Comput. Math. Applic. Vol. 14, No. 8, pp. 591–622, 1987
Printed in Great Britain.

0097-4943/87 $3.00 + 0.00

SEQUENTIAL AND PARALLEL COMPLEXITY OF APPROXIMATE EVALUATION OF POLYNOMIAL ZEROS

V. PAN
Computer Science Department, SUNY Albany, Albany, NY 12222, U.S.A.

(*Received* 17 *July* 1987)

Communicated by E. Y. Rodin

Abstract—Our new sequential and parallel algorithms establish new record upper bounds on both arithmetic and Boolean complexity of approximating to complex polynomial zeros. $O(n^2 \log b \log n)$ arithmetic operations or $O(n \log n \log (bn))$ parallel steps and $n \log b/\log (bn)$ processors suffice in order to approximate with absolute errors $\leqslant 2^{m-b}$ to all the complex zeros of an nth degree polynomial $p(x)$ whose coefficients have mod $\leqslant 2^m$. If we only need such an approximation to a single zero of $p(x)$, then $O(n \log b \log n)$ arithmetic operations or $O(\log^2 n \log (bn))$ steps and $(n/\log n)\log b/\log (bn)$ processors suffice (which places the latter problem in NC, that is, in the class of problems that can be solved using polylogarithmic parallel time and a polynomial number of processors). Those estimates are reached in computations with $O(bn)$ binary bits where the polynomial has integer coefficients. We also reach the sequential Boolean time bounds $O(bn^3 \log (bn) \log \log(bn))$ for approximating to all the zeros (very minor improvement of the bound announced in 1982 by Schönhage) and $O(bn^2 \log \log n \log(bn) \log \log(bn))$ for approximating to a single zero. Among further implications are the improvements of the known algorithms and complexity estimates for computing matrix eigenvalues, for polynomial factorization over the field of complex numbers and for solving systems of polynomial equations. The computations rely on recursive application of Turan's proximity test of 1968, on its more recent extensions to root radii computations, on contour integration via Fast Fourier transform (FFT) within geometric constructions for search and exclusion, and (for the final minor improvements of the complexity bounds) on the recursive factorization of $p(x)$ over discs on the complex plane via numerical integration and Newton's iterations.'

1. INTRODUCTION

1.1. Some major methods and the arithmetic (algebraic) complexity estimates

In the vast bibliography on the evaluation of complex polynomial zeros, only relatively few works specifically address the issue of the complexity of those computations [1–11]. The complexity estimates in those papers have been obtained via several different algorithms, relying on Newton's iterations and on the various techniques for computing power sums and contour integrals, compare Refs [12–14]. Those three basic approaches and techniques have also been combined together and manipulated with in different ways, including some geometric constructions for search and exclusion on the complex plane (Lehmer, Weyl). In particular Smale [1, 4] and Shub and Smale [7, 8] proved that Newton's method is highly effective in the average case, while Turan's power sum method of 1968 [14–16] and its recent extension [9], turned out to be good and reliable in the worst case.

In Tables 1 and 2 we will trace the history of the progress in estimating the sequential and parallel arithmetic complexity of computing polynomial zeros with absolute error bound 2^{-b}, provided that the polynomial $p(x)$ has been scaled so that all its zeros lie in a unit circle. As our comments to those tables, we note that the algorithms of Refs [9, 15, 16] satisfy a bit stronger requirements of assuring *relative output precision* $\leqslant 2^{-b}$; the algorithms of pioneering paper [1] and of Ref. [4] satisfy a little weaker requirement, that of computing x such that $|p(x)| < 2^{-b}$ and imply the average case, in Ref. [1], or probabilistic, in Ref. [4], estimates, while all other estimates of those two tables are the worst case estimates. We complemented the older algorithms of Ref. [15–17] with modern estimates for the complexity of their main blocks, which are reduced to polynomial multiplications and divisions and to discrete Fourier transforms (hereafter referred to as to DFTs). We also included parallel complexity bounds in the cases where those bounds are implicit but not stated in the original papers. For parallel computations we assume the customary machine model, where

Table 1. Arithmetic complexity of approximating to all the zeros of a polynomial

Paper	Sequential time	Parallel time	Processors
[17]	$O(n^3b)$		
[9]	$O(n^2\log\ n(b+n\ \log\ n))$	$O(n\ \log\ n(b+n\ \log\ n))$	n
[11]	$O(n^2\log\ n(n+\log\ b))$	$O(n\ \log\ n(n+\log\ b))$	n
This paper	$O(n^2\log\ b\ \log\ n)$	$O(n\ \log\ n\ \log(bn))$	$n\ \log\ b/\log(bn)$

Table 2. Arithmetic complexity of approximating to a single zero of a polynomial

Paper	Sequential time	Parallel time	Processors
[15, 16]	$O(bn\ \log\ n)$	$O(b\ \log\ n)$	n
[1, 4]	$O((b+n)n)$	$O((b+n)\log\ n)$	$n/\log\ n$
[11]	$O(n\ \log\ n(n^2+\log\ n\ \log\ b))$	$O(\log\ n(n+\log\ b))$	n^2
This paper	$O(n\ \log\ b\ \log\ n)$	$O(\log^2 n\ \log(bn))$	$n\ \log\ b/(\log\ n\ \log(bn))$

in each step each processor performs at most one arithmetic operation, that is, $+$, $-$, $*$, $\div$, or the evaluation of an Nth root of a positive number for a natural N [18–20]. (We will also include here the comparison of two real numbers.) We estimate the number of processors up to within constant factors, for we may always slow down the computations K times and use K times fewer processors (but $\geqslant 1$).

As can be seen from Tables 1 and 2, our new algorithms of this paper establish new record upper estimates for the parallel and sequential arithmetic complexity of approximating to all the zeros of $p(x)$ and to a single zero of $p(x)$. For comparison, recall the known lower bound $\Omega(n+\log\ b)$ on the sequential arithmetic complexity, valid in both cases of approximating to all the zeros of $p(x)$ and to its single zero. (Surely at least $n/2$ arithmetic operations are needed already in order to process the n input coefficients of $p(x)$; the lower bound $\Omega(\log\ b)$ follows from the results of Refs [11, 21].)

Table 2 shows in particular that parallel evaluation of a single zero of $p(x)$ requires only polylogarithmic parallel time and n or fewer processors, so we added a new problem to NC, that is, to the class of problems that can be solved using polylogarithmic parallel time and a polynomial number of processors, compare Refs [18–20]. The parallel time bound of the last line of Table 1 can be decreased to $(\log\ n)^{O(1)}$ if all the zeros of $p(x)$ are real [10] and/or if $b \leqslant (\log\ n)^{O(1)}$, see Theorem 3.3 and Corollary 5.1.

In Section 12.5 we compare the main techniques used in this paper and in previous important works of Refs [2, 11]. Of course, to deduce our improvements of the known complexity bounds, we had to apply some new tools and constructions, but we would like to emphasize the power of Turan's ingenious result of 1968, which served as the springboard of our work, finally resulted in the improvement of all the record estimates of 1987 for the arithmetic complexity of computing polynomial zeros.

1.2. Precision of computations and the Boolean circuit complexity

Without loss of generality we may assume that the coefficients of $p(x)$ are real. (Otherwise, we may shift to the polynomial $p(x)\bar{p}(x)$, whose coefficients are real; here $\bar{p}(x)$ is obtained from $p(x)$ via complex conjugation of the coefficients.) Furthermore, we may truncate the floating point mantissas (fractions) to a finite number of digits and then turn them into integers by scaling the polynomial. Let those integers lie between -2^m and 2^m. In that case $O(bn)$ bit precision of computations suffices in order to support the arithmetic complexity estimates of Tables 1 and 2, where $b = q + m$, $\epsilon = 2^{-q}$, see Section 11. (The study of the perturbation of polynomial zeros due to variation of their coefficients, see Ref. [12, 2, 22], suggests that the precision of computations must be at least of an order of bn in the worst case; on the other hand, even performing our algorithms with precision $O(b)$ should suffice for many polynomials $p(x)$.)

We may immediately extend our estimates of Tables 1 and 2 to the case of the Boolean circuit complexity model, since $O(t(B))$ Boolean operations or $O(\log^2 B)$ Boolean parallel steps, $t(B)$ processors suffice to perform an arithmetic operation over two integers mod 2^B [23–27], where

$$t(B) = B\ \log\ B\ \log\ \log\ B \tag{1}$$

and since in our case $B = O(bn)$. In particular we arrive at the bound $O(bn^3\log\ n\ \log^2(bn)\ \log\ \log$

(bn)) on the Boolean sequential time required for computing all the zeros of $p(x)$. That bound can be decreased by a factor of $\log n$ using the special algorithms for polynomial arithmetic of [27–30]. Even a slightly better bound, $O(n^3(b + \log n) \log (bn) \log \log (bn))$, was stated in Ref. [2, Sect. 19]. (The worst case arithmetic cost of the numerical integration stages of the algorithm of Ref. [2] roughly n times exceeds the bounds of the last lines of Tables 1 and 2, respectively, but the algorithm of Ref. [2] computes with lower precision in its integration stages.) The proof of the Boolean complexity bound of Ref. [2] is very much involved and has not been completed yet. There seems to be two difficulties with that proof. One difficulty is due to the intention of A. Schönhage to supply various techniques for the study of the asymptotic complexity of arithmetic computations with multiple precision. That study is important theoretically and may lead to the results of practical value. Computing polynomial zeros is a good example where multiple precision is required, although the hard and uncompleted proofs of Ref. [2] would be much simpler and clearer if the methods were first demonstrated in the study of the *arithmetic* complexity of the same main and auxiliary problems. Another difficulty seems to be due to the complications in the algorithm of Ref. [2] at its crucial and hardest stage of approximate recursive splitting of $p(x)$ into pairs of factors over a disc, in particular the complications are where a splitting disc must be found such that its boundary lies far enough from all the polynomial zeros. As a side effect of those complications, the overhead constants, hidden in the "O" notation of the asymptotic estimates of Ref. [2], substantially grow.

In contrast to that, having modified Lehmer's and Weyl's constructions and having incorporated them into our algorithms, we arrived at much simpler proofs, smaller overhead constants, and also decreased the arithmetic cost bounds of Ref. [2] roughly by a factor of n. In Sections 2–11 we avoid the recursive splitting, and thus greatly simplify our presentation for the price of only a minor deterioration of the complexity bounds. In Sections 12.2 and 12.3 we incorporate the splitting in order to arrive at the record estimates. In particular, in the sequential case we reach the Boolean complexity estimates $O(bn^3 \log (bn) \log \log (bn))$ and $O(bn^2 \log \log n \log (bn) \log \log (bn))$ for approximating to all the zeros and to a single zero of $p(x)$, respectively [to be compared with the cited bound of Ref. [2] for all the zeros and with the bound $O((b + n)n^2 \log n \log(bn) \log \log(bn))$ for a single zero implicit in Ref. [2].

1.3. Some advantages in practical implementation

Together with each approximation x_s^* to a zero of $p(x)$, our algorithms output a positive $\epsilon_s < \epsilon$ and the number of the zeros of $p(x)$ lying in the ϵ_s-neighborhood of that approximation and counted with their multiplicities. (In the case of an input polynomial $p(x)$ with integer coefficients, our algorithms isolate its zeros, that is, for each distinct zero of $p(x)$ they compute a disc that contains only that zero. This follows from the known lower bound on the minimum distance between the pairs of the distinct zeros of such a polynomial, see Refs [2, 10, 22].) The clusters of zeros and multiple zeros of $p(x)$ are treated alike. In fact, polynomials with clustered zeros are frequent in applications as numerical approximants to polynomials with multiple zeros; many otherwise effective algorithms fail to converge to the clustered zeros of $p(x)$ or converge very slowly. In contrast to that, our algorithms do not slow down at all where the zeros of $p(x)$ are clustered together. Furthermore, Lehmer's and Weyl's constructions are self-correcting (that is, each their iteration corrects the errors of the previous ones); so are our algorithms. Due to the above properties of our algorithms and to their low computational cost (both asymptotic and with accounting the overhead), they are certainly good candidates for practical implementation, particularly regarding our algorithms for computing all the zeros of $p(x)$. They surely promise to be superior (both in speed and in reliability of handling the clustered zeros) to the currently used and already highly effective algorithms, such as one of Ref. [31]. Furthermore, the arithmetic complexity bounds of the last line of Table 1 seem to be overly pessimistic in the case of many polynomials $p(x)$: deducing our worst case estimates, we pessimistically assume that *all* our recursive subdivisions of the set of zeros of $p(x)$ are highly unbalanced, while such a systematic disbalance is certainly a rather exceptional case in practice.

