Nearly Optimal Refinement of Real Roots of a Univariate Polynomial

Victor Y. Pan, Elias Tsigaridas

Nearly Optimal Refinement of Real Roots of a Univariate Polynomial

Victor Y. Pan

Depts. of Mathematics and Computer Science, Lehman College and Graduate Center of the City University of New York, Bronx, NY 10468 USA http://comet.lehman.cuny.edu/vpan

Elias P. Tsigaridas

INRIA, Paris-Rocquencourt Center, POLSYS *Project*
Sorbonne Universités, UPMC Univ Paris 06, POLSYS, *UMR 7606, LIP6, F-75005, Paris*
CNRS, UMR 7606, LIP6, F-75005, Paris, France.

Abstract

We assume that a real square-free polynomial A has a degree d, a maximum coefficient bitsize τ and a real root lying in an isolating interval and having no nonreal roots nearby (we quantify this assumption). Then we combine the *Double Exponential Sieve* algorithm (also called the *Bisection of the Exponents*), the bisection, and Newton iteration to decrease the width of this inclusion interval by a factor of $t = 2^{-L}$. The algorithm has Boolean complexity $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$. This substantially decreases the known bound $\widetilde{\mathcal{O}}_B(d^3 + d^2L)$ and is optimal up to a polylogarithmic factor. Furthermore we readily extend our algorithm to support the same upper bound on the complexity of the refinement of r real roots, for any $r \leq d$, by incorporating the known efficient algorithms for multipoint polynomial evaluation. The main ingredient for the latter is an efficient algorithm for (approximate) polynomial division; we present a variation based on structured matrix computation with quasi-optimal Boolean complexity.

Keywords: real root refinement, polynomial, Boolean complexity, fast polynomial division, precision of computing

1. Introduction

The problem of the approximation of the real roots of a univariate polynomial appears very often as an important ingredient of various algorithms in computer algebra and nonlinear computational geometry, for example the algorithms that compute the topology of real plane algebraic curves [11, 26], solve

Email addresses: victor.pan@lehman.cuny.edu (Victor Y. Pan), elias.tsigaridas@inria.fr (Elias P. Tsigaridas)

the systems of polynomial equations [31, 14, 19], isolate the real roots of polynomials with coefficients in an extension field [55, 24], or compute cylindrical algebraic decomposition [2, 13]. Typically one starts the approximation of a real root with its isolation, by including it into an interval that contains no other roots. This gives a crude initial approximation, and then one rapidly refines this approximation, e.g., by Newton's method. In the present paper we study just the refinement stage, that is, given a polynomial A, which has a degree d and a maximum coefficient bitsize τ, and an interval with rational endpoints that contains one of its real roots (an isolating interval), we devise an algorithm that refines this *inclusion interval* to decrease its width by a factor $t = 2^{-L}$, for some positive integer L.

The overall Boolean complexity of the known real root-finding algorithms is the same as for complex root-finders, obtained in [35] (see also [36], [39], [32, Chapter 15]), that is, $\widetilde{\mathcal{O}}_B(d^3 + d^2L)$. For the complex root refinement, this is within polylogarithmic factors from the optimum provided $\tau = \mathcal{O}(L)$. The upper bound is also the record bound for the real root-refinement problem, but the lower bound does not apply to the real case anymore, and our new record upper bound $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$ of this paper covers the complexity of the refinement of a single real root as well as all real roots of a univariate polynomial. Even for the refinement of an approximation to a single root, this bound is nearly optimal, being within polylogarithmic factor from the information lower bound. Next we supply more details and recall some related results.

The known real root-finders and root-refiners are most efficient in the important special case where the polynomial has only real roots. In this context we refer to the work of Ben-Or and Tiwari [3] that introduced interlacing polynomials and *Double Exponential Sieve*. Pan and Linzer [42] and Bini and Pan [7], see also [5, 6], modified the approach of [3] (they called it *Bisection of Exponents*) to approximate the eigenvalues of a real symmetric tridiagonal matrix by using Courant-Fischer minimax characterization theorem. In [43] a variant of the refinement algorithms in [42, 7] is used, for the approximation of all the real roots of a polynomial. We also wish to cite the real root-finders (for polynomials with only real roots) by means of Quasi-Laguerre Iteration [15], [16]. They have highly efficient implementation and, like the algorithms cited above, also support nearly optimal complexity bounds for this task.

Collins and Krandick [12] presented a variant of Newton's algorithm where all the evaluations involve only dyadic numbers, as well as a comparison with the case where operations are performed with rationals of arbitrary sizes. Quadratic convergence of Newton's iterations is guaranteed by point estimates and α-theory of Smale, e.g. [9], [47]. For robust approximation of zeros based on bigfloats operations we refer the reader to [54]. A very interesting and efficient algorithm that combines bisection and Newton iterations is the *Quadratic Interval Refinement* (`qir`) by Abbott [1]. For a detailed analysis of the Boolean complexity of `qir` we refer the reader to [25]. Kerber and Sagraloff [26] modify `qir` to use interval arithmetic and approximations; they achieve a bound of $\widetilde{\mathcal{O}}_B(d^3\tau^2 + dL)$. A factor of τ could be saved if we use fast algorithms for root

isolation of univariate polynomials, e.g. [49], [39, 52]. Such an approach is used in [33] that makes adaptive the algorithm of Pan [39] for root approximation and achieves a bound of $\widetilde{\mathcal{O}}_B(d^3 + d^2\tau + dL)$ for refinement. The algorithm of [48], based on Kantorovich point estimates, is efficient in practice but has unknown complexity.

We revisit the approach of [3], [42, 7] to devise our *Real Root Refinement* ($\mathtt{R_3}$) algorithm and present a detailed analysis under the bit complexity model, based on exact operations with rationals (Thm. 12). We also introduce an approximate variant ($\alpha\mathtt{R_3}$) based on interval arithmetic, Sec. 2.1, where we use multi-precision floating point numbers for computations and for the representation of the endpoints of intervals, and where we estimate in advance the maximum precision needed. To obtain this estimate we use some tools from Kerber and Sagraloff [26] for evaluating a polynomial at a rational number by using interval arithmetic. We also study the Newton operator under both exact and approximate models of computing (Sec. 2.3), and we estimate the Boolean complexity of approximate variants of Double Exponential Sieve (Lem. 4) and Newton iteration (Lem. 10).

The Boolean complexity of $\mathtt{R_3}$ and $\alpha\mathtt{R_3}$ is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$ (Theorem 12). To deduce our complexity estimates we assume that there is no complex root of the polynomial in the complex disc that has the isolating interval as its diameter. Such an interval could be the outcome of root-finding algorithms. We detail this issue in Section 2.5.

In Section 2.4 we readily extend our algorithms and both complexity bounds $\widetilde{\mathcal{O}}_B(d^2\tau + d^2L)$ and $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$ to the refinement of up to d real roots, by incorporating the known efficient algorithms for multipoint polynomial evaluation. The latter algorithms essentially amount to recursive application of polynomial multiplication and division, which are in turn reduced to multiplication and division of long integers. The latter reduction technique (known as "binary segmentation") employs Kronecker products and involves d-fold precision increase. We present FFT-based alternative, which uses structured matrices, avoids the precision increase, and still supports nearly optimal Boolean complexity bounds (see Sections 3). The results can be of independent interest. In Section 4 we present a simpler alternative, which works for the average (but not worst case) input.

Since we published the proceedings version [44] of our paper in ISSAC 2013, further progress was achieved on polynomial root-finding, and in particular on the real root approximation, isolation and refinement, reported in the papers [50, 45, 53, 41, 46]. The paper [46] presents a nearly optimal real root-finding algorithm under the assumptions that every real root has no other roots nearby, but the paper assumes no initial approximations available for any root.

Our present paper is structured as follows. First we introduce our notation. Section 2 presents a high level description of the real root refinement algorithm. We detail its three steps, in Sec. 2.1, Sec. 2.2, and Sec. 2.3. In Section 2.4 we estimate the overall complexity of refining a single root and all the real roots. In Section 3 we present and analyze the algorithms for polynomial multiplica-

tion (Section 3.1) and division (Section 3.2). Section 4 estimates the expected number of steps of DES and αDES when the input polynomial is random of type Weyl, Sec. 4.1, or $SO(2)$, Sec. 4.2. Finally, in Section 5 we conclude and suggest some directions for further study.

Notation.. In what follows $\mathcal{O}_B$, resp. $\mathcal{O}$, means bit, resp. arithmetic, complexity and $\widetilde{\mathcal{O}}_B$, resp. $\widetilde{\mathcal{O}}$, means that we are ignoring logarithmic factors. "Ops" stands for "arithmetic operations". "lg" stands for "$\log_2$". For a polynomial $A = \sum_{i=0}^{d} a_i\, x^i \in \mathbb{Z}[x]$, $\mathtt{deg}(A) = d$ denotes its degree and $\mathcal{L}(A) = \tau$ the maximum bitsize of its coefficients, including a bit for the sign. For $a \in \mathbb{Q}$, $\mathcal{L}(a) \geq 1$ is the maximum bitsize of the numerator and the denominator. $\mu(\lambda)$ denotes the bit complexity of multiplying two integers of size λ; we have $\mu(\lambda) = \widetilde{\mathcal{O}}_B(\lambda)$. 2^{Γ} is an upper bound on the magnitude of the roots of A. We write $\Delta_\alpha(A)$ or just Δ_α to denote the minimum distance between a root α of a polynomial A and any other root. We call this quantity *local separation bound*. We also write Δ_i instead of Δ_{α_i}. $\Delta(A) = \min_\alpha \Delta_\alpha(A)$ or just Δ denotes the *separation bound*, that is the minimum distance between all the roots of A. The Mahler bound (or measure) of A is $\mathcal{M}(A) = a_d \prod_{|\alpha|\geq 1} |\alpha|$, where α runs through the complex roots of A, e.g. [34, 58]. If $A \in \mathbb{Z}[x]$ and $\mathcal{L}(A) = \tau$, then $\mathcal{M}(A) \leq \|A\|_2 \leq \sqrt{d+1}\|A\|_\infty = 2^{\tau}\sqrt{d+1}$. If we evaluate a function F (e.g. $F = A$) at a number c using interval arithmetic, then we denote the resulting interval by $[F(c)]$, provided that we fix the evaluation algorithm and the precision of computing. We write $D(c, r) = \{x : |x - c| \leq r\}$.

2. The $\mathtt{R}_3$ and $\alpha\mathtt{R}_3$ algorithms

In what follows $A = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$ with $\mathcal{L}(A) = \tau$. Let α_1 be the real root of A that lies in an (isolating) interval $I = [a\,..\,b]$. The width of I, $w = (b-a)/2$, has bitsize $\mathcal{L}(w) = \mathcal{O}(|\lg \Delta(A)|) = \mathcal{O}(d\tau)$, in the worst case (see also Prop. 1). We write $|I| = b - a$ and $m = (a+b)/2$. We wish to refine I to include α_1 into a subinterval of the width $t = 2^{-L}w$. We define the isolation ratio of a real isolating interval I of a root α of A as $\mathtt{ir}(I) = 2\,|m - \alpha_c|/|I|$, where α_c is the root of A that is the closest to α.

Our high-level description follows [3] and [42]. For details and various improvements we refer the reader to [42, 5, 7, 43]. The algorithm for refining the isolating interval of a real root α_1 consists of three steps: Double Exponential Sieve (DES), Bisection (BIS) and Newton iteration (NEWTON). We denote the approximate variants by αDES, αBIS, and αNEWTON, respectively. The three procedures are as follows:

(1) DES or αDES achieves an isolation ratio at least 3.

(2) Sufficiently many bisections (BIS or αBIS), but $\widetilde{\mathcal{O}}(\lg d)$, increase the isolation ratio.

(3) Then Newton iteration, NEWTON or αNEWTON, converges quadratically and yields an inclusion interval of the desired width.

In the sequel we describe the three sub-algorithms in detail and analyze them. The pseudo-code of $\mathtt{R_3}$ appears in Alg. 1. The pseudo-code for $\alpha\mathtt{R_3}$ is similar. We only need to change the way we perform the evaluations.

The following proposition estimates the separation bounds for a univariate (integer) polynomial. For variants and proofs see, e.g. [13, 17, 34, 24]. We use a variant from [27, Thm. 4].

