New Hybrid Nearly Optimal Polynomial Root-finders*

Victor Y. Pan

Department of Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA and
Ph.D. Programs in Mathematics and Computer Science
The Graduate Center of the City University of New York
New York, NY 10036 USA
victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/

Abstract

Univariate polynomial root-finding has been studied for four millennia and still remains the subject of intensive research. Hundreds if not thousands of efficient algorithms for this task have been proposed and analyzed. Two nearly optimal solution algorithms have been devised in 1995 and 2016, based on recursive factorization of a polynomial and subdivision iterations, respectively, but both of them are superseded in practice by Ehrlich's functional iterations. By combining factorization techniques with Ehrlich's and subdivision iterations we devise a variety of hybrid root-finders, which improve both of these iteration processes in various ways. We also improve initialization of subdivision iterations specialized for real root-finding.

Key Words: Polynomial root-finding; Polynomial factorization; Functional iterations; Subdivision; Root-counting; Real roots; Root radii

2000 Math. Subject Classification: 65H05, 26C10, 30C15

1 Introducton

1. The problem and three known efficient algorithms. Univariate polynomial root-finding has been the central problem of mathematics since Sumerian

*Some results of this paper are scheduled to be presented at CASC 2019.

times (see [1], [2], [43], [44]) and still remains the subject of intensive research due to applications to signal processing, control, financial mathematics, geometric modeling, and computer algebra (see the books [36], [38], a survey [21], the recent papers [54], [11], [57], [51], [12], [30], and the bibliography therein).

Hundreds if not thousands of efficient polynomial root-finders have been proposed. The algorithm of [42] and [47], extending the previous progress in [17], [59], [39], first computes numerical factorization of a polynomial into the product of its linear factors and then approximates the roots; it solves both tasks in nearly optimal Boolean time – almost as fast as one can access the input coefficients with the precision required for these tasks.[1]

This algorithm is quite involved and has never been implemented. Since 2000 the root-finder of the user's choice has been the package MPSolve,[2] implementing Ehrlich's iterations of [20], also known from their rediscovery by Aberth in 1973. In 2016 a distinct nearly optimal polynomial root-finder appeared in [11] and [12], based on subdivision iterations. It performs slightly faster than MPSolve for root-finding in a disc on the complex plain [30] but is still inferior for the approximation of all roots of a polynomial.

2. New hybrid algorithms. Our *synergistic combination* of the techniques of these three root-finders enhances their power. We incorporate two distinct efficient deflation techniques into subdivision and Ehrlich's iterations, which one can similarly incorporate into other functional root-finding iterations such as Weierstrass's (aka Durand–Kerner's) and Newton's.

Our even more significant advance is a faster and more robust algorithm for counting roots in a disc on the complex plain, which is a key stage of subdivision algorithms. Its previous improvement versus the immediate predecessors [58] and [45] was claimed to be a major algorithmic novelty of the papers [11] and [12]. Unlike [11] and [12] our root-counting algorithm does not involve polynomial coefficients and a can be applied to polynomial given in any basis, e.g., Bernstein's or Chebyshev's, or more generally to a polynomial represented by a subroutine for its evaluation,[3] which is an important benefit for dealing with sparse polynomials. Namely we reduce root counting for a polynomial of degree d just to polynomial evaluation at $\log_2(2d+1)$ points, that is, at 11 points in the case of degree 1,000 and at 21 points in the case of degree 1,000,000.

Our third substantial improvement is of subdivision iterations is simplification of their initialization in the case of real root-finding.

Our hybrid root-finders are nearly optimal and should become the user's choice. Their implementation, testing and refinement are major challenges.

[1]Required precision and Boolean time are smaller by a factor of d, the degree of an input polynomial, for the problem of numerical polynomial factorization, which has various important applications to modern computations, besides root-finding, e.g., to time series analysis, Wiener filtering, noise variance estimation, co-variance matrix computation, and the study of multi-channel systems (see Wilson [69], Box and Jenkins [7], Barnett [3], Demeure and Mullis [18] and [19], Van Dooren [66]).

[2]Some competition came in 2001 from the package EigenSolve of [22], but the latest version of MPSolve of [10] has combined the benefits of both packages.

[3]Hereafter we refer to them as *black box polynomials*, assuming that such a black box subroutine also covers evaluation of the derivative $p'(x)$.

3. Our technical background and a research challenge. To a large extent our progress relies on some advanced techniques hidden in Scönhage's paper [59]. That long paper contains a realm of intricate and powerful techniques for theoretical estimation of asymptotic Boolean complexity where extremely accurate polynomial factorization is required. These advanced techniques have been too little (if at all) used by researchers since [47].[4] In this paper we recalled an efficient deflation algorithm developed by Schönhage in [59], traced back to Delves and Lyness [17], and hereafter referred to as *DLS algorithm* or *DLS deflation.* Subdivision iterations turned out to be quite friendly to this algorithm and particularly to the specialization of its simple sub-algorithm to root-counting. Furthermore we advanced subdivision iterations specialized to real root-finding by combining DLS deflation with new application of Zhukovsky's function and by applying an efficient root radii algorithm from [59]. The DLS deflation can be incorporated into Ehrlich's iterations as well, but for that task we devised more efficient deflation techniques and compared them with alternative ones of [62]. In our analysis of proposed algorithms we applied some results from [32], [54], and [55].

We hope that our work will motivate further efforts towards synergistic combination of some efficient techniques well- and less-known for polynomial root-finding (see, e.g., the little explored methods of [51] and [57]).

Devising practical and nearly optimal factorization algorithms is still an importnat research challenge because for that task both Ehrlich's and subdivision iterations are slower by at least a factor of d than nearly optimal solution in [42] and [47], which, however, is too involved in order to be practically efficient.

5. Organization of the paper. We state variations of the polynomial root-finding and factorization problems in Section 2 and deduce lower bounds on their Boolean complexity in Section 3. We cover subdivision iterations with our three improvements in Section 4. We describe DLS deflation in Sections 5 and 6 and a more straightforward but less accurate deflation in Section 7. We devote Section 9 to incorporation of deflation into Ehrlich's iterations. In Section 10 we cover the complexity of the presented algorithms. We conclude with Section 11. In Parts I and II of the Appendix we briefly recall some other algorithms of [59] and [32] directed towards polynomial factorization and its extension to root-finding. In Part III of the Appendix we cover some auxiliary and complementary algorithms and techniques for polynomial root-finding.

2 Four fundamental computational problems

Problem 1. *Univariate Polynomial Root-finding.* Given a real b' and the

[4] To some extent such comments can be also applied to the paper [32] by Kirrinnis, which elaborates upon the techniques of [59] towards simultaneous splitting of polynomial into the product of a number of factors and is revisited in [55].

coefficients $p_0, p_1, \ldots, p_d$ of a univariate polynomial $p(x)$,

$$p(x) = \sum_{i=0}^{d} p_i x^i = p_d \prod_{j=1}^{d} (x - x_j),\ \ p_d \neq 0. \tag{1}$$

approximate all d roots[5] $x_1, \ldots, x_d$ within the error bound $1/2^{b'}$ provided that $\max_{j=0}^{d} |x_j| \leq 1$. We can ensure the latter customary assumption at a dominated computational cost by first approximating the root radius

$$r_1 = \max_{j=1}^{d} |x_j|, \tag{2}$$

and then scaling the variable: $x \to x/r_1$ (this implies increasing the computational (*working*) precision by $\log_2(\max\{r_1, 1/r_1\})$). We can readily approximate r_1 (cf., e.g., [43]) by

$$\tilde{r} = \max_{j \geq 1} (p_{d-j}/p_d)^{1/j},\ \ \tilde{r}/d \leq r_1 < 2\tilde{r}, \tag{3}$$

$$\tilde{r}\sqrt{2/d} \leq r_1 \leq (1 + \sqrt{5})\tilde{r}/2 < 1.62\tilde{r} \text{ if } p_{d-1} = 0. \tag{4}$$

With a modest amount of computations we can approximate the root radius within, say, a 1% error by applying the algorithm that supports the following result.

Theorem 1. *See [59, Corollary 14.3]. Given a polynomial $p = p(x)$ of (1), one can approximate its* root radii *$r_j := |x_j|$ for $j = 1, \ldots, d$ within a relative error bound $1 + 1/d^k$ for a real k at a Boolean cost in $O(d^2 \log^2(d))$,*

All these estimates for r_1 and other root radii involve coefficients of an input polynomial and can be applied neither to polynomials $p(x)$ represented in Bernstein, Chebyshev and other non-monomial bases of (1) nor to sparse polynomials defined by a subroutine for their evaluation, e.g., $p = p_h(x)$ for a fixed positive integer h such that

$$p_0(x) = 0,\ \ p_{i+1}(x) = p_i(x)^2 + x \text{ for } i = 0, 1, \ldots, h. \tag{5}$$

The roots of this polynomial are known as the centers of hyperbolic components of Mandelbrot's set of period h. For another example polynomials can be defined by the following similar sequence of equations:

$$p_0(x) = x,\ \ p_{i+1}(x) = p_i(x)^2 + 2 \ \text{ for } i = 0, 1, \ldots, h. \tag{6}$$

For both classes the degree of $p_i(x)$ is squared in the transition to $p_{i+1}(x)$. One can estimate the root radius r_1 of such polynomials by monitoring the behavior of Newton's iterations initialized sufficiently far from the origin and applying the estimates of [58] or [65].

Before proceeding any further we recall some **Basic Definitions.**

[5]We count m times a root of multiplicity m.

- Hereafter we freely denote polynomials $p(x)$, $t(x) = \sum_i t_i x^i$, $u(x) = \sum_i u_i x^i$ etc. by p, t, u etc. unless this can cause confusion.
- We use the norms $|u| = \sum_i |u_i|$ for $u = \sum_i u_i x^i$ and $|u|_\infty = \max_i |u_i|$.
- $d_u := \deg(u)$ denotes the degree of a polynomial u; in particular $d_p = d$.
- ϵ-*cluster* of roots of p is a root set lying in a disc of radius ϵ; in particular a 0-cluster of m roots of p is its root of multiplicity m.

Problem 2. *Approximate Factorization of a Polynomial.* Given a real b and the coefficients $p_0, p_1, \dots, p_d$ of a polynomial $p = p(x)$ of (1), compute $2d$ complex numbers u_j, v_j for $j = 1, \dots, d$ such that

$$|p - \prod_{j=1}^{d} (u_j x - v_j)| \leq 2^{-b} |p|. \tag{7}$$

Problem 3. *Polynomial root-finding in a disc.* This is Problem 1 restricted to root-finding in a disc on the complex plain for a polynomial p that has no roots lying outside the disc but close to it.

Problem 4. *Polynomial root-finding in a line segment.* This is Problem 1 restricted to root-finding in a line segment for a polynomial p that has no roots lying outside the segment but close to it.

The above concept "close" is quantified in Definition 9 for Problem 3 and is extended to Problem 4 via its reduction to Problem 3 in Section 9.

Remark 2. It is not easy to optimize working precision for the solution of Problems 1 – 4 a priori, but we can nearly optimize it *by action* – by applying the solution algorithms with recursively doubled or halved precision and monitoring the results (see Section 9.2 and recall similar policies in [5], [10], [54], [12]).

Remark 3. It is customary to reduce Problems 3 and 4 to root-finding in the unit disc

$$D(0,1) := \{x : \ |x| < 1\}$$

and unit segment

$$S[-1,1] := \{x : \ -1 \leq x \leq 1\}$$

by means of shifting and scaling the variable (cf. (3) and (4)). Then working precision and Boolean cost grow but within the nearly optimal bounds.

3 Boolean complexity: lower estimates

Proposition 4. The solution of Problem 2 involves at least db bits of memory and at least as many Boolean (bit-wise) operations.

Proof. The solution of Problem 2 is given by the $2d$ coefficients u_j and v_j of the d linear factors $u_jx - v_j$ of p for $j = 1, \dots, d$. Let $u_j = 1$ and $1/2 < |v_j| < 1$ for all j. Then each v_j must be represented with b bits and hence all v_j must be represented with db bits in order to satisfy (7). A Boolean operation outputs a single bit, and so we need at least db operations in order to output db bits. $\square$

Next we bound from below the Boolean complexity of Problems 1, 3 and 4.

Definition 5. $\omega = \omega_K := \exp(\frac{2\pi\mathbf{i}}{K})$ denotes a primitive Kth root of unity, such that $\omega_K^K = 1$, $\omega_K^i \neq 1$ for $0 < i < K$.

Lemma 6. *Let $p(x) = (x - x_1)^m f(x)$ for a polynomial $f(x)$ and a positive integer m. Fix a real b. Then the polynomial $p_j(x) = p(x) + 2^{(j-m)b}(x - x_1)^j f(x)$ has $m - j$ roots $x_1 + \omega_{m-j}^i 2^{-b}$ for $i = 0, \dots, m - j - 1$.*

Proof. Observe that $p_j(x) = ((x - x_1)^{m-j} + 2^{(j-m)b})(x - x_1)^j f(x)$ and consider the roots of the factor $(x - x_1)^{m-j} + 2^{(j-m)b}$. $\square$

Corollary 3.1. Under the assumption of Lemma 6 write $f := \lceil \log_2 |f| \rceil$, $g_j(x) = (x - x_1)^j f(x)$ and $j = 1, \dots, m - 1$, and $g := \sum_{j=1}^{m-1} \lceil \log_2 |g_j| \rceil$. Then one must process at least

$$\mathcal{B}_p = \Big(d - m + 1 + \frac{m-1}{2}\Big)mb - f - g \tag{8}$$

bits of the coefficients of p and must perform at least $\mathcal{B}_p/2$ Boolean operations in order to approximate the m-multiple root x_1 of p within $1/2^b$.

Proof. By virtue of Lemma 6 the perturbation of the coefficients $p_0,\dots,p_{d-m}$ of $p(x)$ by $|f|/2^{mb}$ turns the $(m - j)$-multiple root x_1 of the factor $(x - x_1)^{m-j}$ of $p(x)$ into $m - j$ simple roots $p_j(x)$, all lying at the distance $1/2^b$ from x_1. Therefore one must access at least $(d - m + 1)mb - f$ bits of the coefficients $p_0, \dots, p_{d-m}$ of p in order to approximate the root x_1 within $1/2^b$.

Now represent the same polynomial $p(x)$ as $(x - x_1)^{m-j} g_j(x)$. Apply Lemma 6 for m replaced by $m - j$ and for $f(x)$ replaced by $g_j(x)$ and deduce that a perturbation of the coefficient p_{d-m+j} of p by $|g_j|/2^{(m-j)b}$ turns the j-multiple root x_1 of $g_j(x) = (x - x_1)^j f(x)$ into j simple roots, all lying at the distance $1/2^b$ from x_1. Therefore one must access at least $\sum_{j=1}^{m-1}((m - j)b - g = \frac{m-1}{2}mb - g$ bits of the coefficients $p_{d-m+1}, \dots, p_{d-1}$ in order to approximate the root x_1 within $1/2^b$. Sum the bounds $(d - m + 1)mb - f$ and $\frac{m-1}{2}mb - g$ and arrive at the bound (8) on the overall number $\mathcal{B}_p$ of bits to be accessed; this requires at least $\mathcal{B}_p/2$ Boolean operations – at least one operation per each pair of bits. $\square$

Let us specify bound (8) in two cases.