1.4. Some further applications

Further extensions and applications of the results of this paper include, in particular, fac-

torization of a polynomial over the field of complex numbers via the same algorithms as for computing all its zeros (with more favorable Boolean complexity bounds for factorization, see Remark 11.1 in Section 11) and computing the greatest common divisor (gcd) of two or several univariate polynomials. Numerical treatment of the latter problem can be immediately reduced to numerical evaluation of all the zeros of the input polynomials, although there are superior algorithms for *parallel* evaluation of the gcd of a pair or of a set of any input polynomials [19, 32] and also for its sequential evaluation in the case where the input polynomials have *integer* coefficients [30, Sect. 15; 33].

Another important example of applications: Renegar [34] relies on the elimination theory [35] in order to reduce solving a system of polynomial equations to the evaluation of the zeros of a single polynomial.

Finally, computing the eigenvalues of a matrix can be reduced to computing (a) its characteristic polynomial and then (b) the zeros of that polynomial. That method is rarely used, due to the need for high precision computations at stage (a). In many cases, however, the eigenvalues themselves are sought with high precision; then the above objection to that approach looses its ground (in fact we may exactly compute the characteristic polynomial where the input matrix is filled with integers [36]). In that case we apply the current best algorithms at both stages (a) and (b) (see Refs [36, 37] for a stage (a), this paper and Ref. [2] for stage (b)) and arrive at the record worst case complexity estimates (both arithmetic and Boolean) for computing matrix eigenvalues.

1.5. Contents

We will organize this paper as follows. In Section 2 we introduce some basic definitions. In Section 3 we recall Turan's proximity test and its extension to computing a single zero of $p(x)$ via Lehmer's construction. In Section 4 we will recall (and at some point slightly simplify) Schönhage's algorithms for the root radii computations. In Section 5 we will incorporate Turan's test into Weyl's construction. Both Lehmer–Turan's algorithm of Section 3 and Turan–Weyl's algorithm of Section 5 are quite effective for computing relatively rough approximations to polynomial zeros [within the errors of orders of, say $1/n^{O(n)}$]. In Sections 9 and 10 we will extend those two algorithms to effective approximation to the zeros of $p(x)$ with (arbitrarily) high precision. In Sections 6–8 we will present the auxiliary constructions and algorithms required in Sections 9 and 10. In Section 11 we will estimate the precision of computations that we need to use and the Boolean complexity estimates (to be slightly improved in Section 12.3). In Sections 12.1–12.4 we will examine some alternatives to our algorithms. In particular the alternative of recursive splitting of a polynomial over a disc (Sections 12.2, 12,3) can be used to replace the auxiliary constructions of Sections 7 and 8. In Section 12.5 we will briefly compare the techniques of Refs [2, 11], and our paper. In Section 12.6 we will state some open problems.

2. DEFINITIONS AND AN AUXILIARY ALGORITHM

Definition 2.1

Compare Refs [18–20]. $O_A(t, p)$ will mean t parallel arithmetic steps and p processors. $O_A(T) = O_A(T, 1)$ will denote the sequential arithmetic time, that is, the number of arithmetic operations used. Replacing arithmetic operations by Boolean ones we will arrive at the Boolean model of computations with the notation $O_B(t, p)$, $O_B(T)$.

Hereafter let a polynomial $p(x)$ of degree n be fixed,

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - x_j), \quad p_n \neq 0. \tag{2}$$

Definition 2.2

$D = D(X, R)$ denotes the disc of radius R with center X on the complex plane; $S = S(X, R)$ denotes the square with vertices $X + R + R\sqrt{-1}$, $X - R - R\sqrt{-1}$, $X - R + R\sqrt{-1}$, $X + R - r\sqrt{-1}$. We will write $\rho(S) = \rho(D) = R$.

Remark 2.1

Hereafter in different sections each of the characters $R, X, r, x, Y, X_g, R_g, r_g, Y_g$, etc. may denote *distinct* values; all the rectangles and squares in complex domains have their sides *parallel to the coordinate axes.*

Definition 2.3

$z(U)$ denotes the set of all the zeros of $p(x)$ in a complex domain U. Two complex domains U and V are called *equivalent* if $z(U) = z(V)$. Transformation of a domain U into its equivalent subdomain is called *shrinking* or *contraction.* If U denotes a square, or a disc, we define its *rigidity ratio* r.r. (U), and its isolation ratio, i.r. (U), as follows,

$$\text{r.r.}(U) = \min\ (\rho(U^-)/\rho(U)),$$

$$\text{i.r.}(U) = \max\ (\rho(U^+)/\rho(U)).$$

Here the minimization and maximization are over all the domains U^- and U^+ equivalent to U and such that $U^- \subseteq U$ and $U \subseteq U^+$.

Definition 2.4

$d(U) = \max|x_g - x_h|$ for a complex domain U. Here the maximum is over all the pairs of zeros of $p(x)$ in U.

Proposition 2.1

r.r. $(S) \geqslant d(S)/(2\sqrt{2}\rho(S))$ for a square S; r.r. $(D) \geqslant d(D)/(2\rho(D))$ for a disc D.

Definition 2.5

A complex point X is called an *isolated ϵ-approximation* to a zero x_j of $p(x)$ if the disc $D(X, \epsilon)$ contains x_j and has isolation ratio at least $1 + 1/n$.

Definition 2.6

The number of zeros of a polynimial $p(x)$ in a complex domain U, counted with their *multiplicities*, is called the *index* of $p(x)$ in U and is denoted $i(p(x), U)$.

Definition 2.7

The distances $r_1(X) \geqslant r_2(X) \geqslant \cdots \geqslant r_n(X)$ from point X to the n zeros of $p(x)$ are called the *root radii* of $p(x)$ at X; $r_s(X)$ is called the sth root radius of $p(x)$ at X, and we set $r_s(X) = \infty$ for $s \leqslant 0$, $r_s(X) = 0$ for $s > n$.

Proposition 2.2

$1/r_s(X)$ equals the $(n+1-s)$th root radius at X of the (reverse) polynomial $(x - X)^n p(1/(x - X))$.

Definition 2.8

For fixed positive R and ϵ, denote

$$b = \log_2(R/\epsilon), \tag{3}$$

provided that R upper bounds the moduli of all the zeros of $p(x)$,

$$R \geqslant |x_j| \quad \text{for} \quad j = 1, \ldots n. \tag{4}$$

This is the case, in particular, for

$$R = 2 \max_{h=1,\ldots,n} |p_{n-h}/p_n|^{1/h}, \tag{5}$$

see Ref. [13, Sect. 6.4].

Algorithm 2.1. Superscription of a square about a given set of points

Input. A (finite) set H of complex points.

Computation. Compute the max M and the min m in the set of the real parts of the points of H. Output $(M+m)/2$ and $(M-m)/2$. Then repeat the same computations for the set of the imaginary parts of the points of H.

Proposition 2.3 (compare with the end of Remark 2.1)

The two half-sums in the outputs of Algorithm 2.1 equal the real and imaginary parts of the center of the minimum rectangle containing the set H. The two half-differences equal the half-lengths of the sides of that rectangle. The center x and the half-length r of the longer side define a square $S(x,r)$ containing the set H. Moreover, $r \leqslant \rho(S)$ for any square S containing the set H.

In the sequel we will use the known effective algorithms for some basis operations with polynomials (such as their multiplication and division with a remainder, DFT, scaling and shift of the variable) and for solving a triangular Toeplitz system of equations; the arithmetic cost of those computations is $O_A(\log n, n)$, see Refs [9, 23–25].

3. TURAN'S PROXIMITY TEST AND LEHMER–TURAN'S APPROXIMATION TO A ZERO OF A POLYNOMIAL

In this section we will recall Turan's proximity test, which will enable us to compute a distance from a complex point X to the nearest zero of $p(x)$, so that a relative error at most $5^{1/N}-1$ is assured for the cost $O_A(\log N \log n, n)$. Then we will apply such tests recursively (for $N=32$) to approximate to a single zero of $p(x)$.

Algorithm 3.1 (Turan's proximity test)

Inputs. A degree n polynomial $p(x)$ of equation (2) and a complex number X that is not a zero of $p(x)$.

Stage 1. Choose a natural N and compute the values of the power sums,

$$s_{gN} = \sum_{j=1}^{n} y_j^{gN}, \quad y_j = 1/(X - x_j), \quad g, j = 1, 2, \ldots, n.$$

Stage 2. Compute and output

$$r = \max_{g=1,\ldots,n} |s_{gN}/n|^{1/(gN)} \text{ and } r^* = 5^{1/N} r.$$

Theorem 3.1 [14, p. 299]

For the output r of Algorithm 3.1,

$$1 \leqslant \min_{j=1,\ldots,n} |X - x_j|/r \leqslant 5^{1/N},$$

that is, $i(p(x), D) = 0$, i.r. $(D) \leqslant 5^{1/N}$ where $D = D(X, r)$, compare with Definitions 2.3, 2.6, Remark 3.2 below and Corollary 4.1 in Section 4.

For our purpose it will suffice if $N = 32$; then

$$1.051581 < 5^{1/N} < 1.051582. \tag{6}$$

Turan chooses $N = 2^h$ to be a power of 2 and performs Stage 1 as follows.

Subalgorithm 3.1

Stage (a). Shift the variable $y = x - X$ and compute the coefficients of the nth degree polynomial $q(y)$ with the zeros $y_j = 1/(x_j - X)$, $j = 1, \ldots, n$, such that

$$p(x) = p(X+y) = \sum_{i=0}^{n} p_i(X)\, y^i,$$

$$q(y) = y^n p(X + 1/y) = \sum_{i=0}^{n} p_i(X)\, y^{n-i} = p_0(X) \prod_{j=1}^{n} (y - y_j). \tag{7}$$

Stage (*b*). Let $q_0(y) = q(y)/p_0(X)$ and successively compute the coefficients of the polynomials

$$q_{i+1}(y) = q_i(\sqrt{y})q_i(-\sqrt{y}), \quad i = 0, 1, \ldots, h-1. \tag{8}$$

Iteration i of that algorithm *squares* the zeros of the polynomial $q_i(y)$, so

$$q_h(y) = (-1)^{hn}\prod_{j=1}^{n}(y - y_j^N) = \sum_{j=1}^{n} q_{i,h}y^i, \quad N = 2^h. \tag{9}$$

That algorithm is due to *Dandelin*, which was shortly afterwards rediscovered by *Lobachevsky*, but is commonly known as *Graeffe's* [12, 13].

Stage (*c*). Compute the (shifted inverse) power sums s_{gN} for $g = 1, \ldots, n$ from the following triangular Toeplitz system of *Newton's identities* [12, p. 36]

$$\begin{aligned} q_{n,h}s_N + q_{n-1,h} &= 0, \\ q_{n,h}s_{2N} + q_{n-1,h}s_N + 2q_{n-2,h} &= 0, \\ q_{n,h}s_{nN} + q_{n-1,h}s_{(n-1)N} + \cdots + nq_{0,k} &= 0. \end{aligned} \tag{10}$$

At Stage (a) we just shift the variable x; every iteration i of Stage (b) is a polynomial multiplication (convolution); Stage (c) of solving the triangular Toeplitz system (10) amounts to polynomial division, see Ref. [30]. Thus the overall cost of Algorithm 3.1 is $O_A(1 + h \log n, n)$.

