ and $p(x)$ within a factor of $\gamma^{1/2^\ell}$.

Given the coefficients of $p_i(x)$ we can reduce the ith root-squaring iteration, that is, the computation of the coefficients of $p_{i+1}(x)$, to polynomial multiplication and perform it in $O(d\log(d))$ arithmetic operations. Unless the positive integer ℓ is small, the absolute values of the coefficients of $p_\ell(x)$ vary dramatically, and realistically one should either stop because of severe problems of numerical stability or apply the stable algorithm by Gregorio Malajovich and Jorge P. Zubelli [**?**], which performs a single root-squaring at arithmetic cost of order d^2.

For a black box polynomial $p(x)$, however, we apply DLG iterations without computing the coefficients, and the algorithm turns out to be quite efficient: for ℓ iterations evaluate $p(x)$ at 2^ℓ equally spaced points on a circle and obtain the values of the polynomial $p_\ell(x) = \prod \left(x - x_j^{2^\ell}\right)$ at these 2^ℓ points.

Furthermore, we evaluate the ratio $p'(x)/p(x) = p_0'(x)/p_0(x)$ at these points by applying the recurrence

$$\frac{p_{i+1}'(x)}{p_{i+1}(x)} = \frac{1}{2\sqrt{x}}\left(\frac{p_i'(\sqrt{x}\ )}{p_i(\sqrt{x}\ )} - \frac{p_i'(-\sqrt{x}\ )}{p_i(-\sqrt{x}\ )}\right),\ i = 0, 1, \dots \tag{2}$$

Recurrences (1) and (2) reduce the evaluation of $p_\ell(c)$ to the evaluation of $p(c)$ at $q = 2^\ell$ points $c^{1/q}$ and for $c \neq 0$ reduce the evaluation of the ratio $p_\ell'(x)/p_\ell(x)$ at $x = c$ to the evaluation of the ratio $p'(x)/p(x)$ at the latter $q = 2^\ell$ points $x = c^{1/q}$. We will see that we can apply recurrence (2) to support fast convergence to the convex hull of the roots.

For $x = 0$, the recurrence (2) can be specialized as follows:

$$\frac{p_\ell'(0)}{p_\ell(0)} = \left(\frac{p_{\ell-1}'(x)}{p_{\ell-1}(x)}\right)'_{x=0} = \left(\frac{p'(x)}{p(x)}\right)^{(\ell)}_{x=0},\ \ell = 1, 2, \dots \tag{3}$$

Notice an immediate extension:

$$\frac{p_\ell^{(h)}(0)}{p_\ell(0)} = \prod_{g=1}^{h}\left(\frac{p^{(g)}(x)}{p^{(g-1)}(x)}\right)^{(\ell)}_{x=0},\ h = 1, 2, \dots. \tag{4}$$

Equations (3) and more generally (4) enable us to strengthen upper estimates in Eq. 5 & Eq. 6 and more generally Eq. 15 for root radii $r_j(0,p)$ at the origin because $r_j(0,p_\ell) = r_j(0,p)^{2^\ell}$ for $j = 1, \dots, d$ (see Eq. 5 & Eq. 6); we can approximate the higher order derivatives $\left(\frac{p^{(g)}(x)}{p^{(g-1)}(x)}\right)^{(\ell)}$ at $x = 0$ by following Remark. 2. Besides the listed applications of root-squaring, one can apply DLG iterations to randomized exclusion tests for sparse polynomials. One can apply root-squaring $p(x) \mapsto p_\ell(x)$ to improve the error bound for the approximation of the power sums of the roots of $p(x)$ in the unit disc $D(0,1)$ by Cauchy sums, but the improvement is about as much as the additional cost incurred by increasing the number q of points of evaluation of the ratio $\frac{p'}{p}$.

Remark 1. One can approximate the leading coefficient p_d of a black box polynomial $p(x)$. This coefficient is not involved in recurrence (2), and one can apply recurrence (1) by using a crude approximation to p_d and if needed can scale polynomials $p_i(x)$ for some i.

3.2 Extremal root radii

We cover some known estimates for extremal root radii in Sec. 3.3. The most used estimates (Eq. 15 for $i = 1$ and $c = 0$) readily follow from straightforward algebraic manipulations of $p_{\rm rev} := p_d + p_{d-1}x^1 + ... + p_0x^d$:

$$r_d(c, p) \le d \left| \frac{p(c)}{p'(c)} \right| \tag{5}$$

and

$$r_1(c, p) \ge \left| \frac{p'_{\rm rev}(c)}{dp_{\rm rev}(c)} \right|. \tag{6}$$

We strengthen these estimates in the case of $c = 0$ by recalling DLG root-squaring iterations Eq. 2 of Sec. 3.1 and Equation 3. This immediately implies the following extension of Eq. 5 & Eq. 6 for $c = 0$ and all non-negative integers ℓ :

$$r_d(0, p)^{2^\ell} \le d \left| \left(\frac{p'(x)}{p(x)} \right)^{(\ell)}_{x=0} \right|^{-1} \tag{7}$$

and

$$r_1(0, p)^{2^\ell} \ge d^{-1} \left| \left(\frac{p'_{\rm rev}\ (x)}{p_{\rm rev}\ (x)} \right)^{(\ell)}_{x=0} \right| \tag{8}$$

Remark 2. Given a complex c and a positive integer ℓ one can approximate the values at $x = c$ of $(p(x)/p'(x))^{(\ell)}$ for a black box polynomial $p(x)$ by using divided differences, by extending expressions $p^{(i+1)}(c) = \lim_{x\to c} \frac{p^{(i)}(x)-p^{(i)}(c)}{x-c}$ for $i = 0, 1, \dots, \ell - 1$ and based on the mean value theorem [**?**].