(i) If $m = d$, $f(x) = 1$, and $|x_1| \leq \frac{0.5}{d}$, then $f = 0$, $|g_j| \leq 2$ for all j, $g \leq d - 1$, and

$$\mathcal{B}_p \geq (d + 1)db/2 - d + 1. \tag{9}$$

(ii) If x_1 is a simple root, well-isolated from the other roots of p, then substitute $m = 1$ and $g = 0$ into equation (8), thus turning it into $\mathcal{B}_p = db - f$ for $f(x) = \frac{p(x)}{x-x_1}$ such that $|f| \le d|p|$. Consequently

$$\mathcal{B}_p \ge (b - |p|)d.$$

Remark 7. Corollary 3.1 defines lower bounds on the Boolean complexity of Problems 1, 3, and 4 as long as an input polynomial p has an m-multiple root in the complex plain, a disc, and a segment, respectively. One can extend all these bounds to the case where a polynomial has an ϵ-cluster of m roots for a sufficiently small positive ϵ rather than an m-multiple root.

The algorithm of [42] and [47] solves Problem 2 by using $\tilde{O}(db)$ bits of memory and $\tilde{O}(db)$ Boolean operations.[6] This Boolean cost bound is within a poly-logarithmic factor from the information-theoretic lower bound db of Proposition 4. Based on [61, Theorem 2.7] one can extend that estimate to the solution of Problems 1, 3 and 4 at a Boolean cost in $\tilde{O}(d^2b)$, which is also nearly optimal by virtue of (9), and to nearly optimal solution of the problem of polynomial root isolation (see Corollaries B.1 and B.2).

Finally we state the following information lower bound in the case of a black box polynomial.

Theorem 8. *Approximation of k roots of a black box polynomial p of a degree $d \ge k$ with an error bound $2^{-b'}$ requires at least $2^{-b'}k$ Boolean operations.*

Proof. Notice that a Boolean operation outputs just a single bit. ☐

4 Subdivision iterations with three improvements

4.1 Subdivision iterations and a Quadtree

Subdivision iterations extend the classical bisection iterations from root-finding on a line to polynomial root-finding in the complex plain. Under the name of *Quad-tree Construction* these iterative algorithms have been studied in [27], [25], [58], and [45] and extensively used in Computational Geometry. The algorithm has been introduced by Herman Weyl in [68] and advanced in [27], [25], [58], and [45]; under the name of subdivision Becker et al. modified it in [11] and [12].[7] Let us briefly recall subdivision iterations for Problem 1; they are similar for Problem 3.

At the beginning of a subdivision (quad-tree) iteration all the d roots of p are covered by at most cd congruent *suspect squares* on the complex plain that have horizontal and vertical edges, all of the same length where c is a fixed constant. The iteration outputs a similar cover of all d roots of p with a new

[6]Here and hereafter we write $\tilde{O}(s)$ for $O(s)$ defined up to a poly-logarithmic factor in s.
[7]The algorithms of [11] and [12] are quite similar to one another.

set of at most cd suspect squares whose edge length is halved. One needs a *nonexistence test*, which decides whether a square contains no roots of p, in which case it is discarded. Otherwise it is a suspect square to be processed in the next subdivision iteration. Subdivision process can be described by a Quadtree, with vertices in suspect squares and edges connecting them to their suspect subsquares of the next subdivision.

4.2 Nonexistence, proximity and counting tests

Nonexistence test for the roots in a disc can be performed by means of a *proximity test*, applied in the center of the disc and outputting the distance from the center to the nearest roots of p.

Conversely, by recursively applying a nonexistence test we can approximate the distance to a root from a complex point c. Namely apply a nonexistence test to a sequence of concentric discs $D(c, \rho_i)$ with $\rho_{i+1} = 2\rho_i$, for $i = 1, \ldots, k$, until a disc $D(c, \rho_k)$ contains a root. At that point the distance from c to a nearest root lies in the semi-open segment $(\rho_{k-1}, \rho_k]$. We can bisect it by applying the test to disc $D(c, (\rho_{k-1} + \rho_k)/2)$ and continuing this process recursively, but instead of doubling the disc radii and halving the error bound, we can apply the optimal policy of the double exponential sieve of [58].

Nonexistence test in [45] relies on fast estimation (within errors of, say, at most 1%) of the distances from the origin or any selected point on the complex plain to all roots based on Theorem 1. This is is actually a *counting test*, which approximates the number of the roots of p in every disc centered at the origin; it outputs their exact number if the disc is 1.01-isolated. The test is restricted to polynomials given with their coefficients.

At any subdivision iteration that uses this nonexistence test, every root can make at most four squares suspect, and thus all d roots can make at most $4d$ squares suspect. Moreover this single test replaces nonexistence tests for all suspect squares that have no overlap with the d narrow annuli defined by the fast algorithm that supports Theorem 1. This is a significant simplification of subdivision iterations, which is particularly dramatic where subdivision is applied to real root-finding, but it seems that this resource has not been used so far.

4.3 Isolation of a complex domain from external roots

For our study we must quantify isolation of a disc on the complex plain from the external roots of p. Next we define such isolation for more general domains.

Definition 9. *Isolation of a domain and its boundary.* Let a domain D on the complex plain allow its dilation from a fixed center. Then this domain has an *isolation ratio* of at least θ and is θ*-isolated* for a polynomial p and real $\theta > 1$ if the root set of p in the domain D is invariant in the θ-dilation of D. The boundary of such a domain D has an *isolation ratio* at least θ and is θ-isolated if the root set of p in D stays invariant in both θ- and $1/\theta$-dilation of the domain.

4.4 A root counting algorithm

First we count roots in the unit disc $D(0,1) = \{x : \ |x| = 1\}$. Fix a sufficiently large integer q, let ω denote a primitive qth root of unity of Definition 5, and approximate the number s of the roots of p in the θ-isolated unit disc as follows (compare equation (16) for $h = 0$):

$$s \approx s^* = \frac{1}{q} \sum_{g=0}^{q-1} \omega^g \ \frac{p'(\omega^g)}{p(\omega^g)}, \ \omega = \exp\Big(\frac{2\pi}{q}\sqrt{-1}\Big). \tag{10}$$

We obtain s by rounding s^* if $|s - s^*| < 1/2$, which holds if $2d + 1 < \theta^q$ or equivalently if $q > \log_\theta(2d + 1)$ by virtue of Theorem 14 (cf. Remark 15). For example, if $\theta = 2$, then choosing any $q \geq 11$ is sufficient where $d = 1,000$ and choosing any $q \geq 21$ is sufficient where $d = 1,000,000$.

We compute s^* by means of the evaluation of both $p(x)$ and $p'(x)$ at the qth roots of unity, which we reduce to performing discrete Fourier transform at q points twice, and in addition performing discrete Fourier transform at q points once again. We can perform transforms faster by applying FFT if we choose q being the power of 2, although the gain decreases as the integer q decreases.

We can extend this algorithm to counting roots of p in any θ-isolated disc $D(c, \rho)$ on the complex plain. Write $t(x) := p(\frac{x-c}{\rho})$ and compute s^* for the polynomial $t(x)$ replacing $p(x)$. We do not need to compute the coefficients of the polynomial $t(x)$ because $t'(x) = p'(x)/\rho$, and so we can just evaluate $p(x)$ and $p'(x)$ at the q points $c + \rho\omega^g$, $g = 0, \ldots, q - 1$, equally spaced on the boundary circle of the disc $D(c, \rho)$. For this evaluation we can apply "Horner's" algorithm $2q$ times or the algorithms of [37], [48], or [50] for fast multipoint polynomial evaluation.

4.5 Convergence to the roots and its acceleration

Let us come back to subdivision iterations with a nonexistence test. At every iteration the centers of the suspect squares approximate the root set of p with an error of at most one half of their diameter. Every subdivision decreases this bound by twice, but at some point we can accelerate this linear convergence.

Suppose that at a subdivision step suspect squares form s connected components. We can readily compute their minimal covering discs D_i, $i = 1, \ldots, u$. Let $D'_i(c'_i, R'_i)$ denote the minimal sub-disc that covers the root sets of p in the disc and let θ_i^2 denote its isolation ratio. Then $\theta = \max_{i=1}^u \theta_i \geq 1$ is noticeably separated from 1 in $O(\log(d))$ subdivision iterations [58], [45]. The algorithms of [58], [45], [11] and [12] yield super-linear convergence at this point.

[58] and [45] apply Newton's iterations, whose convergence to a disc D'_i is quadratic right from the start if they begin in its θ_i-dilation $D(c'_i, \theta R'_i)$ and if θ_i is sufficiently large. Tilli in [65] proves that it is sufficient if $\theta \geq 3d - 3$, which improves the earlier estimate $\theta \geq 5d^2$ of [58].

[11] and [12] achieve super-linear convergence to the roots by extending to the complex plain the QIR iterations, proposed by Abbott for a line segment.

We accelerate subdivision iterations by means of DLS deflation under a mild assumption that $\theta - 1$ exceeds a positive constant, e.g., that $\theta = 2$.

4.6 Root-finding in a disc via DLS deflation: an outline

Let a subdivision iteration define a well-isolated disc $D = D(c, \rho)$ containing $w > 0$ roots of a polynomial p, say, $x_1, \dots, x_w$. In this case the polynomial $t(x) = p(\frac{x-c}{\rho})$ contains w roots $y_j = \frac{x_j - c}{\rho}$, $j = 1, \dots, w$, in the unit disc $D(0, 1)$. By applying DLS deflation we closely approximate the polynomial $f = f(x) = \prod_{j=1}^{w}(x - y_j)$, then approximate its roots $y_1, \dots, y_w$ by solving Problem 1 for that polynomial rather than for p, and finally recover the roots $x_j = c + ry_j$ of p. Appendices A.2 and B bound the errors of the approximation of the roots y_j in terms of the error of the approximation of the polynomial f.

Remark 10. Transition from p to f simplifies root-finding because $w = \deg(f) < d = \deg(p)$ and potentially because the maximal distance between a pair of roots and the number and the sizes of root clusters f may decrease, but the impact of these favorable factors can be weakened or lost due to the coefficient growth and loss of sparseness. Even for the worst case inputs, however, these deficiencies become less serious as w decreases: the norm $|f|$ reaches its maximum 2^w for $f = (x-1)^w$, and f may have $w+1$ nonzero coefficients.

Remark 11. We can extend the counting algorithm of Section 4.4 and DLS deflation to any well-isolated domain on the complex plain rather than a disc (cf. Section 5.3) but cannot bound the computational cost and output accuracy as strong as in the case of a disc.

4.7 Concurrent and recursive deflation

We can deflate a polynomial p simultaneously for a number of discs D'_i (cf. Part II of the Appendix) and can recursively deflate the computed lower degree polynomials. One should guard against coefficient growth and loss of sparseness in deflation. As a remedy one can delay deflation until the degree of a factor decreases to a safe level.[8] Another motivation for such a delay is potential dramatic increase of the overall computational complexity of root-finding.

In the following example computational cost becomes too high if we perform deflation $d - 1$ times but stays nearly optimal if we properly delay deflation.

Example 12. Let $p = \prod_{j=1}^{d}(x - 1 + 1/2^j)$ for a large integer d. In this case the roots $1 - 1/2^j$ are stronger isolated and better conditioned for smaller j, and so Ehrlich's iterations may peel out one such a root of p at a time. Then we would $d-1$ times approximate polynomials of the form $p_i := g_i(x)\prod_{j=1}^{i}(x - 1 + 1/2^j)$ where $g_i = g_i(x)$ is a polynomial of degree $d-i$ that shares all its $d-i$ roots with p. Approximation of such a factor of p for $i \geq d/2$ involves at least $bd^2/4$ bits and at least $bd^2/8$ Boolean operations. Indeed such a cluster consists of at least $d/2$

[8] An alternative possible remedy is implicit deflation (cf. [31]).

roots of p, and so we must operate with them with a precision of at least $bd/4$ bits (see Corollary 3.1). Hence approximation of these factors for $i = 1, \dots, d-1$ involves at least $bd^3/8$ bits and at least $bd^3/16$ Boolean operations. We can decrease this cubic lower bound to the nearly optimal quadratic level if we skip deflation at the ith step unless $\frac{\deg(g_i)}{\deg(g_{i+1})} \geq \gamma$ for a fixed $\gamma > 1$, e.g., $\gamma = 2$. Such a degree bound ensures that we apply deflation not more than $\lceil \log_\gamma(d) \rceil$ times.

4.8 Correctness verification for counting and deflation

Recipe 1. Count the roots of p in all suspect squares at the current subdivision iteration and verify that the sum over all squares is d.

Recipe 2. Verify correctness of both counting and deflation by means of extending them to the approximation of the respective roots of p and verify correctness of the computed approximations either based on the known criteria (cf. [5], [10]) or heuristically by action: apply p Newton's or other root-finding iterations initialized at these approximations and monitor convergence.

Recipe 3. For another heuristic verification of correctness recall that both counting and deflation begin with approximation of the first power sums of the roots of the auxiliary polynomial $t(x) = p(\frac{x-c}{\rho})$ in an input disc $D(c, \rho)$ and that the errors of these approximations are bounded in terms of the isolation ratio of the disc and the number of points used for the auxiliary evaluation of p and its derivative. By recursively adding new evaluation points and monitoring the improvement behavior of the approximations, in particular by verifying convergence to an integer of the computed 0-th power sum, we can empirically verify correctness of our auxiliary computation of the power sums and estimate the level of isolation (cf. Theorem 14).

5 DLS deflation: approximation of the power sums of the roots

5.1 The power sums of the roots in the unit disc $D(0, 1)$

Next we define the power sums s_h of the $w = d_f$ roots of the factor f of p:

$$s_h := \sum_{j=1}^{d_f} x_j^h, \; h = 0, \pm 1, \pm 2, \dots, \tag{11}$$

and their approximations $s_0^*, s_1^*, \dots, s_{q-1}^*$ for a positive q such that

$$s_h^* := \frac{1}{q} \sum_{g=0}^{q-1} \omega^{(h+1)g} \, \frac{p'(\omega^g)}{p(\omega^g)}, \quad h = 0, 1, \dots, q-1, \tag{12}$$

where $\omega = \exp(2\pi\sqrt{-1}/q)$ denotes a primitive qth root of unity for an integer $q > 1$ (cf. Definition 5). Compute these approximations by applying the following algorithm.

Algorithm 13. *Approximation of the power sums of the roots in the unit disc.*

(i) Given the coefficients of the polynomial $p(x)$ of (1), compute the coefficients $p'_i = (i+1)p_{i+1}$, $i = 1, \dots, d-1$, of the polynomial $p'(x)$ (by using $d-2$ multiplications).

(ii) For a fixed integer q, $1 < q < d$, compute the values

$$\bar{p}_g = \sum_{u=0}^{\lfloor (d-g)/q \rfloor} p_{g+uq}, \quad \bar{p}'_g = \sum_{v=0}^{\lfloor (d-1-g)/q \rfloor} \bar{p}'_{g+vq}, \text{ for } g = 0, 1, \dots, q-1, \tag{13}$$

by performing less than $2d$ additions.

(iii) Compute the values

$$p(\omega^g) = \sum_{h=0}^{q-1} \bar{p}_h \omega^{gh}, \quad p'(\omega^g) = \sum_{h=0}^{q-1} \bar{p}'_h \omega^{gh}, \text{ for } g = 0, 1, \dots, q-1, \tag{14}$$

by applying two DFTs,[9] each on q points, to the coefficient vectors of the polynomials $\bar{p}(x) = \sum_{i=0}^{q-1} \bar{p}_i x^i$ and $\hat{p} = \sum_{i=0}^{q-1} \bar{p}'_i x^i$ (cf. (13)).

(iv) Compute the vector $\left(\frac{p(\omega^g)}{p'(\omega^g)}\right)_{g=0}^{q-1}$ by performing q divisions.

(v) By applying a single DFT on q points multiply the DFT matrix $(\omega^{(h+1)g})_{g,h=0}^{q-1}$ by this vector.