Remark 3.1

Subalgorithm 3.1 can be replaced by numerical integration,

$$s_K = \frac{1}{2\pi i}\int_{y\in\Gamma}(y^K q'(y)/q(y))\,dy, \tag{11}$$

see Ref. [13]. Here Γ denotes the boundary of a region containing all the zeros of $q(y)$. Choosing a circular region we may reduce the numerical integration to DFT, compare Section 12.2 below.

Next we will recall the algorithm of Ref. [14–16], which approximates to a single zero of $p(x)$ by using Algorithm 3.1 within Lehmer's geometric construction [12, pp. 111–112; 38]. We will assume that two sufficiently large integers N and Q have been fixed, say $Q = N = 32$.

Algorithm 3.2

Inputs. Polynomial $p(x)$ of equation (2), a complex X_0, an integer j, and positive r_0^* and $\epsilon < r_0^*$ such that $1 \leqslant r_j(X)/r_0^* \leqslant 5^{1/N}$, $1 \leqslant j \leqslant n$, where $r_j(X_0)$ denotes the jth root radius of $p(x)$ at X_0, see Definition 2.7.

Stage g, $g = 0, 1, \ldots$. Apply Algorithm 3.1 at $X = Y_i$ for $i = 0, 1, \ldots, Q-1$, where $Y_i = X_g + r_g^*\omega^i$, ω denotes a primitive Q-root of 1, $\omega^Q = 1$, $\omega^i \neq 1$ if $0 < i < Q$. Choose i where the output r^* of Algorithm 3.1 takes minimum value; let r_{g+1}^* denote that output and let $X_{g+1} = Y_i$. If $r_{g+1}^* < \epsilon$, output X_{g+1} and r_{g+1}^*. Otherwise enter Stage $g+1$.

Theorem 3.1 implies that

$$1 \leqslant r_{g+1}^*/r_n(X_{g+1}) \leqslant 5^{1/N}, \quad r_{g+1}^* < ar_g^* \text{ for } g = 0, 1, \ldots, \tag{12}$$

where $a = a(N, Q)$ converges to 0 as N and Q converge to ∞; $a(32, 32) < 0.2$.

Apply expression (12) recursively for $g = 0, 1, \ldots$ and arrive at the following result.

Theorem 3.2 [14–16]

For a positive ϵ, a single zero of a polynomial $p(x)$ can be evaluated with absolute error $< \epsilon$ for the cost $O_A(b^* \log n, n)$, where

$$b^* = \log_2(r/\epsilon),\ r = r_n(X) = \min_{j=1,\ldots,n}|X - x_j|,$$

and X denotes an initial approximation to a zero of $p(x)$; $r \geqslant \epsilon$, and an upper bound on r is given by equation (18), see below.

Remark 3.2

Turan [14–16] uses his algorithm also in order to approximate to

$$r_1(X) = \max_{j=1,\dots,n} |X - x_j|,$$

the first root radius of $p(x)$ at X, recall Definition 2.7. Proposition 2.2 implies immediate reduction of computing $r_1(X)$ to computing $r_n(X)$ and vice versa, but Turan just presents two dual algorithms for those two dual problems. In particular, here are Turan's inequalities dual to ones of Theorem 3.1,

$$1 \leqslant |r_1(x)| / \max_{g=1,\dots,n} |s^*_{gN}/n| \leqslant 5^{1/N}, \quad \text{where } s^*_K \text{ denotes } \sum_{j=1}^{n} x_j^K.$$

In fact in Algorithm 3.2 he chooses $j = 1$ at Stage 0, but the choice $j = n$ seems to be a little better, for $r_n(X) < r_1(X)$ as a rule.

Remark 3.3

In Section 6 of Ref. [9] Turan's algorithm is extended to computing all the zeros of $p(x)$ with the error bound $\epsilon = 2^{-b}$ for the cost $O_A(n \log n(n \log n + b), n)$. In that extension, when a zero $y_j = y_{j(1)}$ of $q(y)$ becomes available, the same algorithm of Ref. [14] is applied again [but with s_{gN} replaced by $s_{gN} - y^{gN}_{j(1)}$ throughout]. The process is recursively repeated until all the n zeros are computed.

We will conclude this section using iterations (8) in order to obtain the following result.

Theorem 3.3

Let all the zeros of $p_n(\lambda)$ be real. Then they can be computed with relative output error bound $\epsilon = 2^{-b}$ for the cost $O_A((b + \log n) \log n, n)$.

Proof. Deduce from equations (9) that

$$|q_{n-g,h}| = \sum_{j(1),\dots,j(g)} |y_{j(1)} y_{j(2)} \dots y_{j(g)}|^N, N = 2^h, \ h = 1, 2, \dots, n. \tag{13}$$

The latter sum is over all the sets $\{j(1), j(2), \dots, j(h)\}$ of distinct values in the range from 1 to n. For even $N = 2^h$, equation (13) implies that

$$1 \leqslant |q_{n-g,h}/(y_1 y_2 \dots y_g)^N| \leqslant n!/(g!(n-g)!) \leqslant 2^n, \quad g = 1, \dots, n.$$

Therefore, $O(b + \log n)$ iterations (8) suffice in order to compute the products $|y_1 y_2 \dots y_g|$ for $g = 1, \dots, n$ with relative errors $\leqslant \epsilon/2 = 2^{-1-b}$, and then the absolute values $|y_1|, \dots, |y_n|$ with relative errors $\leqslant \epsilon = 2^{-b}$, remaining within the complexity bounds of Theorem 3.3. Similarly we compute the absolute values of the zeros $z_g = y_h - \Delta$ of the polynomial $s(z) = q(z + \Delta)$ (where, say Δ is close to $|y_n|/2$) and finally recover the values $y_g = \pm |y_g|$ from $|y_g|$ and $|y_g - \Delta| = |z_g|$, $g = 1, \dots, n$. This proves Theorem 3.3. Q.E.D.

4. ROOT RADII COMPUTATIONS

In Sections 7 and 9 (and also in Remark 5.2 below) we will apply some extensions of Theorem 3.1. We will derive them by following and slightly simplifying Ref. [2, Sect. 14]. In this section we will assume that $X = 0$ (otherwise we will shift the variable letting $y = x - X$) and will denote $r_s = r_s(X)$ (compare Definition 2.7),

$$r_0 = \infty, \quad r_{n+1} = 0. \tag{14}$$

Consider the two following tasks.

Task r. Given positive r and Δ, find a (generally nonunique) integer s such that $r_{s+1}/(1 + \Delta) < r < (1 + \Delta) r_s$.

Task s. Given an integer s, $1 \leqslant s \leqslant n$, and a positive Δ, find a positive r such that $r/(1 + \Delta) < r_s < (1 + \Delta) r$.

We will solve Tasks r and s for $1 + \Delta = 2n$, with immediate extension to arbitrary positive Δ via

$$g = g(\Delta) = \lceil \log_2(\log(2n)/\log(1 + \Delta)) \rfloor \tag{15}$$

iteration (8); indeed, such an iteration amounts to squaring $1+\Delta$ in the context of Tasks r and s. The cost of those iterations should be added to the overall cost of the solution, of course. Note that

$$g(\Delta)=0 \text{ if } 1+\Delta \geqslant 2n;\ g(\Delta)=O(\log\log n) \text{ if } 1/\Delta=O(1), \tag{16}$$

$$g(\Delta)=O(\log n) \text{ if } 1/\Delta \leqslant n^{O(1)}. \tag{17}$$

Frequently throughout this paper we will need to solve Task s for $s=n$; we will use the following corollary from Theorem 3.1, compare Remark 3.2.

Corollary 4.1

(a) For $s=1$ and $s=n$ Task s can be solved via application of Theorem 3.1 for the cost $O_A((1+g)\log n, n)$, where g is defined by expressions (15–17). (b) Moreover, the cost decreases to $O_A(\log n, n)$ if $1/\Delta=O(1)$.

Reference [2] indicates an alternative (slightly inferior) way, based on the following well known inequalities (whose derivation is simpler than the proof of Theorem 3.1, compare Ref. [14] with Ref. [13, pp. 451, 452, 457]):

$$t_1^*/n \leqslant r_1 < 2t_1^*, \quad t_1^*=\max_{h>0}|p_{n-h}/p_n|^{1/h}.$$

Here p_i denote the coefficients of $p(x)$, compare equations (2) and (5). Apply Proposition 2.2 and extend those bounds as follows:

$$t_n^*/2 < r_n \leqslant nt_n^*, \quad t_n^*=\min_{h>0}|p_0/p_h|^{1/h}. \tag{18}$$

Therefore, $r=r_1^*=t_1^*\sqrt{2/n}$ is a solution to Task s for $s=1$, while $r=r_n^*=t_n^*\sqrt{n/2}$ is a solution to Task s for $s=n$, where in both cases $1+\Delta=\sqrt{2n}$. This can be extended to the solution of Task s for $s=1$ and $s=n$ with arbitrary $1+\Delta>1$ for the cost $O_A(g\log n, n)$ of g iterations (8), where g is defined by expressions (15)–(17). That cost bound is the same as in part (a) of Corollary 4.1.

In this paper, apart from Task s for $s=n$, we will also need to solve Task s for $s<n$ in Section 7 and Task r in Remark 5.2 of the next section and in Section 10. Next we will supply solutions in those cases, relying on the following useful and elegant result.

Theorem 4.1 [2]

If $1 \leqslant m \leqslant n$ and if $|p_{n+1-m-h}/p_{n+1-m}| \leqslant av^h$ for $h=1,\ldots,n+1-m$, then $r_m < m(a+1)v$.

Proof. See Ref. [2]. Due to VanVleck's theorem, [13, p. 459]

$$|p_{n+1-m}|r_m^{n+1-m} \leqslant \binom{n}{n+1-m}|p_0|+\binom{n-1}{n-m}|p_1|r_m+\cdots+\binom{m}{1}|p_{n-m}|r_m^{n-m}.$$

Divide both sides by $|p_{n+1-m}|r^{n+1-m}$, apply the assumed bound on the ratios $|p_{n+1-m-h}/p_{n-m}|$, and deduce for $x=v/r_m$ that

$$1 \leqslant ax^{n+1-m}\binom{n}{m-1}+ax^{n-m}\binom{n-1}{m-1}+\ldots+ax^2\binom{m+1}{m-1}+ax\binom{m}{m-1}. \tag{19}$$

If $x \geqslant 1$, then $r_m \leqslant v$, and Theorem 4.1 follows. Otherwise expression (19) implies that

$$1+a \leqslant a/(1-x)^m, \quad 1/x < (a+1)^d/((a+1)^d-a^d), \quad d=1/m,$$

so $r_m/v=1/x<(a+1)/d$, and then again Theorem 4.1 follows. Q.E.D.

Theorem 4.1 and Proposition 2.2 also imply a similar upper bound on $1/r_m$, which is the $(n+1-m)$th root radius of the *reverse polynomial* $x^n p(1/x)$, having the same coefficients as $p(x)$, but appearing in the reverse order.

Theorem 4.1 immediately implies the solution of Task r for $r=1$ and $1+\Delta=2n$. Indeed, compute m such that

$$|p_{n+1-m}|=\max_{0\leqslant i\leqslant n}|p_i|.$$

If $m = n + 1$ then $t_n^* \geqslant 1$, $r_n > 1/2$, $r_{n+1} = 0$, see expressions (14) and (18), so $s = n$ is a desired solution to Task r. Otherwise $1 \leqslant m \leqslant n$. Then applications of Theorem 4.1 with $a = v = 1$ to $p(x)$ and to $x^n p(1/x)$ yield that $1/(2(n + 1 - m)) < r_m < 2m$, so $1/(2n) < r_m < 2n$, and $s = m - 1$ is a solution to Task r, where $r = 1$ and $1 + \Delta = 2n$ (take into account equations (14), where $m = 1$, $s = 0$). The extensions to arbitrary r is via scaling the variable x and to arbitrary Δ via iterations (8), see above. We arrive at Proposition 4.1.