For $\ell = 1$ one can instead apply the algorithm supporting the following elegant theorem and implicit in its constructive proof; the complexity of straightforward recursive extension to $\ell = 2, 3, \dots$ increases exponentially in ℓ.

Theorem 1. *Given an algorithm that evaluates a black box polynomial $p(x)$ at a point x over a field $\mathcal{K}$ of constants by using A additions and subtractions, S scalar multiplications (that is, multiplications by elements of the field $\mathcal{K}$), and M other multiplications and divisions, one can extend this algorithm to the evaluation at x of both $p(x)$ and $p'(x)$ by using $2A + M$ additions and subtractions, $2S$ scalar multiplications, and $3M$ other multiplications and divisions.*

Proof. [**?**] and [**?**] prove the theorem for any function $f(x_1, \dots, x_s)$ that has partial derivatives in all its s variables $x_1, \dots, x_s$.

Next we prove that estimates 7 and 8 are extremely poor for worst case inputs.

Theorem 2. *The ratios* $\left|\frac{p(0)}{p'(0)}\right|$ *and* $\left|\frac{p_{\rm rev}(0)}{p'_{\rm rev}(0)}\right|$ *are infinite for* $p(x) = x^d - h^d$, *while* $r_d(c,p) = r_1(c,p) = r_d\,(c, p_{rev}\,) = r_1\,(c, p_{rev}\,) = |h|$. *Proof. Observe that the roots* $x_j = h\exp\left(\frac{(j-1)\mathbf{i}}{2\pi d}\right)$ *of* $p(x) = x^d - h^d$ *for* $j = 1, 2, \dots, d$ *are the* d *th roots of unity up to scaling by* h.

Clearly, the problem persists at the points x where $p'(x)$ and $p'_{\rm rev}\,(x)$ vanish; rotation of the variable $\mathcal{R}_a : p(x) \mapsto t(x) = p(ax)$ for $|a| = 1$ does not fix it but shifts $\mathcal{T}_c : p(x) \mapsto t(x) = p(x - c)$ for $c \neq 0$ can fix it, thus enhancing the power of estimates 7 and 8. Furthermore,

$$\frac{1}{r_d(c,p)} \le \frac{1}{d}\left|\frac{p'(c)}{p(c)}\right| = \frac{1}{d}\left|\sum_{j=1}^{d}\frac{1}{c - x_j}\right|$$

by virtue of inequalities 5 & 6, and so the approximation to the root radius $r_d(c,p)$ is poor if and only if severe cancellation occurs in the summation of the d roots, and similarly for the approximation of $r_1(c,p)$. Such cancellations only occur for a narrow class of polynomials $p(x)$, with a low probability under random root models, although it occurs for the polynomials $p(x) = x^d - h^d$ of Thm. 5 and with high probability for random polynomials of [**?**].

3.3 Classical estimates for extremal root radii

Next we recall some non-costly estimates known for the extremal root radii $r_1 = r_1(0,p)$ and $r_d = r_d(0,p)$ in terms of the coefficients of p (cf. [**?**], [**?**], and [**?**]) and the two parameters

$$\tilde{r}_- := \min_{i\ge 1}\left|\frac{p_0}{p_i}\right|^{\frac{1}{i}}, \tilde{r}_+ := \max_{i\ge 1}\left|\frac{p_{d-i}}{p_d}\right|^{\frac{1}{i}} \tag{9}$$

These bounds on r_1 and r_d hold in dual pairs since $r_1(0,p)r_d\,(0, p_{\rm rev}\,) = 1$. Furthermore, we have that

$$\frac{1}{d}\tilde{r}_+ \le r_1 < 2\tilde{r}_+, \frac{1}{2}\tilde{r}_- \le r_d \le d\tilde{r}_-, \tag{10}$$

$$\tilde{r}_+\sqrt{\frac{2}{d}} \le r_1 \le \frac{1+\sqrt{5}}{2}\tilde{r}_+ < 1.62\tilde{r}_+ \text{ if } p_{d-1} = 0, \tag{11}$$

$$0.618\tilde{r}_- < \frac{2}{1+\sqrt{5}}\tilde{r}_- \le r_d \le \sqrt{\frac{d}{2}}\tilde{r}_- \text{ if } p_1 = 0, \tag{12}$$

$$r_1 \le 1 + \sum_{i=0}^{d-1}\left|\frac{p_i}{p_d}\right|, \frac{1}{r_d} \le 1 + \sum_{i=1}^{d}\left|\frac{p_i}{p_0}\right|. \tag{13}$$

$M(p) := |p_d| \max_{j=1}^{d} \{1, |x_j|\}$ is said to be the Mahler measure of p, and so $M(p_{\text{rev}}) := |p_0| \max_{j=1}^{d} \left\{1, \frac{1}{|x_j|}\right\}$. It holds that

$$r_1^2 \leq \frac{M(p)^2}{|p_d|} \leq \max_{i=0}^{d-1} \left|\frac{p_i}{p_d}\right|^2, \frac{1}{r_d^2} \leq \frac{M(p_{\text{rev}})^2}{|p_0|^2} \leq \max_{i=1}^{d} \left|\frac{p_i}{p_0}\right|^2 \tag{14}$$

It is shown in [**?**] that we can get a very fast approximation of all root radii of p at the origin at a very low cost, which complements the estimates 9, 10 11, 12, 13, and 14.

One can extend all these bounds to the estimates for the root radii $r_j(c,p)$ for any fixed complex c and all j by observing that $r_j(c,p) = r_j(0,t)$ for the polynomial $t(x) = p(x-c)$ and applying Taylor's shift; *i.e.,* applying the mapping $\mathcal{S}_{c,\rho} : p(x) \mapsto p\left(\frac{x-c}{\rho}\right)$.