5.2 Error estimates for the power sums of the roots in the unit disc $D(0,1)$

The following upper estimates from [59] show that the errors of the approximation of the power sums s_h decrease exponentially in the number q of the knots of evaluation in (12).

Theorem 14. *Let the annulus*

$$A(0, 1/z, z) = \{x : \ 1/z \le |x| \le z\} \tag{15}$$

contain no roots of p. Then

$$|s_h^* - s_h| \le \frac{d_f z^{q+h} + (d-d_f) z^{q-h}}{1 - z^q} \text{ for } h = 0, 1, \dots, q-1 \tag{16}$$

(compare this bound for $h = 0$ with (10)).

Proof. Deduce the following expressions from Laurent's expansion:

$$\frac{p'(x)}{p(x)} = \sum_{j=1}^{d} \frac{1}{x - x_j} = -\sum_{h=1}^{\infty} S_h x^{h-1} + \sum_{h=0}^{\infty} s_h x^{-h-1} := \sum_{i=-\infty}^{\infty} c_i x^i \tag{17}$$

[9] Here and hereafter "DFT" is the acronym for "discrete Fourier transform."

for $|x|=1$, s_h of (11), and

$$S_h = \sum_{j=d_f+1}^{d} \frac{1}{x_j^h},\ h=1,2,\dots.$$

The leftmost equation of (17) is verified by the differentiation of the polynomial $p(x)=\prod_{j=1}^n (x-x_j)$. gb The middle equation of (17) is implied by the following equations for $|x|=1$:

$$\begin{aligned} \frac{1}{x-x_j} &= \frac{1}{x}\sum_{h=0}^{\infty}\left(\frac{x_j}{x}\right)^h && \text{for } j\le d_f,\\ \frac{1}{x-x_j} &= -\frac{1}{x_j}\sum_{h=0}^{\infty}\left(\frac{x}{x_j}\right)^h && \text{for } j> d_f. \end{aligned}$$

Equations (12) and (17) combined imply that

$$s_h^* = \sum_{l=-\infty}^{+\infty} c_{-h-1+lq} \text{ for } h\ge 0.$$

Moreover, (17) for $i=-h-1,\ h\ge 0$ implies that $s_h=c_{-h-1}$, while (17) for $i=h-1,\ h\ge 1$ implies that $S_h=-c_{h-1}$. Consequently

$$s_h^* - s_h = \sum_{l=1}^{\infty}(c_{lq-h-1}+c_{-lq-h-1}) \text{ for } h\ge 0.$$

We assumed in (12) that $0\le h\le q-1$. It follows that $c_{-lq-h-1}=s_{lq+h}$ and $c_{lq-h-1}=-S_{lq-h}$ for $l=1,2,\dots$, and we obtain

$$s_h^* - s_h = \sum_{l=1}^{\infty}(s_{lq+h}-S_{lq-h}) \text{ for } h\ge 0. \tag{18}$$

Furthermore $s_0=d_f$, $|s_h|\le d_f z^h$, $|S_h|\le (d-d_f)z^h$, $h=1,2,\dots$, for z of (15). Substitute these bounds into (18) and obtain (16). □

Remark 15. $|s^*-s|\le \frac{dz^q}{1-z^q}$ for $s=s_0$ and $s^*=s_0^*$ by virtue of Theorem 14.

5.3 The power sums of the roots in any region

Expression (12) for the power sums of the roots in the disc $D(0,1)$ is a special case of more general expression for the power sums s_h of the roots in a complex domain $\mathcal{D}$ via discretization of the contour integrals

$$\mathcal{I}_h := \int_\Gamma \frac{p'(x)}{p(x)}\ x^h\ dx$$

where Γ denotes the boundary of the domain $\mathcal{D}$ and where $s_h=\mathcal{I}_h$ for all h by virtue of Cauchy's theorem. Kirrinnis in [33] presents a fast, although quite involved algorithm for the approximation of such integrals at a nearly optimal Boolean cost provided that the contour Γ is θ-isolated for a constant $\theta>1$, but the bounds of Theorem 14 are superior in the case where $\mathcal{D}$ is the unit disc.

Next we cover two other quite favorable cases, where the region $\mathcal{D}$ is the unit circle and an arbitrary disc.

5.4 The power sums of the roots in any disc

Let us be given a θ-isolated disc $D(c, \rho) = \{x : \ |x - c| \leq \rho\}$ for a complex $c \neq 0$ and a positive ρ. Then the disc $D(0, 1)$ is θ-isolated for the polynomial $t(x) = p(\frac{x-c}{\rho})$, and roots x_j of $p(x)$ can be readily recovered from the roots $y_j = \frac{x_j - c}{\rho}$ of $t(x)$ (cf. the end of Section 4.4). We can approximate the power sums of the roots of the polynomial $t(x)$ by applying Algorithm 13 to that polynomial rather than to $p(x)$. The computation by Algorithm 13 is reduced essentially to the evaluation of the polynomials $t(x)$ and $t'(x) = \frac{1}{\rho}p'(x)$ at the roots of unity $x_g = \omega^g$ for $\omega = \omega_K$ of Definition 5, $g = 0, 1, \ldots, K-1$, $K = 2^k$, and $k = \lceil \log_2(q) \rceil$, followed by the application of inverse FFT at K points.

We can perform the evaluation of these two polynomials by applying any of the two following algorithms.

Algorithm 16. Evaluate the polynomials $p(x)$ and $\frac{1}{\rho}p'(x)$ at the points $v_g = c + \rho\omega^g$ for $g = 0, 1, \ldots, K-1$.

Algorithm 17. (i) Compute the coefficients of $t(x)$ by applying shift and scaling of the variable x to $p(x)$. (ii) Then evaluate the polynomials $t(x)$ and $t'(x)$ at the K-th roots of unity ω^g for $g = 0, 1, \ldots, K-1$.

By applying Algorithm 16 we avoid potential coefficient growth and the loss of sparseness of the polynomial p in its transition to $t(x)$. The computation of the values $p(c + \rho\omega^g)$ and $p'(c + \rho\omega^g)$ is not reduced to performing three DFTs anymore, but we can perform it in $O(d \log^2(K))$ arithmetic operations by using the algorithm of [37] (see also Appendix F).

6 Transition from the power sums of the roots to the coefficients

Given a real c and the power sums $s_0, \ldots, s_{d_f}$ of the d_f roots of a monic polynomial $f(x) = x^{d_f} + \sum_{j=1}^{d_f - 1} f_j x^j \ f(x)$, the following algorithm approximates the coefficients $f_0, f_1, \ldots, f_{d_f - 1}$ within the norm bound $1/2^{cd}$.

Algorithm 18. *Recovery of the factor by means of applying Newton's iterations* (see [59, Section 13]).

Write $f_{\rm rev}(x) := 1 + g(x) = \prod_{j=1}^{d_f}(1 - x x_j)$ and deduce that

$$(\ln(1 + g(x)))' = \frac{g'(x)}{1 + g(x)} = -\sum_{j=1}^{d_f} \frac{x_j}{1 - x x_j} = -\sum_{j=1}^{h} s_j x^{j-1} \mod x^h \tag{19}$$

for $h = 1, 2, \ldots, d_f + 1$.

Write $g_r(x) := g(x) \mod x^{r+1}$, observe that $g_1(x) = -s_1 x$ and $g_2(x) = -s_1 x + (s_1^2 - s_2)x^2$, and express the polynomial $g_{2r}(x)$ as follows:

$$1 + g_{2r}(x) = (1 + g_r(x))(1 + h_r(x)) \mod x^{2r+1} \tag{20}$$

where

$$h_r(x) = h_{r+1}x^{r+1} + \cdots + h_{2r}x^{2r} \tag{21}$$

is an unknown polynomial. Equation (20) implies that

$$\frac{h_r'(x)}{1+h_r'(x)} = h_r'(x) \mod x^{2r+1}.$$

Equations (20) and (21) together imply that

$$\ln(1+g_{2r}(x))' = \frac{g_{2r}'(x)}{1+g_{2r}(x)} = \frac{g_r'(x)}{1+g_r(x)} + \frac{h_r'(x)}{1+h_r(x)} \mod x^{2r}.$$

Combine the latter identities with equation (19) for $d_f = 2r+1$ and obtain

$$\frac{g_r'(x)}{1+g_r(x)} + h_r'(x) = -\sum_{j=1}^{2r} s_j x^{j-1} \mod x^{2r}. \tag{22}$$

Having the power sums $s_1, \ldots, s_{2d_f}$ and the coefficients of the polynomials $g_1(x)$ and $g_2(x)$ available, recursively compute the coefficients of the polynomials $g_4(x), g_8(x), g_{16}(x), \ldots$ by using identities (20)–(22).

Namely, having the polynomial $g_r(x)$ available, compute the polynomial $\frac{1}{1+g_r(x)} \mod x^{2r}$ and its product with $g_r'(x)$. Then obtain the polynomials $h_r'(x)$ from (22), $h_r(x)$ from (21), and $g_{2r}(x)$ from (20).

Notice that $\frac{1}{1+g_r(x)} \mod x^r = \frac{1}{1+g(x)} \mod x^r$ for all r and reuse the above polynomials for computing the polynomial $\frac{1}{1+g_r(x)} \mod x^{2r}$. Its coefficients approximate the coefficients $f_1, \ldots, f_{d-1}$ of $f(x)$ and its reverse polynomial.

The algorithm performs $\lceil \log_2(d_f) \rceil$ iterations, at ith iteration amounts to performing single polynomial division modulo x^{2^i} and single polynomial multiplication modulo x^{2^i}, for $i = 2, \ldots, d_f$. We can perform them by using $O(d_f \log(d_f))$ arithmetic operations.

An alternative solution. We can also recover the factor f as follows. Define the reverse polynomial

$$f_{\text{rev}}(x) = x^{d_f} f(1/x) = 1 + \sum_{i=1}^{d_f} f_{d_f-i} x^i$$

and compute the coefficients of $f(x)$ from a linear system of d_f *Newton's identities* (cf. [46, equations (2.5.4) and (2.5.5)] for extension to a linear system of d Newton's identities and beyond – to a system of any number of Newton's identities):

$$-if_{d_f-i} = s_1 f_{d_f-i+1} + \cdots + s_i f_{d_f-1} + s_{i+1} \text{ for } i = 1, \ldots, d_f. \tag{23}$$

7 Lazy deflation in the case of an isolated cluster of a small number of roots

Suppose that w roots of p, say, $x_1, \dots, x_w$ form a cluster strongly isolated from the other roots of p, covered by a small disc $D(c, \rho)$. This is frequently observed as a by-product of subdivision iterations. In that special case one can perform DLS deflation in a lazy way, simply by computing the sum of $w + 1$ trailing terms of the polynomial $t(y) = p(\frac{x-c}{\rho})$ obtained from $p(x)$ by means of shifting and scaling the variable x (cf. the study in [34, Section 3.2] of real root-finding). This deflation amounts to invoking a readily available subroutine, but our next estimates for its output errors suggest that it tends to be inferior in accuracy to the DLS deflation.

Write $t(x) := p(x-c) = \sum_{j=0}^{d} t_j x^j$, suppose that $t_w \neq 0$, and approximate f by the following polynomial,

$$\tilde{f} := \frac{1}{t_w} \sum_{j=0}^{w} t_j x^j = \frac{1}{t_w} t(x) \mod x^{w+1}. \tag{24}$$

If $x_j = 0$ for $j = 1, \dots, w$ is a w-multiple root of p, then $f = x^w = \tilde{f}$ and $p = x^w g(x)$ for some polynomial $g = g(x)$ of degree $d - w$. If these roots are moved anywhere within the disc $D(0, \rho)$ for a positive ρ, then the factor g does not change but the factor changes into a polynomial $\bar{f}_\rho$ such that $|\bar{f}_\rho - f| \leq (1+\rho)^w - 1$. If $s = w\rho < 1$, then

$$|\bar{f}_\rho - f| \leq (1+\rho)^w - 1 \leq \frac{w\rho}{1 - w\rho},$$

$$|\bar{f}_\rho g \mod x^{w+1}| = |\bar{f}_\rho (g \mod x^{w+1}) \mod x^{w+1}| \leq |\bar{f}_\rho| \ |(g \mod x^{w+1}|,$$

and so

$$|\bar{f}_\rho g \mod x^{w+1}| \leq ((1+\rho)^w - 1) \ |g \mod x^{w+1}|.$$

Perform the routine scaling by $1/r_1$ of the variable x that moves all roots of p into the unit disc $D(0,1)$ and add $\log_2(r_1)$ bits to working precision. Then the norm $|g \mod x^{w+1}|$ is maximized for $g = (x+1)^{d-w}$, that is, $|g \mod x^{w+1}| \leq m_{d,w} = \sum_{i=1}^{w} \binom{d-w}{i}$. Assume that $2w\rho \leq 1$ and summarizing obtain

$$|\bar{f} - f| \leq ((1+\rho)^w - 1) m_{d,w} \leq \frac{m_{d,w} w\rho}{1 - w\rho} \leq 2 m_{d,w} w\rho,$$

and then we would need to have $\rho \leq 2^{-b}/(2m_{d,w}w)$ in order to ensure an upper bound $|\bar{f}_\rho g \mod x^{w+1}| \leq 2^{-b}$.

Output error norm of lazy deflation. The latter bound is a strong restriction on the size of a root cluster $\{x_1, \dots, x_w\}$, and so *formal support of lazy deflation is restricted to the clusters of very small size made up of a small number of roots. Moreover, as the above analysis shows, the error norm of the approximation by means of lazy deflation exceeds the half-diameter of the cluster.* According to [34], however, lazy deflation was useful for real root-finding, apparently in the cases where very crude root approximation was sufficient.

8 Improved root-finding on a line

The lower bounds of Section 3 equally apply to Problems 1, 3 and 4, and so the algorithms of [42], [47], [11] and [12] are nearly optimal for all three problems. For Problem 4 the algorithms of [54] and [34] also run in nearly optimal Boolean time. The algorithm of [34] specializes subdivision iterations to real root-finding; it is currently the user's choice algorithm, but next we substantially upgrade it by extending our improvements of subdivision to real root-finding.

In particular the improvement based on application of Theorem 1 is immediately extended with significantly enhanced power. Indeed the algorithm supporting Theorem 1 and running at nearly optimal Boolean cost, computes d narrow annuli covering all roots of p. Their intersection with a line defines at most $2d$ small segments that contain all roots of p lying on that line. By weeding out the extraneous empty segments containing no roots of p obtain close approximations to all roots on the line, including those lying in the segment $S[-1,1]$. See [56] for further details and extensive tests that showed particularly high efficiency of this algorithm for the approximation of the real roots of p that are sufficiently well-isolated from the other roots.

Remark 19. The paper [51] studies combination of Theorem 1 with random shift of the variable for root-finding on the complex plain.

Remark 11 motivates extension of the other improvements of subdivision iterations to Problem 4 by means of reducing it to root-finding in the unit disc. We first reduce the task to the case of the unit circle. Towards this goal we recall the two-to-one Zhukovsky's function $z = J(x)$, which maps the unit circle $C(0,1)$ onto the unit segment $S[-1,1]$, and its one-to-two inverse:

$$z = J(x) := \frac{1}{2}\Big(x + \frac{1}{x}\Big);\ x = J^{-1}(z) := z \pm \sqrt{z^2 - 1}. \tag{25}$$

Here x and z are complex variables. Now perform the following steps:

1. Compute the polynomial $s(z) := x^d p(x)p(1/x)$ of degree $2d$ by applying [9, Algorithm 2.1], based on the evaluation of the polynomials $p(x)$ and $x^d p(1/x)$ at the Chebyshev points and the interpolation to $s(z)$ at the roots of unity. Recall that the set of the roots of $p(x)$ lying in the segment $S[-1,1]$ is well-isolated and observe (see Remark 21) that it is mapped in one-to-two mapping (25) into a well-isolated set of the roots of $s(z)$ lying on the unit circle $C(0,1)$.