Proposition 4.1

Task r can be solved for the cost $O_A((1 + g) \log n, n)$, where g is defined by expressions (15)–(17); the cost can be decreased to $O_A(\log n, n/\log n)$ if $1 + \Delta \geqslant 2n$.

We could solve Task s by recursively applying Proposition 4.1 in a binary search algorithm, but we will prefer a more direct algorithm outlined in Ref. [2]. In its description we will use the following definitions;

$$y = x/\rho, \quad q(y) = \gamma p(\rho y) = \sum_{u=0}^{n} q_u y^u, \quad q_u = 0 \quad \text{if} \quad u < 0 \text{ or } u > n. \tag{20}$$

Algorithm 4.1

Choose the scaling factors γ and ρ in relations (20) and two integers t and h such that

$$q_t = 1 \leqslant q_{t+h} = \xi^h < 2^h, \quad t < n + 1 - s \leqslant t + h, \tag{21}$$

and the following *convexity property* holds: in the plane $\{(u, w)\}$ for none u the point $(u, w(u))$ lies above the straight line passing through the two points $(t, w(t))$ and $(t + h, w(t + h))$, where $w(u)$ denotes $\log_2 |q_u|$. Compute and output $r = \rho/\xi$.

The relations (21) and the above convexity property imply that

$$q_{t+g}/q_t \leqslant \xi, \quad q_{t+h-g}/q_{t+h} \leqslant 1/\xi^g \quad \text{for all positive } g. \tag{22}$$

In the sequel we will use expression (22) in order to prove that the output r of Algorithm 4.1 is a solution to Task s for $1 + \Delta = 2n$, but at first we will specify the computations in Algorithm 4.1.

Note that scaling relation (20) turns the coefficients p_u of $p(x)$ into the coefficients

$$q_u = \gamma p_u/\rho^u \tag{23}$$

of $q(y)$, so $\log_2 q_u - \log_2 p_u = \log_2 \gamma - u \log_2 \rho$ for all u; consequently, the convexity property required in Algorithm 4.1 is invariant in scaling relation (20). Therefore, Algorithm 4.1 can be performed as follows.

Algorithm 4.2

Compute the convex hull H of the finite set $\{(u, \log_2|p_u|), u = 0,1, \ldots, n\}$ on the plane, find the edge of the boundary of H whose orthogonal projection onto the u-axe is an interval including the point $n + 1 - s$. Choose t and $t + h$ to be the endpoints of that interval. Choose ρ and γ such that expression (21) holds for the latter choice of t and h, that is, such that

$$\rho^{t+h} = \gamma p_t \rho^h \leqslant \gamma p_{t+h} = \xi^h \rho^{t+h} < 2^h \rho^{t+h}.$$

[Here we scaled expression (21) by ρ^{t+h} and applied equation (23).] Finally compute and output $r = \xi/\rho$ where $\xi = q_{t+h}^{1/h} = (\gamma p_{t+h}/\rho^{t+h})^{1/h}$, see expressions (21) and (23).

To prove that $r = \xi/\rho$ is indeed a solution to Task s for $1 + \Delta = 2n$, let $r_j^* = r_j/\rho$ for all j, so r_j^* denotes the jth root radius of $q(y)$ at 0. Due to expressions (22) we may apply Theorem 4.1 to $q(y)$ with $a = 1$, $v = 1/\xi$ and to $y^n q(1/y)$ with $a = 1$, $v = \xi$ and yield the desired bounds

$$(r/\rho)/2n = 1/(2n\xi) < r_{n+1-t}^* \leqslant r_s^* \leqslant r_{n+1-t-h}^* < 2n/\xi = 2nr/\rho, \tag{24}$$

so $r_s/(2n) < r < 2nr_s$.

Proposition 4.2

Task s can be solved for the cost $c_A(CH) + O_A(g \log n, n)$ where g is defined by expressions

(15)–(17) and where $c_A(\mathrm{CH})$ denotes the cost of computing the values $\log_2|p_u|$ for $u = 0, 1, \ldots, n$ and the convex hull of the set $\{(u, \log_2|p_u|), u = 0, 1, \ldots, n\}$ of $n+1$ points on the plane.

Note that we compute the convex hull H of the same set to solve Task s for all s. If we allow CREW PRAM (rather than arithmetic circuits), then we may compute H for the cost $O_A(\log n, n)$ [39–41] although the overhead constants are very large. Ref. [2] suggests using Newton's diagram to reach the cost $O_A(n \log n)$.

In the applications of Section 7, however, we only need to solve Task s where $1 + \Delta = 2n$ and where in addition we have a disc D such that

$$i(p(x), D) = k = n + 1 - s, \quad \text{i.r.}(D) = (1+\mu)^2 \geqslant 4n^2. \tag{25}$$

In that case we may solve Task s on arithmetic circuits for a lower cost, $O_A(1, n)$ with small overhead.

Proposition 4.3

Task s for $1 + \Delta = 2n$ can be solved for the cost $O_A(1, n)$ if equations (25) hold.

Proof. Equations (25) imply that $r_{s-1}/r_s = r^*_{s-1}/r^*_s \geqslant 4n^2$. On the other hand, the last two inequalities of relation (21) imply that $r^*_{n+1-t-h} \geqslant r^*_s \geqslant r^*_{n+1-t}$, so either $t = n - s + 2 - h$ or otherwise $r^*_{n+1-t-h}/r^*_{n+1-t} \geqslant r^*_{s-1}/r^*_s \geqslant 4n^2$. The latter inequalities contradict relation (24), so $t = n + 2 - h - s$ if equations (25) hold. Then it remains to choose h such that the convexity property of Algorithm 4.1 holds. This can be done for the cost $O_A(1, n)$, and *we will avoid computing the convex hull.* Indeed, the convexity property is invariant in scaling relation (20), so we may deal with the coefficients p_u of $p(x)$ [rather than with the coefficients q_u of $q(y)$] and may compute h as a positive integer maximizing the value $(p_{t+h}/p_t)^{1/h}$. This proves Proposition 4.3.

5. TURAN–WEYL'S EXCLUSION ALGORITHM, COMPUTING THE NUMBER OF POLYNOMIAL ZEROS IN A DISC, AND AN OUTLINE OF A FURTHER IMPROVEMENT

In this section we will complement Weyl's exclusion algorithm [13, pp. 517–521] by Turan's proximity test, so that for a prescribed positive ϵ our next algorithm will compute isolated ϵ_s-approximations to all the zeros of $p(x)$ for some $\epsilon_s < \epsilon$, see Definition 2.5. Then we will devise Algorithm 5.2 in order to compute the indices of $p(x)$ in the ϵ_s-neighborhoods of the computed approximations. The resulting cost bounds will be quite satisfactory for larger ϵ and will be improved for smaller ϵ in Sections 9 and 10 (using the auxiliary results of Sections 6–8), see an outline of that improvement at the end of this section.

Algorithm 5.1 (Turan–Weyl)

Inputs. Polynomial $p(x)$ of equation (2), positive integers J, k and N, complex X and positive R such that the square $S(X, R)$ contains at most k distinct zeros of $p(x)$ and has isolation ratio $\geqslant \sqrt{2}$. In Stage 0 call the square $S(X, R)$ suspect.

Stage j, j = 0,1, ..., J. Subdivide each suspect square with side length $R/2^{j-1}$ into four squares with side length $R/2^j$ and apply Algorithm 3.1 at their centers. Call the tested square *suspect* unless Algorithm 3.1 outputs $r > R/2^{j-0.5}$, that is, unless the test proves that the square contains no zeros of $p(x)$. In Stage J output the centers of all the suspect squares having side length $R/2^J$.

Proposition 5.1

Iteration j of Algorithm 5.1 has cost $O_A(\log n, kn)$ and defines at most $4k$ suspect squares with side length $R/2^j$. The centers of those squares approximate to all the zeros of $p(x)$ in $S(X, R)$ with errors $\leqslant R/2^{j-0.5}$, and the center of every suspect square lies at the distance at most $(R/2^{j-0.5})\, 5^{1/N}$ from some zero of $p(x)$.

Proposition 5.1 immediately follows from Theorem 3.1.

Theorem 5.1

Let Algorithm 5.1 be applied, let ϵ be a positive constant, $b = \log_2(R/\epsilon)$, compare with equation (3). Then the center X_s of every suspect square output by iteration $J = \lceil b + \log_2 n \rceil + 5$ of

Algorithm 5.1 is an isolated ϵ_s-approximation to a zero of $p(x)$ for $\epsilon_s \leqslant \epsilon$. ϵ_s for all the suspect squares can be computed for the cost $O_A(\log k, k^2)$.

Corollary 5.1

Let positive b, R and ϵ satisfy expressions (3) and (4). Then isolated ϵ_s-approximations to all the n zeros of $p(x)$ where $\epsilon_s \leqslant \epsilon$ for all s can be computed for the cost $O_A((b + \log n)\log n, n^2)$.

Proof of Theorem 5.1. Due to Proposition 5.1, each output suspect square $S(X_s, r)$ has center X_s approximating to a zero of $p(x)$ within $r\sqrt{2}\ 5^{1/N}$, and

$$r < \epsilon/((12n + 1)\sqrt{2}). \tag{26}$$

Also other required properties immediately follow, except that it remains to prove the ϵ_s-isolation of X_s with $\epsilon_s \leqslant \epsilon$ and to estimate ϵ_s for each center X_s. Define

$$r_0 = r\sqrt{2}, \quad r_{i+1} = r_i + 3r_0 = (3i + 4)r_0 \quad \text{for} \quad i = 0, 1, 2, \ldots. \tag{27}$$

Fix X_s and successively (for $i = 0, 1, \ldots$) check if there is any output suspect square that does not lie inside the disc $D(X_s, r_i)$ but lies in or intersects the disc $D(X_s, r_i(n + 1)/n)$. Such a square will be called *adjacent* to the disc $D(X_s, r_i)$. By definition, it lies inside the discs $D(X_s, r_j)$ for all $j > i$ and therefore is not adjacent to them. Since there are at most $4k$ output suspect squares, checking step i will give answer "no" for some $i = i(s) \leqslant 4k$. Then $r_i = (3i + 1)r_0 \leqslant (12k + 1)r_0 < \epsilon$, due to expressions (26) and (27), and i.r. $(D(X_s, r_i)) \geqslant 1 + 1/n$, so X_s is a desired isolated ϵ_s-approximation to a zero of $p(x)$ for $\epsilon_s = r_i$. For every fixed s perform all the $O(k)$ checking steps in parallel. Checking step i for each of $O(k)$ output suspect squares amounts to computing the distance from its center to the center of its nearest neighbor among the output suspect squares, so all the $O(k)$ checking steps for all $O(k)$ output suspect squares have overall cost $O_A(\log k, k^2)$. Q.E.D.

Next we will compute the index of $p(x)$ in the ϵ_s-neighborhood of each isolated ϵ_s-approximation to a zero of $p(x)$ and, more generally, in a disc D with isolation ratio $1 + \mu > 1$.

Proposition 5.2

Let i.r. $(D) \geqslant 1 + v > 1$ for a disc $D = D(X, r)$. Then the index $i(p(x), D)$ can be computed for the cost $O_A(h \log n, n)$ where

$$h = 1 + \lceil \log_2(\log 9/\log (1 + v)) \rceil = O(\log (1/v)) \quad \text{as} \quad v \to 0. \tag{28}$$

Proof. See also Remark 5.2 below. The well known *winding number algorithms* (whose cost is $O_A(\log n, n)$) [11, 13, pp. 239–241], compute the index $i(p(x), D)$ of $p(x)$ in a disc $D = D(X, r)$ provided that all the zeros of $p(x)$ lie far enough from the boundary of D or of a fixed disc $D(x, r^*)$ equivalent to D. Reference [11] proves that the bound i.r. $(D) \geqslant 9$ already suffices in order to assure the latter provision. The next algorithm reduces the case of arbitrary disc D with i.r. $(D) > 1$ to the case i.r. $(D) \geqslant 9$.