Proposition 1. *Suppose $A \in \mathbb{Z}[x]$ is a square-free univariate polynomial of degree d, $\mathcal{L}(A) = \tau$, and $\alpha_1, \ldots, \alpha_d$, are the d distinct roots of A. Let $\mathbf{SR}_0(A, A')$ be the resultant of A and its derivative, A', w.r.t. x, and $\Delta_i = \Delta_i(A) = \Delta_{\alpha_i}(A)$, where $i = 1, \ldots, d$. Then*

$$\begin{aligned} |\alpha_i| &\leq 2^{\Gamma} \leq 2\|A\|_\infty \leq 2^{\tau+1} \ , & (1)\\ -\sum_i \lg \Delta_i(A) &\leq 30\, d\, \lg \mathcal{M}(A) + 3\, \mathcal{L}(\mathbf{SR}_o(A, A')) & (2)\\ &\leq 36\, d\, \tau + 42\, d \lg d \ . \end{aligned}$$

Lemma 2. [26] *Suppose we evaluate A at c, where $|c| \leq 2^{\Gamma+2}$, and suppose we use a working precision (or fixed precision arithmetic) ρ. Then*

$$\mathtt{width}[A(c)] \leq 2^{-\rho+1} (d+1)^2\, 2^{\tau}\, |c|^d \ .$$

The following lemma generalizes [26, Lemma 4].

Lemma 3. *Let x_0 be such that $|x_0 - \alpha_i| \geq \Delta_i/c$ for all* real *α_i such that $i \neq 1$ and $c \geq 2$. Then*

$$|A(x_0)| > |a_d|\, |x_0 - \alpha_1|\, c^{1-d}\, \mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \ .$$

Proof: Let $\Im(\alpha)$ be the imaginary part of $\alpha \in \mathbb{C}$. Then $|x_0 - \alpha_i| \geq |\Im(\alpha_i)| \geq \Delta_i/2 \geq \Delta_i/c$, and so $|x_0 - \alpha_i| \geq \Delta_i/c$ is true for all the roots of A. Now

$$\begin{aligned} |A(x_0)| &= |a_d| \prod_{i=1}^{d} |x_0 - \alpha_i| = |a_d||x_0 - \alpha_1| \prod_{i=2}^{d} |x_0 - \alpha_i| \\ &\geq |a_d||x_0 - \alpha_1| \frac{c}{\Delta_1} \prod_{i=1}^{d} \frac{\Delta_i}{c} \\ &\geq |a_d||x_0 - \alpha_1| c^{1-d} \mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \ . \end{aligned}$$

For the last inequality we use $\Delta_1 \leq 2\,\mathcal{M}(A)$, which in turn relies on $\Delta_1 = |\alpha_1 - \alpha_{c_1}| \leq |\alpha_1| + |\alpha_{c_1}| \leq 2\,\mathcal{M}(A)$, where α_{c_1} is a the root closest to α_1. □

2.1. Double Exponential Sieve

In this subsection we follow [3], [7] and [43] to compute an interval that contains the real root and has endpoints "far away" from the endpoints of the initial interval. The difficult case is when the real root is very close to one of the endpoints of I. Next we outline this procedure referring the reader to [43] for its detailed treatment and efficient implementation.

Algorithm 1: $\mathtt{R}_3$ $(A,\, I_0)$

Input: $A \in \mathbb{Z}[X]$ and isolating interval I_0
Output: An isolating interval I such that $\mathtt{width}(I) = \mathtt{width}(I_0)/t$.

```
/* Apply the Double Exponential Sieve Algorithm.          */
/* The pseudo code for this algorithm is in Alg. 3.       */
```
1 $(I_1, r) \leftarrow \mathrm{DES}(A, I_0)$

```
/* Increase the isolation ratio.  We perform at most O~(lg d)
   bisections.                                            */
/* The pseudo code is in Alg. 4                           */
```
2 $(I_2, r) = \textsc{IncrIsolationRatio\_by\_subdiv}(A, I_1, r, 5d^2)$

```
/* Apply Newton operator.  It converges quadratically right
   from the beginning.  The pseudo-code is in Alg. 5.     */
```
3 $I \leftarrow \textsc{Newton}(A, I_2, t)$

4 RETURN I

Initially let $\alpha_1 \in I = [a\,..\,b]$ for $a < b$. We compute a new interval $\bar{I} = [\bar{a}\,..\,\bar{b}]$ containing α_1 and such that either $0 \le \bar{b} - \bar{a} \le 2\,t$ or $\mathtt{ir}(\bar{I}) \ge 3$. In the first case the midpoint of $\bar{I}$, $\bar{m} = (\bar{a} + \bar{b})/2$, approximates α_1 within a desired error bound t, and hence we return either $[\bar{a}\,..\,\bar{m}]$ or $[\bar{m}\,..\,\bar{b}]$, depending on the sign of $A(\bar{m})$. In the second case we can apply bisections to increase the isolation ratio to the level supporting Newton's iteration. We present the analysis of the bisection iteration in the next subsection. The pseudo-code of the algorithm appears in Alg. 3.

During the first step of DES, we decide whether α_1 lies in $[a\,..\,\frac{a+b}{2}]$ or $[\frac{a+b}{2}\,..\,b]$. W.l.o.g. assume that $a = 0$ and $b = 2$ because our claims are invariant in the shifts and scaling the variable x. Furthermore assume that $\alpha_1 \in [0\,..\,1]$ for the other case is treated similarly. Now the bound $\mathtt{ir}(\bar{I}) \ge 3$ follows where

$$\bar{b} \le 2\,\bar{a}\ . \tag{3}$$

Next, we write $a_0 = 0$, $b_0 = 1$, $I_0 = [0\,..\,1]$ and evaluate A at the sequence of points $c_k = a_0 + 2w_0/2^{2^k}$, $k = 1, \ldots, g_1$, where $2w_0 = b_0 - a_0$ and $g_1 - 1$ is the maximum index such that $\alpha_1 \in [0\,..\,c_{g_1}]$. If $g_1 = 1$, then we write $a' = c_{g_1}$, $b' = 1$, obtain that $\alpha_1 \in [a'\,..\,b']$ and $\mathtt{ir}([a'\,..\,b']) \ge 5/3$, and yield an interval $\bar{I}$ with $\mathtt{ir}(\bar{I}) \ge 3$ in at most two bisections. Otherwise (if $g_1 > 1$) we reapply the DES procedure to the interval $[c_{g_1}\,..\,1]$, denote by g_2 the number of evaluations of $A(x)$ with $x < \alpha_1$ in this process and ensure (3) unless $g_2 < g_1$. Recursively we obtain a strictly decreasing sequence of intervals I_i, each defined by means of g_i evaluations of $A(x)$ where the sequence $g_1, g_2, \ldots$ strictly decreases. This means that the overall number of evaluations of $A(x)$ in the DES procedure is at most $1 + \sum_{i=1}^{g} g_i \le 1 + (g_1 + 1)g_1/2$ for $u \le g_1$. Moreover, $g_1 = \lceil \lg(\lg w + L - 1) \rceil = \mathcal{O}(\lg(\tau + L))$ because otherwise we would have $0 \le \bar{b} - \bar{a} \le 2\,t$.

The next lemma provides an approximate variant of the algorithm where at

each step of the procedure we use exactly the number of bits needed. We call this variant αDES, by using the acronym of *approximation* Double Exponential Sieve.

Lemma 4 (αDES). *The procedure αDES compresses the isolating interval I to an interval J such that $|J| \leq 1/2^L$ or $\mathtt{ir}(J) \geq 3$ using a working precision and time*

$$\mathcal{O}(-g_1 \lg w + 2^{g_1} + g_1(\lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i)) \quad \textit{bits and}$$

$$\widetilde{\mathcal{O}}_B(d^2\tau g_1^2 - dg_1^2 \lg w + dg_1 2^{g_1} + dg_1^2(\lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i)),$$

that is $\widetilde{\mathcal{O}}(d\tau + L)$ and $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$, respectively.

Proof: Initially α_1 lies in the interval $I = I_0 = [a\,..\,b]$. Let $w = w_0 = |b - a|$ be its width. We want to compute the maximum integer g_1 such that $\alpha_1 \in (a\,..\,a + w/2^{2^{g_1}})$. For this we need to evaluate A on $a + w/2^{2^k}$, for $k = 1, \ldots, g_1$. It might happen that the evaluation of A at one of these numbers is zero. To avoid that, at each step, we evaluate A at two points, instead of one. This multiple evaluation technique is borrowed from [26].

For each step k we fix two real points, m_1 and m_2 such that

$$a < m_1 = a + w/2^{2^k+1} < m_2 = a + w/2^{2^k} < b$$

and evaluate the polynomial A over them. At least one of them is not a root of A. Let $j \in \{1, 2\}$. Then for all $i \neq 1$ we obtain $|m_j - \alpha_i| \geq w/2^{2^k+1} \Rightarrow w \leq 2^{2^k+1}|m_j - \alpha_i|$ and $|\alpha_1 - m_j| \leq w/2^{2^k}$. Therefore

$$\begin{aligned}\Delta_i \leq |\alpha_1 - \alpha_i| &\leq |\alpha_1 - m_j| + |m_j - \alpha_i| \\ &\leq w/2^{2^k} + |m_j - \alpha_i| \leq |m_j - \alpha_i|2^{2^k+1}/2^{2^k} + |m_j - \alpha_i| \\ &\leq 3|m_j - \alpha_i| \ .\end{aligned}$$

Using Lemma 3 with $c = 3$ we deduce that

$$|A(m_j)| > |m_j - \alpha_1||a_d|\, 3^{1-d} \mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \ .$$

For at least one of the m_j's it holds that $|m_j - \alpha_1| > w/2^{2^k+2}$ (actually this is the half of the distance between m_1 and m_2), and hence

$$|A(m_j)| > |a_d| \frac{w}{2^{2^k+2}}\, 3^{1-d} \mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \geq w\, 3^{1-d}\, 2^{-2^{g_1} - 3 + \lg \prod_i \Delta_i - \lg \mathcal{M}(A) + \lg |a_d|} \ .$$

By virtue of Lemma 2 the precision , ρ_1, needed for this step, satisfies the equation

$$2^{-\rho_1+1}(d+1)^2 2^{\tau + (\Gamma+2)d} < w\, 3^{1-d}\, 2^{-2^{g_1} - 3 + \lg \prod_i \Delta_i - \lg \mathcal{M}(A) + \lg |a_d|} \ ,$$

and thus

$$\rho_1 = \widetilde{\mathcal{O}}(-\lg w + 2^{g_1} + \lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i) \ .$$

To support our computation of the desired interval J we double the precision of computing at every DES step. We perform $\lceil \lg \rho_1 \rceil$ steps overall; each is essentially an evaluation of A. By applying Horner's rule we yield the cost bound $\widetilde{\mathcal{O}}_B((d\tau + \rho_1)d)$. Similarly, at the i-th step of αDES we perform $g_i \lceil \lg \rho_i \rceil$ evaluations, each at the cost of $\widetilde{\mathcal{O}}_B((d\tau + \rho_i)d)$, where

$$\rho_i = \widetilde{\mathcal{O}}(-\lg w_{i-1} + 2^{g_i} + \lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i) \ .$$

Summarizing, we bound the overall cost of performing v steps by

$$\widetilde{\mathcal{O}}_B\left(\sum_{i=1}^{v} (d\tau + \rho_i)\, d\, g_i \lg \rho_i\right) \ .$$

The sequence of g_i's is strictly decreasing, that is $g_i > g_{i+1}$, and so $v < g_1^2$. The output intervals I_i have widths $w_i < w/2^{2^i}$, and so

$$\sum_{i=0}^{v-1} \lg w_i = g_1 \lg w - 2^{g_1} + 1 \ ,$$

$$\sum_i \rho_i = \mathcal{O}(-g_1 \lg w + 2^{g_1} + (\lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i)\, g_1) \ ,$$

and the overall cost is bounded by

$$\widetilde{\mathcal{O}}_B(d^2\tau g_1^2 - d g_1^2 \lg w + d g_1 2^{g_1} + (\lg \mathcal{M}(A) + \tau + d\Gamma - \lg \prod_i \Delta_i)\, d\, g_1^2) \ .$$

By noticing that $|a_d| \geq 1$, $\lg w = \mathcal{O}(d\tau)$, $\Gamma = \mathcal{O}(\tau)$, $\mathcal{O}(\lg \mathcal{M}(A)) = \mathcal{O}(\tau + \lg d)$ and using Prop. 1 to bound $\lg \prod_i \Delta_i$, we obtain that the maximum precision needed is $\widetilde{\mathcal{O}}(d\tau + L)$ and that the complexity of αDES is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$. □

Remark 5 (DES). *We call this procedure* DES *if it uses only exact arithmetic with rational numbers. It performs $g_1^2 = \mathcal{O}(\lg^2(\tau + L))$ evaluations of A at numbers of bitsize $\mathcal{O}(L)$. Using Horner's rule, each evaluation costs $\widetilde{\mathcal{O}}_B((\tau + L)d^2)$. Hence, the overall complexity is $\widetilde{\mathcal{O}}_B((\tau + L)\, d^2\, \lg^2(\tau + L)) = \widetilde{\mathcal{O}}_B((\tau + L)\, d^2)$. This bound is greater by a factor of d than the one supported by αDES. However, since we are working with exact arithmetic in the bit complexity model, Horner's arithmetic is not optimal. To see this, notice that the output of the evaluation is of bitsize $\widetilde{\mathcal{O}}((\tau + L)\, d)$. If we use the divide and conquer approach [10, 23], each evaluation costs $\widetilde{\mathcal{O}}_B((\tau + L)\, d)$ and the overall complexity is $\widetilde{\mathcal{O}}_B((\tau + L)\, d\, \lg^2(\tau + L)) = \widetilde{\mathcal{O}}_B((\tau + L)\, d)$. This bound is the same as the one supported by αDES.*

Nevertheless, this approach has some drawbacks. First, we are forced to work with full precision right from the beginning. Even though this does not affect the worst case bound it is a serious disadvantage for implementations. Second, this approach does not scale well, in the case where we want to refine all the real roots of A. Then we have to multiply the bound by d, which is not the case for the approximate algorithm. For further details we refer the reader to Section 2.4.