2. Let $g(z)$ denote the monic factor of the polynomial $s(z)$ with the root set made up of the roots of $s(z)$ lying on the unit circle $C(0,1)$ and such that $\deg(g(z)) = \deg(f)$. By applying the algorithm of Corollary 10.2 (cf. [59, Section 12]) approximate the power sums of the roots of the polynomial $g(z)$.

3. By applying Algorithm 18 approximate the coefficients of $g(z)$.

4. Compute the polynomial $h(x) := x^{2d}g(\frac{1}{2}(x+\frac{1}{x}))$ of degree $4\deg(f)$ in x by means of evaluation of the polynomial $h(x)$ at all Kth roots of unity ω_K for $K=2^k$, $k=\lceil\log_2(4d+1)\rceil$ (cf. Definition 5) via the evaluation of $g(y)$ at the Chebyshev's points $y_j=\frac{1}{2}(\omega_K^j+\omega_K^{-j})$, $j=0,1,\dots,K-1$ (cf. [9, Section 2]), followed by interpolation to $h(x)$ by means of FFT. The root set of the polynomial h is made up of the roots of the polynomial p lying in the segment $S[-1,1]$ and of their reciprocals; in the transition to $h(x)$ the multiplicity of the roots of p either grows 4-fold (for the roots 1 and -1 if they are the roots of p) or is doubled, for all other roots.

5. Approximate all roots of the polynomial $h(x)$ by applying a fixed root-finder, e.g., MPSolve or the real root-finder of [34].[10]

6. Among them identify and output $\deg(f)$ roots that lie in the segment $S[-1,1]$; they are precisely the roots of $p(x)$.

Remark 20. We can simplify stage 5 by replacing the polynomial $h(x)$ with its half-degree square root $j(x) := x^a f(x)f(1/x)$ at stage 5, but further study is needed to find out whether and how much this could decrease the overall computational cost.

Remark 21. Represent complex numbers as $z := u + \mathbf{i}v$. Then Zhukovsky's map transforms a circle $C(0,\rho)$ for $\rho \neq 1$ into the ellipse $E(0,\rho)$ whose points (u,v) satisfy the following equation,

$$\frac{u^2}{s^2}+\frac{v^2}{t^2}=1 \text{ for } s=\frac{1}{2}\Big(\rho+\frac{1}{\rho}\Big),\ \ t=\frac{1}{2}\Big(\rho-\frac{1}{\rho}\Big).$$

Consequently it transforms the annulus $A(0,1/\theta,\theta)$ into the domain bounded by the ellipses $E(0,1/\theta)$ and $E(0,\theta)$, so the circle $C(0,1)$ is θ-isolated if and only if no roots of p lie in the latter domain.

Remark 22. The paper [57] proposes a number of new advanced real root-finders based on iterating Jukovsky's function and its matrix analog, called *matrix sign iterations* and applied to the companion or generalized companion matrix of p. Such an application was against the customary advice for matrix sign iterations, but extensive tests in [57] showed its good practical promise.

9 Ehrlich's iterations and deflation

9.1 Ehrlich's iterations and their super-linear convergence

The papers [5] and [10] present two distinct versions of MPSolve based on two distinct implementations of Ehrlich's functional iterations.

[10]Since all roots of $h(x)$ are real, we can compute them also by applying the algorithms of [13], [8], or [16], which are nearly optimal real root-finders for a polynomial whose all roots are real.

[5] applies original Ehrlich's iterations by updating current approximations z_i to all or selected roots x_i as follows:

$$z_i \leftarrow z_i - E_{p,i}(z_i),\ i = 1, \dots, d, \tag{26}$$

$$E_{p,i}(x) = 0 \text{ if } p(x) = 0;\ \frac{1}{E_{p,i}(x)} = \frac{p'(x)}{p(x)} - \sum_{j=1, j\neq i}^{d} \frac{1}{x - z_j} \text{ otherwise.} \tag{27}$$

[10] modifies these iterations by replacing polynomial equation $p(x) = 0$ by an equivalent rational *secular equation*[11]

$$S(x) := \sum_{j=1}^{d} \frac{v_j}{x - z_j} - 1 = 0 \tag{28}$$

where $z_j \approx x_j$ and $v_j = \frac{p(z_j)l(x)}{x - z_j}$ for $l(x) = \prod_{j=1}^{d}(x - z_j)$ and $j = 1, \dots, d$.

Cubic convergence of these iterations simultaneously to all roots of a polynomial has been proved locally, near the roots, but under some standard choices of initial approximations very fast global convergence to all roots, right from the start, has been consistently observed in all decades-long applications of the iterations worldwide.

9.2 Tame and wild roots and precision management

The condition number of a root defines computational precision sufficient in order to ensure approximation within a fixed relative output error bound (see the relevant estimates in [5] and [10]). The value of the condition is not known a priori, however, and MPSolve adopts the following policy: at first apply Ehrlich's iterations with a fixed low precision (e.g., the IEEE double precision of 53 bits) and then recursively double it until all roots are approximated within a selected error tolerance.

More precisely MPSolve updates approximations only until they become close enough in order to satisfy a fixed stopping criterion, verified at a low computational cost. We call such roots *tame* and the remaining roots *wild*.

Remark 23. Similarly our study in this section can be applied where all roots of p are partitioned into tame and wild by another root-finding iteration, e.g., by a subdivision iteration (cf. [29]).

MPSolve stops applying Ehrlich's iterations to a root as soon as it is tamed but keeps applying them to the wild roots and then recursively double working precision unless approximations are improved. When a root is tamed we know that the minimal necessary precision has been exceeded by less than a factor of two. Recall from [59] and [61, Section 2.7] that working precision does not

[11] The paper [10] elaborates upon expression of Ehrlich's iterations via secular equation, shows significant numerical benefits of root-finding by using this expression, and traces the previous study of this approach back to [6].

need to exceed the output precision b by more than a factor of d, and so at most $O(\log(db))$ steps of doubling precision are sufficient. This natural policy has been proposed and elaborated upon in [5] and [10], greatly improving the efficiency of the previous implementations of functional iterations for polynomial root-finding.

9.3 Deflation for Ehrlich's iterations: initial comments

Suppose that Ehrlich's iterations have tamed all roots of p but the w wild roots $x_1, \dots, x_w$. We propose (a) to deflate p by computing its factor $f(x) = \prod_{j=1}^{w}(x - x_j)$ and then (b) to apply Ehrlich's iterations to this factor rather than to p. The potential gain of this deflation may be lost because of coefficient growth, the loss of sparseness, and the cost of performing deflation, but we can monitor the adverse factors (cf. Remark 10) and delay deflation if they prevail.

Subsequent Ehrlich's iterations may in turn tame a part of the roots of the polynomial $f(x)$, and then we can recursively deflate the computed factors as long as the gain of performing deflation exceeds its cost.[12]

9.4 Two deflation algorithms for Ehrlich's iterations

One can try to accelerate Ehrlich's iterations by means of combining them with DLS deflation, but next we describe two alternative algorithms, which we further study in Sections 10.6 and 10.7.

In that description we scale the variable x: $x \to x/h$ for $h = r_1\theta$, which moves all roots of p into the disc $D(0, 1/\theta)$ for $\theta > 1$. Having computed the roots $y_j = hx_j$ of the polynomial $t(x) = p(x/h)$ for all j we readily recover $x_j = y_j/h$ for all j, but we must approximate y_j within the error bound $2^{-b}/h$ in order to approximate x_j within the error bound 2^{-b}. For $h > 1$ this scaling means the increase of working precision by $\lceil \log_2(h) \rceil$ and accordingly raises the Boolean cost but may benefit us overall if it lets us apply more efficient algorithms.

Algorithm 24. Fix $K = 2^k$, for an an integer k such that $K/2 \le w < K$, let $\omega^j := \exp(\frac{2\pi}{K})$ and compute (i) $p(\omega^j)$ for $\omega = \omega_K = \exp(\frac{2\pi\mathbf{i}}{K})$ denoting a primitive Kth root of unity (cf. Definition 5), $j = 0, \dots, K-1$, (ii) $f(\omega^j) = p(\omega^j)/\prod_{g=w+1}^{d}(\omega^j - x_g)$ for $j = 0, \dots, K-1$, and (iii) the coefficients of the polynomial $f(x) := p(x)/\prod_{g=w+1}^{d}(x - x_g)$.

This is just evaluation – interpolation technique traced back to Toom's paprer [64]. We ensure as large isolation as we like when we remove the impact of tame roots.

[12] In a typical partition of a root set observed in Ehrlich's, Weierstrass's and other functional iterations that simultaneously approximate all roots of p as well as in Newton's iterations in [63], wild roots are much less numerous than tame roots, that is, the ratio $\deg(f_{i+1})/\deg(f_i)$ is small, and then the coefficient growth and the loss of sparseness cannot be dramatic in the transition from f_i to f_{i+1}.

Algorithm 25. Cf. [62].

(a) Write $p'_{d-i} := \frac{p_{d-i}}{p_d}$ and $s_i := \sum_{j=1}^{d} x_j^i$, for $i = 0, 1, \ldots, w-1$, so that $s_0 = d$, and compute the power sums s_i of the roots of p, for $i = 1, \ldots, w-1$, by solving the following triangular linear system of w Newton's identities (cf. (23)),

$$s_i + \sum_{j=1}^{i-1} p'_{d-j} s_{i-j} = -ip'_{d-i}, \ , \ i = 1, \ldots, w, \tag{29}$$

such that

$s_1 = -p'_{d-1}$,
$p'_{d-1}s_1 + s_2 = -2p_{d-2}$,
$p'_{d-2}s_1 + p'_{d-1}s_2 + s_3 = -3p_{d-3}$,
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

(b) By subtracting the $(i+1)$st powers of all $d-w$ tame roots, compute the $(i+1)$st power sums $s_{i+1,f}$, for $i = 0, 1, \ldots, w-2$, of the roots of the polynomial $f(x) = \sum_{i=0}^{w} f_i x^i$.

(c) Finally recover its coefficients from the associated triangular linear system of Newton's identities:

$$\sum_{j=1}^{i-1} s_{i-j,f} f'_{w-j} + if'_{w-i} = -s_{i,f} \text{ for } f'_{w-i} = \frac{f_{w-i}}{f_w}, \ i = 1, \ldots, w, \tag{30}$$

such that

$f'_{w-1} = -s_{1,f}$,
$s_{1,f}f'_{w-1} + 2f'_{w-2} = -s_{2,f}$,
$s_{2,f}f'_{w-1} + s_{1,f}f'_{w-2} + 3f'_{w-3} = -s_{3,f}$,
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

In the next section we supply further details for the latter algorithm, point out its potential numerical problems, analyze both algorithms, and estimate the worst case complexity of Algorithm 24.

Remark 26. Algorithm 24 streamlines deflation compared to the DLS algorithm. This improvement is possible because we implicitly achieve θ-isolation of the wild roots for infinitely large θ when we suppress the impact of the tame roots.

Remark 27. We can solve Newton's identities at stage (c) of Algorithm 25 by applying Algorithm 18, but this cannot make significant improvement because the arithmetic and Boolean cost of that stage is dominated at stage (b).

Algorithm 24 can be applied to a black box polynomial p. Stage (a) of Algorithm 25 involves the $w+1$ leading coefficients of p, but they can be readily computed at a dominated arithmetic and Boolean cost.

9.5 Extension to other functional iterations

The recipes of doubling working precision and consequently of partitioning the roots into tame and wild ones and our recipes for deflation and its analysis can be extended to Weierstrass's [67], Werner's [70], various other functional iterations for simultaneous approximation of all roots of p [36, Chapter 4], and Newton's iterations applied to the approximation of all roots of p. E.g., Schleicher and Stoll in [63] apply Newton's iterations to the approximation of all roots of a polynomial of degree $d = 2^{17}$ and arrive at $w \approx d/1000$ wild roots.

10 Complexity estimates

10.1 Some basic estimates

Hereafter $\mu(u) = O((u \log(u))$ denotes the Boolean complexity of integer multiplication modulo a prime of order 2^u (cf. [28]), and we can represent Boolean complexity $\mathbb{B}$ of an algorithm by the product $\mathbb{A}\mu(\lambda)$ where $\mathbb{A}$ denotes the number of arithmetic operations involved and λ denotes the maximal precision of computations required in order to approximate the output within a fixed error bound $2^{-\lambda}$ provided that all input parameters are known up to within $2^{-\lambda}$ and either an input polynomial p of (1) of degree d has norm $|p|_\infty = 2^\tau$ and the root radius r_1 or two input polynomials u and v of degree at most d have norms $|u|_\infty = 2^{\tau_1}$ and $|v|_\infty = 2^{\tau_2}$ and the root radii r_u and r_v.

Remark 28. We bound the norms of polynomials and the errors of the approximation of their values at the roots of unity by 2^{-b}, rather than $2^{-b'}$, which we keep for the bounds on root errors, in line with the notation of Problems 1 – 4.

Theorem 29. *Write* $k := \lceil \log_2 d \rceil$ and $K := 2^k$. *Then*

$$\mathbb{A} = O(K \log(K)) \text{ and } \lambda = b + \tau + k + 3$$

for the evaluation within the error bound 2^{-b} *of a polynomial* p *of (1) at all the* K*-th roots of unity* ω^j, $\omega = \exp(\frac{2\pi\mathbf{i}}{K})$, $j = 0, 1, \ldots, K-1$. *Furthermore*

$$|p(\omega^j)| \le K|p|_\infty = 2^{k+\tau} \text{ for all } j.$$

Theorem 30. *Write* $k := \lceil \log_2(2d+1) \rceil$ and $K := 2^k$. *Then*

$$\mathbb{A} = O(K \log(K)) \text{ and } \lambda = b + O(\tau + d)$$

for the evaluation within 2^{-b} *of the polynomial* p *at the Chebyshev points*

$$v_j = \frac{1}{2}(\omega_K^j + \omega_K^{-j}),$$

for $j = 0, \ldots, K-1$ *and* $\omega_K = \exp(\frac{2\pi\mathbf{i}}{K})$. *Furthermore*

$$|p(v_j)| \le K|p|_\infty = 2^{k\tau}.$$

Theorem 31. *Write* $k := \lceil \log_2(2d+1) \rceil$ *and* $K := 2^k$. *Then*

$$\mathbb{A} = O(K\log(K)) \text{ and } \lambda = b + 2\tau_1 + 2\tau_2 + 5.1k + 4$$

for the computation within 2^{-b} *of the product* uv *of two polynomials* u *and* v. *Furthermore*

$$|uv|_\infty \le K\ |u|_\infty |v|_\infty = 2^{k+\tau_1+\tau_2}.$$

Theorem 32. *Write* $k := \lceil \log_2(2d+1) \rceil$ *and* $K := 2^k$. *Suppose that* $|p| \le 1$ *and* $|s| \le 2$ *for a complex number* s. *Then*

$$\mathbb{A} = O(K\log(K)) \text{ and } \lambda = b + O(\tau + d)$$

for the computation of the coefficients of the polynomial $\tilde{t}(x)$ *such that* $|\tilde{t}(x) - p(x-s)| \le 2^{-b}$. *Furthermore*

$$|p(x-s)|_\infty \le 1.5|p|_\infty 3^K.$$

10.2 The complexity of the approximation of the power sums of the roots in the disc $D(0,1)$, circle $C(0,1)$, and segment $S[-1,1]$

Perform the three DFTs in Algorithm 13 by applying FFT or generalized FFT (see [46, Sections 2.1-2.4]). Then less than $3d + O(q\log(q))$ arithmetic operations are involved at stages (i) – (v) overall.