The algorithms in Sec. 4 closely approximate root radii $r_j(c,p)$ for a black box polynomial p and a complex point c at reasonably low cost, but the next well-known upper bounds on r_d and lower bounds on r_1 (cf. [**?**],[**?**],[**?**],[**?**], and [**?**]) are computed at even a lower cost, defined by a single fraction $\frac{p_0}{p_i}$ or $\frac{p_{d-i}}{p_d}$ for any i, albeit these bounds are excessively large for the worst case input. Finally, we have that $r_d \leq \rho_{i,-} := \left(\binom{d}{i}\left|\frac{p_0}{p_i}\right|\right)^{\frac{1}{i}}, \frac{1}{r_1} \leq \frac{1}{\rho_{i,+}} := \left(\binom{d}{i}\left|\frac{p_d}{p_{d-i}}\right|\right)^{\frac{1}{i}}$ and therefore, since $p^{(i)}(0) = i!p_i$ for all $i > 0$, that

$$r_d \leq \rho_{i,-} = \left(i!\binom{d}{i}\left|\frac{p(0)}{p^{(i)}(0)}\right|\right)^{\frac{1}{i}}, \frac{1}{r_1} \leq \frac{1}{\rho_{i,+}} = \left(i!\binom{d}{i}\left|\frac{p_{\text{rev}}(0)}{p_{\text{rev}}^{(i)}(0)}\right|\right)^{\frac{1}{i}} \tag{15}$$

for all i; from which we obtain relations 5 and 6 for $i = 1$.

4 Algorithm Design

In this section we present the design of the general algorithm for approximating the root radius given by Eq. 3. There are two main steps: 1) going down the rational root tree, *i.e.,* performing Alg. 1, and 2) going up the rational root tree, *i.e.,* performing Alg. 3. The going-down is depicted in Fig. 1 and the going-up is depicted in Fig. 3. The other procedures are simple bookkeeping/preprocessing steps in between the two main going-down/going-up steps. In particular we 1) first convert a complex number x into polar coordinates r, θ where $\theta \approx \frac{p}{q} = \frac{p}{2^\epsilon}$, *i.e.,* perform Alg. 4, 2) perform the first going-down pass which gives the rational angles for the roots of x in fraction form, *i.e.,* perform Alg. 1, 3) perform the second pass of the going-down algorithm where we compute the values $|x|^{\frac{1}{2^m}} \exp(2\pi i \frac{p}{q})$ at the m^{th} level, and 4) finally compute the values given by Eq. 2 going back up the rational root tree.

The intuition behind Alg. 1 is that the square root operation satisfies

$$p\%q \neq 0 \implies \sqrt{\exp\left(2\pi i \frac{p}{q}\right)} = \exp\left(2\pi i \frac{p}{2q}\right) \tag{16}$$

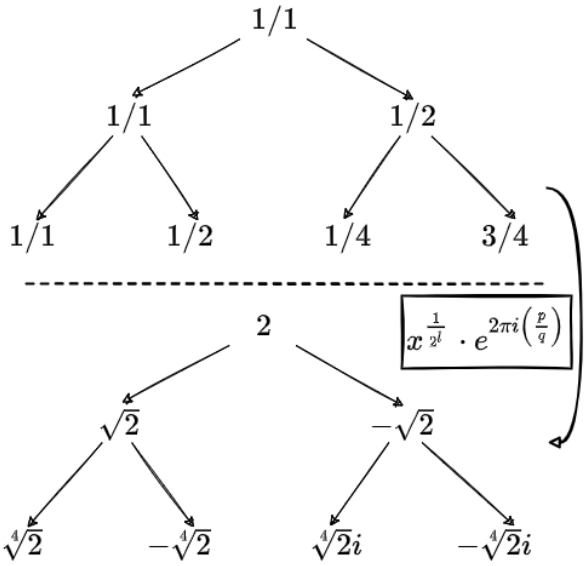

Fig. 1. The upper tree depicts the steps of CIRCLE_ROOTS_RATIONAL_FORM(p, q, l) in Alg.1 for $l = 2$, $p = 1$, and $q = 1$. The lower tree depicts the steps of ROOTS(r, t, u, l) in Alg.2 for $r = 2$, $l = 2$, $p = 1$, and $q = 1$

Fig. 2.

and

$$p\%q = 0 \implies \sqrt{\exp\left(2\pi i\frac{1}{1}\right)} = \exp\left(2\pi i\frac{1}{1}\right) = 1, \tag{17}$$

and the negation operation satisfies

$$p\%q \neq 0 \implies -\exp\left(2\pi i\frac{r}{s}\right) = \exp\left(2\pi i\frac{2r+s}{2s}\right) \tag{18}$$

and

$$p\%q = 0 \implies \sqrt{\exp\left(2\pi i\frac{1}{1}\right)} = \exp\left(2\pi i\frac{1}{2}\right) = -1; \tag{19}$$

therefore, the first four lines of Alg. 1 compute the (angle of the) positive square root, $\sqrt{x}$, of complex number on the unit circle and the next four lines Alg. 1 computes the (angle of the) negative square root, $-\sqrt{x}$. Therefore, we have:

Theorem 3. *For a complex number x with a rational angle $\frac{p}{q}$,* i.e., $x = |x| \exp\left(2\pi i\frac{p}{q}\right)$, *Alg. 1 correctly computes the roots in Eq. 2. In particular, for a rational angle in fractional form, it does so with exact precision.*

Proof. Equations 16, 17, 18, and 19 give the base case and the theorem follows by a straightforward induction.