2.2. Bisection(s)

This section covers the second step of the refinement algorithm. Recall that the isolation ratio of a real isolating interval I of a root α_1 is defined as $\mathtt{ir}(I) = 2\,|m - \alpha_{c_1}|/|I|$, where α_{c_1} is the root of A closest to α [42]. Our goal is, by using bisections, to achieve an isolation ratio of $\mathtt{ir}(I) \geq 5d^2$, which ensures the quadratic convergence of Newton's iteration right from the start [47, Corollary 4.5].

If $\mathtt{ir}(I) = 1 + \delta$ for an interval $I = [a\,..\,b]$, then after k bisection steps, we obtain an interval I_k, such that $\mathtt{ir}(I_k) \geq 1 + 2^k\delta$. The details appear in Alg. 4. We increase the isolation ratio by applying bisections. We set *dir* equal to -1, 0, or 1 to indicate the search direction. If the initial isolation ratio is $\mathtt{ir}(I) = 1 + 2\delta$, then k bisections increase it at least to $\mathtt{ir}(I_k) = 1 + 2^k\delta$, where I_k is the refined isolating interval. The value `left` or `right` of the variable *dir* specifies whether the closest real root of the polynomial A not belonging to the interval I lies to the left or the right of the midpoint of I. We may know this value in advance, but if we do not know it, then we temporarily set $dir = \emptyset$, perform a single bisection, and readily obtain the correct value of *dir*. Even if this observation does not affect the asymptotic behavior of the algorithm, it can help us save a constant number of bisections, which can be useful in practical implementation of the algorithm.

If the initial isolation ratio is $r = 1 + 2 \cdot 2^{\lg(r-1)-1}$, then after k steps it becomes $1 + 2^k \cdot 2^{\lg(r-1)-1}$. If our goal is to achieve an isolation ratio R, then

$$1 + 2^k \cdot 2^{\lg(r-1)-1} \geq R \Leftrightarrow 2^k \cdot 2^{\lg(r-1)-1} \geq R - 1 \Leftrightarrow$$
$$k + \lg(r-1) - 1 \geq \lg(R-1) \Leftrightarrow k \geq 1 + \lg\frac{R-1}{r-1}\ .$$

In our case $R = 5d^2$. From the previous step α`DES` guarantees an isolation ratio at least 3, and thus we need to perform $k = \mathcal{O}(\lg d)$ bisections.

Each bisection consists of an evaluation of A over the midpoint of the corresponding interval and setting *dir* accordingly. We will perform this in an approximate way using the algorithm from [26]. We need the following lemma for the approximate variant, α`BIS`.

Lemma 6. [26] *The approximate bisection for a root $\alpha_1 \in I = [a\,..\,b]$ of A requires a working precision of $\rho = \widetilde{\mathcal{O}}(-\lg w + \tau + d\Gamma + \lg\prod_i \Delta_i))$ bits and has bit complexity $\widetilde{\mathcal{O}}_B(d(-\lg w + \tau + d\Gamma + \lg\prod_i \Delta_i))$, where w is the width of I.*

A single bisection halves an interval of width w, k bisections decrease the width to $w_k = w/2^k$.

In the worst case the interval has the width $w\,2^{-L}$, and so the number of bits needed in the worst case is $\rho = \widetilde{\mathcal{O}}(-\lg w + L + \tau + d\Gamma + \lg \prod_i \Delta_i)$. We perform $\mathcal{O}(\lg d)$ bisections, and so the overall cost is $\widetilde{\mathcal{O}}_B(d(-\lg w + L + \tau + d\Gamma + \lg \prod_i \Delta_i))$.

Recall that $-\lg w = -\mathcal{O}(\lg \Delta(A)) = \widetilde{\mathcal{O}}(d\tau)$.

Lemma 7 (αBIS). *The cost of αBIS is $\widetilde{\mathcal{O}}_B(d(-\lg w + L + \tau + d\Gamma + \lg \prod_i \Delta_i))$, that is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.*

Remark 8 (BIS). *A single bisection using only exact arithmetic with rational numbers involves $\mathcal{O}(\lg d)$ evaluations of polynomal A over numbers of bitsize $\mathcal{O}(L - \lg w)$, in the worst case. Each evaluation costs $\widetilde{\mathcal{O}}_B(d(\tau - \lg w + L))$, using the divide and conquer scheme [23, 10]. Hence, the overall complexity is $\widetilde{\mathcal{O}}_B(d(\tau - \lg w + L)\lg d) = \widetilde{\mathcal{O}}_B(d(d\tau + L)) = \widetilde{\mathcal{O}}_B(d^2\tau + dL)$.*

2.3. Bounding the Newton operator

The last step of our algorithm consists in performing a suitable number of Newton iterations sufficient to refine the isolating interval up to the required width. The bisections of the previous step ensure that Newton iteration converges quadratically, right from the beginning. This follows by virtue of [47, Corollary 4.5], which we reproduce next.

Theorem 9. *Suppose both discs $D(m, r)$ and $D(m, r/s)$ for $s \geq 5d^2$ contain a single simple root α of a polynomial $A = A(x)$ of degree d. Then Newton's iteration*

$$x_{k+1} = x_k - A(x_k)/A'(x_k), k = 0, 1, \ldots \tag{4}$$

converges quadratically to the root α right from the start provided $x_0 = m$.

First, we estimate the precision needed at each step of Newton iteration. Given an interval $[a_k \,..\, b_k]$, let $m_k = (a_k + b_k)/2$ be its middle point and apply the Newton operator, that is

$$NA(m_k) = m_k - \frac{A(m_k)}{A'(m_k)} \ ,$$

where $A_k = A(m_k)$, $A'_k = A'(m_k)$, and A' is the derivative of A. Assume that $A_k > 0$ and $A'_k < 0$. The other sign combinations could be treated similarly. Suppose we compute A_k and A'_k using interval arithmetic and a working precision ρ to be specified later. We can assume that their interval representation, $[A_k] = [A_k - \epsilon \,..\, A_k + \epsilon]$ and $[A'_k] = [A'_k - \epsilon \,..\, A'_k + \epsilon]$, have the same width, ϵ.

The interval evaluation of Newton operator, using the same working precision ρ, results in the interval

$$[NA(m_k)] = \left[m_k - \epsilon - \frac{A_k - \epsilon}{A'_k - \epsilon} \,..\, m_k + \epsilon - \frac{A_k + \epsilon}{A'_k + \epsilon}\right] \ .$$

Its width, $[NA(m_k)]$ is $2\epsilon - \frac{A_k+\epsilon}{A'_k+\epsilon} + \frac{A_k-\epsilon}{A'_k-\epsilon}$, and now we ensure the upper bound $t = 2^{-L}w$.

$$2\epsilon - \frac{A_k+\epsilon}{A'_k+\epsilon} + \frac{A_k-\epsilon}{A'_k-\epsilon} \leq t \Rightarrow$$
$$P(\epsilon) = 2\epsilon^3 - t\epsilon^2 - 2(A_k - A'_k + {A'_k}^2)\epsilon + t{A'_k}^2 \geq 0 \ .$$

The coefficient list of P has two sign variations, and hence from Descartes' rule of signs it follows that P has at most two positive real roots. If such a pair of roots exists, let them be $\epsilon_1 < \epsilon_2$ and assume that P is positive between 0 and ϵ_1. For the width of $[NA(m_k)]$ to be smaller than $t = 2^{-L}w$, it suffices if $0 < \epsilon \leq \min\{1, \epsilon_1\}$. To guarantee this we estimate a (positive) lower bound on the roots of the P and require ϵ to be smaller than it. Combine Lemma 2 and Cauchy's bound, e.g. [58, 34] to obtain

$$\epsilon \leq \frac{t{A'_k}^2}{t + 2(A_k - A'_k + {A'_k}^2)} \ ,$$

and assume working with a precision ρ that satisfies

$$2^{-\rho+1}(d+1)^2 2^{\tau+d(\Gamma+2)} \leq \epsilon \leq \frac{t{A'_k}^2}{t + 2(A_k - A'_k + {A'_k}^2)} \ . \tag{5}$$

Hence, we can express ρ as a function of the desired width.

It remains to bound the evaluations A_k and A'_k. At the k-th step, given an interval I_k, we apply Newton operator on its midpoint, m_k, and deduce that $\Delta_i < 2|m_k - \alpha_i|$ for all $i \neq 1$. Indeed $|m_k - \alpha_i| \geq |\alpha_1 - \alpha_i| \geq \Delta_i$ where $m_k \leq \alpha_1 \leq \alpha_i$, whereas $|\alpha_1 - m_k| \leq |m_k - \alpha_i|$ where $\alpha_1 \leq m_k \leq \alpha_i$ because α_i lies outside the isolating interval. Therefore $\Delta_i \leq |\alpha_i - \alpha_1| \leq |\alpha_i - m_k| + |\alpha_1 - m_k| \leq 2|m_k - \alpha_i|$.

So using Lemma 3 we obtain

$$|A_k| \geq |m_k - \alpha_1||a_d|\, 3^{1-d}\mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \ . \tag{6}$$

For the approximations by Newton iterations [43, 7], having quadratic convergence, we have

$$|m_k - \alpha_1| = 2^{4-2^k}|m_0 - \alpha_1| \ . \tag{7}$$

Furthermore

$$|m_0 - \alpha_1| \geq t = 2^{-L}w \ ,$$

for otherwise the iteration would not refine the initial interval to the required length. Therefore

$$|m_k - \alpha_1| = 2^{4-2^k}|m_0 - \alpha_1| \geq 2^{4-2^k - L + \lg w} \ , \tag{8}$$

which leads to the bound

$$|A_k| \geq |a_d|\, 2^{4-2^k-L+\lg w}\, 3^{1-d}\mathcal{M}(A)^{-1}\, 2^{\lg \prod_i \Delta_i - 1} \ . \tag{9}$$

We need a similar bound on $|A'_k|$. Let α'_i be the roots of A'. We assume that A' is square-free. This is no loss of generality since we can estimate the required quantities using the square-free part of A'.

Let α'_1 and α'_2 denote the two roots of A' that are located on the left and on the right of α_1, respectively, and are the closest to α_1. Let α' denote any other root of A'. Then $|m_k - \alpha'_i| \geq |\alpha' - \alpha'_i| \geq \Delta'_i$, where Δ'_i is the local separation bound of α'_i and where α' is either α'_1 or α'_2 depending on which side from m_k the root α'_i lies. To see this assume that α'_i lies on the right of α'_2. Then $m_k \leq \alpha'_2 \leq \alpha'_i$, and so $|m_k - \alpha'_2| \geq |\alpha'_2 - \alpha'_i| \geq \Delta'_i$. A similar argument applies when α'_i lies on the left of α'_1. Therefore

$$\begin{aligned}
|A'_k| = |A'(m_k)| = |d\, a_d| \prod_{i=1}^{d-1} |m_k - \alpha'_i| = \\
= |d\, a_d||m_k - \alpha'_1||m_k - \alpha'_2| \prod_{i=3}^{d} |m_k - \alpha'_i| \\
\geq |d\, a_d||m_k - \alpha'_1||m_k - \alpha'_2| \, \frac{1}{\Delta'_1 \Delta'_2} \prod_{i=1}^{d} \Delta'_i \\
\geq |d\, a_d||m_k - \alpha'_1||m_k - \alpha'_2| \, \mathcal{M}\left(A'\right)^{-2} 2^{-2}\, 2^{\lg \prod_i \Delta'_i} \ ,
\end{aligned}$$

where we use the inequality $\Delta'_i \leq 2\mathcal{M}\left(A'\right)$.