Theorem 33. *Assume that the disc* $D(0,1/\theta)$ *is* θ^2*-isolated for* $\theta > 1$, *that is, let no roots of* $p(x)$ *lie in the annulus* $A(0,1/\theta,\theta) = \{z :\ 1/\theta \le |z| \le \theta\}$. *Then*

$$\mathbb{A} = O(d\log(q)) \text{ and } \lambda = b + \tau + 2\Big\lceil \log_2\Big(d_f + \frac{d}{\theta-1}\Big)\Big\rceil + 6\log_2 d + 11$$

for the approximation within 2^{-b} *of the power sums* $s_0, \dots, s_{q-1}$ *of the roots of* p *lying in the unit disc* $D(0,1)$ *by means of application of Algorithm 13. Furthermore*

$$\lambda = b + \tau + O(\log(d))$$

if $\theta - 1$ *exceeds a positive constant.*

Proof. Recall that $|p'(x)/p(x)| = |\sum_{j=1}^d \frac{1}{x-x_j}| \le \sum_{j=1}^d \frac{1}{|x-x_j|}$ and the disc $D(0,1/\theta)$ is θ^2-isolated. Hence

$$\Big|\frac{p'(x)}{p(x)}\Big| = \sum_{j=1}^{d} \frac{1}{|x-x_j|} \le \frac{d_f}{1-1/\theta} + \frac{d-d_f}{\theta-1} = d_f + \frac{d}{\theta-1} \text{ for } |x| = 1. \quad (31)$$

Combine this bound with Theorem 29 and deduce that it is sufficient to compute the ratios $\frac{p(\omega^g)}{p'(\omega^g)}$ for all g within $2^{-\hat{b}}$, $\hat{b} = b + \lceil \log_2(d_f + \frac{d}{\theta-1})\rceil + \lceil \log_2 q\rceil + 3$, in order to output the values s_h^* within 2^{-b} for all h.

We can ensure this bound by computing the values $p(\omega^g)$ and $p'(\omega^g)$ for all g with the precision increased by $\lceil \log_2(d_f + \frac{d}{\theta-1})\rceil$ bits.

By virtue of Theorem 29 this requires to add just

$$\lceil \log_2(q)\rceil + \left\lceil \log_2\left(\frac{d}{q}\max\{|p|_\infty, |p'|_\infty\}\right)\right\rceil + 3$$

bits to the precision in the computation at stage (iii). This bound is at most

$$\lceil \log_2(q)\rceil + \left\lceil \log_2\left(\frac{d^2}{q}|p|_\infty\right)\right\rceil + 3$$

because $|p'|_\infty \le d|p|_\infty| = d\tau$.

It is sufficient to add $\lceil \log_2(d)\rceil$ and $\lceil \log_2(2d)\rceil$ bits to the precision in order to cover its growth at stages (i) and (ii), respectively.

Summarizing we obtain that it is sufficient to perform stages (i) – (v) with the precision λ of

$$b + 2\left\lceil \log_2\left(d_f + \frac{d}{\theta-1}\right)\right\rceil + \left\lceil \log_2\left(\frac{d^2}{q}|p|_\infty\right)\right\rceil + 6 + 3\lceil \log_2(d)\rceil + \lceil \log_2(2d)\rceil$$

bits in order to compute the values s_h^* within 2^{-b} for all h.

Simplify this expression by recalling that $q > 1$, $\log(|p|_{infty}) = \tau$, $\log_2(x) + \log_2(y) = \log_2(y)$ and $\lceil x\rceil \le x + 1$ for all real x and y. □

Corollary 10.1. For a polynomial p of (1), an integer $q > 1$, real b, and θ exceeding 1 by a positive constant, let the disc $D(0,1)$ be θ-isolated, let $\{x_1, \dots, x_{d_f}\}$ denote the set of the roots of p lying in the disc $D(0,1)$, and let $f(x) := \prod_{j=1}^{d_f}(x - x_j)$. Then

$$\mathbb{A} = O(d\log(q)) \text{ and } \lambda = b + \tau + O(\log(d))$$

for the approximation within 2^{-b} of the power sums $s_0, \dots, s_k$ of the roots of the polynomial $f = f(x)$.

Proof. Combine Theorems 33 and 14 and deduce the corollary in the case where the disc $D(0,1/\theta)$ is θ^2-isolated. By scaling the variable x extend this result to the case where the disc $D(0,1/\theta)$ is θ^2-isolated. Replacing θ^2 by θ does not change the claimed cost bound, and if $\theta - 1$ exceeds a positive constant, then so does θ^2 as well. □

Corollary 10.2. Corollary 10.1 still holds if $\{x_1, \dots, x_{d_f}\}$ is the root set of a polynomial p on the θ-isolated circle $C(0,1)$ rather than in the θ-isolated disc $D(0,1)$ and if the task is the approximation of the power sums $s_1, \dots, s_k$ of these roots.

Proof. By applying the argument used at the very end of the proof of Corollary 10.1 deduce that it is sufficient to prove Corollary 10.2 under the assumption that the circle $C(0,1)$ is θ^2-isolated. Under that assumption the discs $D' = D(0,1/\theta)$ and $D'' = D(0,\theta)$ are θ-isolated.

By extending the algorithm that supports Corollary 10.1 approximate within 2^{-b-1} the first k power sums $s'_1,\dots,s'_k$ and $s''_1,\dots,s''_k$ of the roots of p in the discs D' and D'', respectively. Essentially the same proof is applied, with a slight modification for the accommodation of the increase of the numbers of roots of p from d_f on the circle $C(0,1)$ to at most d in the discs D' and D''.

Finally approximate the power sums $s_j = s''_j - s'_j$ for $j = 1,\dots,2d'_f$ within 2^{-b}, that is, by at most doubling the previous bound. □

Corollary 10.3. Corollary 10.1 still holds if $\{x_1,\dots,x_{d_f}\}$ is the root set of a polynomial p on the segment $S[-1,1]$ rather than in the θ-isolated disc $D(0,1)$, if all roots of the polynomial p of (1) lie either in the segment $S[-1,1]$ or outside the ellipse $E(0,\theta)$ of Remark 21 for $\theta - 1$ exceeding a positive constant, and if the task is the approximation of the power sums $s_1,\dots,s_k$ of these roots.

Proof. Combine Corollary 10.2 with Theorem 30. □

Remark 34. The cost bounds of the corollaries do not cover the cost of computing θ, but we do not use θ in our computations. We just perform them by first using a smaller number of points q and then recursively double it until the algorithm succeeds. See Section 4.8 on verification of the success.

10.3 The complexity of the approximation of the power sums of the roots in any disc

Based on [55, Lemma 21] for $m = d$, $n = 1$, and $P_j = x-(c+r\omega^j)$, $j = 0,\dots,q-1$ estimate the complexity of the computation of the values $p(v_g)$ for $v_g = c + r\omega^g$ and $g = 0,1,\dots,q-1$ in Algorithm 16.

Lemma 35. *For a polynomial $p(x)$ of (1), an integer q, $\omega = \exp(\frac{2\pi\mathbf{i}}{q})$, a complex c, a positive ρ, a real b, and $r_1 := \max_{j=1}^d |x_j|$, let the coefficients of $p(x)$, and the values $c + \rho\omega^g$ for all g be given within $2^{-\lambda}$ for*

$$\lambda = b + \tau\log_2(d) + 30(\log_2(r_1) + 3)d + \log_2(d) + 60\log_2(d)\log_2^2(d+1),$$

which implies that

$$\lambda = b + (\tau + \log^2(d+1)))\log(d) + d\log(r_1).$$

Then one can approximate within 2^{-b} the values $p(c+\rho\omega^g)$ for all g by performing $\mathbb{A} = O(d\log^2(d+1))$ arithmetic operations with precision λ. Furthermore

$$\log_2|p(c+\rho\omega^g| \le \tau + \log_2((r_1+1)d) + 1 \text{ for all } g.$$

The lemma can be immediately extended to cover the Boolean cost of the computation of the values $\frac{1}{\rho}p'(v_g)$ for $g = 0,1,\dots,q-1$ if we increase the precision by at most $\log_2(d) + O(1)$ bits.

Notice that bound (31) is extended to the ratios $\max_{g=0}^{q-1}\left|\frac{t'(v_g)}{t(v_g)}\right|$, combine it with the estimates of this lemma, its extension to the latter evaluation and Theorem 33, and arrive at the following bound on the overall computational cost of performing Algorithm 16.

Theorem 36. *Under the assumptions of Algorithm 16 let* $q-1$ *exceeds a positive constant. Then one can approximate the value* $p(c+\rho\omega^g)$ *and* $p'(c+\rho\omega^g)/\rho$ *for* $g = 0, 1, \dots, K-1$ *within* 2^{-b} *for all* g *by performing Algorithm 16 at the cost bounded by*

$$\mathbb{A} = O(d\log^2(d)) \text{ and } \lambda = b + \tau + O(\log(d) + d\log(r_1) + (d_f + \frac{d}{\theta - 1})K).$$

Next move a disc $D(c, \rho)$ into the unit disc $D(0, 1)$ by means of scaling the variable $x \to x/r_1$ and then would become ill-conditioned, and then again we would not ensure any output accuracy.

Nevertheless in the tests by Schleicher [62] the algorithm has consistently output accurate solutions in the tests of its combination with Newton's iterations polynomials of the sequences (5) and (6) having high degree in the range from 10^7 to 10^9. Apparently in these applications the wild and tame roots are always spread more or less in the same regions, so that the ratios $|s_g/s_g(f)|$ are reasonably bounded.

One is challenged to study how typically the wild and tame roots are distributed in such a way, favorable to the application of Algorithm 16.

10.4 Recovery of the coefficients from the power sums of the roots

Theorem 37. *Given the power sums* $s_1, \dots, s_{2d_f-1}$ *of the roots of a monic polynomial* $f(x)$ *of degree* d_f *having all its roots in the unit disc* $D(0, 1)$*, one can approximate its coefficients within* 2^{-b} *at the cost*

$$\mathbb{A} = O(d_f \log(d_f)) \text{ and } \lambda = b + \tau + O(d_f).$$

Proof. See [59, Lemma 13.1]. □

Remark 38. We can move the roots of a polynomial f into the unit disc $D(0, 1)$ by means of scaling of the variable $x \to x/r_1$. Then we could ensure the same output error bound if we increase working precision by $\lceil d_f \log_2(r_1)\rceil$.

10.5 Complexity of deflation algorithms: a table

Remark 39. The above complexity bound for lazy deflation is a little more favorable than that for DLS deflation over any disc, but the opposite is true for their accuracy according to the estimates in Sections 5 – 7.

Table 1: Complexity of deflation, $\tau = \log(|p|_\infty)$

Task/Algorithm	$\mathbb{A}$	$\lambda - b$
Unit disc	$q\log(q)$	$\tau + \log(d)$
Any disc	$d\log^2(d)$	$(q + \log(r_1+1))d + \tau + \log(d)$
Recovery	$d_f \log(d_f)$	$d_f + \tau$
Lazy deflation	$d\log(d)$	$d + \tau$

10.6 Complexity of Algorithm 24

Theorem 40. *Suppose that the polynomials p and f are monic and that*

$$r_1 := r_1(p) := \max_{j=1}^{d} |x_j| \le \frac{1}{\theta} < 1, \tag{32}$$

in which case all roots of p lie in the disc $D(0, 1/\theta) = \{|x| : \ |x| \le \frac{1}{\theta}\}$ for a constant $\theta > 1$. Then Algorithm 24 computes the coefficients of the polynomial $f(x)$ with errors at most $1/2^b$ at the cost

$$\mathbb{A} = O(d\log(w)) \text{ and } \lambda = b + B_{d,w,\theta} + \lceil \log_2(wd) \rceil + 6$$

where

$$B_{d,w,\theta} = \Big\lceil d\log_2\Big(\frac{\theta+1}{\theta}\Big)\Big\rceil + \Big\lceil w\log_2\Big(\frac{\theta+1}{\theta}\Big)\Big\rceil + (d-w)\Big\lceil \log_2\Big(\frac{\theta}{\theta-1}\Big)\Big\rceil.$$

Proof. We perform $(d-w)q$ arithmetic operations at stage (ii), $\lceil d/w \rceil$ FFTs on q points at stage (i), and a single inverse FFT on q points at stage (iii). This involves $O(d\log(w))$ arithmetic operations overall.

Apply Theorem 29 to the polynomial f replacing p and obtain that it is sufficient to perform stage (iii) with the precision

$$b'' = b' + \lceil \log_2(|f|_\infty) \rceil + \lceil \log_2(w) \rceil + 3$$

within a dominated Boolean cost bound in $O(w\log(w)\mu(b''))$.

Bound (32) implies that $|\omega^j - x_g| \ge \frac{\theta-1}{\theta}$ for all g and j, and so in the w division by $\omega^j - x_g$ for all g at stage (ii) we lose at most $\Big\lceil w\log_2\Big(\frac{\theta}{\theta-1}\Big)\Big\rceil$ bits of precision. Therefore it is sufficient to compute the values $p(\omega^j)$ for all j with the precision $\hat{b} = b'' + \lceil \log_2((d-w)\frac{\theta}{\theta-1}) \rceil$ at stage (i).

Estimate the Boolean cost of performing stage (i) by applying Theorem 29 for b replaced by $\hat{b}$ and obtain that at that stage it is sufficient to perform $O(d\log(d))$ arithmetic operations with the precision

$$\hat{b} + \lceil \log_2(|p|_\infty) \rceil + \lceil \log_2(d) \rceil + 3.$$

Substitute bound (32) and obtain the theorem. □

Corollary 10.4. Suppose that under the assumptions of Theorem 40 scaling of the variable x has ensured an isolation ratio $\theta \geq 2$. Then it is sufficient to choose the precision bound

$$\lambda = b + (\log_2(3) + 1)d + (\log_2(3) - 1)w + \lceil \log_2(dw) \rceil + 8$$
$$\leq 2.585\ d + 0.585\ w + \lceil \log_2(dw) \rceil + 8.$$

10.7 Analysis of Algorithm 25

We can solve each of the linear systems (29) and (30) by means of substitution, by using w^2 arithmetic operations. Both systems can be ill-conditioned. E.g., the inverse of the matrix of (30) has the southwestern entry $\pm(f'_{w-1})^{w-1}$ if $f'_{w-i} = 0$ for all $i > 1$.

We can, however, ensure strong numerical stability of the solution of linear systems (29) and (30) by means of substitution if we scale the variable x so that, say, $\sigma_{p,w} = \sum_{i=0}^{w-1} |p'_i| \leq 1/2$ and $\sigma_{s,w} = \sum_{i=0}^{w-1} |s_{i,f}| \leq 1/2$, respectively.

Then clearly $\sigma_{p,w} \leq 1/2$ if we move all roots of p into the disc $D(0, \frac{\gamma}{2w}) = \{x : |x| \leq \frac{1}{4w}\}$, but by moving them into a smaller disc $D(0, \frac{1}{2d+4w}) = \{x : |x| \leq \frac{1}{2d+4w}\}$ we ensure that $|s_{i,f}| \leq \frac{1}{4w}$ at stage (b), which implies that $\sigma_{s,w} \leq 1/2$. We can achieve this by scaling the variable

$$x \to x/h \text{ for } h = (2d + 4w)r_1$$

and for r_1 denoting the root radius.

Under this scaling we can bound by $d + 2w + \log_2(1/2) = d + 2w - 1$ the increase of working precision versus the output precision b'.

Since (29) is a triangular Toeplitz linear system, one can solve it by using $O(w \log(w))$ arithmetic operations (see, e.g., [46, Section 2.5]), but the overall arithmetic cost of recipe 3 is dominated at stage (b), which involves about $3(d - w)w$ arithmetic operations.