Algorithm 5.2

Inputs. Complex X, positive r and v, and polynomial $p(x)$ of equation (2) such that i.r. $(D(X, r)) \geqslant 1 + v > 1$, compare with Definition 2.3.

Stage 1. Compute the coefficients of the monic polynomial $q_0(y) = p(X + ry)/(r^n x_n)$.

Stage 2. Apply h iterations (8) where h is defined by equations (28).

Due to the transformation of the variable, Stage 1 of the algorithm transforms the discs $D(X, r)$ and $D(X, (1 + v)r)$ into the discs $D(0, 1)$ and $D(0, 1 + v)$, respectively. Stage 2 transforms the disc $D(0, 1)$ into itself and the disc $D(0, 1 + v)$ into the disc $D(0, (1 + v)^N)$, where $N = 2^h$, $(1 + v)^N \geqslant 9$ due to equations (28), so the isolation ratio of $D(0.1)$ with respect to $q_h(y)$ is at least 9. Therefore, we may apply a winding number algorithm and compute the index $i(q_h(y), D(0, 1)) = i(p(x), D(X, r))$. This proves Proposition 5.2. Q.E.D.

Corollary 5.2

Given an isolated ϵ-approximation X to a zero of $p(x)$, the index of $p(x)$ in the disc $D(X, \epsilon)$ can be computed for the cost $O_A(\log^2 n, n)$.

Proof. Apply Proposition 5.2 with $r = \epsilon$, $v = 1/n$. Q.E.D.

Remark 5.1

The proof of Proposition 5.2 would little change if we set $q_0(y) = p(X + y)/p_n$ in Stage 1 of Algorithm 5.2.

Remark 5.2

Given a disc $D = D(0, \rho)$ such that i.r.$(D) \geqslant (1 + v)^2$, we may compute $s = n + 1 - i(p(x), D)$ by solving Task r of Section 4 for $r = (1 + v)\rho$ and for any $\Delta \leqslant v$. The cost of the solution is $O_A((1 + g) \log n, n)$ where g is defined by expressions (15)–(17), so $g = O(h + \log \log n)$, compare equations (15) with (28). This implies an alternative proof of Proposition 5.2, provided that its cost bound changes respectively. That change would not affect Corollary 5.2.

Finally let us outline our further improvement of Turan–Weyl's algorithm that we will present in Sections 6–9.

We will separately treat two cases.

Case 1. See Section 9. For an input square $S = S(Y, R^*)$, r.r. $(S) > 1/(8\sqrt{2}n^2)$. Then in O(log n) recursive subdivisions of S in Turan–Weyl's algorithm, the set of all the suspect squares will form at least two connected components C_g, included into some squares equivalent to them and having isolation ratios $\geqslant 6$. The index of $p(x)$ in each component will be positive but strictly less than in S. We will repeat that process recursively for each such square unless its rigidity ratio is smaller than $1/(8\sqrt{2}n^2)$. This surely will be the case if $i(p(x), C_g) = 1$, so this will be the case in $\leqslant n - 1$ subdivisions of the square into strongly isolated components.

Case 2. See Sections 7 and 8. r.r.$(S) \leqslant 1/(8\sqrt{2}n^2)$ or equivalently i.r.$(D) \geqslant 8n^2$, where $D = D(Y, R)$, $R = R^*/(8n^2)$. Then we will compute an approximation M^* to the center of gravity $M = s_1/k$ of all the k zeros of $p(x)$ in S using numerical integration (11) along the boundary Γ of the disc D. Then we will apply Proposition 4.3 in order to approximate $r_{n+1-k}(M^*)$, the $(n + 1 - k)$th root radius of $p(x)$ at M^*, equal to the distance from M^* to the farthest zero of $p(x)$ in S. The disc $D(M^*, r_{n+1-k}(M^*))$ is equivalent to S and, as we will prove, has rigidity ratio $> 1/(8n^2)$ (so we will go back to Case 1) unless $r_{n+1-k}(M^*)$ is smaller than, say $|M - M^*|/2$. In the latter case M^* approximates to all the k zeros of $p(x)$ in S with errors $\leqslant |M^* - M|/2$. It remains to assure that $|M^* - M|/2 < \epsilon$ (which we will do by repeating the integration recursively along the boundaries of smaller and smaller discs centered at the current approximations to M and equivalent to S).

Implementing that outline in Sections 6–9, we will distinguish between Cases 1 and 2 for the input square S by recursively applying a special algorithm (Subalgorithm 6.1 of Section 6) based on the application of Turan's test at the four vertices of the square S. That special algorithm will be used also in Section 10 within our accelerated algorithm for computing a single zero of $p(x)$.

6. HOW TO CONTRACT A SQUARE REGION

The next algorithm will contract a square $S(Y, R)$ having isolation ratio $\geqslant 4$ into its equivalent subsquare $S(Z, r)$ such that either

$$r < 0.8R, \tag{29}$$

that is, the input square is contracted into a substantially smaller subsquare, or otherwise

$$\text{r.r.}\,(S(Z, r)) > 0.1 \quad \text{and} \quad d(S(Z, r)) > 0.3R \geqslant 0.3r, \tag{30}$$

compare Definitions 2.3 and 2.4. If expression (30) holds, the algorithms of Sections 9 and 10 will effectively subdivide all the zeros of $S(Y, R)$ into two or several isolated groups. The algorithm will rely on the application of Turan's test at the four vertices of $S(Y, R)$. Then the points of the four resulting discs free of zeros of $p(x)$ will be eliminated from $S(Y, R)$, and the remaining domain will be covered by the square $S(Z, r)$ via application of Algorithm 2.1.

Subalgorithm 6.1

Inputs. Polynomial $p(x)$ of equation (2), complex Y, and positive R, such that i.r. $(S(Y, R)) \geqslant 4$.

Stage 1. Let g and h take the values -1 and 1 and let

$$Y(g, h) = Y + (g + h\sqrt{-1})R$$

denote the four vertices of the square $S(Y, R)$. Fix N, apply Algorithm 3.1 to the polynomial $p(x)$ at the four points $Y(g, h)$, and denote the output values $r(g, h)$, $r^*(g, h)$ (rather than r, r^*). Denote $D^*_{g,h} = D(Y(g, h), r^*(g, h))$.

Stage 2. Compute the distances between the two pairs of discs, $d^*(1, 1) = \text{dist}(D^*_{1,-1}, D^*_{1,1})$, $d^*(1, -1) = \text{dist}(D^*_{1,-1}, D^*_{-1,1})$, where $\text{dist}(D^*_{g,h}, D^*_{i,j}) = \max\{0, |Y(g, h) - Y(i, j)| - r^*(g, h) - r^*(i, j)\}$. Output the values h, $d^* = d^*(1, h)$, $Y(1, h)$, $Y(-1, -h)$, $r^*(1, h)$, $r^*(-1, -h)$ and denote $D_1 = D(Y(-1, -h), r^*(-1, -h))$, $D_2 = D(Y(1, h), r^*(1, h))$, where $h = 1$ if $d^*(1, 1) \geqslant d^*(1, -1)$ and $h = -1$ otherwise.

$$\text{r.r.}(D_g) \geqslant 1/5^{1/N} \text{ for } g = 1, 2, \tag{31}$$

due to Theorem 3.1.

Stage 3. Consider the four discs $D(Y(g, h), r(g, h))$ (for g, h taking the values 1 and -1) and the square $S(Y, R)$. For each pair of those five domains compute all the intersection points of their boundaries not lying outside of the square $S(Y, R)$ or inside any of the four discs. Let $V = \{v_1, \ldots, v_q\}$, $q \leqslant 10$, denote the set of all those intersection points defined by all the pairs of the five boundaries. Apply Algorithm 2.1 to the set V in order to compute a square $S(Z, r)$ of minimum size that covers the set V (and consequently contains all the zeros of $p(x)$ in $S(y, R)$), compare Proposition 2.3. Output Z and r.

The cost of Subalgorithm 6.1 is $O_A(\log n, n)$.

Proposition 6.1

Subalgorithm 6.1 outputs complex Z and positive r and d^* such that (i) the square $S = S(Z, r)$ is equivalent to the square $S(Y, R)$; i.r. $(S) \geqslant 4$; (ii) $d(S) \geqslant d^*$ and (iii)

$$2\sqrt{2}R - d^* \leqslant (4R - 2r)5^{1/N}. \tag{32}$$

Proof.

$$S(Y, R) - \bigcup_{g,h} D(Y(g, h) \subseteq S \subseteq S(Y, R)$$

by the definition of S, and, as follows from Theorem 3.1, the discs $D(Y(g, h), r(g, h))$ contain no zeros of $p(x)$ for all g, h. This immediately implies condition (i). On the other hand, each disc $D(Y(g, h), r^*(g, h))$ must contain at least one zero of $p(x)$; since i.r. $(S(Y, R)) \geqslant 4$, that zero belongs to $S(Y, R)$ unless $d^* = 0$. This implies condition (ii). To prove expression (32), define the straight line $L(-1, -1)$ connecting the two points $Y(-1, -1) + r(-1, -1)$ and $Y(-1, -1) + r(-1, -1)\sqrt{-1}$ of the intersection of the boundaries of the square $S(Y, R)$ and of the disc $D(Y(-1, -1), r(-1, -1))$. Similarly define the straight lines $L(1, 1)$, $L(-1, 1)$ and $L(1, -1)$. Let $d(h)$ denote the distance between the parallel lines $L(-1, -h)$ and $L(1, h)$ for $h = 1$ and $h = -1$. Observe that no points of the set V (defined in Stage 3 of Subalgorithm 6.1) lie in the rectangle bordered by the four lines $L(g, h)$ and deduce that

$$\max\{d(1), d(-1)\} \geqslant \sqrt{2}r. \tag{33}$$

Next prove for $h = 1$ that

$$d(h) = 2\sqrt{2}R - (2R\sqrt{2} - d(1, h))/\sqrt{2}), \tag{34}$$

where $d(1, h)$ denotes $\text{dist}(D(Y(-1, -h), r(-1, -h)),$

$$D(Y(1, h), r(1, h))) = |Y(-1, -h) - Y(1, h)| - r(-1, -h) - r(1, h).$$

Let E_{-1}, F_{-1}, F_1, E_1 denote four successive points of the diagonal of the square $S(Y, R)$ passing through the points $Y(-1, -1)$ and $Y(1, 1)$, namely, let them denote the four successive intersection points of that diagonal with the four following lines: with $L(-1, -1)$, with the two boundaries of the two discs $D(Y(g, g)), r(g, g))$ for $g = -1$ and $g = 1$, and with $L(1, 1)$. Let A_g and B_g denote the two intersection points of $L(g, g)$ with the two sides of the square $S(Y, R)$ for $g = -1$ and $g = 1$. Then

$$d(1) = |E_1 - E_{-1}|, d(1, 1) = |F_1 - F_{-1}|, |Y(g, g) - E_g| = |Y(g, g) - A_g|/\sqrt{2} = |Y(g, g) - F_g|/\sqrt{2}$$

for $g = 1$ and $g = -1$, so

$$2\sqrt{2}R - d(1) = |Y(1,1) - Y(-1,-1)| - d(1) = \sum_g |Y(g,g) - E_g| = \sum_g |Y(g,g) - F_g|/\sqrt{2}$$
$$= (2\sqrt{2}R - d(1,1))/\sqrt{2}.$$

This proves equation (34) for $h = 1$; similarly equation (34) can be proven for $h = -1$. Combine expressions (33) and (34) and deduce that

$$2\sqrt{2}R - d \leqslant 4R - 2r, \quad \text{where} \quad d = \max\{d(1,1), d(1,-1)\}.$$

Now expression (32) follows because $2\sqrt{2}R - d^* = (2\sqrt{2}R - d)5^{1/N}$. Q.E.D.