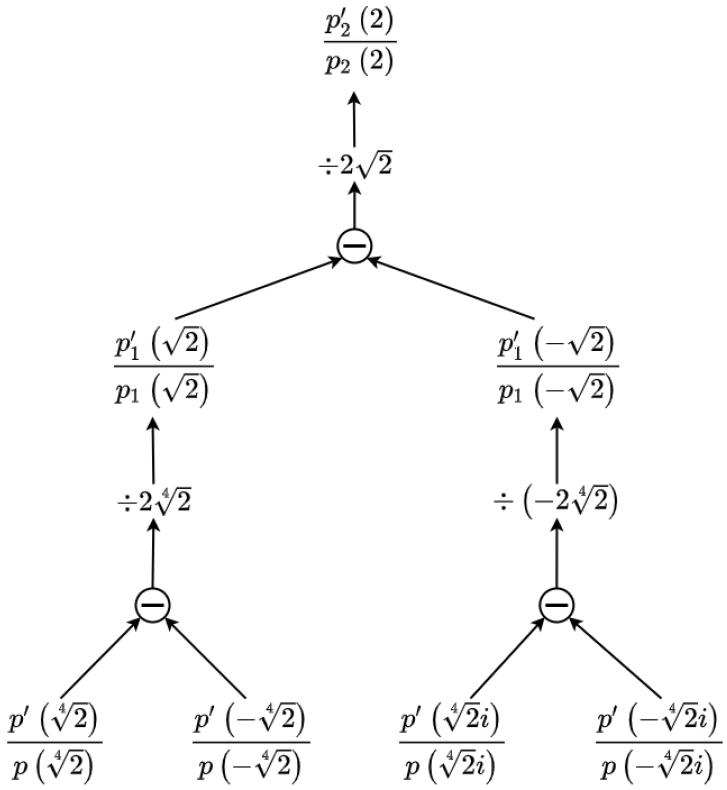

Fig. 3. The steps of DLG_Rational_Form(p, p', r, t, u, l) in Alg.3 for $r = 2$, $l = 2$, $t = 1$, and $u = 1$.

Fig. 4.

Alg. 2 and Alg. 4 are both straightforward, so for the rest of this section we focus on the intuition behind Alg. 3.

Alg. 3 is a dynamic programming on Equation 2. Therefore, by Thm. 3, we have that Alg. 3 correctly computes Eq. 2. Specifically, Alg. 3 does the following: 1) it computes the last layer of the recursion Eq. 2 (*i.e.,* it computes $p'(x^{\frac{1}{2^l}})/p(x^{\frac{1}{2^l}})$) and then 2) it recursively applies Eq. 2 via dynamic programming until it finally computes $\frac{p_\ell'(x)}{p_\ell(x)}$ which is the desired quantity. The former is given by the line "base_step[i] := $\frac{p'(r_i)}{p(r_i)}$" and the latter is given by the line "diff[$i+1$][j]:=$\frac{1}{2}\frac{\text{diff}[i][2j]-\text{diff}[i][2j+1]}{\text{root}[2j]}$" in Alg. 3; the desired output (*i.e.,* $p'(x^{\frac{1}{2^l}})/p(x^{\frac{1}{2^l}})$) is given by the line "diff[l][0]".

The final step in the algorithm is to use Alg. 4 to compute root radius approximations r_d and r_1. The procedure is given by Alg. 5. The rationale for generating a random x is that there may be roots close to 0 and thus, by taking

Algorithm 1 CIRCLE_ROOTS_RATIONAL_FORM(p, q, l)

if $p\%q == 0$ **then**
  $r, s := (1,1)$
else
  $r, s := (p,2q)$
end if
if r%s $== 0$ **then**
  $t, u := (1,2)$
else
  $t, u := (2r + s, 2s)$
end if
if $l == 1$ **then**
  return $[(r, s),(t, u)]$
else if l != 0 **then**
  left := CIRCLE_ROOTS_RATIONAL_FORM$(r, s, l - 1)$
  right := CIRCLE_ROOTS_RATIONAL_FORM$(t, u, l - 1)$
  return left $\cup$ right
else
  return $[(p, q)]$
end if

Algorithm 2 ROOTS(r, t, u, l)

root_tree = CIRCLE_ROOTS_RATIONAL_FORM(p, q, l)
circ_root = $[\exp\left(2 \cdot \pi \cdot i \cdot \frac{r}{s}\right)$ for r, s in root_tree$]$
roots =$[\sqrt[2^l]{r}$·root for root in circ_root$]$
return roots

Algorithm 3 DLG_RATIONAL_FORM(p, p', r, t, u, l)

root := ROOTS(r, t, u, l)
for $r_i \in$ root **do**
  base_step$[i]$:= $\frac{p'(r_i)}{p(r_i)}$
end for
diff[0] := base_step
for $i \leq l$ **do**
  for $j \leq 2^{l-i-1}$ **do**
    diff$[i + 1][j]$:=$\frac{1}{2}\frac{\text{diff}[i][2j]-\text{diff}[i][2j+1]}{\text{root}[2j]}$
    root = roots$(r, t, u, l - 1 - i)$
  end for
end for
return diff$[l][0]$

Algorithm 4 DLG(p, p', l, x, ϵ)

angle := $\frac{1}{2\pi i}\log(x)$
$u := 2^{\epsilon}$
$t := (\text{angle} \cdot u)\%1$
$r := |x|$
return DLG_RATIONAL_FORM(p, p', r, t, u, l)

Algorithm 5 DLG_ROOT_RADIUS$(p, p', p_{\text{rev}}, p'_{\text{rev}}, l, \epsilon, \delta)$

Uniformly Randomly Generate x in the unit circle
$d := \deg(p)$
$r_{\min} := d/\text{DLG}(p, p', l, x \cdot 2^{-\delta}, \epsilon)$
$r_{\max} := \text{DLG}(p_{\text{rev}}, p'_{\text{rev}}, l, x \cdot 2^{-\delta}, \epsilon)/d$

the limit in certain directions, we avoid these possible poles; in particular, we have that

$$\lim_{\delta,\epsilon\to\infty} \text{DLG}(p, p', l, x \cdot 2^{-\delta}, \epsilon) = \frac{p'_\ell(0)}{p_\ell(0)} = \Big(\frac{p'_{\ell-1}(x)}{p_{\ell-1}(x)}\Big)'_{x=0} = \Big(\frac{p'(x)}{p(x)}\Big)^{(\ell)}_{x=0}, \quad (20)$$

for any x, if $p(0) \neq 0$ (See Thm 6).