Moreover, in the subtractions at that stage we lose about

$$b_{\text{loss}} = \log_2 \left(\left| \frac{\sum_{j=1}^{d} x_j^g}{\sum_{j=1}^{w} x_j^g} \right| \right)$$

bits of precision of the power sum $s_{g,f}$ of the roots of the polynomial f for any g. This value is unbounded for the worst case input, and so the output errors of Algorithm 25 are unbounded for the worst case input under any fixed bound on working precision.

The value b_{loss}, however, can be moderate for many inputs, and possibly in some sense for most of inputs. According to [62] the tests of the combination of Algorithm 25 with Newton's iterations consistently succeeded in approximation of all roots of the polynomials defined by the recurrence sequences (5) and (6) and having degrees in the range from 10^7 to 10^9.

10.8 Boolean complexity of Problems 1 – 4 with and without MPSolve

Let us briefly compare the complexity of Ehrlich's iterations and MPSolve versus the algorithms of [42], [47], [11], and [12] for the solution of Problems 1 – 4.

Empirically Ehrlich's iterations have simpler structure and smaller overhead in comparison with the algorithms of the latter papers, but unlike them have only empirical support for being nearly optimal.

Moreover super-linear convergence of Ehrlich's iterations has only been observed for simultaneous approximation of all roots, and so these iterations solve Problems 3 and 4 of root-finding in a disc and on a line interval about as fast and as slow as Problem 1 of root-finding on the complex plain, while the nearly optimal cost of root-finding by the algorithms of [42], [47], [11], and [12] decreases at least proportionally to the number of roots in the input domain.

As we have already said in the introduction, MPSolve, [11] and [12] can solve Problem 2 of factorization of p within the same Boolean cost bound as Problem 1, whereas the algorithm of [42] and [47] solves Problem 2 faster by a factor of d, reaching a nearly optimal Boolean cost bound.

11 Conclusions

1. Synergistic combination of efficient techniques proposed and developed for distinct polynomial root-finders is a natural direction but requires effort and maybe luck. Our advance in this direction is rather exceptional if not pioneering, but we hope that it will encourage new research effort (see [31] for another attempt of synergistic combination of distinct root-finding techniques).

2. As we said, Ehrlich's and Weierstrass's iterations globally converge to all roots much faster than their formal study ensures, and one is challenged to explain excellent empirical performance of these and other functional iterations for root-finding that have much weaker formal support. This is a well- and long-known challenge, but our study leads to three new challenges of this kind.

(a) We have already cited usefulness of lazy deflation for real root-finding according to [34] and

(b) of Algorithm 24 in Newton's iterations for very high degree polynomials given by recurrences (5) and (6), according to [62].

(c) We can add that the initial tests of our new recipe for root-counting suggest that its actual power is significantly stronger than its formal support indicates.

3. For an example of a natural technical extension of our study consider the task of polynomial root counting in a triangle, a square and a rectangle with potential applications to subdivision iterations. One can try to extend our counting algorithm by combining our techniques of Sections 4.4 and 8 and to compare the efficiency of such extension with that of the algorithms of [72] and of Section 5.3.

APPENDIX

PART I: RECURSIVE FACTORIZATION OF A POLYNOMIAL AND ITS EXTENSION TO ROOT-FINDING

A Recursive splitting into factors

A.1 Auxiliary norm bounds

We first state two simple lemmas.

Lemma 41. *It holds that* $|u + v| \le |u| + |v|$ *and* $|uv| \le |u|\ |v|$ *for any pair of polynomials* $u = u(x)$ *and* $v = v(x)$.

Lemma 42. *Let* $u = \sum_{i=0}^{d} u_i x^i$ *and* $|u|_2 = (\sum_{i=0}^{d} |u_i|^2)^{1/2}$. *Then* $|u|_2 \le |u|$.

The following lemma relates the norms of a polynomial and its factors.

Lemma 43. *If* $p = p(x) = \prod_{j=1}^{k} f_j$ *for polynomials* $f_1, \dots, f_k$ *and* $\deg p \le d$, *then* $\prod_{j=1}^{k} |f_j| \le 2^d |p|_2 \le 2^d |p|$.

Proof. The leftmost bound was proved by Mignotte in [35]. The rightmost bound follows from Lemma 42. □

Remark 44. [61, Lemma 2.6] states with no proof a stronger bound as follows: $\prod_{j=1}^{k} |f_j| \le 2^{d-1}|p|$ under the assumptions of Lemma 43. From various factors of the polynomial $p(x) = x^d - 1$ such as $\prod_{j=1}^{d/2}(x - \omega_d^j)$ for even d, one can see some limitations on strengthening this bound even further.

A.2 The errors and complexity of recursive factorization

Suppose that we split a polynomial p into a pair of factors over some θ-isolated discs and recursively apply this algorithm to the factors until they become linear of the form $ux + v$; some or all of them can be repeated. Finally we arrive at complete approximate factorization

$$p \approx p^* = p^*(x) = \prod_{j=1}^{d} (u_j x + v_j). \tag{33}$$

Next, by following [59, Section 5], we estimate the norm of the residual polynomial

$$\Delta^* = p^* - p. \tag{34}$$

We begin with an auxiliary result.

Theorem 45. *Let*

$$\Delta_k = |p - f_1 \cdots f_k| \le k\epsilon |p|/d, \tag{35}$$

$$\Delta = |f_1 - fg| \le \epsilon_k |f_1|, \tag{36}$$

for some non-constant polynomials $f_1, \dots, f_k, f$ *and* g *and for*

$$\epsilon_k \le \epsilon/(d2^d \prod_{j=1}^{k} |f_j|). \tag{37}$$

Then

$$|\Delta_{k+1}| = |p - fgf_2 \cdots f_k| \le (k+1)\epsilon|p|/d. \tag{38}$$

Proof. $\Delta_{k+1} = |p - f_1 \cdots f_k + (f_1 - fg)f_2 \cdots f_k|$. Apply Lemma 41 and deduce that $\Delta_{k+1} \le \Delta_k + |(f_1 - fg)f_2 \cdots f_k|$ and furthermore that

$$|(f_1 - fg)f_2 \cdots f_k| \le |f_1 - fg|\ |f_2 \cdots f_k| = \Delta|f_2 \cdots f_k|.$$

Combine the latter inequalities and obtain $\Delta_{k+1} \le \Delta_k + \Delta|f_2 \cdots f_k|$. Combine this bound with (35)–(37) and Lemmas 41 and 43 and obtain (38). □

Write $f_1 := f$ and $f_{k+1} = g$. Then (38) turns into (35) for k replaced by $k+1$. Now compute one of the factors f_j as in (36), apply Theorem 45, then recursively continue splitting the polynomial p into factors of smaller degrees, and finally arrive at factorization (33) with

$$|\Delta^*| \le \epsilon|p|$$

for Δ^* of (34). Let us call this computation *Recursive Splitting Process* provided that it begins with $k = 1$ and $f_1 = p$ and ends with $k = d$.

Theorem 46. *To support (35) for all* $j = 1, 2, \dots, d$ *in the Recursive Splitting Process for a positive* $\epsilon \le 1$*, it is sufficient to choose* ϵ_k *in (36) satisfying*

$$\epsilon_k \le \epsilon/(d2^{2d+1}) \text{ for all } k. \tag{39}$$

Proof. Prove bound (35) by induction on j. Clearly, the bound holds for $k = 1$. It remains to deduce (38) from (35) and (39) for any k. By first applying Lemma 43 and then bound (35), obtain

$$\prod_{i=1}^{k} |f_i| \le 2^d |\prod_{i=1}^{k} f_i| \le 2^d(1 + k\epsilon/d)|p|.$$

The latter bound cannot exceed $2^{d+1}|p|$ for $k \le d$, $\epsilon \le 1$. Consequently (39) ensures (37), and then (38) follows by virtue of Theorem 45.

□

Remark 47. The theorem shows that by using working precision of

$$\bar{b} \ge \bar{b}_{\inf} = 2d + 1 + \log_2 d + b'$$

bits throughout the Recursive Splitting Process we can ensure the output precision b'.

A.3 Overall complexity of recursive factorization

The overall complexity of recursive is bounded by the sum of the bounds on the complexity of all splittings into pairs of factors. In this recursive process in [42] and [47] the degrees of the computed factors decreased by at least a factor of 12/11, and so the overall cost of recursive splitting was proportional to the cost of the first splitting. Including also complexity of the algorithms that supported the isolation of the factors and those that controlled the decrease of their degrees the overall Boolean cost of recursive factorization was bounded by $O(\mu(b')d\log(d)(\log^2 d + \log(b)))$. No increase from the bound $O(d\log(d)\mu(b'))$, however, is required in the application of splitting algorithms within Ehrlich's and subdivision iterations provided that an isolation ratio $\theta \geq 1 + g/\log^h d$ is ensured in every splitting.

B From factors to roots

Theorem 48. *[61, Theorem 2.7]. Suppose that*

$$p = p_d \prod_{j=1}^{d}(x - x_j),\ \tilde{p} = \tilde{p}_d \prod_{j=1}^{d}(x - y_j),\ |\tilde{p} - p| \leq \epsilon|p|,\ \epsilon \leq 1/2^{7d}$$

and

$$|x_j| \leq 1 \text{ for } j = 1, \dots, d' \leq d,\ |x_j| \geq 1 \text{ for } j = d'+1, \dots, d.$$

Then up to reordering the roots of p it holds that

$$|y_j - x_j| < 9\epsilon^{1/d} \text{ for } j = 1, \dots, d';\ |1/y_j - 1/x_j| < 9\epsilon^{1/d} \text{ for } j = d'+1, \dots, d.$$

By virtue of Theorem 48 for $b' = O(bd)$ we can bound the Boolean complexity of the solution of Problem 1 by increasing the estimate for the complexity of factorization in Section A.3 by a factor of d.

Corollary B.1. *Boolean complexity of the solution of Problem 1.* Given a polynomial p of degree d and a positive b, one can approximate all roots of that polynomial within the error bound $1/2^b$ at a Boolean cost in $O(\mu(b')d^2\log(d)(\log^2 d + \log(b))) = \tilde{O}(bd^2)$.

By combining this study with [59, Section 20] we estimate Boolean complexity of the following problem.

Problem 5. *Polynomial root isolation.* Given a polynomial p of (1) that has integer coefficients and only simple roots, compute d disjoint discs on the complex plane, each containing exactly one root of p.

Corollary B.2. *Boolean complexity of polynomial root isolation.* Suppose that a polynomial p of (1) has integer coefficients and has only simple roots. Let σ_p denotes its *root separation*, that is, the minimal distance between a pair of its roots. Write $\epsilon := 0.4\sigma_p$ and $b' := \log_2(1/\epsilon)$. Let $\epsilon < 1$ and let $m = m_{p,\epsilon}$

denote the maximal number of the roots of the polynomial $p(x)$ in ϵ-clusters of its roots. Then Problem 5 of Root Isolation for p can be solved in Boolean time $\tilde{O}(bdm)$ for $b = b'/m$.

PART II: NEWTON'S REFINEMENT OF SPLITTING A POLYNOMIAL INTO FACTORS

C Refinement of splitting into two factors

C.1 Refinement algorithm for two factors

The algorithms of Part I combined with those of Sections 5 and 6 compute accurate approximate splitting of a polynomial p into the product of two factors at a nearly optimal Boolean cost $O(d\log(d)\mu(b))$. It increases proportionally to $\mu(b)$ as b grows, but for large b one can save a factor of $\log(d)$ by applying Kronecker's map for multiplication of polynomials with integer coefficients (see [23], [40, Section 40]). This application motivated Schönhage in [59, Sections 10 − 12] to devise efficient algorithms for Newton's refinement of splitting a polynomial into factors, which enabled super-linear decrease of the error of splitting and equivalently super-linear increase of b. Consequently he decreased the overall Boolean cost of highly accurate splitting, for large b, to $O(d\mu(b))$. It is not clear whether this decrease has any practical promise, but the result is interesting for the theory, and the algorithms seem to be of independent interest. Next we outline Schönhage's algorithms and estimates.

Given the coefficients of a polynomial p and its approximate splitting into the product of two polynomials $f_{1,0}$ and $f_{2,0}$,

$$p \approx f_{1,0}f_{2,0}, \tag{40}$$

we update this initial splitting as follows:

$$p \approx f_{1,1}f_{2,1},\ \ f_{1,1} = f_{1,0} + h_{1,0}\ \ f_{2,1} = f_{2,0} + h_{2,0}, \tag{41}$$

where the polynomials $h_{1,0}$ and $h_{2,0}$ satisfy

$$p - f_{1,0}f_{2,0} = f_{1,0}h_{2,0} + h_{1,0}f_{2,0},\ \ \deg(h_{i,0}) < \deg(f_{i,0}) \text{ for } i = 1, 2. \tag{42}$$

This is Newton's iteration. Indeed substitute (41) into exact splitting $p = f_{1,1}f_{2,1}$ and arrive at (42) up to a single term $h_{1,0}h_{2,0}$ of higher order. Given two polynomials $f_{1,0}$ and $f_{2,0}$ one completes this iteration by computing polynomials $h_{1,0}$ and $h_{2,0}$ and then can continue such iterations recursively.

From equation (42) we obtain that $p = h_{1,0}f_{2,0} \mod f_{1,0}$, and so $h_{1,0} = p\bar{h}_1 \mod f_{1,0}$ where the multiplicative inverse $\bar{h}_1$ satisfies $\bar{h}_1 f_{2,0} \mod f_{1,0} = 1$. Having computed the polynomials $\bar{h}_1$ and $h_{1,0}$ we can obtain the polynomial $h_{2,0}$ from equation (42) within a dominated cost bound by means of approximate polynomial division (cf. [60], [4], [52], [53], or [32]).

C.2 Fast initialization of Newton's refinement of splitting

Next we recall Schönhage's numerically stable algorithm of [59, Sections 10 – 12] for Newton's refinement of a close initial approximate splitting (40) computed by the algorithms of Sections 5 and 6. Given an initial approximation $\bar{h}_{1,0}$ to $\bar{h}_1$ the algorithm recursively updates it by computing the polynomials $j_{i,0} = 1 - \bar{h}_{i,0} f_{2,0} \mod f_{1,0}$ and $\bar{h}_{i+1,0} = \bar{h}_{i,0}(1 + j_{i,0}) \mod f_{1,0}$ for $i = 1, 2, \dots$ [13]

For any $b > d$ the computations ensure the bound $|\bar{h}_{i,0}| \le 2^{-b}$ by using $O(d\mu(b))$ Boolean operations provided that

$$|\bar{h}_{1,0}| \le \nu^2/(w^2 2^{2d}) \tag{43}$$

where $w = \deg(f_1) \le d$, $\nu = \min_{x:\ |x|=1} |p(x)|$ (see [59, Lemma 11.1]) and $1/\nu \le 2^{cn}$) for a constant c [59, equation (16.7)]. If in addition

$$|p - f_{1,0} f_{2,0}| \le \nu^4/(w^4 2^{3d+w+1}), \tag{44}$$

then the new factors $f_{1,1}$ and $f_{2,1}$ can be computed by using $O(d\mu(b))$ Boolean operations such that $|p - f_{1,1} f_{2,1}| \le |p|\ 2^{-b}$.

We can ensure bound (44) by performing the algorithms of Sections 5 and 6 within the same Boolean cost estimate. This completes the computation of splitting in overall Boolean time in $O(d\mu(b))$ provided that within this cost bound one can also compute an initial polynomial $\bar{h}_{1,0}$ satisfying (43).