Corollary 6.1

If $R \geqslant r \geqslant 0.8R$ then

$$d(S(Z,r)) \geqslant \text{dist}(D_1, D_2) = d^* > (2\sqrt{2} - (2.4)5^{1/N})R. \tag{35}$$

Otherwise expression (29) holds.

Next let $N = 32$, apply expression (6) and Proposition 2.1, and deduce relations (30) from expression (35).

Since i.r. $(S(Z,r)) \geqslant 4$, Subalgorithm 6.1 can be applied again [with the inputs $p(x)$, Z, r, k]. Continue that process recursively (call it *Algorithm 6.1*) and arrive at Proposition 6.2.

Proposition 6.2

For a positive E, a square $S(Y,R)$ having isolation ratio $\geqslant 4$ can be contracted for the cost $O_A(g \log n, n)$, $g = \log(R/E)$, into a square $S(Z,r)$ such that either $r < E$ or else the relations (6.2) are satisfied.

Remark 6.1

We could obtain more favorable bounds on the values r and r^* of the output of Subalgorithm 6.1 if we moved the four points $Y(g,h)$ outside of the square $S(Y,R)$ along the diagonals of that square or if we replaced those four points, say by eight points $Y + (g + h\sqrt{-1})R$ where $g + h$ were odd and g and h took the values -2, -1, 1 and 2. Both of those modifications would require that i.r. $(S(Y,R))$ exceed 4.

7. ACCELERATED SHRINKING OF AN ISOLATED SQUARE

Algorithm 7.1 of this section rapidly contracts a square $S(Y,R)$ having isolation ratio at least 4 either (i) into a disc of diameter $< \epsilon$ [so its center approximates to all the zeros of $p(x)$ lying in $S(Y,R)$ with errors $< \epsilon$] or (ii) into a square $S(Z,r)$ output by Subalgorithm 6.1 and satisfying the relations (30). At first we will apply Subalgorithm 6.1 in order to contract the square $S(Y,R)$ either into a desired square $S(Z,r)$ or into a disc having isolation ratio $\geqslant 8n^2$, see Proposition 6.2. We only need to consider the latter case. We will start with some auxiliary results. Let $i(p(x), U) = k$ and let

$$M = M(U) = \sum_{h=1}^{k} x_{j(h)}/k \tag{36}$$

denote the *center of gravity* of the k zeros of $p(x)$ lying in a domain U.

Remark 7.1

If $k = n$, then the value $M = -p_{n-1}/(p_n n)$ is immediately available, see equation (2).

In the next section we will prove the following result.

Proposition 7.1

For two positive constants c and $\bar{c}$, a natural k, and a disc D such that i.r. $(D) \geqslant (1+v)^2$, $v > 1$,

$i(p(x), D) = k$, the center of gravity $M = M(D)$ of the k zeros of $p(x)$ in D can be approximated by a point $M^* \in D$ with absolute error $\leqslant (1+v)^{-\bar{c}n^c} r$ for the cost $O_A(\log n, n^c)$.

In the sequel we will also use the following proposition.

Proposition 7.2

Let $i(p(x), D) = k$ for a disc D and let M denote the center of gravity of the k zeros of $p(x)$ lying in D. Then $d(D) \geqslant r_{n+1-k}(M)k/(k-1)$, where $r_{n+1-k}(M)$ denotes the $(n+1-k)$th root radius of $p(x)$ at M, compare Definitions 2.4, 2.7

Proof. Let $x_{j(1)}, \ldots, x_{j(k)}$ denote the k zeros of $p(x)$ lying in the disc D and let $|M - x_{j(1)}| = r_{n+1-k}(M)$. Assume that $M = 0$, $x_{j(1)}$ is negative. (Otherwise assure those properties by shifting and scaling the variable x.) Then

$$\sum_{h=1}^{k} \operatorname{Re} x_{j(h)} = 0.$$

Therefore,

$$\sum_{h=2}^{k} \operatorname{Re} x_{j(h)} = -x_{j(1)} = r_{n+1-k}(M), \quad \max_{h=2,\ldots,k} \operatorname{Re} x_{j(h)} \geqslant r_{n+1-k}(M)/(k-1),$$

so

$$d(D) \geqslant r_{n+1-k}(M)k/(k-1). \quad \text{Q.E.D.}$$

Suppose that we can compute the center of gravity M exactly [2]. Then we would apply algorithm 4.2, compute a value $r^*_{n+1-k}(M)$, slightly exceeding the root radius $r_{n+1-k}(M)$, and arrive at the disc $D(M, r^*_{n+1-k}(M))$, having rigidity ratio $> 1/2$, due to Proposition 7.2. Then we would apply Subalgorithm 6.1, arrive at the relations (30), and effectively apply algorithms of Sections 9 or 10. Actually we will compute approximation M^* to M with some error estimated from above in Proposition 7.1. If that upper estimate on the error is much smaller than $r^*_{n+1-k}(M^*)$, then we will just ignore the error and will go ahead with Subalgorithm 6.1 as above. Otherwise $r^*_{n+1-k}(M^*)$ is small, so that we contracted the input square $S(Z, r)$ into a small disc $D(M^*, r^*_{n+1-k}(M^*))$. In that case we will again apply the integration, this time along a circular contour with the center at M^* and having appropriate radius, much smaller than the initial r. Then the error of the approximation to M will greatly decrease. Recursive application of that process will lead us either to (6.2) (a desired option!) or to rapid contraction of $S(Z, r)$ into a disc of radius $< \epsilon$ [an option not less desired, because the center of that disc will approximate to all the zeros of $p(x)$ in $S(Z, r)$ with errors less than ϵ].

Next we will elaborate that approach.

Corollary 7.1

Let, under the assumptions of Proposition 7.2, $|M^* - M|/r_{n+1-k}(M) = \alpha$ for some complex M^*, and let D^* denote the disc $D(M^*, (1+\alpha)r_{n+1-k}(M))$. Then $D^* \supseteq D(M, r_{n+1-k}(M))$, and therefore

$$r_{n+1-k}(M^*) \leqslant (1+\alpha)r_{n+1-k}(M), \quad d(D^*) \geqslant d(D) \geqslant (r_{n+1-k}(M^*)/(1+\alpha))(k/(k-1)).$$

Next we will present the desired algorithm for a rapid contraction of a disc. In this section, unlike Section 4, r_j does not denote a root radius of $p(x)$, compare Remark 2.1.

Algorithm 7.1

Inputs. Polynomial $p(x)$ of equation (2), complex X, natural k, and positive r and v such that

$$i(p(x), \quad D(X, r)) = k, \quad \text{i.r.} \quad (D(X, r)) = (1+v)^2 \geqslant 8n^2. \tag{37}$$

Let M denote the unknown center of gravity of the k zeros of $p(x)$ lying in $D(X, r)$.

Initialization. $j = 0$, $M_0 = X$, $r_0 = r$, $v_0 = v$, $D_0 = D(M_0, r_0)$.

Stage j, $j = 0, 1, \ldots, J-1$. At first compute [for the cost $O_A(\log n, n)$] an approximation $M_{j+1} \in D(M_j, r_j)$ to M with absolute error bound

$$\epsilon_{j+1} = (1+v_j)^{-4n} r_j \tag{38}$$

(apply Proposition 7.1 where $c = 1$, $\bar{c} \approx 4$, $X = M_j$, $r = r_j$, $v = v_j$, compare also Proposition 7.3

below). Then compute an approximation r^*_{j+1} to the $(n+1-k)$th root radius $r_{n+1-k}(M_{j+1})$ of $p(x)$ at M_{j+1} such that the ratio $r^*_{j+1}/r_{n+1-k}(M_{j+1})$ lies between $1/(2n)$ and $2n$. The cost of computing r^*_{j+1} is $O_A(1, n)$, due to Proposition 4.3; indeed in our case $1+\Delta \geqslant 2n$, and relation (25) holds for $D = D(M_j, r_j)$. [To deduce relation (25), combine the inequality of conditions (37) with the inclusions $M_{j+1} \in D(M_j, r_j) \subseteq D(M_0, 2r_0)$; the latter inclusions hold for $j=0$ and are recursively extended to all j.] Denote

$$r_{j+1} = 2n\, r^*_{j+1}, \quad D_{j+1} = D(M_{j+1}, r_{j+1}), \quad (1+v_{j+1})^2 = (1+v_j)^{2n}. \tag{39}$$

If $r_{j+1} < \epsilon$, set $J = j+1$, output M_J and r_J, and end the computations. If

$$\epsilon \leqslant r_{j+1} < 8n^2(1+v_j)^{-4n} r_j = 8n^2 \epsilon_{j+1}, \tag{40}$$

enter Stage $j+1$. Otherwise set $J = j+1$ and apply Algorithm 6.1 to the square $S(M_J, r_J\sqrt{2})$ superscribing the disc D_J [until the relations (30) are satisfied].

Let us analyze Algorithm 7.1 and estimate the number of its iterations (Stages). At first consider the case where bound (40) holds for all j. Then recursively apply the last equation of expression (39) and obtain that $(1+v_j)^2 = (1+v)^{2^j}$, $j = 0,1,\ldots,J-1$. Substitute this into bound (40) and obtain that $r_{j+1} < 8n^2(1+v)^{-4n^{j+1}} r_j$. Apply the latter relation recursively, and deduce that

$$\log_2(r/r_j) > 4n^j \log_2(1+v) \quad \text{if} \quad j > 1,$$

$$r_J < \epsilon \text{ if } J = \lceil (\log_2\log_2(r/\epsilon) - 2 - \log_2\log_2(1+v))/\log_2 n \rceil,$$

which gives a desired upper bound on the number of iterations J of Algorithm 7.1 in this case. We need, however, the following result in order to apply Proposition 7.1 when we deduce equation (38).

Proposition 7.3

If bound (40) holds for $j = 0, 1, \ldots, g$, then $(1+v_j)^2 \leqslant$ i.r. (D_j) for $j = 0, 1, \ldots, g$.

Proof. Combine the definition of r^*_{j+1} and the equations (39) and obtain that all the discs D_j are equivalent to D_0 and to each other. Therefore $|M_{j+1} - M_j| \leqslant r_j + r_{j+1}$ for all j. On the other hand,

$$\begin{aligned}(\text{i.r. } (D_{j+1}))r_{j+1} &= r_{n-k}(M_{j+1}) \geqslant r_{n-k}(M_j) - |M_{j+1} - M_j| \\ &= (\text{i.r. } (D_j))r_j - |M_{j+1} - M_j| \geqslant (\text{i.r. } (D_j))r_j - (r_j + r_{j+1})\end{aligned}$$

for all j. It follows that

$$(\text{i.r. } (D_{j+1}) + 1)/(\text{i.r. } (D_j) - 1) \geqslant r_j/r_{j+1}$$

for all j. Apply bound (40) and deduce that

$$(\text{i.r. } (D_{j+1}) + 1)/(\text{i.r. } (D_j) - 1) \geqslant (1+v_j)^{4n}/(8n^2) \tag{41}$$

for all j. Finally recall that $(1+v_0)^2 =$ i.r. (D_0) and observe that i.r. (D_j) grows more rapidly than $(1+v_j)^2$ does as j grows. [The latter fact follows via comparison of relation (41) with the last equation of expression (39) for $j = 0, 1, \ldots,$ since $(1+v_j)^{2n} > (1+v)^2 \geqslant 8n^2$, see condition (37).] Q.E.D.