5 Theoretical Analysis

We now give some theoretical guarantees: Thm. 4 and Thm. 5 give computational complexity bounds, and Thm. 6 proves the correctness of Alg. 5.

Theorem 4. *Alg. 3 performs q floating point subtractions, divisions, and multiplications and q applications of* sin *and* cos*, where $q = 2^l$; furthermore, Alg. 3 performs at most Cq integer additions, "multiplications-by-2", and $\%2^\epsilon$ (*i.e., *mod 2^ϵ) operations, where $C = 1, 3, 2$ respectively.*

Proof. Looking at Fig. 1 and Fig. 3, we can see that the computational tree for the Alg. 3 is a binary tree with $2^l = q$ nodes; the proof for the constants $C = 1, 3, 2$ follows similarly follows from the inspection of the operations performed in Alg. 1.

Theorem 5. *The Cq integer additions, "multiplications-by-2", and $\%2^\epsilon$ (*i.e., *mod 2^ϵ) operations in Alg. 3 have negligible overhead. More precisely, integer additions are always additions of $2\,\epsilon \log \ell$-bit integers and "multiplications-by-2" and $\%2^\epsilon$ (*i.e., *mod 2^ϵ) operations have constant time overhead.*

Proof. Since Alg. 4 always passes in a denominator which is a power of two all of the integer $\%$ and $\cdot$ operations are in fact "multiplications-by-2", and $\%2^\epsilon$ (*i.e.,* mod 2^ϵ) operations by a straightforward proof similar to the one in Alg. 3. Thus these operation are essentially constant overhead bit shift operations on a computing machine with binary words. Since $p\%r$ always reduces $p \mapsto 1$, we have that whenever an overflow of more than $\log r$-bits happens in Alg. 1 it gets converted to an $\log r$-bit integer; therefore, it suffices to prove that $\log r$ is bounded by $\epsilon \log \ell$. However, this once again follows by induction on the binary computation tree: since this tree has depth ℓ we see that any denominator is bounded by $\epsilon \log \ell$ by a simple induction.

In order to prove Thm. 6 we must first prove Lem. 1.

Lemma 1. *If $p(0) \neq 0$, then the limit $\lim_{x\to 0} \frac{p'_\ell(x)}{p_\ell(x)}$ is always well-defined.*

Proof. By applying induction on Eq. 1, we have that $p_l(0) \neq 0$ if $p(0) \neq 0$, and thus it suffices to consider the behavior of the numerator in Eq. 2. The case $\ell = 1$ follows from an application of L'Hopital's rule. For $\ell \neq 1$, there are two cases: either $\sqrt{x}$ divides the numerator of $\frac{p'_\ell(\sqrt{x}\,)}{p_\ell(\sqrt{x}\,)}$ or it does not. If it does, then we are done since we can once again apply L'Hopital's rule. Otherwise, we get that $p'_\ell(\sqrt{x}\,) = c_0 + c_1\sqrt{x}\, + ... + c_k(\sqrt{x}\,)^k$ for some c_i with $c_0 \neq 0$. But then we have

$$\begin{aligned}
\frac{p'_{\ell+1}(\sqrt{x}\,)}{p_{\ell+1}(\sqrt{x}\,)} &= \frac{1}{2\sqrt{x}}\Big(\frac{p'_\ell(\sqrt{x}\,)}{p_\ell(\sqrt{x}\,)} - \frac{p'_\ell(-\sqrt{x}\,)}{p_\ell(-\sqrt{x}\,)}\Big) \\
&= \frac{1}{2\sqrt{x}}\frac{b_0c_0 + \sqrt{x}\cdot N_1(\sqrt{x}) - b_0c_0 - \sqrt{x}\cdot N_2(\sqrt{x})}{p'_\ell(\sqrt{x}\,)p'_\ell(-\sqrt{x}\,)} \\
&= \frac{1}{2}\frac{N_1(\sqrt{x}) - N_2(\sqrt{x})}{p'_\ell(\sqrt{x}\,)p'_\ell(-\sqrt{x}\,)},
\end{aligned}$$

for some polynomials N_1 and N_2 with $b_0 = p'_\ell(0)$. Therefore the limit at zero is once again well-defined.

Theorem 6. *If $p(0) \neq 0$, then Alg. 5 computes the bounds given by Equation 5 and Equation 6 with probability 1.*

Proof. By Lem. 1 we have that the limit in Eq. 1 is well defined. Since there are at most finitely many roots for p_ℓ with a high probability Alg. 5 computes the correct approximation to the bounds in Eq. 5 and Eq. 6.

Lemma 2. *The (relative) condition number operator satisfies the following properties:*

1. $\kappa\{f\}(x) = |x\log'(f(x))|$
2. $\kappa\left\{\frac{f}{g}\right\}(x) = ||\kappa\{f\}(x)| - |\kappa\{g\}(x)||$
3. $\kappa\{x^d\}(x) = d$

Proof. This follows from the definition of the condition number.

Theorem 7. *If $p_\ell(0) \neq 0$, then $\frac{p'_\ell(x)}{p_\ell(x)}$ is well-conditioned at any point sufficiently close to 0.*

Proof. The proof of Lem. 1 gives us that $\frac{p'_\ell(x)}{p_\ell(x)}$ is a well behaved rational function at any point close to zero and thus the condition number $\kappa\{\frac{p'_\ell}{p_\ell}\}(x)$ is well defined at this point since the condition number of arithmetic operations of functions are themselves arithmetic operations in those same functions and the conditions number for a polynomial of degree $d \in \mathbb{R}$ is exactly d; therefore, $\kappa\{\frac{p'_\ell}{p_\ell}\}(x)$ has a well defined/bounded condition number in the limit to zero.