We can do this based on the following expression of [59, equation (12.19)]:

$$\bar{h}_{1,0} = \sum_{i=0}^{w-1} \Big(\sum_{j=i+1}^{w} u_{w-j} v_{j-i} \Big) x^i$$

where u_k and v_k are the coefficients of the polynomial $f_{1,0}(x) = x^w + u_1 x^{w-1} + \dots + u_w$ and of the Laurent expansion $\frac{1}{f_{1,0}(x) f_{2,0}()x} = \sum_k v_k x^k$, respectively.

C.3 Alternative computation of multiplicative inverse

Alternatively we can compute the coefficient vector $\bar{\mathbf{h}}_1$ of the multiplicative inverse $\bar{h}_1$ by solving the Sylvester linear system of equations

$$S_{f_{2,0}, f_{1,0}} \mathbf{v} = \mathbf{e}_d \tag{45}$$

where $S_{f,g}$ denotes the Sylvester matrix of the polynomials f and g, $\mathbf{e}_1$ is the last coordinate vector of dimension d, $\mathbf{v}^T = (\mathbf{h}_1^T \mid \mathbf{h}_2^T)^T$, $\mathbf{w}^T$ denotes the transpose of a vector $\mathbf{w}$, and $\bar{\mathbf{h}}_1$ and $\bar{\mathbf{h}}_2$ are the coefficient vectors of the polynomials $\bar{h}_1$ and $\bar{h}_2$ such that $\bar{h}_1 f_{2,0} + \bar{h}_2 f_{1,0} = 1$.

A Sylvester matrix has structure of Toeplitz type with displacement rank two, and so the symbolic solution of Sylvester linear system (45) is as fast as the extended Euclidean algorithm (cf., e.g., [46, Chapter 5]).

[13] We can compute the multiplicative inverse $\bar{h}_1$ by applying the extended Euclidean algorithm (cf., e.g., [46]), but it is not reliable in numerical implementation and keeps an extraneous factor of $\log^2(d)$ in the Boolean complexity bound.

We propose an iterative numerical algorithm for the solution. It is slower than one of [59, Sections 10 – 12] and has no convergence guarantee but requires no initial approximate splitting of p and thus can be of independent interest.

Namely we can compute a low precision generator for the rank-2 displacement of the inverse $X = S^{-1}_{f_{2,0},f_{1,0}}$ by means of the algorithm of [71].[14]

Then we can refine the solution output by that algorithm by applying classical iterative refinement [26], which decreases the initial approximation error by a factor of $\gamma = 1/||I - S_{f_{2,0},f_{1,0}}X||$ at every iteration where I denotes the identity matrix and $||\cdot||$ denotes any fixed (e.g., spectral) matrix norm.

In our case every refinement iteration amounts essentially to multiplication of eight circulant and skew-circulant matrices by vectors and further to performing 24 Fourier transforms (see [46, Examples 4.4.1, 4.4.2 and Theorem 2.6.4]), where one needs no increase of working precision. This implies the overall Boolean cost in $\tilde{O}(db)$ for a fixed target precision $b \geq d$ provided that $\gamma > 1$.

If $\gamma \leq 1$ we can perform one or a few steps of Euclidean algorithm to the pair of polynomials $(f_{1,0}, f_{2,0})$, then use the output pair of polynomials in order to update Sylvester matrix of (45), and finally reapply the algorithm of [71] and iterative refinement to the updated Sylvester linear system of equations.

A bounded number of iterative refinement steps can be performed at dominated Boolean cost. These techniques increase our chances for success but, unlike the algorithm of [59, Sections 10 and 11], do not ensure success of our numerical computation of $\bar{h}_1$ satisfying $h_{1,0} = p\bar{h}_1 \mod f_{1,0}$.

D Newton's refinement of splitting a polynomial into any number of factors

D.1 Newton's refinement algorithm

Next, by following Kirrinnis [32], we generalize Newton's refinement to the case of splitting p into any number s of factors. For a monic polynomial p of (1) he seeks pairwise prime monic polynomials $f_1, \dots, f_s$ and polynomials $h_1, \dots, h_s$, $\deg\ h_j < \deg\ f_j = d_j$, $j = 1, \dots, s$, defining the factorization $p = f_1 \cdots f_s$ and the pfd

$$\frac{1}{p} = \frac{h_1}{f_1} + \cdots + \frac{h_s}{f_s}. \tag{46}$$

Suppose that initially $2s$ polynomials $f_{j,0}$ and $h_{j,0}$, $j = 1, \dots, s$, are given such that

$$f_0 = f_{1,0} \cdots f_{s,0} \approx p,\ \ f_{j,0} \approx f_j \text{ for all } j, \tag{47}$$

$$\frac{1}{f_0} = \frac{h_{1,0}}{f_{1,0}} + \cdots + \frac{h_{s,0}}{f_{s,0}} \text{ and } \deg(h_{j,0}) < \deg(f_{j,0}) \text{ for all } j. \tag{48}$$

[14] The algorithm of [71] is fast and has very strong numerical stability. By following the line of [41] (cf. [48]) it reduces the solution of a Toeplitz system of equations to computations with Cauchy-like matrices and then to the application of the Fast Multipole Method. It can be immediately adjusted to the solution of Sylvester and other linear systems of equations with displacement structure of Toeplitz type.

Then define Newton's iteration for the refinement of initial approximation (47) and pfd (48) by performing the following computations:

$$q_{j,k} = \frac{f_k}{f_{j,k}},\ h_{j,k+1} = (2 - h_{j,k}q_{j,k})h_{j,k} \bmod f_{j,k},\ j = 1, \dots, s, \tag{49}$$

$$f_{j,k+1} = f_{j,k} + (h_{j,k+1}p \bmod f_{j,k}),\ j = 1, \dots, s, \tag{50}$$

$$f_{k+1} = f_{1,k+1} \cdots f_{s,k+1} \tag{51}$$

for $k = 0, 1, \dots$ We can compress equations (49) and (50) as follows,

$$q_{j,k} = \frac{f_k}{f_{j,k}},\quad f_{j,k+1} = f_{j,k} + ((2 - h_{j,k}q_{j,k})h_{j,k}p \bmod f_{j,k}),\ j = 1, \dots, s,$$

Clearly the refinement iterations are simplified where the factors $l_{j,k}$ have small degrees. In particular

$$h_{j,k+1} = (2 - h_{j,k}f'_k(z_{j,k}))h_{j,k}$$

and both $h_{j,k}$ and $h_{j,k+1}$ are constants for all k where $f_{j,k} = x - z_{j,k}$ is a monic linear factor and $f'_k(x)$ denotes the derivative of the polynomial $f_k(x)$.

D.2 The overall complexity of Newton's refinement

Kirrinnis in [32] assumes that all roots of p have been moved into the unit disc $D(0,1)$, the s root sets of the s factors $f_1, \dots, f_s$ as well as the s root sets of the initial approximations $f_{1,0}, \dots, f_{s,0}$ to these factors are pairwise well isolated from each other, and a given initial factorization and pfd (48) is sufficiently close to (46). His theorem below shows that, under these assumptions and for $f_k = f_{1,k} \cdots f_{s,k}$, sufficiently large k, and $\mu(u) = O((u \log(u))$, in k iterations his algorithm ensures the approximation error norm bounds

$$\delta_k = \frac{|p - f_k|}{|p|} \le 1/2^b,\quad \sigma_k = \Big|1 - h_{1,k}\frac{f_k}{f_{1,k}} - \dots - h_{s,k}\frac{f_k}{f_{s,k}}\Big| \le 1/2^b,$$

at the Boolean cost in $O(d\mu(b')\log(d))$. By applying the algorithms of [59] $s-1$ times we arrive at similar cost estimates, but the Kirrinnis algorithm streamlines the supporting computations.

Theorem 49. *Let s, d, $d_1, \dots, d_s$ be fixed positive integers such that*

$$s \ge 2,\ d_1 + \dots + d_s = d.$$

Let p, $f_{i,0}$, $h_{i,0}$, $i = 1, \dots, s$, be $2s+1$ fixed polynomials such that p, $f_{1,0}, \dots, f_{s,0}$ are monic and

$$\deg(f_{i,0}) = d_i > \deg(h_{i,0}),\ i = 1, \dots, s; \deg(p) = d.$$

Let all roots of the polynomial p lie in the disc $D(0,1)$. Furthermore, let

$$|p|\delta_0 = |p - f_{1,0}\cdots f_{s,0}| \le \min\{2^{-9d}/(sh)^2, 2^{-4d}/(2sh^2)^2\},$$
$$\sigma_0 = |1 - f_0 h_{1,0}/f_{1,0} - \ldots - f_0 h_{s,0}/f_{s,0}| \le \min\{2^{-4.5d}, 2^{-2d}/h\},$$
$$f^{(0)} = \prod_{j=1}^{s} f_{j,0} \text{ and } h = \max_{i=1,\ldots,s} |h_i|$$

(see (46)). Let

$$l = l(d_1,\ldots,d_s) = \sum_{i=1}^{s} \frac{d_i}{d}\log(\frac{d_i}{d})$$

(which implies that $l \le \log_2(d)$ for all choices of $s, d_1, d_2, \ldots, d_s$ and that $l = O(1)$ for $s = 2$ and all choices of d_1 and d_2).

Finally let $b \ge 1$, $b_1 \ge 1$, and k in $O(\log(b+b_1))$ be sufficiently large. Then in k steps Algorithm (49) – (51) computes the polynomials $f_{1,k}$, $h_{1,k}$, $\ldots$, $f_{s,k}$, $h_{s,k}$ such that $f_{1,k},\ldots,f_{s,k}$ are monic,

$$\deg(h_{i,k}) < \deg\ (f_{i,k}) = d_i,\ i = 1,\ldots,s,\ \delta_k|p| < 2^{-b}, \text{ and } \sigma_k < 2^{-b_1}.$$

These steps involve $O((dl\log(d))\log(b+b_1))$ arithmetic operations in $O(b+b_1)$-bit precision; they can be performed by using $O(\mu((b+b_1)dl))$ Boolean operations, that is, $O(\mu((b')dl))$ for $b' = b + b_1$. Moreover,

$$\max_{1\le i\le s} |f_{i,k} - f_i| < 2^{3d} M\delta_0|p|$$

where $p = f_1 \cdots f_s$ and $f_{1,k},\ldots,f_{s,k}$ are the computed approximate factors of p.

D.3 Computation of an initial splitting

Given sufficiently close initial approximations $f_{1,0}$,$\ldots$,$f_{s,0}$ to the factors f_1, $\ldots$,f_s of a polynomial p of (1), e.g., computed by means of the algorithms of Sections 5 and 6, we can complete the initialization of Kirrinnis's algorithm by computing the initial polynomials $h_{1,0},\ldots,h_{s,0}$ by applying $s-1$ times the initialization algorithm of [59, Sections 10 and 11].

We can also extend the semi-heuristic alternative algorithm of Section C.3. Namely multiply both sides of equation (48) by f_0 and obtain the equation

$$h_{1,0}u_{1,0} + \cdots + h_{s,0}u_{s,0} = 1 \tag{52}$$

where $u_{i,0} = f_0/f_{i,0}$ for all i.

Rewrite this polynomial equation as a linear system of d equations in the coefficients of the polynomials $h_{1,0},\ldots,h_{s,0}$ and observe that its coefficient matrix has structure of Toeplitz type and has displacement rank at most s (see [46]).

Then again we can apply the algorithm of [71] and the iterative refinement of [26]. Its every iteration is reduced essentially to multiplication of $4s$ circulant and skew-circulant matrices by vectors and therefore to performing $12s$ Fourier transforms (see [46, Examples 4.4.1, 4.4.2 and Theorem 2.6.4]).

PART III: AUXILIARY AND COMPLEMENTARY RESULTS

E Simplified counting, isolation and exclusion tests

E.1 Simplified counting tests

We can use the power sum algorithm as an empirical exclusion test, that is, for testing whether a disc contains no roots of p, which is a basic test in subdivision root-finders. Indeed if there is no roots of p in a disc, then clearly all their power sums vanish, and we conjecture that conversely for all or almost all inputs the disc contain no roots if the power sum s_0 vanishes.

If the latter conjecture is true, we could substantially decrease working precision in the known counting tests because we would just need to compute the integer s_0 with an error smaller than 1/2. So far, however, we can only prove the conjecture where we deal with a reasonably well-isolated disc.

E.2 Fast estimation of the distances to the roots

We can readily verify the following sufficient conditions that a polynomial p has a root in a fixed disc on complex on the complex plane, although it is not clear how large is the class of polynomials for which this condition helps.

Bini and Fiorentino in [5, Theorems 8–10, 13, and 14] and Bini and Robol in [10, Sections 3.1 and 3.2] estimate the distances to the roots from a point ξ by computing the values $p^{(k)}(\xi)$ for some fixed k, $1 \leq k \leq d-1$, and applying the following well-known bound (cf. [45, Remark 6.4], [5, Theorem 9], [25], [14]).

Theorem 50. *For a polynomial p of (1), a complex point ξ, write*

$$r_k := \Big(k! \binom{d}{k} \Big| \frac{p(\xi)}{p^{(k)}(\xi)} \Big| \Big)^{1/k}, \text{ for } k = 1, \dots, d.$$

Then each disc $D(\xi, r_k) = \{z: \ |z - \xi| \leq r_k\}$ contains a root of p. In particular for $k = 1$ this is the disc $D(\xi, d\Big|\frac{p(\xi)}{p'(\xi)}\Big|)$.

The above bounds are computed very fast but can be too crude to be useful.

The next simple but apparently novel application of a result by Coppersmith and Neff in [15] leads to a lower bound on the distance to the roots.

Theorem 51. *(*See Coppersmith and Neff [15].*) For any integer k satisfying $0 < k < n$, for every disc $D(X, r)$ containing at least $k+1$ zeros of a polynomial p of degree d, and for any $s \geq 3$ if $k = d-1$ and any $s \geq 2 + 1/\sin(\pi/(d-k))$ if $k < d-1$, the disc $D(X, (s-2)r)$ contains a zero of $p^{(k)}(x)$, the kth order derivative of $p(x)$.*

We can count the roots of p in $D(X, r)$ by combining this result with a proximity test for $p^{(k)}(x)$. This can be simpler than performing a proximity test for p if some roots of p but not any roots of $p^{(k)}(x)$ lie close to the boundary circle $C(X, r)$. Now, if a test shows that no roots of $p^{(k)}(x)$ lie in the disc

$D(X,(s-2)r)$ (which is the desired outcome), then we would learn that the disc $D(X,r)$ contains at most k roots of $p(x)$. Hence we learn that the polynomial $p(x)$ has no roots in the disc $D(X,r)$ if our test detects that the polynomial $u^{(k)}(x)$ for $u(x)=(x-X)^k p(x))$ has no roots in the disc $D(X,(s-2)r)$.

F Fast multi-point polynomial evaluation in polynomial root-finders

Every Weierstrass's and every Ehrlich's iteration perform multi-point polynomial evaluation, but so do various other polynomial root-finders as well. E.g., multi-point polynomial evaluation is involved in a typical proximity test applied in order to estimate the distance to a nearest root of p from the center of every suspect square processed in each subdivision iteration. The same can be said about counting the numbers of roots of p in each of the current suspect squares, and in counting algorithms of Remark 15 it is sufficient to perform multipoint evaluation with low precision. Likewise multi-point evaluation of a polynomial p is performed in DLS deflation and in Newton's iterations initialized at a large number of points in order to approximate all roots of p (cf. [63]).

Moenck and Borodin in [37] evaluate p at s points where $d=O(s)$ by using $O(s\log^2(d))$ arithmetic operations. The Boolean cost of performing their algorithm with high precision is nearly optimal [32], [55], but the algorithm fails numerically, with double precision for $d>50$.