Let us bound the distances $|m_k - \alpha'_1|$ and $|m_k - \alpha'_2|$. Consider the complex disc $D(m_0, |I_0|/2)$, having the interval I_0 as its diameter. It follows from the quadratic convergence of Newton operator that this disc satisfies the assumptions of Thm. 9. Therefore the roots α'_1 and α'_2 lie outside the $5d^2$-dilation of this disc, that is lie much farther than α_1 from the center m_0. Likewise α'_1 and α'_2 lie outside the $5d^2$-dilations of the discs $D(m_k, |I_k|/2)$, with the centers at m_k and radii $|I_k|/2$, that is the centers m_k lie much closer to α_1 than to α'_1 and α'_2. So using Eq. (8) we obtain $|m_k - \alpha'_1| \geq |m_k - \alpha_1| \geq 2^{4-2^k-L+\lg w}$. Hence

$$|A'_k| \geq |d\, a_d|\, 2^{8-2^{k+1}-2L+2\lg w}\, \mathcal{M}\left(A'\right)^{-2} 2^{-2+\lg \prod_i \Delta'_i} \ . \tag{10}$$

Now we can bound the right-hand side of inequality (5). First, we bound the numerator. For all m_k it holds that $|m_k| \leq 2^{\Gamma} \leq 2^{\tau+1}$, see Eq. (1). Thus

$$|A'_k| = |A'(m_k)| \leq \sum_{i=1}^{d} i\, |a_i|\, 2^{(i-1)\,\Gamma} \leq d\, 2^{\tau}\, 2^{(d-1)\Gamma} \leq 2^{d\tau+d+\lg d-1} \leq 2^{2d\tau+\lg d} \ ,$$

and so

$$t{A'_k}^2 \leq 2^{-L+\lg w+4d\tau+2\lg d} \ . \tag{11}$$

Next, we will bound the denominator of the upper bound in (5). Represent the right-hand sides of equations (9) and (10) as the powers of 2 and thus rewrite the bounds given by these equations as follows, $|A_k| \geq 2^{-\ell_1} > 0$ and

$|A'_k| \geq 2^{-\ell_2} > 0$. Recall that we assume A'_k to be negative and hence $-A'_k > 0$. Therefore

$$t + 2(A_k - A'_k + {A'_k}^2) \geq 2^{\min\{-\ell_1, -2\ell_2, -L+\lg w\}} \geq 2^{-\max\{\ell_1, 2\ell_2, L-\lg w\}} ,$$

and so

$$\frac{1}{t + 2(A_k - A'_k + {A'_k}^2)} \leq 2^{\max\{\ell_1, 2\ell_2, L-\lg w\}} . \tag{12}$$

By combining bounds (5), (9)–(12) we deduce that the precision

$$\widetilde{\mathcal{O}}\left(L - \lg w + 2^k + d\tau + \lg\left(\mathcal{M}\left(A\right)\mathcal{M}\left(A'\right)\right) - \lg(\prod_i \Delta_i \prod_i \Delta'_i)\right)$$

is sufficient.
We want to achieve the bound $|m_k - \alpha_1| \leq w/2^L$. Using Eq. (7) we obtain

$$|m_k - \alpha_1| \leq 2^{4-2^k}\,\frac{w}{2} \leq \frac{w}{2^L} \Rightarrow k \geq \lg(L+3) .$$

Hence, we need to apply the Newton operator $k = \mathcal{O}(\lg L)$ times, to refine the interval by a factor of 2^{-L}. Note that $|a_d| \geq 1$, $\mathcal{O}(\lg \mathcal{M}\left(A\right)) = \mathcal{O}(\tau + \lg d)$, $\mathcal{O}(\lg \mathcal{M}\left(A'\right)) = \mathcal{O}(\tau \lg d)$, $-\lg w = \mathcal{O}(d\tau)$, use Prop. 1 to bound $\lg \prod_i \Delta_i$ and $\lg \prod_i \Delta'_i$, and obtain that the precision $\widetilde{\mathcal{O}}(d\tau + L)$ is sufficient for us. So the Boolean complexity of α`DES` is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.

Lemma 10 (α`NEWTON`). *The maximum number of bits needed by Newton iterations is $\widetilde{\mathcal{O}}(d\tau + L)$ and the overall complexity of the Newton step is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.*

We should mention that there is no need to realize the Newton iteration using interval arithmetic. However, it is easier to estimate theoretically the working precision needed using the formalization of interval arithmetic.

Remark 11 (`NEWTON`). *We can also estimate the complexity of Newton iterations in the case where only exact arithmetic with rational numbers is used. We need to perform $\mathcal{O}(\lg L)$ Newton iterations, each consisting of an evaluation of A and its derivative over the numbers of bitsize $\mathcal{O}(\tau + L - \lg w)$, in the worst case. The cost of the pair of these evaluations is $\widetilde{\mathcal{O}}_B(d(\tau + L - \lg w))$. Hence the overall complexity is $\widetilde{\mathcal{O}}_B(d(\tau + L - \lg w)\lg L) = \widetilde{\mathcal{O}}_B(d^2\tau + dL))$.*

2.4. Overall complexity of $\mathtt{R_3}$ *and* $\alpha\mathtt{R_3}$

The following theorem summarizes our estimates of the previous sections.

Theorem 12 ($\alpha\mathtt{R_3}$). *We can refine an isolating interval of a real root of A to decrease its width by a factor of 2^{-L} by using $\alpha\mathtt{R_3}$ or $\mathtt{R_3}$ with Boolean complexity $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.*

The bound of Th. 12 for $\alpha\mathtt{R}_3$ holds even where we refine all the real roots of the polynomial A. Indeed the bound is dominated by the cost of the evaluation of a polynomial at at most r rational points $x_1, \ldots, x_r$, where $r \leq d$ is the number of real roots that we refine. We can perform this evaluation efficiently by applying the following theorem by Kirrinnis [28, Thm. 3.9] for $p_j = x - x_j$ and $j = 1, \ldots, r$.

Theorem 13. *Let p_j be monic polynomials with their roots inside the unit disc of degree n_j, for $1 \leq j \leq n$. Let f be a polynomial of degree $m \geq n$ such that $\|f\|_1 \leq 1$ and $s \in \mathbb{N}$. Then the polynomials $\tilde{f}_j$ of degree $n_j - 1$ with $|f \mod p_j - \tilde{f}_j| < 2^{-s}$ can be computed simultaneously in time $\mathcal{O}_B(\mu(n \lg n + m)(s + m))$, that is $\widetilde{\mathcal{O}}_B((n + m)(s + m))$.*

Results similar to Kirrinnis' appear in [30], and in the case where all the polynomials p_j are linear (still sufficient for our applications), in [56, Lemma 11].

To extend Theorem 13 to polynomials that can have any norm we should normalize them. If the polynomial f in the theorem has coefficients of maximum bitsize τ, then we just need to add the term $m\tau$ to the complexity bound. We are interested in evaluating a polynomial at rational numbers that lie between its real roots, thus in our case all p_j are linear ($n_j = 1$ for all j in Theorem 13). To comply with the assumptions of the theorem we apply a homothetic transformation to move all the roots into the unit disc, and then we divide the coefficients of the polynomial with a suitable number to ensure that the norm is less than 1. This adds the term $d\,\tau$ to the complexity bound. The discussion leads to the following theorem.

Theorem 14 ($\alpha\mathtt{R}_3$). *We can refine the isolating intervals of all the real roots of A to decrease their width by a factor of 2^{-L} using $\alpha\mathtt{R}_3$ with Boolean complexity $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.*

The algorithm supporting the Kirrinnis' theorem employs the classical reduction of multipoint evaluation to polynomial multiplication via polynomial division (cf. [4], [57]) and by following [21], [52] represents polynomials $A(x) = \sum_{i=0}^{d} a_i x^i$ with nonnegative integer coefficients as integers $\sum_{i=0}^{d} a_i 2^{hi}$ for a sufficiently large integer 5p, $h > \lg ||A||_\infty$ and reduces polynomial multiplication to integer multiplication, for which one can apply the algorithms having low asymptotic Boolean complexity [57], [22]. The representation of a polynomial as an integer is called sometimes binary segmentation (cf. [52], [4, Remark 3.9.2]), can be linked to the Kronecker's product, and implies dramatic growth of the precision of the binary numbers involved, which is not always desired. Is such a growth necessary for multipoint evaluation of polynomials at a low Boolean cost? The answer is "no" because we can properly use FFT instead of binary segmentation and Kronecker's product. In the next section we elaborate upon this approach for polynomial multiplication and division. Further extension to multipoint polynomial evaluation and interpolation is much simpler and is better known (see [28], [30], or [56]). We refer the reader to [40]

Algorithm 2: ISOLATION_DISCS(A, $I_1, \dots, I_r$)

Input: $A \in \mathbb{Z}[X]$, and isolating intervals $I_1, \dots, I_r$
Output: Isolation ratio for convergence of Newton.

1 Apply a fixed number of bisections as well as bisections of exponent to all r input intervals $I_1, \dots, I_r$, which transforms them into subintervals $\bar{I}_1, \dots, \bar{I}_r$.

2 Recall that the Möbius map $z = \frac{x+\sqrt{-1}}{x-\sqrt{-1}}$ maps the real line $\{x : \Im(x) = 0\}$ into the unit circle $C_1 = \{z : |z| = 1\}$. Note that $x = \sqrt{-1}\frac{z-1}{z+1}$ and compute the coefficients of the polynomial $B(z) = (z+1)^n A(\sqrt{-1}\frac{z-1}{z+1})$.

3 Apply the root-radii algorithm (cf. [39], [52]) to approximate the root radii of this polynomial. In particular this defines an annulus about the circle C_1, which contains the images of the r real roots of the polynomial $A(x)$ and no images of its nonreal roots.

4 Compute the boundary circles of the image of this annulus in the converse map $x = \sqrt{-1}\frac{z-1}{z+1}$.

5 For all subintervals $\bar{I}_1, \dots, \bar{I}_r$ compute at first the distances $\bar{d}_1, \dots, \bar{d}_r$ from their midpoints to these two boundary circles and then the ratios $2d_1/|\bar{I}_1, \dots, 2d_r/|\bar{I}_r|$.

6 RETURN the minimum of the ratios as the guaranteed isolation ratio for the input intervals.

on a distinct approach to bounding the precision of computing for multipoint polynomial evaluation and interpolation.

In the exact version, $\mathtt{R}_3$, we perform evaluations at numbers of bitsize L. The output of such an evaluation results in rational numbers of bitsize $\widetilde{\mathcal{O}}(d(\tau + L))$. When we isolate all the r real roots, the bitsize of the output is $\widetilde{\mathcal{O}}(r\, d(\tau + L)) = \widetilde{\mathcal{O}}(d^2(\tau + L))$, which is also a lower bound on the Boolean complexity of the refinement process. This exceeds the bound for $\alpha\mathtt{R}_3$ by a factor of d. This result shows the superiority of approximation methods.

2.5. Requirements for the isolating intervals

Our algorithms support the complexity bound of Thm. 12 provided that we are given a real m and a positive r such that the *root-isolation disc* $D(m, r) = \{x : |x - m| < r\}$ contains a single simple real root α of the polynomial A and no other its roots, and furthermore α is not very close to the boundary circle of the disc, namely

$$|\alpha - m|(1 + c'/d^c) \le r \ , \tag{13}$$

for two real constants $c' > 0$ and c. Namely, under the latter assumption it is sufficient to apply $\mathcal{O}(\log d)$ bisections to strengthen bound (13) to the level $5d^2|\alpha - m| \le r$. Then we can apply Theorem 9 to ensure quadratic convergence of Newton's iteration, and then complete our algorithms and proofs.

Is it simple to ensure bound (13) at a low cost? For the worst case input this is not simpler than to approximate the root α very closely. Indeed the divide-and-conquer algorithms (cf. [39], [52]) can compute a real isolation interval for a single simple root, but produce such intervals already well isolated from all other roots, and then our construction is not needed. On the other hand root-finders working on the real line such as the subdivision algorithms produce such intervals independently of the distribution of nonreal roots on the complex plane. In this case we cannot exclude any unfavorable distribution of them. On the average input, however, violation of the isolation assumption (13) seems to be rather pathological (see Section 4).

A natural question arises: How can we test whether this assumption holds for a given polynomial A and a real interval I containing its single simple root α? In fact very easily: we can just apply our algorithms. They compute a sequence of real inclusion intervals $(a_h \,..\, b_h)$, for $h = 0, 1, \dots$, where $(a_0 \,..\, b_0) = I$ and $b_h > a_h$ for all h. We verify the inclusion property by checking whether $A(a_h)A(b_h) < 0$ and either observe that h bisection steps decrease the width of the isolating interval by a factor of 2^h or otherwise conclude that the assumption (13) is certainly violated. This *test by action* requires negligible extra cost.