Remark 3. Even though Thm. 7 states that $\frac{p'_\ell(x)}{p_\ell(x)}$ is highly stable, in practice, computating this function requires the use of trigonometric functions (*i.e.,* the roots in Alg. 2); therefore, our added precision in Alg.. 1 helps with the instability associated with Alg. 2. Intuitively this helps the instability because the trigonometric functions are the only subroutines in the algorithm that have a non constant condition number and thus, special care must be taken with them.

6 Experimental Results

In this section, we give the results of our experiments where we use DLG iterations to estimate the extremal root radii of the polynomial test suite included as part of the MPSolve package. To measure accuracy, we compute the relative error of the estimates in comparison to the extremal roots obtained by MPSolve using the formula

$$\text{relative error}_{r_i} = \frac{|\tilde{r_i} - r_i|}{|r_i|},$$

where r_i is the root radius found using MPSolve and $\tilde{r_i}$ our estimate. The results overall verify our theoretical analysis that our root radii approximations perform well when $p(x)$ has no roots extremely close to zero whereas the estimates are poor otherwise. Remarkably, despite the fact $x^n - a^n$ being the worst family of input, as follows from Sec. 3, our implementation performs well for the `nroots` and `nrooti` families of polynomials.

All experiments were run using Python 3.7.7 and MPSolve 3.2.1 on MacOS 11.6.1 with 2.8 GHz Dual-Core Intel Core i5 with 8 GB memory. The columns of the tables, in order, are

- d = degree of the input polynomial
- ℓ = number of iterations
- $e = -\log(|x|)$
- mp.dps: the `mpmath` precision level used
- the relative errors for the minimum and maximum root radii
- total runtime,
- the extremal root radii as computed by MPsolve for the particular given polynomial.

The entries '-' in the tables indicate the test was terminated before completion.

7 Conclusion

We present a novel algorithm that uses the DLG formulae to approximate the root radii and give many theoretical and experimental guarantees. In particular, we prove that the algorithm has low time complexity and is well conditioned when there are no roots around 0 (for which we can simply test beforehand). We tested our algorithm against many of the MPSolve benchmarks. For future work we would like to directly implement our algorithm in subdivision root-finders and test the performance.

Table 1. Experimental Data for `chebyshev`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.155	0.0939	0.27	$[0.0785, 0.997]$
40	5	616	332	0.0981	0.0589	1.13	$[0.0393, 0.999]$
80	6	617	334	0.0593	0.0353	5.26	$[0.0196, 1.0]$
160	7	617	334	2.71	0.0205	20.88	$[0.00982, 1.0]$
320	8	617	334	37.6	0.465	88.4	$[0.00491, 1.0]$

Table 2. Experimental Data for `chrma`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
21	4	616	332	0.215	0.139	0.22	$[1.0, 3.17]$
85	6	617	334	0.0369	0.0452	5.63	$[1.0, 3.25]$
341	8	617	334	0.717	0.547	105.48	$[0.884, 3.41]$

Table 3. Experimental Data for `chrma_d`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.16	0.175	0.26	$[1.3, 3.01]$
84	6	617	334	0.0548	0.00507	5.53	$[1.1, 3.06]$
340	8	617	334	0.658	0.699	93.82	$[0.741, 3.11]$

Table 4. Experimental Data for `chrmc`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
22	4	616	332	0.211	0.138	0.3	$[1.0, 3.03]$
342	8	617	334	0.718	0.279	94.64	$[0.897, 4.13]$

Table 5. Experimental Data for `chrmc_d`

d	ℓ	e	MP.- DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
11	3	616	332	0.251	0.242	0.05	$[1.27, 2.8]$
43	5	616	332	0.101	0.0996	1.23	$[1.02, 2.97]$
171	7	617	334	0.268	1.13	22.05	$[0.715, 3.07]$
683	9	619	338	0.164	0.257	374.69	$[0.519, 3.1]$

Table 6. Experimental Data for `curz`

d	ℓ	e	MP.- DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.156	0.854	0.21	$[0.452, 1.15]$
40	5	616	332	0.199	0.776	1.13	$[0.379, 1.26]$
80	6	617	334	0.086	0.227	5.49	$[0.318, 1.34]$
160	7	617	334	0.0385	0.509	20.79	$[0.271, 1.38]$

Table 7. Experimental Data for `easy`

d	ℓ	e	MP.- DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
100	6	617	334	0.12	0.0809	6.35	$[0.949, 0.98]$
200	7	617	334	0.068	0.047	25.57	$[0.971, 0.99]$
400	8	617	334	0.0381	0.0265	104.47	$[0.983, 0.995]$
1600	10	619	338	0.01	0.00	2228.22	$[0.995, 0.999]$
3200	11	619	338	0.00	-	-	$[0.997, 0.999]$

Table 8. Experimental Data for `exp`

d	ℓ	e	MP.- DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
50	5	616	332	3.76	0.126	1.48	$[14.9, 39.4]$
100	6	617	334	0.367	0.0598	7.37	$[28.9, 83.9]$
200	7	617	334	0.714	0.839	28.22	$[56.8, 176.0]$
400	8	617	334	0.965	0.985	107.96	$[113.0, 365.0]$

Table 9. Experimental Data for `geom1`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
10	3	616	332	0.334	0.25	0.05	$[1.0, 1.0\text{E}{+}18]$
15	3	616	332	0.403	1.0	0.08	$[1.0, 1.0\text{E}{+}28]$
20	4	616	332	0.206	1.0	0.28	$[1.0, 1.0\text{E}{+}38]$
40	5	616	332	0.122	1.0	1.18	$[1.0, 1.0\text{E}{+}78]$