Next assume that bound (40) does not hold for some j. Observe that

$$i(p(x), D_{j+1}) = k, \quad r_{n+1-k}(M_{j+1}) < r_{j+1} < 4n^2 r_{n+1-k}(M_{j+1}). \tag{42}$$

Denote $M^* = M_{j+1}$, $\alpha = \alpha_{j+1} = |M - M^*|/r_{n+1-k}(M)$, $D^* = D(M^*, (1+\alpha)r_{n+1-k}(M))$, combine Corollary 7.1 and relation (42), and arrive at the following estimate:

$$d(D^*) = d(D_{j+1}) \geqslant (r_{n+1-k}(M_{j+1})/(1+\alpha_{j+1}))(k/(k-1)),$$

where

$$\alpha_{j+1} \leqslant \epsilon_{j+1}/r_{n+1-k}(M) \leqslant \epsilon_{j+1}/(r_{n+1-k}(M_{j+1}) - \epsilon_{j+1}).$$

Therefore,

$$d(D_{j+1}) \geqslant (r_{n+1-k}(M_{j+1}) - \epsilon_{j+1})k/(k-1). \tag{43}$$

If bound (40) does not hold some j, then

$$8n^2\epsilon_{j+1} = 8n^2(1+v_j)^{-4n}r_j \leqslant r_{j+1} < 4n^2 r_{n+1-k}(M_{j+1}),$$

compare equations (38) and (42). Consequently

$$2\epsilon_{j+1} < r_{n+1-k}(M_{j+1}),\ r_{n+1-k}(M_{j+1}) - \epsilon_{j+1} > r_{n+1-k}(M_{j+1})/2.$$

Then relations (42) and (43) imply that

$$d(D_{j+1}) \geqslant (0.5)r_{n+1-k}(M_{j+1})k/(k-1) \geqslant (r_{j+1}/(8n^2))(k/(k-1)). \tag{44}$$

On the other hand, unless bound (40) holds, Algorithm 7.1 requires to apply Algorithm 6.1 to the square $S(M_{j+1}, r_{j+1}\sqrt{2})$, superscribing the disc D_{j+1}; in that case the output square $S(Z, r)$ will satisfy the relations (6.2) in $O(\log n)$ recursive applications of Subalgorithm 6.1 [due to relations (44)], and then the computations of Algorithm 7.1 will end. Summarizing we arrive at the following result.

Proposition 7.4

Let positive b and ϵ satisfy equation (3) for $R \geqslant r$. Then Algorithm 7.1 contracts its input disc $D(X, r)$ satisfying condition (37) either for the cost $O_A(\log b \log n, n)$ into a disc of radius $< \epsilon$ [whose center approximates to all the k zeros of $p(x)$ in $S(X, r)$ with absolute errors $< \epsilon$] or else (this may occur only for $k > 1$) for the cost $O_A(\log \log(R/r_{j+1})\log n, n)$ into a disc D_{j+1} satisfying relation (44). In the latter case, $O(\log n)$ further recursive applications of Subalgorithm 6.1 starting with the square $S(M_{j+1}, r_{j+1}\sqrt{2})$ suffice in order to compute a square $S(Z, r)$ satisfying the relations (30) [for the additional cost $O_A(\log^2 n, n)$].

Remark 7.2

In the cases where Algorithm 7.1 is recursively applied in order to approximate to the *same* zero of $p(x)$, the cost of its H recursive applications is $O_A(\log n \log b, n)$ (*independent* of H because we just need to sum $\log \log (R/r_{j+1})$ in j and because in all those applications r_{j+1} monotone decreases to ϵ as j grows), but the cost of the subsequent applications of Subalgorithm 6.1 is $O_A(H \log^2 n, n)$.

8. COMPUTING THE CENTER OF GRAVITY OF A SET OF POLYNOMIAL ZEROS

In this section we will prove Proposition 7.1. Extending equation (11) we arrive at the following formula [13]:

$$M = \frac{1}{2\pi k i}\int_\Gamma (xp'(x)/p(x))\,dx, \tag{45}$$

where $i = \sqrt{-1}$, the value $M = M(U)$ is defined by equation (36), and the domain U bordered by the contour Γ contains exactly k zeros of $p(x)$ (not necessarily distinct), denoted $x_{j(i)}$, $i = 1, \ldots, k$. We will assume that $U = D(X, r)$ is a disc with isolation ratio $\geqslant (1+v)^2$, will choose Γ being the boundary of the disc $D(X, R)$, $R = (1+v)r$, and will approximate to the integral (45) using the integral sum

$$M^* = \frac{1}{2\pi k Q i}\sum_{q=0}^{Q-1} (X + R\omega^q)\,p'(X + R\omega^q)/p(X + R\omega^q). \tag{46}$$

Here ω is a primitive Qth root of 1, $\omega^Q = 1$, $\omega^q \neq 1$ for $0 < q < Q$. We bound the error of the approximation to M by M^* basing on the Laurent expansion

$$p'(x)/p(x) = \sum_{m=0}^{\infty} s_m x^{1-m} - \sum_{m=1}^{\infty} S_m x^{m-1}, \tag{47}$$

where

$$s_m = \sum_{i=1}^{k} x_{j(i)}^m; \quad S_m = \sum_{i=k+1}^{n} x_{j(i)}^m;$$

$\{x_{j(1)}, \ldots, x_{j(n)}\}$ is the set of all the zeros of $p(x)$ numbered such that $|X - x_{j(i)}| < R$ if and only if $i \leqslant k$, compare with Ref. [2, Sect. 12]. Equations (45)–(47) immediately imply that

$$|M^* - M| \leqslant 2R(kg^{Q+1} + (n-k)g^{Q-1})/(k(1-g^Q)), \tag{48}$$

where

$$g = \min_{1 \leqslant j \leqslant n} \min \{|X - x_j|/R, R/|X - x_j|\}. \tag{49}$$

$g \leqslant 1/(1+v)$ since $R = (1+v)r$, i.r. $(D(X,r)) \geqslant (1+v)^2$. Expressions (48) and (49) imply that

$$|M^* - M| < 4Rn/(k(1+v)^{Q-1}) = 4nr/(k(1+v)^Q). \tag{50}$$

We will keep Q of an order of c^*n^c for a constant c^*, so the cost of the integration will be $O_A(\log n, n^c)$; we will choose the constant c^* such that $|M^* - M| \leqslant (1+v)^{-\bar{c}n^c}r$, as this is required in Proposition 7.1. Q.E.D.

Remark 8.1

We may similarly estimate the error $|s_K^* - s_K|$ of the numerical integration for arbitrary K, where s_K is defined by equation (11) and

$$s_K^* = \frac{1}{2\pi Qi} \sum_{q=0}^{Q-1} (x + R\omega^q)^K p'(X + R\omega^q)/p(X + R\omega^q). \tag{51}$$

We apply equation (47) again and deduce that

$$|s_K^* - s_K| \leqslant 2R^K(kg^{Q+K} + (n-k)g^{Q-K})/(1-g^Q), \tag{52}$$

where k denotes $i(p(x), D(X,r))$ and g is defined by equation (49).

9. TURAN–WEYL'S ISOLATION ALGORITHM

Using the machinery of the previous sections we will now extend (Turan–Weyl's) Algorithm 5.1 of Section 5 and will arrive at the bounds of the last line of Table 1.

Let $S(Y,R)$ denote an input square of some recursive step such that i.r.$(S(Y,R)) \geqslant 6$, $i(p(x), S(Y,R)) = k$. [At the initial step $Y = 0$, R is defined by equation (5), $K = n$.] Applying Algorithm 7.1, we will contract the square $S(Y,R)$ and will either approximate all the k zeros of $p(x)$ in $S(Y,R)$ with errors $\leqslant \epsilon$ or will arrive at the case where Subalgorithm 6.1 has been applied and its output square S satisfies relations (30); in particular, in that case $S(Y,R)$ shrinks to a square $S = S(Z,r)$ such that r.r. $(S) > 0.1$. Then we will apply (Turan–Weyl's) Algorithm 5.1 stopping in

$$J = \lceil \log_2(nr/\epsilon) \rceil + 5 \tag{53}$$

iterations, which will solve our problem if r is already close to ϵ. We will, however, end the computations in j iterations for $j < J$ if we can group the suspect squares output by iteration j into at least two maximal connected components, *strongly isolated* in the sense to be defined below.

The idea is to partition the zeros of $p(x)$ into two or several nonempty sets, each included into a square with isolation ratio $\geqslant 6$. Then our algorithms will be recursively applied to each of such squares independently of others. With each subdivision of the zeros of $p(x)$, the index of $p(x)$ in new squares decreases. Therefore, there can be at most $n-1$ subdivisions until we either contract all the squares into discs of radii $< \epsilon/\sqrt{2}$ and then end the computations or will arrive at the squares each containing only a single zero of $p(x)$. In the latter case we will apply Algorithm 7.1 and will rapidly contract those squares into discs of radii $< \epsilon$. Next we will formally describe the desired recursive subdivision of the zeros of $p(x)$ assuming relations (30) and applying Algorithm 5.1.

Partition the union of all the suspect squares returned by iteration j into $H(j)$ maximal connected components, $C_1, \ldots, C_{H(j)}$; each component contains at least one zero of $p(x)$, so

$$H(j) \leqslant k. \tag{54}$$

For every g apply Algorithm 2.1 to the set of all the vertices of all the suspect squares of C_g and arrive at a square $S(X_g, R_g^*)$ covering C_g. Propositions 2.3 and 5.1 imply that

$$R_g^* \leqslant (k+2)R/2^{j+1}. \tag{55}$$

Call component C_g and square $S(X_g, R_g^*)$ *strongly isolated* if

$$|X_h - X_g| > \sqrt{2}\,(R_h^* + 6R_g^*) \quad \text{for all} \quad h \neq g. \tag{56}$$

Condition (56) implies that the square $S = S(X_g, R_g^*)$ is equivalent to C_g and that

$$\text{i.r.}\,(S) \geqslant 6. \tag{57}$$

Let exactly $h(g)$ strongly isolated components be output by iteration g of Algorithm 5.1, let $h(g) = 1$ for $g < j$, $h(j) > 1$. Combining expressions (54)–(56) we deduce that $j \leqslant \lceil j(0) \rceil$ such that

$$1/(k+1) = 7\sqrt{2}(k+2)/2^{j(0)+1},$$

so

$$j \leqslant \lceil -0.5 + \log_2 7 + \log_2((k+1)(k+2)) \rceil, \tag{58}$$

compare also condition (58). Having completed that iteration j, fix all the $h(j)$ strongly isolated components and continue applying Algorithm 5.1 to all the suspect squares of all other components until only strongly isolated squares are returned. We will use the name *Turan–Weyl's isolation algorithm* for that modification of Algorithm 5.1. The cost of that algorithm is $O_A(H \log n, n)$, where H denotes the total number of all the suspect squares processed in all the iterations.

Next we will prove that $H = O(h)$, where h denotes the number of strongly isolated components output by the final iteration. Moreover, we will prove that $H = O(h)$ even for a modification of the algorithm where the suspect squares of each strongly isolated component are subdivided further as long as the diameter of the component exceeds the diameter of a suspect square more than twice. Then each output component consists of not more than four suspect squares. Certainly the cost of the original algorithm may only increase due to that modification. To show that $H = O(h)$, we will retrace back the process of the recursive subdivision of suspect squares, beginning from its end, that is, from the last iteration, which returns h strongly isolated components. We will respectively reverse the basis property of the forward subdivision process, that is, a subdivision of a suspect square decreases its diameter by 50%, but that diameter is doubled when we retrace the process back; therefore every backtrack step expands the components in all directions. (Exception: the strongly isolated output components will stay unchanged by the backtrack steps where they remained unchanged by the associated steps of the forward process.) The distance between every two components output by iteration j is lower bounded by the length of an edge of a suspect square; we may at least double such a bound unless in a backtrack step (from iteration j to iteration $j-1$) these two components are output components or meet each other. Therefore, each component C either is a strongly isolated output component or meets another component in at most $\lceil \log_2(3k(C)) \rceil$ backtrack steps, where $k(C)$ is the number of suspect squares in that component C, compare condition (56). Let us represent all the components in all iterations by the nodes of a tree whose h leaves correspond to the h output components of the algorithm and whose each edge represents one or several backtrack steps needed in order that one component could meet another. The total number of the nodes of the tree is at most $2h-1$, which also means at most $2h-2$ edges of the tree. At the leaves level there are at most $4h$ suspect squares. This immediately implies that $H = O(h \log^2 h)$ since the number of suspect squares cannot grow in the backtrack process, which in particular bounds the number of suspect squares in *each* component by h. The stronger bound $H = O(h)$ follows from the simple observation that in each step of the backtrack process each connected set of g suspect squares is imbedded into a set of at most $2 + g/2$ larger suspect squares; therefore the number of suspect squares processed in all the components having

at least five suspect squares decreases at least by 10% in each backtrack step. Consequently a total number of suspect squares in all steps is less than $40h$, not counting the suspect squares in the components consisting of at most five suspect squares. If a component consists of $k \leqslant 5$ suspect squares, then the edge in the tree from that component in the direction to the root corresponds to at most $\lceil \log_2(3k) \rceil \leqslant 4$ backtrack steps and therefore to at most 20 suspect squares. There are at most $2h-2$ edges in the tree, so we arrive at the *rough* upper bound $H < 40h + 20(2h-2) < 80h$.