Alternatively, given m and r, we can test whether the disc $D(m, r)$ contains only one root by applying the Schur–Conn test, partial inverses of Descartes' rule of sign, e.g. [18], or the root-radii algorithms of [52] (cf. [37, Section 4]), which approximate the distances from m to all roots of A within, say 1% error. These a priori tests, however, have no advantage versus the test by action and have a little greater Boolean cost.

Remark 15. *Suppose we have r real intervals $I_1, \dots, I_r$, each containing a single simple root of A. In this case our algorithm* ISOLATION_DISCS *(Alg. 2) is a single a priori test of the existence of all the r root-isolation discs. Our present paper does not use this algorithm, but it may be of independent interest, and our next research plan includes estimation of its Boolean complexity, which seems to be dominated at Stage 3. This stage is quite inexpensive, according to the estimates in [52] and [39].*

3. Approximate FFT-based polynomial multiplication and division

In this section we present efficient algorithms and the complexity results for multiplying and dividing univariate polynomials approximately. These results are the main ingredients of the fast algorithms for multipoint evaluation and interpolation. The evaluation is involved into our record fast real root-refinement, but all these results are also interesting on their own right because, unlike the previous papers such as [51], [52] and [28], we keep the Boolean cost bounds of these computations at the record level by employing FFT rather than the Kronecker product and thus decreasing the precision of computing dramatically.

3.1. Approximate polynomial multiplication

We need the following lemma on the evaluation of a polynomial at the powers of a root of unity. A similar result appears in [51, Section 3] where Bluestein's

technique from [8] is applied (see also [29, Chapter 4.3.3, Exercise 16]). We use that lemma to provide a bound on the Boolean complexity of multiplying two univariate polynomials when their coefficients are known up to a fixed precision. An algorithm for this problem appeared in [51, Theorem 2.2] based on employing Kronecker's product, but we rely on FFT instead.

Lemma 16. *Suppose $A \in \mathbb{C}[x]$ of a degree at most d such that $\|A\|_\infty \le 2^\tau$. Let $K = 2^k \ge d$ for a positive integer k. Assume that we know the coefficients of A up to the precision $-\ell-\tau-\lg K-3$; that is the input is assumed to be a polynomial $\widetilde{A}$ such that $\|A-\widetilde{A}\|_\infty \le 2^{-\ell-\tau-\lg K-3} \ge 10$. Let $\omega = \exp(\frac{2\pi}{K}\sqrt{-1})$ denote a K-th root of unity. Then we can evaluate the polynomial A at $1, \omega, \ldots, \omega^{K-1}$ in $\widetilde{\mathcal{O}}_B(K \lg K\, \mu(\ell+\tau+\lg K))$ such that*

$$\max_{0\le i\le K-1} |A(\omega^i) - \widetilde{A(\omega^i)}| \le 2^{-\ell} \ . \tag{14}$$

Moreover, $|A(\omega^i)| \le K\,\|A\|_\infty \le 2^{\tau+\lg K}$, for all $0 \le i \le K-1$.

Proof: Let $\mathbf{fft}_\lambda(A)$ denote the vector output by the `FFT` algorithm applied with rounding to the precision of λ bits and let $\mathbf{dft}(A)$ denote the exact vector of the discrete Fourier transform. Then Corollary 4.1 (Chapter 3, Section 4) from [4] implies that

$$\|\mathbf{fft}_\lambda(A) - \mathbf{dft}(A)\|_\infty = \max_i |A(\omega^i) - \widetilde{A(\omega^i)}| \le 5\cdot 2^{-\lambda}\sqrt{K}\,\lg K\,\|A\|_\infty \le 2^{-\lambda+\tau+\lg K+3} \ .$$

Therefore, we can obtain the output with the accuracy of $2^{-\ell}$ bits if we use a precision of $\lambda \ge \ell+\tau+\lg K+3$ bits in the input and throughout the computations by this algorithm. This implies the claimed complexity bound because the algorithm uses $1.5k\lg K$ ops [4, Problem 2.2a]. Moreover $|A(\omega^i)| \le \sum_{i=0}^d |a_i| \le K\,\|A\|_\infty \le 2^{\tau+\lg K}$, for all $0 \le i \le K-1$, because $|\omega| = 1$. □

Lemma 17. *Let $A, B \in \mathbb{C}[x]$ of degree at most d, such that $\|A\|_\infty \le 2^{\tau_1}$ and $\|B\|_\infty \le 2^{\tau_2}$. Let C denote the product AB and let $K = 2^k \ge 2d+1$ for a positive integer k. Write $\lambda = \ell + 2\tau_1 + 2\tau_2 + 5.1\lg K + 4$. Assume that we know the coefficients of A and B up to the precision λ, that is that the input includes two polynomials $\widetilde{A}$ and $\widetilde{B}$ such that $\|A-\widetilde{A}\|_\infty \le 2^{-\lambda}$ and $\|B-\widetilde{B}\|_\infty \le 2^{-\lambda}$. Then we can compute in $\mathcal{O}_B(d\lg d\,\mu(\ell+\tau_1+\tau_2+\lg d))$ a polynomial $\widetilde{C}$ such that $\|C-\widetilde{C}\|_\infty \le 2^{-\ell}$. Moreover, $\|C\|_\infty \le 2^{\tau_1+\tau_2+2\lg K}$ for all i.*

Proof: We briefly recall polynomial multiplication algorithm based on FFT and on the seminal evaluation–interpolation techniques of Toom 1963. We refer the reader to [4] for a more detailed presentation. Let $\omega = \exp(\frac{2\pi}{K}\sqrt{-1})$ denote a primitive K-th root of unity. First we apply FFT to evaluate A and B at $1, \omega, \ldots, \omega^{K-1}$ by using $O(K\lg K)$ ops. Then, in K multiplications, we obtain the values

$$C(1) = A(1)B(1),\ C(\omega) = A(\omega)B(\omega),\ \ldots,\ C(\omega^{K-1}) = A(\omega^{K-1})B(\omega^{K-1}).$$

Let $\mathbf{c}_\omega$ denote the vector of these K values. From its coordinates we recover the coefficients of the polynomial C by interpolation, which amounts to performing the inverse FFT of size K, that is FFT at the numbers $1, \frac{1}{\omega}, \ldots, \frac{1}{\omega^{K-1}}$ whose output values are divided by K. Note that $\frac{1}{\omega^i} = \omega^{K-i}$, and so, up to scaling by $1/K$, the inverse FFT is just a "shuffled" version of FFT; its cost is $\mathcal{O}(K \lg K)$.

For the values of the polynomials A and B at the powers of ω we deduce from Lemma 16 that

$$\max_i |A(\omega^i) - \widetilde{A(\omega^i)}| \le 2^{-\lambda+\tau_1+\lg K+3} \quad \text{and} \quad \max_i |B(\omega^i) - \widetilde{B(\omega^i)}| \le 2^{-\lambda+\tau_2+\lg K+3} \ . \tag{15}$$

By combining the two inequalities of Eq. (15) with the assumed bounds $\|A\|_\infty \le 2^\tau$ and $\|B\|_\infty \le 2^\tau$, we obtain

$$\|\mathbf{c}_\omega - \widetilde{\mathbf{c}_\omega}\|_\infty = \max_i \left| C(\omega^i) - \widetilde{C(\omega^i)} \right| \le 2^{-\lambda+\tau_1+\tau_2+2\lg K+4} \ ,$$

that is we yield the coordinates of the vector $\mathbf{c}_\omega$ up to $(\lambda - \tau_1 - \tau_2 - 2\lg K - 4)$ bits of precision.

Moreover, Lemma 16 implies that $|A(\omega^i)| \le 2^{\tau_1+\lg K}$ and $|B(\omega^i)| \le 2^{\tau_2+\lg K}$, for all $0 \le i \le K$.

Multiply pairwise with no errors the K pairs of the values of A and B at ω^i for all i and obtain the numbers $C(\omega^i)$ such that

$$|C(\omega^i)| = |A(\omega^i)\, B(\omega^i)| \le 2^{\tau_1+\tau_2+2\lg K} \text{ for all } i \ ,$$

and so

$$\|\mathbf{c}_\omega\|_\infty = \max_i |C(\omega^i)| \le 2^{\tau_1+\tau_2+2\lg K} \ .$$

By applying again Corollary 4.1 (Chapter 3, Section 4) from [4] to the vector $\mathbf{c}_\omega$, we obtain

$$\begin{aligned} \|\mathbf{fft}_\lambda(\mathbf{c}_\omega) - \mathbf{dft}(\mathbf{c}_\omega)\|_\infty &\le 5 \cdot 2^{-\lambda+\tau_1+\tau_2+2\lg K+4} \sqrt{K}\, \lg K\, \|\mathbf{c}_\omega\|_\infty \\ &\le 5 \cdot 2^{-\lambda+\tau_1+\tau_2+2\lg K+4} \sqrt{K}\, \lg K\, 2^{\tau_1+\tau_2+2\lg K} \\ &\le 5 \cdot 2^{-\lambda+2\tau_1+2\tau_2+5.1\lg K+4} \end{aligned}$$

because $\sqrt{K}\, \lg K \le 1.1K$ for $K \ge 1$.

We can compute the output values accurate up to ℓ bits of precision if we employ the coefficients of the input polynomials up to $\ell + 2\tau_1 + 2\tau_2 + 5.1 \lg K + 4$ bits and if we perform all computations with the precision of $\ell + 2\tau_1 + 2\tau_2 + 5.1 \lg K + 4$ bits. We use $\mathcal{O}(K \lg K)$ ops, and so the bit complexity of the algorithm is $\mathcal{O}_B(K \lg K\, \mu(\ell+\tau_1+\tau_2+\lg K))$ or $\mathcal{O}_B(d \lg d\, \mu(\ell+\tau_1+\tau_2+\lg d))$. □

3.2. Approximate polynomial division

Assume two polynomials $s(x) = \sum_{i=0}^m s_i x^i$ and $t(x) = \sum_{i=0}^n t_i x^i$ such that $s_m t_n \ne 0$, $m \ge n$, and seek the quotient $q(x) = \sum_{i=0}^{m-n} q_i x^i$ and the remainder $r(x) = \sum_{i=0}^{n-1} r_i x^i$ of their division such that $s(x) = t(x)\, q(x) + r(x)$ and $\deg(r) <$

$\deg(t)$. Further assume that $t_n = 1$. This is no loss of generality because we can divide the polynomial t by its nonzero leading coefficient. We narrow our task to computing the quotient $q(x)$ because we can compute the remainder $r(x) = s(x) - t(x)\,q(x)$ at the dominated cost as soon as the quotient $q(x)$ is available, if we multiply $t(x)$ by $q(x)$ and subtract the result from $s(x)$. We begin with an algorithm for the exact evaluation of the quotient. Represent division with a remainder by the vector equation

$$\begin{bmatrix} 1 & & & \\ t_{n-1} & 1 & & \\ \vdots & \vdots & & \\ t_1 & & & \\ t_0 & t_1 & \cdots & 1 \\ & t_0 & t_1 & \vdots \\ & & & t_1 \\ & & & t_0 \end{bmatrix} \begin{bmatrix} q_{m-n} \\ q_{m-n-1} \\ \vdots \\ q_1 \\ q_0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ \\ 0 \\ r_{n-1} \\ r_{n-2} \\ \vdots \\ r_0 \end{bmatrix} = \begin{bmatrix} s_m \\ s_{m-1} \\ \vdots \\ s_n \\ s_{n-1} \\ \vdots \\ \\ s_0 \end{bmatrix}.$$

The first $m - n + 1$ equations form the following vector equation,

$$\begin{bmatrix} 1 & & & \\ t_{n-1} & 1 & & \\ \vdots & \vdots & & \\ t_1 & & & \\ t_0 & t_1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} q_{m-n} \\ q_{m-n-1} \\ \vdots \\ q_1 \\ q_0 \end{bmatrix} = \begin{bmatrix} s_m \\ s_{m-1} \\ \vdots \\ s_{n+1} \\ s_n \\ \vdots \\ s_{m-n-1} \end{bmatrix} \Leftrightarrow T\,\mathbf{q} = \mathbf{s}, \qquad (16)$$

where $\mathbf{q} = (q_i)_{i=0}^{m-n}$, $\mathbf{s} = (s_i)_{i=m-n+1}^{m}$, and T is the nonsingular lower triangular Toeplitz matrix, defined by its first column vector $\mathbf{t} = (t_i)_{i=0}^{n}$, $t_n = 1$. Write $T = Z(\mathbf{t})$ and $Z = Z(\mathbf{e}_2)$ where $\mathbf{e}_2 = (0, 1, 0 \ldots, 0)^T$ is the second coordinate vector, and express the matrix T as a polynomial in a generator matrix $Z = Z_{n+1}$ of size $(n+1) \times (n+1)$ as follows,

$$Z = \begin{pmatrix} 0 & & \cdots & & 0 \\ 1 & \ddots & & & \\ \vdots & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & \\ 0 & & \cdots & 1 & 0 \end{pmatrix}, \quad T = Z(\mathbf{t}) = t(Z) = \sum_{i=0}^{n} t_i Z^i, \quad Z^{n+1} = O.$$

The matrix T is nonsingular because $t_n \neq 0$, and the latter equations imply that the inverse matrix $T^{-1} = t(Z)^{-1} \mod Z^{n+1}$ is again a polynomial in Z, that is again a lower triangular Toeplitz matrix defined by its first column. We compute this column by applying a divide and conquer algorithm. Assume that

$n+1=2^k$ is a power of two, for a positive integer k. If this is not the case, embed the matrix T into a lower triangular Toeplitz $q \times q$ matrix $\bar{T} = \bar{t}(Z_q)$ for $q = 2^k$ and $k = \lceil \lg(n+1) \rceil$ with the leading (that is northwestern) block $T = t(Z_q)$, such that $t(Z_q) = \bar{t}(Z_q) \mod Z_q^{n+1}$, compute the inverse matrix $\bar{T}^{-1}$ and output its leading $(n+1) \times (n+1)$ block T^{-1}.