Table 10. Experimental Data for `geom2`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
10	3	616	332	0.334	0.25	0.06	$[1.0\text{E}{-}18, 1.0]$
15	3	616	332	$8.09\text{E}{+}4$	0.287	0.06	$[1.0\text{E}{-}28, 1.0]$
20	4	616	332	$2.63\text{E}{+}26$	0.171	0.23	$[1.0\text{E}{-}38, 1.0]$
40	5	616	332	$1.61\text{E}{+}72$	0.109	1.14	$[1.0\text{E}{-}78, 1.0]$

Table 11. Experimental Data for `geom3`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
10	3	616	332	0.334	0.25	0.05	$[9.54\text{E}{-}7, 0.25]$
20	4	616	332	2.41	0.171	0.24	$[9.09\text{E}{-}13, 0.25]$
40	5	616	332	$1.95\text{E}{+}18$	0.109	1.15	$[8.27\text{E}{-}25, 0.25]$
80	6	617	334	$1.83\text{E}{+}45$	0.0662	5.73	$[6.84\text{E}{-}49, 0.25]$

Table 12. Experimental Data for `geom4`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
10	3	616	332	0.334	0.25	0.05	$[4.0, 1.05\text{E}{+}6]$
20	4	616	332	0.206	0.707	0.27	$[4.0, 1.1\text{E}{+}12]$
40	5	616	332	0.122	1.0	1.15	$[4.0, 1.21\text{E}{+}24]$
80	6	617	334	0.0709	1.0	5.28	$[4.0, 1.46\text{E}{+}48]$

Table 13. Experimental Data for `hermite`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.155	0.129	0.28	$[0.245, 5.39]$
40	5	616	332	0.0981	0.0876	1.31	$[0.175, 8.1]$
80	6	617	334	0.0593	0.0555	5.97	$[0.124, 11.9]$
160	7	617	334	0.0348	0.0335	23.56	$[0.0877, 17.2]$
320	8	617	334	2.08	0.785	88.88	$[0.062, 24.7]$

Table 14. Experimental Data for `kam1`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
7	2	615	331	0.368	1.0	0.01	[3.0E-12, 15.8]
7	2	615	331	0.368	1.0	0.01	[3.0E-40, 1.0E+4]
7	2	615	331	2.37E+93	1.0	0.01	[3.0E-140, 1.0E+14]

Table 15. Experimental Data for `kam2`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
9	3	616	332	0.107	1.0	0.03	[1.73E-6, 251.0]
9	3	616	332	0.107	1.0	0.03	[1.73E-20, 1.0E+8]
9	3	616	332	4.23E+46	1.0	0.03	[1.73E-70, 1.0E+28]

Table 16. Experimental Data for `kam3`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
9	3	616	332	0.107	1.0	0.03	[1.73E-6, 251.0]
9	3	616	332	0.107	1.0	0.03	[1.73E-20, 1.0E+8]
9	3	616	332	4.23E+46	1.0	0.03	[1.73E-70, 1.0E+28]

Table 17. Experimental Data for `kir1`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
44	5	616	332	4.41E-5	0.000443	1.28	$[0.5, 0.5]$
84	6	617	334	2.29E-5	0.000464	2.9	$[0.5, 0.5]$
164	7	617	334	1.16E-5	0.000476	9.79	$[0.5, 0.5]$
8	3	616	332	0.000244	0.000244	0.02	$[0.5, 0.5]$

Table 18. Experimental Data for `kir1_mod`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
44	5	616	332	0.000983	0.00095	1.26	$[0.5, 0.5]$
84	6	617	334	0.00364	0.00364	2.99	$[0.498, 0.502]$
164	7	617	334	0.00734	0.00749	10.35	$[0.496, 0.504]$

Table 19. Experimental Data for `lagurerre`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.206	0.167	0.22	$[0.0705, 66.5]$
40	5	616	332	0.122	0.108	1.28	$[0.0357, 142.0]$
80	6	617	334	0.0709	0.0659	5.63	$[0.018, 297.0]$
160	7	617	334	3.09	0.954	22.21	$[0.00901, 610.0]$
320	8	617	334	41.4	0.996	89.64	$[0.00451, 1.24\text{E}{+3}]$

Table 20. Experimental Data for `lar1`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	7.06E+9	1.0	0.1	$[3.73\text{E}{-22}, 1.0\text{E}{+50}]$
200	7	617	334	9.69E+19	0.311	0.83	$[3.73\text{E}{-22}, 41.0]$

Table 21. Experimental Data for `legendre`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.155	0.0969	0.25	$[0.0765, 0.993]$
40	5	616	332	0.0981	0.0604	1.15	$[0.0388, 0.998]$
80	6	617	334	0.0593	0.0359	6.08	$[0.0195, 1.0]$
160	7	617	334	2.72	0.0637	19.97	$[0.00979, 1.0]$
320	8	617	334	37.7	0.456	81.12	$[0.0049, 1.0]$

Table 22. Experimental Data for `lsr`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
24	4	616	332	2.88E+8	1.0	0.29	[1.0E-20, 1.0E+20]
52	5	616	332	1.81E+14	1.0	0.25	[1.0E-20, 1.0E+10]
52	5	616	332	1.81E+34	1.0	0.22	[1.0E-40, 1.0E+20]
52	5	616	332	1.81E+74	1.0	0.2	[1.0E-80, 1.0E+40]
224	7	617	334	3.62E+18	1.0	9.8	[1.0E-20, 1.0E+20]
500	8	617	334	1.92E+3	1.0	6.79	[0.0001, 2.0E+4]
500	8	617	334	5.77E+3	0.995	3.27	[3.33E-5, 1.0E+3]
500	8	617	334	1.05	1.0	3.05	[0.0916, 1.0E+200]