The algorithms of [48] and [50] for multipoint evaluation are numerically stable although are only efficient where the relative output error norms are in $O(1/d^c)$ for a constant c. So they are of no value in the case of ill-conditioned roots but could be of interest for the approximation of well-conditioned roots.

Acknowledgements: This research has been supported by NSF Grants CCF–1563942 and CCF–1733834 and PSC CUNY Award 69813 00 48.

References

[1] Bell, E.T.: The Development of Mathematics. McGraw-Hill, New York (1940) doi: 10.2307/2268176

[2] Boyer, C.A.: A History of Mathematics. Wiley, New York (1991) doi: 10.1177/027046769201200316

[3] Barnett, S.: Polynomial and Linear Control Systems. Marcel Dekker, New York (1983) doi: 10.1112/blms/16.3.331

[4] Bini, D.A.: Parallel Solution of Certain Toeplitz Linear Systems. SIAM J. Comput. **13**(2), 268–279 (1984) doi: 10.1137/0213019

[5] Bini, D.A., Fiorentino, G.: Design, analysis, and implementation of a multiprecision polynomial rootfinder. Numer. Algorithms **23**, 127–173 (2000) doi: 10.1023/A:1019199917103

[6] Bini, D.A., Gemignani, L., Pan, V.Y.: Inverse power and Durand/Kerner iteration for univariate polynomial root-finding. Comput. Math. Appl. **47**(2/3), 447–459 (2004)

doi: 10.1016/S0898-1221(04)90037-5

[7] Box, G.E.P., Jenkins, G.M.: Time Series Analysis: Forecasting and Control. Holden-Day, San Francisco, California, fifth edition (2015) doi: 10.1111/jtsa.12194

[8] Bini, D., Pan, V.Y.: Computing matrix eigenvalues and polynomial zeros where the output is real. SIAM J. Comput. **27**(4), 1099–1115 (1998). Proc. version in: SODA'91, 384–393. ACM Press, NY, and SIAM Publ., Philadelphia (1991) doi: 10.1137/S0097539790182482

[9] Bini, D., Pan, V.Y.: Graeffe's, Chebyshev, and Cardinal's processes for splitting a polynomial into factors. J. Complex. **12**, 492–511 (1996) doi: 10.1006/jcom.1996.0030

[10] Bini, D.A., Robol, L.: Solving secular and polynomial equations: a multiprecision algorithm. J. Comput. Appl. Math. **272**, 276–292 (2014) doi: 10.1016/j.cam.2013.04.037

[11] Becker, R., Sagraloff, M., Sharma, V., Xu, J., Yap, C.: Complexity analysis of root clustering for a complex polynomial. In: Intern. Symp. Symb. Algebraic Computation (ISSAC 2016), pp. 71–78. ACM Press, New York (2016) doi: 10.1145/2930889.2930939

[12] Becker, R., Sagraloff, M., Sharma, V., Yap, C.: A near-optimal subdivision algorithm for complex root isolation based on the Pellet test and Newton iteration. J. Symb. Comput. **86**, 51–96 (2018) doi: 10.1016/j.jsc.2017.03.009

[13] Ben-Or, M., Tiwari, P.: Simple algorithms for approximating all roots of a polynomial with real roots. J. Complexity **6**(4), 417–442 (1990) doi: 10.1016/0885-064X(90)90032-9

[14] Carstensen, C.: Inclusion of the roots of a polynomial based on Gerschgörin's theorem, Numerische Math., **59**, 349–360, 1991.

doi: 10.1007/BF01385785

[15] Coppersmith, D., Neff, C. A.: Roots of a polynomial and its derivatives, Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94), 271–279, ACM Press, New York, and SIAM Publications, Philadelphia, 1994.

[16] Du, Q., Jin, M., Li, T.Y., Zeng, Z.: The quasi-Laguerre iteration. Math. Comput. **66**(217), 345–361 (1997) doi: 10.1090/S0025-5718-97-00786-2

[17] Delves, L.M., Lyness, J.N.: A numerical method for locating the zeros of an analytic function. Math. Comput. **21**, 543–560 (1967). doi: 10.1090/S0025-5718-1967-0228165-4

[18] Demeure, C.J., Mullis, C.T.: The Euclid algorithm and the fast computation of cross-covariance and autocovariance sequences. IEEE Trans. Acoust, Speech, Signal Process. **37**, 545–552 (1989)

doi: 10.1109/29.17535

[19] Demeure, C.J., Mullis, C.T.: A Newton–Raphson Method for moving-average spectral factorization using the Euclid algorithm. IEEE Trans. Acoust, Speech, Signal Processing **38**, 1697–1709 (1990)

doi: 10.1109/29.60101

[20] Ehrlich, L.W.: A modified Newton method for polynomials. Communs ACM **10**, 107–108 (1967) doi: 10.1145/363067.363115

[21] Emiris, I.Z., Pan, V.Y., Tsigaridas, E.: Algebraic Algorithms. In: Tucker, A.B., Gonzales, T., Diaz-Herrera, J.L. (eds.) Computing Handbook (3rd edition), Vol. **I**: Computer Science and Software Engineering, Ch. 10, pp. 10-1 – 10-40. Taylor and Francis Group (2014)

[22] Fortune, S.: An iterated eigenvalue algorithm for approximating roots of univariate polynomials. In: J. of Symbolic Computation, **33, 5**, 627–646 (2002) doi: 10.1006/jsco.2002.0526. Proc. version in Proc. Intern. Symp. on Symbolic and Algebraic Computation (ISSAC'01), 121–128, ACM Press, New York (2001)

[23] Fischer, M. J., Paterson, M. S.: String matching and other products, SIAM-AMS Proceedings, **7**, 113–125, 1974.

[24] Householder, A.S.: Dandelin, Lobachevskii, or Graeffe? Amer. Math. Monthly **66**, 464–466. (1959) doi: 10.2307/2310626

[25] Henrici, P.: Applied and Computational Complex Analysis, Vol. 1: Power Series, Integration, Conformal Mapping, Location of Zeros. Wiley, New York (1974)

[26] N. J. Higham, Accuracy and Stability in Numerical Analysis, SIAM, Philadelphia, 2002 (second edition).

https://doi.org/10.1137/1.9780898718027

[27] Henrici, P., Gargantini, I.: Uniformly convergent algorithms for the simultaneous approximation of all zeros of a polynomial. In: Dejon, B., Henrici, P. (eds.) Constructive Aspects of the Fundamental Theorem of Algebra. Wiley, New York (1969)

[28] Harvey, D., van der Hoeven, J.: Integer multiplication in time $O(n \log n)$, 2019. hal-02070778.

[29] Inbach, R., Pan, V.Y.: Polynomial root clustering and explicit deflation, arXiv:1906.04920 (2019)

[30] Inbach, R., Pan, V.Y., Yap, C.: Implementation of a near-optimal complex root clustering algorithm. In: Proc. Int. Congress on Math Software (ICMS 2018), 235–244 (2018) doi: 10.1007/978-3-319-96418-8 28

[31] Inbach, R., Pan, V.Y., Yap, C., Kotsireas, I.S., Zaderman, V.: Root-finding with implicit deflation. In: Proc. CASC (2019) (to appear) arXiv:1606.01396, submitted on 21 May (2019)

[32] Kirrinnis, P.: Polynomial factorization and partial fraction decomposition by simultaneous Newton's iteration. J. Complex. **14**, 378–444 (1998) doi: 10.1006/jcom.1998.0481

[33] Kirrinnis, P.: Fast computation of contour integrals of rational functions, Journal of Complexity, **16, 1**, 181–212, 2000.

doi: 10.1006/jcom.1999.0531

[34] Kobel, A., Rouillier, F., Sagraloff, M.: Computing real roots of real polynomials ... and now for real! In: Intern. Symp. Symb. Algebraic Computation (ISSAC 2016), pp. 301 – 310. ACM Press, New York (2016) doi: 10.1145/2930889.2930937

[35] Mignotte, M.: An inequality about factors of polynomials, Math. of Computation, **28**, 1153–1157, 1974. doi: 10.2307/2005373

[36] McNamee, J.M.: Numerical Methods for Roots of Polynomials, Part I, XIX+354 pages. Elsevier (2007) ISBN: 044452729X

ISBN13: 9780444527295

[37] Moenck, R., Borodin, A.: Fast modular transforms via division., In: Proc. 13th Annual Symposium on Switching and Automata Theory (SWAT 1972), 90–96, IEEE Computer Society Press (1972) doi: 10.1109/SWAT.1972.5

[38] McNamee, J.M., Pan, V.Y.: Numerical Methods for Roots of Polynomials, Part II, XXI+728 pages. Elsevier (2013) ISBN: 9780444527301

[39] Neff, C.A., Reif, J.H.: An $o(n^{1+\epsilon})$ algorithm for the complex root problem. In: Proc. 35th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS '94), pp. 540–547. IEEE Computer Society Press (1994) doi: 10.1109/SFCS.1994.365737

[40] Pan, V. Y.: How to Multiply Matrices Faster, Lecture Notes in Computer Science, vol. 179 (XI + 212 pages), Springer, Berlin, 1984. ISBN:3-387-13866-8

[41] Pan, V. Y.: On Computations with Dense Structured Matrices, Math. of Computation, **55, 191**, 179–190, 1990. Proceedings version in Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC'89), 34–42, ACM Press, New York, 1989. doi: 10.2307/2008798

[42] Pan, V.Y.: Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros. In: Proc. 27th Ann. ACM Symp. on Theory of Computing (STOC'95), 741–750. ACM Press, New York (1995) doi: 10.1145/225058.225292

[43] Pan, V.Y.: Solving a polynomial equation: Some history and recent progress. SIAM Rev **39**(2), 187–220 (1997)

doi: 10.1137/S0036144595288554

[44] Pan, V.Y.: Solving polynomials with computers. Am. Sci. **86**, 62–69 (1998) doi: 10.1511/1998.1.62

[45] Pan, V.Y.: Approximation of complex polynomial zeros: modified quadtree (Weyl's) construction and improved Newton's iteration. J. Complexity **16**(1), 213–264 (2000) doi: 10.1006/jcom.1999.

[46] Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser/Springer, Boston/New York (2001) doi: 10.1007/978-1-4612-0129-8

[47] Pan, V.Y.: Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. J. Symb. Comput. **33**(5), 701–733 (2002) doi: 10.1006/jsco.2002.0531

[48] Pan, V.Y.: Transformations of Matrix Structures Work Again. Linear Algebra and Its Applications **465**, 1–32 (2015)

doi: 10.1016/j.laa.2014.09.004

[49] Pan, V.Y.: Root-finding with Implicit Deflation. arXiv:1606.01396, submitted on 4 June (2016)

[50] Pan, V.Y.: Fast approximate computations with Cauchy matrices and polynomials, Math. of Computation, **86** 2799-2826 (2017) doi: 10.1090/mcom/3204

[51] Pan, V.Y.: Simple and nearly optimal polynomial root-finding by means of root radii approximation. In: Kotsireas, I.S., Martinez-Moro, E. (eds.) Springer Proc. in Math. and Statistics, Ch. 23: Applications of Computer Algebra **198** AG. Springer International Publishing (2017). Chapter doi: 10.1007/978-3-319-56932-1 23

[52] Pan, V.Y., Sadikou, A., Landowne, E.: Univariate polynomial division with a remainder by means of evaluation and interpolation. In: Proc.

3rd IEEE Symposium on Parallel and Distributed Processing, pp. 212–217. IEEE Computer Society Press, Los Alamitos, California (1991) doi: 10.1109/SPDP.1991.218277

[53] Pan, V.Y., Sadikou, A., Landowne, E.: Polynomial division with a remainder by means of evaluation and interpolation. Inform. Process. Lett. **44**, 149–153 (1992) doi: 10.1016/0020-0190(92)90055-Z

[54] Pan, V.Y., Tsigaridas, E.P.: Nearly optimal refinement of real roots of a univariate polynomial. J. Symb. Comput. **74**, 181–204 (2016) doi: 10.1016/j.jsc.2015.06.009. Proceedings version in: Kauers, M. (ed.) Proc. ISSAC 2013, pp. 299–306. ACM Press, New York (2013)

[55] Pan, V.Y., Tsigaridas, E.P.: Nearly optimal computations with structured matrices. Theor. Comput. Sci. **681**, 117–137 (2017) doi: 10.1016/j.tcs.2017.03.030.

[56] Pan, V.Y., Zhao, L.: Real root isolation by means of root radii approximation. In: Proceedings of the 17th International Workshop on Computer Algebra in Scientific Computing (CASC'2015), (V. P. Gerdt, V. Koepf, and E. V. Vorozhtsov, editors), Lecture Notes in Computer Science, **9301**, 347–358, Springer, Heidelberg, 2015. doi: 10.1007/978-3-319-24021-3 26

[57] Pan, V.Y., Zhao, L.: Real Polynomial root-finding by means of matrix and polynomial iterations. Theor. Comput. Sci. **681**, 101–116 (2017). doi: 10.1016/j.tcs.2017.03.032.

[58] Renegar, J.: On the worst-case arithmetic complexity of approximating zeros of polynomials. J. Complex. **3**(2), 90–113 (1987) doi: 10.1016/0885-064X(87)90022-7

[59] A. Schönhage. The Fundamental Theorem of Algebra in Terms of Computational Complexity. Math. Dept., University of Tübingen, Tübingen, Germany (1982)

[60] Schönhage, A.: Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In: Proc. European Computer Algebra Conference (EUROCAM 1982), pp. 3–15. Computer Algebra (1982) doi: 10.1007/3-540-11607-9 1

[61] Schönhage, A.: Quasi GCD computations. J. Complex. **1**, 118–137 (1985) doi: 10.1016/0885-064X(85)90024-X

[62] Schleicher, D.: private communication.

[63] Schleicher, D., Stoll, R.: Newton's method in practice: finding all roots of polynomials of degree one million efficiently. Theor. Comput. Sci. **681**, 146–166 (2017) doi: 10.1016/j.tcs.2017.03.025

[64] Toom, A. L., The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers, Soviet Mathematics Doklady, **3**, 714–716 (1963)

[65] Tilli, P.: Convergence conditions of some methods for the simultaneous computation of polynomial zeros. In: Calcolo **35**, 3–15 (1998) doi: 10.1007/s100920050005

[66] Van Dooren, P.M.: Some numerical challenges in control theory. In: Linear Algebra for Control Theory, IMA Vol. Math. Appl. **62**, 177 – 189 (1994) doi: 10.1007/978-1-4613-8419-9 12

[67] Weierstrass, K.: Neuer Beweis des Fundamentalsatzes der Algebra. Mathematische Werke, Bd. **III**, 251–269. Mayer und Mueller, Berlin (1903)

[68] Weyl, H.: Randbemerkungen zu Hauptproblemen der Mathematik. II. Fundamentalsatz der Algebra und Grundlagen der Mathematik. Mathematische Zeitschrift **20**, 131–151 (1924)

[69] Wilson, G.T.: Factorization of the covariance generating function of a pure moving-average. SIAM J. Numer. Anal. **6**, 1–7 (1969) doi: 10.1137/0706001

[70] Werner, W.: Some improvements of classical iterative methods for the solution of nonlinear equations. In: Allgower, E.L. et al. (eds.) Numerical Solution of Nonlinear Equations, Proc. Bremen 1980 (L.N.M. 878), 427 – 440. Springer, Berlin (1982) doi: 10.1007/BFb0090691

[71] Xia, J., Xi, Y., Gu, M.: A Superfast Structured Solver for Toeplitz Linear Systems via Randomized Sampling, SIAM J. Matrix Anal. Appl., **33**, 837–858, 2012. DOI:10.1137/110831982

[72] Zaderman, V., Zhao, L.: Counting Roots of a Polynomial in a Convex Compact Region by Means of Winding Number Calculation via Sampling, In *Proceedings of CASC 2019*.