Now represent T as the 2×2 block matrix, $T = \begin{bmatrix} T_0 & 0 \\ T_1 & T_0 \end{bmatrix}$ where T_0 and T_1 are $\frac{q}{2} \times \frac{q}{2}$ Toeplitz submatrices of the Toeplitz matrix T, T_0 is invertible, and observe that

$$T^{-1} = \begin{bmatrix} T_0 & 0 \\ T_1 & T_0 \end{bmatrix}^{-1} = \begin{bmatrix} T_0^{-1} & 0 \\ -T_0^{-1} T_1 T_0^{-1} & T_0^{-1} \end{bmatrix}. \tag{17}$$

We only seek the first column of the matrix T^{-1}. Its computation amounts to solving the same problem for the half-size triangular Toeplitz matrix T_0 and to multiplication of each of the $\frac{q}{2} \times \frac{q}{2}$ Toeplitz matrices T_1 and T_0^{-1} by a vector. Let $TTI(s)$ and $TM(s)$ denote the arithmetic cost of $s \times s$ triangular Toeplitz matrix inversion and multiplying an $s \times s$ Toeplitz matrix by a vector, respectively. Then the above analysis implies that $TTI(q) \le TTI(q/2) + 2TM(q/2)$. Reapply this bound to $TTI(q/2^g)$ for $g = 1, 2, \dots$ recursively, and deduce that $TTI(q) \le \sum_{g=1}^{h} TM(q/2^g)$. The following simple lemma (cf. [38, equations (2.4.3) and (2.4.4)]) reduce Toeplitz-by-vector multiplication to polynomial multiplication and the extraction of a subvector of the coefficient vector of the product, thus implying that $TM(s) \le cs \lg s$ for a constant c and consequently $TTI(q) < 2cq \lg q$.

Lemma 18. *The vector equation*

$$\begin{pmatrix} u_0 & & O \\ \vdots & \ddots & \\ \vdots & \ddots & u_0 \\ u_m & \ddots & \vdots \\ & \ddots & \vdots \\ O & & u_m \end{pmatrix} \begin{pmatrix} v_0 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} p_0 \\ \vdots \\ \vdots \\ p_m \\ \vdots \\ p_{m+n} \end{pmatrix} \tag{18}$$

is equivalent to the polynomial equation

$$\left(\sum_{i=0}^{m} u_i x^i \right) \left(\sum_{i=0}^{n} v_i x^i \right) = \sum_{i=0}^{m+n} p_i x^i. \tag{19}$$

We wish to estimate the Boolean (rather than arithmetic) cost of inverting a triangular Toeplitz matrix T and then extend this to the Boolean cost bound of computing the vector $T^{-1}\mathbf{s}$ and of polynomial division. So next we assume that

the input polynomials are known up to some precision $2^{-\lambda}$ and employ the above reduction of the problem to recursive (approximate) polynomial multiplications.

To study the Boolean complexity of this procedure, we need the following corollary, which is a direct consequence of Lemma 17, and inequality $\lg(2d+1) \leq 2 + \lg d$.

Corollary 19 (Bounds for the product $P_0^2 P_1$). *Let P_0 be of degree d such that $\|P_0\|_\infty \leq 2^{\tau_0}$, and its coefficients are known up to a precision $2^{-\lambda}$. Similarly, let P_1 be of degree $2d$, $\|P_1\|_\infty \leq 2^{\tau_1}$, and its coefficients are known up to a precision $2^{-\nu}$. Then the polynomial $P = P_0^2 P_1$ is of degree $4d$, $\|P_0^2 P_1\|_\infty \leq 2^{2\tau_0+\tau_1+6\lg d+8}$, and the coefficients are known within the error bound $2^{-\nu+8\tau_0+2\tau_1+15\lg d+40}$.*

We will estimate by induction the cost of inverting the matrix T, by using Eq. (17) recursively.

Lemma 20. *Let $n+1 = 2^h$ for a positive integer h and let T be a lower triangular Toeplitz $(n+1) \times (n+1)$ matrix of. Eq. (16), having ones on the diagonal. Let it subdiagonal entries be complex numbers of magnitude at most 2^τ. Write $T^{-1} = (T^{-1}_{i,j})^n_{i,j=0}$. Then*

$$\left|\lg \max_{i,j} |T^{-1}_{i,j}|\right| \leq (2n-1)\tau + 3n - 20\lg n \ .$$

Furthermore, to compute the entries of T^{-1} up to the precision of ℓ bits, that is to compute a matrix $\widetilde{T}^{-1} = (\widetilde{T}^{-1}_{i,j})^n_{i,j=0}$ such that

$$\max_{i,j} |T^{-1}_{i,j} - \widetilde{T}^{-1}_{i,j}| \leq 2^{-\ell} \ ,$$

it is sufficient to know the entries of T up to precision of $\ell + 16n\tau + 56n + 6\tau + 2\lg n - 68$, that is $\mathcal{O}(\ell + n\tau)$ bits, and the computation of $\widetilde{T}^{-1}$ costs $\mathcal{O}_B(n\lg^2 n \cdot \mu(\ell + n\tau))$, that is $\widetilde{\mathcal{O}}_B(n\ell + n^2\tau)$.

Proof: We will prove the claimed estimates by applying the reduction of the inversion to recursive multiplication of polynomials defined by equation (17) and Lemma 18.

Consider the $\frac{n+1}{2} \times \frac{n+1}{2}$ Toeplitz matrices T_0^{-1} and T_1 of equation (17). The Toeplitz matrix T_0^{-1} is triangular, and so its first column, $\mathbf{p} = (p_i)_{i=0}^{(n-1)/2}$, with $p_0 = 1$, defines this matrix and the polynomial $p(x) = \sum_{i=0}^{(n-1)/2} p_i x^i$ of degree $(n-1)/2$. Likewise the vector $\mathbf{t} = (t_{n-i})_{i=1}^n$ (made up of two overlapping vectors, that is, the reversed first row, $(t_{n-1}, t_{n-2}, \ldots, t_{(n-1)/2})$, of the matrix T_1 and its first column, $(t_{(n-1)/2}, t_{(n-3)/2}, \ldots, t_0)^T$) defines this matrix and the polynomial $t(x) = \sum_{i=1}^n t_{n-i} x^{i-1}$ of degree $n-1$. The first column of the $\frac{n+1}{2} \times \frac{n+1}{2}$ Toeplitz matrix $T_1 T_0^{-1}$ is a subvector, $\mathbf{v}$, of dimension $(n+1)/2$ of the coefficient vector of the polynomial product $t(x)p(x)$, having degree $3(n-1)/2$. Likewise the first column of the $\frac{n+1}{2} \times \frac{n+1}{2}$ matrix $-T_0^{-1} T_1 T_0^{-1}$ is the vector $\mathbf{q} = -T_0^{-1}\mathbf{v}$, which is a subvector of dimension $(n+1)/2$ of the coefficient vector of the polynomial

product $p(x)v(x)$ of degree $n-1$, where the polynomial $v(x)$ of degree $(n-1)/2$ is defined by its coefficient vector $\mathbf{v}$. In sum the vector $\mathbf{q}$ is the coefficient vector of a polynomial $q(x)$ obtained by two successive multiplications of polynomials. (Namely we first compute the polynomial $t(x)p(x)$, then truncate it to obtain the polynomial $v(x)$, then compute the polynomial $-p(x)v(x)$, and finally truncate it to obtain the polynomial $q(x)$.) The truncation can only decrease the degree of a polynomial and the maximum length of its coefficients, and so we can bound the precision and the cost of computing the matrix product $-T_0^{-1}T_1T_0^{-1}$ by the bounds on the precision and the cost of computing the polynomial product $P_0^2\,P_1$, estimated in Corollary 19 for $P_0 = p(x)$, $P_1 = t(x)$, and $d = (n-1)/2$.

Recall that $n+1 = 2^k$ by assumption and that the polynomial $P_0^2\,P_1$ has degree $2n-2$. We write $k = \lceil \lg(2n-2) \rceil$ and then prove by induction that

$$\lg \left\| P_0^2\,P_1 \right\|_\infty \le (2^{k+1}-1)\tau + 3\cdot 2^k - 20k \ . \tag{20}$$

Under the assumption that the input elements are known up to a precision of λ bits, we compute a polynomial $\widetilde{P_0^2\,P_1}$ such that

$$\lg \left\| P_0^2 P_1 - \widetilde{P_0^2\,P_1} \right\|_\infty \le 16\cdot 2^k\tau + 56\cdot 2^k + 6\tau + 2\,k - 68 \ . \tag{21}$$

We proceed by induction. Let the base case be $n+1 = 4$, and then invert the matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ t_2 & 1 & 0 & 0 \\ t_1 & t_2 & 1 & 0 \\ t_0 & t_1 & t_2 & 1 \end{bmatrix} = \begin{bmatrix} T_0 & 0 \\ T_1 & T_0 \end{bmatrix}, \text{ where } T_0 = \begin{bmatrix} 1 & 0 \\ t_2 & 1 \end{bmatrix}, T_0^{-1} = \begin{bmatrix} 1 & 0 \\ -t_2 & 1 \end{bmatrix}, T_1 = \begin{bmatrix} t_1 & t_2 \\ t_0 & t_1 \end{bmatrix},$$

and $|t_i| \le 2^\tau$. The associated polynomials are $P_0(x) = 1 - t_2x$, for T_0^{-1}, and $P_1(x) = t_2 + t_1x + t_0x^2$, for T_1. Therefore $P_1P_0 = (1-t_2x)(t_2+t_1x+t_0x^2) = t_2 + (t_1 - t_2^2)x + (t_0 - t_1t_2)x^2 - t_0t_2x^3$. The subvector $\mathbf{v} = (t_1 - t_2^2, t_0 - t_1t_2)^T$ of the coefficient vector of the polynomial product P_1P_0 is the first column of the matrix product $T_1T_0^{-1}$. Furthermore the vector $-T_0^{-1}\mathbf{v} = (t_2^2 - t_1, 2t_1t_2 - t_2^3 - t_0)^T$ is a subvector of the coefficient vector of the polynomial product $P_0v = (1-t_2x)(t_1 - t_2^2 + (t_0 - t_1t_2)x)$ where v denotes the polynomial $t_1 - t_2^2 + (t_0 - t_1t_2)x$ with the coefficient vector $\mathbf{v}$.

Now we overestimate the magnitude of the coefficients of this product by estimating the norm of the polynomial $P_0^2P_1 = (1-t_2x)^2(t_2+t_1x+t_0x^2) = t_2 + (t_1 - 2t_2^2)x + (t_0 - 2t_1t_2 + t_2^3)x^2 + (t_2^2t_1 - 2t_2t_0)x^3 + t_2^2t_0x^4$. By overestimating the coefficient of x^2 we obtain

$$\|P_0^2P_1\|_\infty \le (2^\tau)^3 + 2\cdot 2^\tau\,2^\tau + 2^\tau \le 3\cdot 2^{3\tau} \le 2^{3\tau+2} \ ,$$

and therefore

$$\lg\|P_0^2P_1\|_\infty \le 3\tau + 2 \le 7\tau - 28 \ ,$$

where the right-hand side represents bound Eq. (20) for $k = 2$ and where the latter bound holds for all $\tau \geq 8$.

We perform the multiplication using the algorithm of Lemma 17 and then deduce that

$$\lg\|P_0^2 P_1 - \widetilde{P_0^2\, P_1}\|_\infty \leq 10\tau + 4\lg 2 + 42 \leq 70\tau + 160 \ ,$$

where the right-hand side represents bound Eq. (21) for $k = 2$.