Table 23. Experimental Data for `mand`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
31	4	616	332	0.295	0.144	0.4	$[0.445, 2.0]$
63	5	616	332	0.127	0.0854	2.16	$[0.403, 2.0]$
127	6	617	334	0.0898	0.0488	8.58	$[0.373, 2.0]$
255	7	617	334	0.0609	0.701	37.41	$[0.351, 2.0]$
511	8	617	334	0.0234	1.63	174.71	$[0.334, 2.0]$
1023	9	619	338	0.357	0.142	569.08	$[0.321, 2.0]$
2047	10	619	338	1.12	0.241	2372.57	$[0.311, 2.0]$
4095	11	619	338	1.68	0.38	9209.19	$[0.303, 2.0]$

Table 24. Experimental Data for `mig1`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.126	1.0	0.08	$[0.01, 2.26]$
50	5	616	332	0.0157	0.987	1.8	$[0.00999, 1.83\text{E}{+3}]$
100	6	617	334	0.0563	1.0	0.5	$[0.01, 1.15]$
100	6	617	334	0.0183	1.0	4.05	$[0.01, 7.92]$
200	7	617	334	2.66	1.0	0.8	$[0.01, 1.07]$
200	7	617	334	2.59	0.999	9.37	$[0.01, 2.33]$
500	8	617	334	18.2	0.99	1.59	$[0.01, 1.03]$
500	8	617	334	18.0	0.984	15.65	$[0.01, 1.36]$

Table 25. Experimental Data for `mult`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
15	3	616	332	0.29	0.189	0.12	$[0.869, 1.07]$
20	4	616	332	0.0782	0.475	0.24	$[0.01, 2.68]$
22	4	616	332	0.213	0.105	0.24	$[1.0, 20.0]$
68	6	617	334	0.0566	0.0556	4.59	$[0.25, 2.24]$

Table 26. Experimental Data for `nroots`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
50	5	616	332	$5.18\text{E}{+13}$	1.0	0.12	$[1.0, 1.0]$
100	6	617	334	$7.06\text{E}{+6}$	1.0	0.19	$[1.0, 1.0]$
200	7	617	334	$2.69\text{E}{+3}$	1.0	0.39	$[1.0, 1.0]$
400	8	617	334	50.5	0.981	0.79	$[1.0, 1.0]$
800	9	619	338	6.34	0.864	1.73	$[1.0, 1.0]$
1600	10	619	338	1.71	0.631	3.19	$[1.0, 1.0]$
3200	11	619	338	0.645	0.392	6.79	$[1.0, 1.0]$
6400	12	623	346	0.289	0.224	14.55	$[1.0, 1.0]$

Table 27. Experimental Data for `nrooti`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
50	5	616	332	4.94E+13	1.0	0.1	[1.0, 1.0]
100	6	617	334	7.07E+6	1.0	0.32	[1.0, 1.0]
200	7	617	334	2.69E+3	1.0	0.46	[1.0, 1.0]
400	8	617	334	50.5	0.981	0.85	[1.0, 1.0]
800	9	619	338	6.34	0.864	1.85	[1.0, 1.0]
1600	10	619	338	1.71	0.631	3.23	[1.0, 1.0]
3200	11	619	338	0.645	0.392	7.02	[1.0, 1.0]
6400	12	623	346	0.289	0.224	13.31	[1.0, 1.0]

Table 28. Experimental Data for `sendra`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
160	7	617	334	0.658	2.14	24.34	[0.987, 2.01]
20	4	616	332	0.283	0.158	0.25	[0.9, 2.05]
320	8	617	334	0.809	1.64	88.08	[0.994, 2.0]
40	5	616	332	0.159	0.101	1.25	[0.95, 2.02]
80	6	617	334	0.287	0.573	5.2	[0.975, 2.01]

Table 29. Experimental Data for `sparse`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
100	6	617	334	0.11	1.0	0.26	[0.968, 1.01]
200	7	617	334	0.0361	0.995	1.16	[0.969, 1.0]
400	8	617	334	0.0211	0.929	2.19	[0.969, 1.0]
800	9	619	338	0.0118	0.739	6.56	[0.969, 1.0]
6400	12	623	346	0.00202	0.158	72.81	[0.969, 1.0]

Table 30. Experimental Data for `spiral`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
10	3	616	332	5.1E-7	1.21E-6	0.05	[1.0, 1.0]
15	3	616	332	3.49E-7	1.15E-6	0.07	[1.0, 1.0]
20	4	616	332	4.55E-7	1.3E-6	0.23	[1.0, 1.0]
25	4	616	332	3.67E-7	1.25E-6	0.29	[1.0, 1.0]
30	4	616	332	3.08E-7	1.21E-6	0.41	[1.0, 1.0]

Table 31. Experimental Data for `toep`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
128	7	617	334	0.0386	0.562	18.71	[1.31, 64.4]
256	8	617	334	0.0219	0.918	73.35	[1.34, 64.4]
128	7	617	334	0.0386	0.0272	17.68	[0.4, 13.2]
256	8	617	334	0.0225	0.599	72.66	[0.383, 13.2]

Table 32. Experimental Data for `wilk`

d	ℓ	e	MP.-DPS	RELATIVE ERROR r_d	RELATIVE ERROR r_1	RUN-TIME	MPSOLVE ROOT RADIUS
20	4	616	332	0.206	0.141	0.22	[1.0, 20.0]
30	4	616	332	0.237	0.0859	0.33	[1.0, 319.0]
40	5	616	332	0.122	0.0927	1.27	[1.0, 40.0]
80	6	617	334	0.0709	0.121	5.59	[1.0, 80.0]
160	7	617	334	0.0404	0.824	21.82	[1.0, 160.0]
320	8	617	334	0.0228	0.983	89.77	[1.0, 320.0]

8 Acknowledgments

The authors of this paper are sincerly indebted to Dr. Victor Pan who provided us with the background, the problem statement, and guided us throughout the research.