It remains to prove the induction. Suppose the claimed bounds are true for $n + 1 = 2^k$ and extend them to $n + 1 = 2^{k+1}$. In this case P_0 is a polynomial of degree $d = 2^{k-1} - 1$. Apply the induction hypothesis and obtain $\|P_0\|_\infty \leq 2^{(2^k-1)\tau+3\cdot 2^{k-1}-20(k-1)}$ and

$$\lg \left\| P_0 - \widetilde{P}_0 \right\|_\infty \leq -\lambda + 16 \cdot 2^{k-1}\tau + 56 \cdot 2^{k-1} + 6\tau + 2\,(k-1) - 68 \ . \qquad (22)$$

The polynomial P_1 has degree $2^k - 2$, and we can assume that we know its coefficients up to the same precision as those of P_0 and furthermore that $\|P_1\|_\infty \leq 2^\tau$, because the coefficients are the entries of the input matrix T.

We form the product $P_0^2 P_1$ and its approximation, based on Cor. 19. In our case we have $d = 2^{k-1} - 1$, $\tau_0 = (2^k - 1)\tau + 3 \cdot 2^{k-1} - 20(k-1)$, $\tau_1 = \tau$, and $-\nu = -\lambda + 16 \cdot 2^{k-1}\tau + 56 \cdot 2^{k-1} + 6\tau + 2\,(k-1) - 68$.

We apply Cor 19 and obtain

$$\lg \left\| P_0^2\, P_1 \right\|_\infty \leq 2\tau_0 + \tau_1 + 6\lg 2^{k-1} + 8 = (2^{k+1}-1)\tau + 3\cdot 2^k - 34k + 42 \leq (2^{k+1}-1)\tau + 3\cdot 2^k - 20k \ ,$$

where the last inequality, which is due to Eq. (20), is true for all $k \geq 3$. We also estimate that it is sufficient to compute the product $P_0^2 P_1$ with the bit-precision

$$\lg \left\| P_0^2 P_1 - \widetilde{P_0^2\, P_1} \right\|_\infty \leq -\nu + 8\tau_0 + 2\tau_1 + 15k + 40 \leq -\lambda + 16\cdot 2^k\tau + 56\cdot 2^k + 6\tau + 2\,k - 68 \ .$$

Combine this bound with the previous ones and deduce that it is sufficient to perform all the computations by using at most $\ell + 16\cdot 2^k\tau + 56\cdot 2^k + 6\tau + 2\,k - 68 = \ell + 16n\tau + 56n + 6\tau + 2\,\lg n - 68$, that is $\mathcal{O}(\ell + n\tau)$ bits of precision.

We perform k steps overall, for $1 \leq k \leq \lg(n+1)$. At each step we apply two multiplications of polynomials of degrees at most 2^{k-1} whose coefficients have magnitude less than $2^{\mathcal{O}(n\tau)}$, and use a precision of at most $\ell + 16n\tau + 56n + 6\tau + 2\,\lg n - 68 = \mathcal{O}(\ell + n\tau)$ bits. By combining these estimates together, we obtain

$$\sum_{k=1}^{\lg n} k \cdot 2\,\mathcal{O}_B(2^k \cdot \lg 2^k \cdot \mu(\ell + n\tau)) = \mathcal{O}_B(n\ \lg^2(n)\,\mu(\ell + n\tau)) \ ,$$

that concludes the proof. □

Estimating the complexity of approximate polynomial division we assume for simplicity that $m = 2n$.

Theorem 21. *Let $s, t \in \mathbb{C}[x]$ denote two polynomals of degrees at most $2n$ and n, respectively, such that $\|s\|_\infty \leq 2^{\tau_1}$ and $\|t\|_\infty \leq 2^{\tau_2}$. Suppose we are given the coefficients of these polynomials up to the precision $\lambda = \ell + 30n\tau_2 + \tau_1 + 80n$, that is we are given two polynomials $\widetilde{s}$ and $\widetilde{t}$ such that $\|s - \widetilde{s}\|_\infty \leq 2^{-\lambda}$ and $\|t - \widetilde{t}\|_\infty \leq 2^{-\lambda}$. Let q and r denote the quotient and the remainder of the division of the polynomials s by t, that is $s = t \cdot q + r$ and $\deg r < \deg t$. Then within the cost bounds in $\mathcal{O}_B(n \lg^2 n \cdot \mu(\ell + n\tau_2 + \tau_1))$, that is in $\widetilde{\mathcal{O}}_B(n\ell + n^2\tau_2 + n\tau_1)$, we can compute two polynomials $\widetilde{q}$ and $\widetilde{r}$ such that $\|q - \widetilde{q}\|_\infty \leq 2^{-\ell}$ and $\|r - \widetilde{r}\|_\infty \leq 2^{-\ell}$. Moreover, $\|q\|_\infty \leq 2^{2n\tau_2 + 10n}$ and $\|r\|_\infty \leq 2^{3n\tau_2 + \tau_1 + 12n}$.*

Proof: We compute the coefficients of $q(x)$ using Eq. (16), that is $\mathbf{q} = T^{-1}\,\mathbf{s}$. Each coefficient of the vector $\mathbf{q}$ comes as an inner product of two vectors, i.e. $q_i = \sum_{j=0}^{n} T_{i,j}^{-1} s_j$.

From Lemma 20 we know that $|\lg|T_{i,j}^{-1}|| \leq (2n-1)\tau_2 + c_1$, where $c_1 = 3n - 20\lg n$ and $\lg|T_{i,j}^{-1} - \widetilde{T}_{i,j}^{-1}| \leq \lambda + 16n\tau_2 + c_2$, where $c_2 = 56n + 6\tau_2 + 2\lg n - 68$. For the coefficients of s, s_j, we deduce that $\lg|s_j| \leq \tau_2$ and $\lg|s_j - \widetilde{s}_j| \leq -\lambda$.

Therefore $\lg|T_{i,j}^{-1} s_j| \leq 2n\tau_2 + c_1$ and $\lg|T_{i,j}^{-1} s_j - \widetilde{T}_{i,j}^{-1}\widetilde{s}_j| \leq -\lambda + 16n\tau_2 + \tau_2 + c_1 + c_2$ for all i and j, and so

$$\lg\|q\|_\infty \leq \max_i \lg|\sum_j T_{i,j}^{-1} s_j| \leq 2n\tau_2 + c_1 + \lg n \leq 2n\tau_2 + 10n$$

and

$$\lg\|q - \widetilde{q}\|_\infty \leq \lg|\sum_j T_{i,j}^{-1} s_j - \sum_j \widetilde{T}_{i,j}^{-1}\widetilde{s}_j| \leq -\lambda + 16n\tau_2 + \tau_2 + c_1 + c_2 + \lg n \leq -\lambda + 23n\tau_2 + 64n \ .$$

To compute the remainder $r(x) = s(x) - t(x)q(x)$ we only need an approximate polynomial multiplication and a subtraction.

Recall the bounds $\lg\|t\|_\infty \leq \tau_2$ and $\lg\|t - \widetilde{t}\|_\infty \leq -\lambda$, apply Lemma 17, and deduce that $\lg\|t\,q\|_\infty \leq 3n\tau_2 + c_1 + 4\lg n$ and $\lg\|t\,q - \widetilde{t}\widetilde{q}\|_\infty \leq -\lambda + 24n\tau_2 + 3c_1 + c_2 + 10\lg n + 20$.

To cover the impact of the subtraction, we apply some simplifications that make bounds less scary (albeit less accurate wrt the constants involved), and thus obtain

$$\lg\|r\|_\infty \leq 3n\tau_2 + c_1 + 4\lg n + \tau_1 \leq 3n\tau_2 + \tau_1 + 12n$$

and

$$\lg\|r - \widetilde{r}\|_\infty \leq -\lambda + 24n\tau_2 + \tau_1 + 3c_1 + c_2 + 10\lg n + 22 \leq -\lambda 30n\tau_2 + \tau_1 + 80n \ .$$

The overall complexity is dominated at the stage of computing the entries of the matrix $\widetilde{T}_{i,j}^{-1}$. □

4. Average case analysis of DES and αDES

The most time consuming part of $\mathtt{R_3}$ and $\alpha\mathtt{R_3}$ is the αDES procedure. It requires, in the worst case,

$$g_1^2 = \lceil \lg(L - 1 + \lg w) \rceil^2 = \mathcal{O}(\lg^2(\tau + L))$$

evaluations of our input polynomial A. The worst case occurs where some extraneous roots of A lie very close to the endpoints of the initial interval I. However, in practice this behavior is rare, if it occurs at all. To explain this phenomenon, Pan and Linzer [42] estimated the average number of steps of DES under the assumption that a real root is uniformly distributed in an interval and concluded that in this case $\mathtt{R_3}$, and hence $\alpha\mathtt{R_3}$, needed a constant number of steps, with a high probability.

Even though the assumption on the equidistribution of the real roots in [42] is plausible, we are not aware of any distribution on the coefficients that results in such a behavior for the roots. We consider an average case analysis where the root we approximate is a real root of a random Weyl or $SO(2)$ polynomial [20]. The density function of the real roots is considerably different from uniform in these cases, but we also arrive at the same conclusion, that is the algorithms $\mathtt{R_3}$ and $\alpha\mathtt{R_3}$ perform a constant number of steps with high probability.

4.1. Weyl polynomials

Weyl polynomials, are random polynomials of the form $A = \sum_{i=0}^{d} a_i x^i / \sqrt{i!}$, with independent standard normal coefficients a_i. Alternatively, we could consider them as $A = \sum_{i=0}^{d} a_i x^i$, where a_i are normal variables having mean 0 and variance $1/\sqrt{i!}$. As the degree grows, the real roots, except for a constant number of them, are in the interval $[-\sqrt{d} \,..\, \sqrt{d}]$. Their (asymptotic) probability density function is $f(t) = \frac{1}{2\sqrt{d}}$. Then the probability value

$$\text{Probability}[\alpha_1 \in [a \,..\, a + w/2^{2^k}]] = \Pr[g_1 > k] = \int_a^{a+w/2^{2^k}} f(t)dt = \frac{w}{2^{2^k+1}\sqrt{d}}$$

rapidly converges to 0 as k or d grows large.

4.2. $SO(2)$ polynomials

Random polynomials of the form $A = \sum_{i=0}^{d} a_i x^i$, where the coefficients are i.d. normals with mean zero and variances $\binom{d}{i}$, where $0 \le i \le d$ are called $SO(2)$ polynomials. Alternatively, we can define them as $A = \sum_{i=0}^{d} \sqrt{\binom{d}{i}}\, a_i\, x^i$, where a_i are i.i.d. standard normals. They are called $SO(2)$ because the joint probability distribution of their zeros is $SO(2)$ invariant, after homogenization.

The (asymptotic) probability density function of the real roots of $SO(2)$ random polynomials is $f(t) = \frac{1}{\pi(1+t^2)}$.

Assume that the isolating interval $I = [a\,..\,b]$ is a subset of $(0\,..\,1)$, and let its width be w. We can treat similarly the case where I is a subset of $(1\,..\,\infty)$. Then the probability value

$$\begin{aligned}
&\text{Probability}\left[\alpha_1 \in [a\,..\,a + w/2^{2^k}]\right] = \Pr[g_1 > k] \\
&= \int_a^{a+w/2^{2^k}} f(t)dt = \arctan(a) - \arctan(a + w/2^{2^k}) \\
&= \arctan \frac{w/2^{2^k}}{1 + a^2\, w/2^{2^k}} \leq \frac{w}{2^{2^k} + a^2 w} - \frac{1}{3}\left(\frac{w}{2^{2^k} + a^2 w}\right)^3 \quad ,
\end{aligned}$$

rapidly converges to 0 as in the case of Weyl polynomials. However, now there is no dependence on the degree but only on the endpoints of the (initial) isolating interval.

5. Conclusions and future work

We present an approximate variant of a real root refinement algorithm based on the *Bisection of the Exponents*, or *Double Exponential Sieve* algorithm, bisection and Newton operator. The complexity of the algorithm is $\widetilde{\mathcal{O}}_B(d^2\tau + dL)$.

Can we combine α`DES` with the approximate version of `qir` in [26] to provide an alternative method that would guarantee quadratic convergence? We believe that this is a very interesting approach to explore.

For random polynomials, it is reasonable to assume that we can derive an even faster algorithm for real root refinement that takes advantage of the distribution of the roots.

Acknowledgments

VP is supported by NSF Grant CCF–1116736 and PSC CUNY Awards 64512–0042 and 65792–0043. ET is partially supported by the EXACTA grant of the National Science Foundation of China (NSFC 60911130369) and the French National Research Agency (ANR-09-BLAN-0371-01), GeoLMI (ANR 2011 BS03 011 06), HPAC (ANR ANR-11-BS02-013) and an FP7 Marie Curie Career Integration Grant. Both authors are also grateful to Michael Sagraloff for his helpful comment to section 2.4.

[1] J. Abbott. Quadratic interval refinement for real roots. *ACM Commun. Comput. Algebra*, 48(1/2):3–12, July 2014.

[2] S. Basu, R. Pollack, and M-F.Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2nd edition, 2006.

[3] M. Ben-Or and P. Tiwari. Simple algorithms for approximating all roots of a polynomial with real roots. *Journal of Complexity*, 6(4):417–442, 1990.

[4] D. Bini and V. Pan. *Polynomial and Matrix Computations*, volume 1: Fundamental Algorithms. Birkhäuser, Boston, 1994.

[5] D. Bini and V. Y. Pan. Parallel complexity of tridiagonal symmetric eigenvalue problem. In *Proc. 2nd Annual ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 384–393, 1991.

[6] D. Bini and V. Y. Pan. Practical improvement of the divide-and-conquer eigenvalue algorithms. *Computing*, 48:109–123, 1992.

[7] D. Bini and V. Y. Pan. Computing matrix eigenvalues and polynomial zeros where the output is real. *SIAM J. Comput.*, 27(4):1099–1115, 1998.

[8] L. Bluestein. A linear filtering approach to the computation of discrete fourier transform. *Audio and Electroacoustics, IEEE Transactions on*, 18(4):451–455, 1970.

[9] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[10] M. Bodrato and A. Zanoni. Long integers and polynomial evaluation with Estrin's scheme. In *Proc. 11th Int'l Symp. on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 39–46. IEEE, 2011.

[11] J. Cheng, S. Lazard, L. M. Peñaranda, M. Pouget, F. Rouillier, and E. P. Tsigaridas. On the topology of planar algebraic curves. In J. Hershberger and E. Fogel, editors, *Proc. 25th Annual ACM Symp. Comput. Geom. (SoCG)*, pages 361–370, Aarhus, Denmark, 2009.

[12] G. E. Collins and W. Krandick. A hybrid method for high precision calculation of polynomial real roots. In *Proc. ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 47–52, 1993.

[13] J. H. Davenport. Cylindrical algebraic decomposition. Technical Report 88–10, School of Mathematical Sciences, University of Bath, England, available at: http://www.bath.ac.uk/masjhd/, 1988.

[14] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the asymptotic and practical complexity of solving bivariate systems over the reals. *J. Symbolic Computation*, 44(7):818–835, 2009. (Special issue on ISSAC 2007).

[15] Q. Du, M. Jin, T. Li, and Z. Zeng. Quasi-laguerre iteration in solving symmetric tridiagonal eigenvalue problems. *SIAM J. Sci. Computing*, 17(6):1347–1368, 1996.

[16] Q. Du, M. Jin, T. Li, and Z. Zeng. The quasi-laguerre iteration. *Mathematics of Computation of the American Mathematical Society*, 66(217):345–361, 1997.

[17] Z. Du, V. Sharma, and C. K. Yap. Amortized bound for root isolation via Sturm sequences. In D. Wang and L. Zhi, editors, *Int. Workshop on Symbolic Numeric Computing*, pages 113–129, School of Science, Beihang University, Beijing, China, 2005. Birkhauser.

[18] A. Eigenwillig. *Real root isolation for exact and approximate polynomials using Descartes' rule of signs.* PhD thesis, Universität des Saarlandes, 2008.

[19] P. Emeliyanenko and M. Sagraloff. On the complexity of solving a bivariate polynomial system. In *Proc. 37th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 154–161, Grenoble, France, July 2012. ACM.

[20] I. Z. Emiris, A. Galligo, and E. P. Tsigaridas. Random polynomials and expected complexity of bisection methods for real solving. In S. Watt, editor, *Proc. 35th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 235–242, Munich, Germany, July 2010. ACM.

[21] M. J. Fischer and M. S. Paterson. String-matching and other products. In R. Karp, editor, *Complexity of Computation*, volume 7, pages 113–125. SIAM-AMS Proc., 1974.

[22] M. Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

[23] W. Hart and A. Novocin. Practical divide-and-conquer algorithms for polynomial arithmetic. In *Proc. CASC*, volume 6885 of *LNCS*, pages 200–214. Springer, 2011.

[24] J. R. Johnson. *Algorithms for Polynomial Real Root Isolation.* PhD thesis, The Ohio State University, 1991.

[25] M. Kerber. On the complexity of reliable root approximation. In *In Proc. 11th Int'l Wkshp on Computer Algebra in Scientific Computing (CASC)*, volume 5743 of *LNCS*, pages 155–167, 2009.

[26] M. Kerber and M. Sagraloff. Efficient real root approximation. In *Proc. 36th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 209–216, San Jose, CA, USA, June 2011. ACM.

[27] M. Kerber and M. Sagraloff. A worst-case bound for topology computation of algebraic curves. *J. Symb. Comput.*, 47(3):239–258, 2012.

[28] P. Kirrinnis. Partial fraction decomposition in $C\langle z\rangle$ and simultaneous Newton iteration for factorization in $C[z]$. *Journal of Complexity*, 14(3):378–444, 1998.

[29] D. E. Knuth. *The art of computer programming, volume 2 (2nd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[30] A. Kobel and M. Sagraloff. Fast approximate polynomial multipoint evaluation and applications. *arXiv preprint arXiv:1304.8069*, 2013.

[31] A. Mantzaflaris, B. Mourrain, and E. P. Tsigaridas. On continued fraction expansion of real roots of polynomial systems, complexity and condition numbers. *Theor. Comput. Sci.*, 412(22):2312–2330, 2011.

[32] J. M. McNamee and V. Y. Pan. *Numerical methods for roots of polynomials (II)*, chapter 15. Elsevier, 2013.

[33] K. Mehlhorn, M. Sagraloff, and P. Wang. From approximate factorization to root isolation with application to cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 66:34–69, 2015.

[34] M. Mignotte. *Mathematics for Computer Algebra*. Springer-Verlag, New York, 1991.

[35] V. Y. Pan. Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros. In *Proc. STOC*, pages 741–750. ACM, 1995.

[36] V. Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Comp. and Math. (with Appl.)*, 31:97–138, 1996.

[37] V. Y. Pan. Approximating complex polynomial zeros: modified Weyl's quadtree construction and improved Newton's iteration. *J. of Complexity*, 16(1):213–264, 2000.

[38] V. Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Birkhäuser / Springer, Boston / New York, 2001.

[39] V. Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *J. Symbolic Computation*, 33(5):701–733, 2002.

[40] V. Y. Pan. Transformations of matrix structures work again. *Linear Algebra and Its Applications*, 465, 1–32 (2015).

[41] V. Y. Pan. Real polynomial root-finding by means of matrix and polynomial iterations. In Proc. *16th International Workshop Computer Algebra in Scientific Computing (CASC)*, (V. P. Gerdt, V. Koepf, W. M. Zeiler, and E. V. Vorozhtsov, editors), Lecture Notes in Computer Science, 8660, 335–349, Springer, Cham (2014). Also at arXiv preprint: arXiv:1501.05390 [cs.SC] (24 pages, 12 tables), 22 January 2015.

[42] V. Y. Pan and E. Linzer. Bisection acceleration for the symmetric tridiagonal eigenvalue problem. *Numerical Algorithms*, 22(1):13–39, 1999.

[43] V. Y. Pan, B. Murphy, R. E. Rosholt, G. Qian, and Y. Tang. Real root-finding. In S. M. Watt and J. Verschelde, editors, *Proc. 2nd ACM Int'l Work. Symbolic Numeric Computation (SNC)*, pages 161–169, 2007.

[44] V. Y. Pan and E. P. Tsigaridas. On the boolean complexity of real root refinement. In *Proc. 38th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 299–306, Boston, USA, Jun 2013. ACM.

[45] V. Y. Pan and E. Tsigaridas. Accelerated Approximation of the Complex Roots of a Univariate Polynomial (Extended Abstract). In *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation*, pages 132–134, Shanghai, China, July 2014. ACM. Also at arXiv preprint: arXiv:1501.05392 [cs.SC] (9 pages), 22 January 2015. and http://hal.inria.fr/hal-00980584

[46] V. Y. Pan, E. Tsigaridas, and L. Zhao. Simple and efficient real root-finding for a univariate polynomial. *arXiv preprint arXiv:1501.05386*, 2015.

[47] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of polynomials. *Journal of Complexity*, 3(2):90–113, 1987.

[48] F. Rouillier. On solving systems of bivariate polynomials. *Mathematical Software–ICMS 2010*, 6327:100–104, 2010.

[49] M. Sagraloff. When Newton meets Descartes: A simple and fast algorithm to isolate the real roots of a polynomial. In *Proc. 37th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 297–304, Grenoble, France, July 2012. ACM.

[50] M. Sagraloff and K. Mehlhorn. Computing real roots of real polynomials-an efficient method based on Descartes' rule of signs and newton iteration. *arXiv preprint arXiv:1308.4088*, 2013.

[51] A. Schönhage. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coeficients. In J. Calmet, editor, *EUROCAM*, volume 144 of *LNCS*, pages 3–15, 1982.

[52] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Manuscript. Univ. of Tübingen, Germany, 1982. URL: http://www.iai.uni-bonn.de/~schoe/fdthmrep.ps.gz.

[53] V. Sharma, P. Batra, and T. Hamburg. Near optimal subdivision algorithms for real root isolation. *arXiv preprint arXiv:1501.07774*, 2015.

[54] V. Sharma, Z. Du, and C. Yap. Robust approximate zeros. *Proc. European Symposium of Algorithms (ESA)*, pages 874–886, 2005.

[55] A. Strzeboński and E. P. Tsigaridas. Univariate real root isolation in an extension field. In A. Leykin, editor, *Proc. 36th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 321–328, San Jose, CA, USA, June 2011. ACM.

[56] J. van der Hoeven. Fast composition of numeric power series. Technical Report 2008-09, Université Paris-Sud, Orsay, France, 2008.

Algorithm 3:
DES(A, I)

Input: $A \in \mathbb{Z}[X], I_0 = [a_0, b_0]$
Output: I, where $\mathtt{ir}(I) \geq 5/3$
Data: This is a recursive procedure. Initially it is called with the interval $I_0 = [0, 1]$

```
1  2w_0 = b_0 − a_0 ;
2  g_1 ← 0 ;
3  I ← I_0 ;
4  repeat
5  |  g_1 ← g_1 + 1 ;
6  |  c_{g_1} ← a_0 + 2w_0/2^{g_1} ;
7  |  TEST IF α_1 ∈ [0 .. c_{g_1}] ;
8  until α_1 ∉ [0 .. c_{g_1}] ;
9  I ← [c_{g_1} .. 1] ;
10 if g_1 = 1 then
   |  /* In this case ir(I) ≥ 5/3.                          */
   |  /* We can ensure ir(I) ≥ 3 with two bisections.       */
11 |  RETURN IncrIsolationRatio_by_subdiv(A, I, 5/3, 3) ;
12 else
13 |  RETURN DES(A, I) ;
```


[57] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 3rd edition, 2013.

[58] C. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.

6. Appendix

Algorithm 4:
IncrIsolationRatio_by_subdiv(A, I, r, R, dir)

Input: $A \in \mathbb{Z}[X], I = [a, b]$, $L \in \mathbb{Z}$
Output: (r, I), where $r \geq R$
Data: Initially $r < R$. The direction of the closest root, dir, may or may not be given.

```
1  if dir = ∅ then
2      m ← (a+b)/2 ;
3      s_m ← sgn(A(m)) ;
4      if s = 0 then I = [m..m]; return (r = ∞, I) ;
5      if s_l · s_m < 0 then I ← [a..m]; dir = left ;
6      if s_r · s_m < 0 then I ← [m..b]; dir = right ;
7      r ← 2(r − 1) + 1 ;
8  while r < R do
9      m ← (a+b)/2 ;
10     s_m ← sgn(A(m)) ;
11     if s = 0 then I = [m..m]; return (r = ∞, I) ;
12     if s_l · s_m < 0 then I ← [a..m]; cdir = left ;
13     if s_r · s_m < 0 then I ← [m..b]; cdir = right ;
14     if cdir ≠ dir then
15         r ← 2(r − 1) + 1
16     else
17         r ← 2(r + 1) − 1
18 return (I, r)
```

Algorithm 5: NEWTON(A, I, t)

Input: $A \in \mathbb{Z}[X], I = [a, b]$, $t = 2^K, L \in \mathbb{Z}$
Output: Interval J such that $w(J) < 2^{-L}w$
Data: It holds that $\mathtt{ir}(I) > 20d^2$

```
1 x_1 ← m(I) ;
2 x_0 ← ∞ ;
3 while |x_1 − x_0| > 2^{−L} w do
4 |   SWAP(x_1, x_0) ;
5 |   x_1 ← x_0 − A(x_0)/A'(x_0) ;
6 RETURN [x_0..x_1]
```