Summarizing, in $H = O(h)$ iterations for the overall cost $O_A(h \log n, n)$, Turan–Weyl's isolation algorithm returns h strongly isolated components C_g, $g = 1, \ldots, h$, each consisting of at most four suspect squares. We cover each of these h components C_g by a square S_g equivalent to C_g and such that i.r. $(S_g) \geqslant 6$ [see condition (57)], superscribe the discs of the minimum size around the squares, and compute the indices of $p(x)$ in all those discs (see Proposition 5.2). Then again we recursively apply Subalgorithm 6.1, Algorithm 7.1, and finally Turan–Weyl's isolation algorithm to each of those squares S_g, until we compute all the zeros of $p(x)$ with absolute errors less than ϵ. To estimate the overall cost, associate the subdivision of the input components for each application of Turan-Weyl's isolation algorithm with the edges of the tree, whose nodes are those input components, whose root is the input square $S(0, R)$ for R satisfying equation (53), and whose leaves are the components of diameters $< \epsilon$. There are at most n leaves, so there are at most $2n-1$ nodes in the tree, and all the required applications of Turan–Weyl's isolation algorithm have overall cost $O_A(n \log n, n)$. Due to the recursive applications of Algorithms 5.2 and 7.1 [required $O(n)$ times in the case of approximating to all the n zeros of $p(x)$; Remark 7.2 is not applied in that case] and of Subalgorithm 6.1 [required $O(n \log n)$ times], the overall cost bound increases to $O_A(n \log n \log(bn), n)$, compare with Proposition 7.4.

Theorem 9.1

Let positive ϵ, b, and R satisfy conditions (3) and (4). Then isolated ϵ_s-approximations to all the zeros of a polynomial $p(x)$ of equation (2) for some $\epsilon_s < \epsilon$ can be computed for the cost $O_A(n \log n \log(bn), n)$.

Turan–Weyl's isolation algorithm also leads to the following result, see also Remark 9.1 below.

Theorem 9.2

Let positive ϵ, v, and R satisfy conditions (3) and (4). Then an isolated ϵ-approximation to a zero of $p(x)$ can be computed with absolute error at most ϵ for the cost $O_A(\log n(\log^2 n + \log b), n(1 + n/\log^2 n + \log b)))$ (under parallel model) or $O_A(n \log n(n + \log b))$ (sequential time).

Proof. Apply Turan–Weyl's algorithm [see conditions (53)–(58)] and observe that the first two strongly isolated components C_1 and C_2 are computed in at most $O(\log n)$ iterations. Compute the indices i_1 and i_2 of $p(x)$ in both of these components, see Proposition 5.2. $i_1 + i_2 \leqslant n$, so $\min\{i_1, i_2\} \leqslant n/2$, say $i_1 = i(p(x), C_1) \leqslant n/2$. Apply algorithms of Sections 6–9 to the component C_1 and repeat that process recursively, defining a sequence of strongly isolated components $C_{g(0)} \geqslant C_{g(1)} \geq \ldots$ such that $C_{g(0)} = C_1$ and

$$i(p(x), C_{g(h)}) \leqslant n/2^h, \quad h = 0, 1, \ldots.$$

The latter relations imply that the component $C_{g(h)}$ contains at most $n/2^{h-2}$ suspect squares, so the cost of the corresponding Turan tests is $O_A(\log n, n^2/2^h)$. Slow down that computation to save processors and arrive at the cost bound $O_A(\log^2 n/)/2^h, n^2/\log n)$. Summing in h from 0 to $\lceil \log n \rceil$ arrive at the overall cost bound $O_A(\log^2 n, n^2/\log n)$ for all the $O(\log n)$ applications of Turan–Weyl's. Slow down the parallel computations once again (to save processors) and replace that cost bound by $O_A(\log n(\log^2 n + \log b), n^2/(\log^2 n + \log b))$. Adding here the cost $O_A(\log n(\log b + \log^2 n), n)$ of $O(\log n)$ recursive applications of Algorithm 7.1 (see Remark 7.2) and of $O(\log^2 n)$ applications of Subalgorithm 6.1, we arrive at the bounds of Theorem 9.2.

Remark 9.1

The asymptotic cost bounds of Theorem 9.2 are improved in Theorems 10.1 and 12.1 for the price of increasing the overhead constants.

10. LEHMER–TURAN'S ISOLATION ALGORITHM

In this section we will extend Lehmer–Turan's construction of Section 3 in order to improve the bounds of Theorem 9.2. We will rely on Subalgorithm 10.1 below that outputs the values X, ρ and $k(M)$ supporting the following result.

Proposition 10.1

Let Subalgorithm 6.1 be applied to a square $S(Y, R)$ such that $i(p(x), S(Y, R)) = k > 1$, i.r. $(S(Y, R)) \geqslant 4$. Let it output two discs $D_1 = D(Y_1, r_1^*)$ and $D_2 = D_2(Y_2, r_2^*)$ satisfying equations (30), (31) and (35), where $N = 32$,

$$Y = (Y_2 + Y_1)/2, \quad R = |Y_2 - Y_1|/(2\sqrt{2}). \tag{59}$$

Then it is possible to compute a complex $X \in S(Y, R)$ and a positive $\rho \leqslant R/2$ for the cost $O_A(M \log n \log\log n, n)$ such that i.r. $(S(X, \rho)) \geqslant 4$, $1 \leqslant M \leqslant \log_2 k$, $1 \leqslant i(p(x), S(X, \rho)) = k(M) \leqslant k/2^M$.

At first we will show how to use Subalgorithm 10.1 and then will present that algorithm.

Lehmer–Turan's isolation algorithm

Apply Subalgorithm 6.1 and then (if needed) Algorithm 7.1 initially to the square $S(0, R)$ with R satisfying equation (5) and then to the output squares $S = S(X, \rho)$ in all applications of Subalgorithm 10.1. Apply Subalgorithm 10.1 whenever the assumptions of Proposition 10.1 are satisfied.

Lehmer–Turan's isolation algorithm may call Subalgorithm 10.1 at most $\log_2 k$ times before it computes an isolated ϵ-approximation to a zero of $p(x)$ [this is due to the bound $k(M) \leqslant k/2^M < k/2$ of Proposition 10.1]. Therefore, all the applications of Subalgorithm 10.1 will contribute $O_A(\log^2 n \log\log n, n)$ to the cost of Lehmer–Turan's isolation algorithm. That contribution will be dominated by the cost bound due to $O(\log^2 n)$ applications of Subalgorithm 6.1, see Proposition 7.4. The $O(\log n)$ required applications of Algorithm 7.1 will contribute $O_A(\log n \log b, n)$, see Remark 7.2, so we will arrive at the following improvement of Theorem 9.2, which in turn will be slightly improved in Theorem 12.1, compare Remark 9.1.

Theorem 10.1

Let positive ϵ, b, and R satisfy (2.2), (2.3). Then an isolated ϵ-approximation to a zero of $p(x)$ can be computed for the cost $O_A(\log n(\log^2 n + \log b), n)$.

It remains to prove Proposition 10.1.

We will start with an outline of Subalgorithm 10.1.

Initially we have the outputs of Subalgorithm 6.1 satisfying relations (30), (31) and (35); in particular we have two discs at the distance $> 0.3R$ from each other, whose centers lie at two opposite vertices of the square $S(Y, R)$. We apply Algorithm 3.2 at those two vertices in order to compute $x^*_{j(0,1)}$ and $x^*_{j(0,2)}$, two approximations to two zeros of $p(x)$. We choose sufficiently large values of the parameters Q and N of Algorithm 3.2, to assure the upper bound $0.01R$ on the approximation errors and the lower bound $4.001r(0) = |x^*_{j(0,2)} - x^*_{j(0,1)}| \geqslant 0.1R$. This is the initial Stage 0 of Subalgorithm 10.1. Then, solving Tasks r of Sections 4, we estimate from above the indices $k_g = i(p(x), D^*_{0,g})$, where $D^*_{0,g} = D(x^*_{j(0,g)}, 2r(0))$, $g = 1, 2$. $k_1 + k_2 \leqslant k$, since the intersection of the two discs, $D^*_{0,1} \cap D^*_{0,2}$ is empty. We assume that $k_1 \leqslant k/2$ and set $g = 1$. (Otherwise we would set $g = 2$.) Then we apply Turan's tests at the points $x^*_{j(0,g)} + r(0)\omega^i$, $i = 0, 1, \ldots, Q-1$, ω being a primitive Qth root of 1. If the distance from at least one of those points to a zero of $p(x)$ is less than $0.8\, r(0)$, we apply Algorithm 3.2 at such a point and at $x^*_{j(0,g)}$, so that the two resulting refined approximations $x^*_{j(1,1)}$ and $x^*_{j(1,2)}$ to the zeros of $p(x)$ lie substantially closer to such zeros than to each other. That was in fact the initial situation; $x^*_{j(1,h)}$ just replace $x^*_{j(0,h)}$ for $h = 1, 2$. Thus we repeat the process recursively. The number of zeros of $p(x)$ in the initial disc (or square) decreases by $\geqslant 50\%$ in each recursive stage, so in at most $\log_2 k$ stages the distances from all the points $x^*_{j(m,g)} + r(m)\omega^i$ for $i = 0, 1, \ldots, Q-1$ to zeros of $p(x)$ exceed $0.8r(m)$. Then a small disc around $x^*_{j(m,g)}$ has isolation ratio > 8, and we may deduce Proposition 10.1.

Now we will formalize that outline.

Subalgorithm 10.1

Inputs. Polynomial $p(x)$ of equation (2), complex Y, Y_1, and Y_2, positive R, r_1^* and r_2^*, and natural $k \leqslant n$, such that relations (31) hold for the two discs $D_g = D(Y_g, r_g^*)$, $g = 1, 2$, and for $N = 32$; condition (59) holds;

$$|Y_2 - Y_1| - (r_1^* + r_2^*) > 0.3R; \tag{60}$$

$$\text{i.r.}(S(Y, R)) \geqslant 4, \quad i(p(x), \quad S(Y, R)) = k. \tag{61}$$

[Condition (60) follows from relations (32) and (35), while condition (61) is our usual assumption on the inputs of Subalgorithm 6.1.]

Stage 0. Apply Algorithm 3.2 with $\epsilon = 0.001R$, $Q = N = 64$, $X_0 = Y_g$, $r_0^* = r_g^*$ twice, for $g = 1$ and for $g = 2$ and let $x_{j(0,g)}^*$ for $g = 1, 2$ denote the two computed approximations to the zeros of $p(x)$ (whose errors are less than $\epsilon = 0.001R$); the cost $O_A(\log n, n)$ suffices due to conditions (59)–(61). Denote $k(0) = k$ and enter Stage 1. Conditions (59)–(61) imply that $|x_{j(0,2)}^* - x_{j(0,1)}^*| > 0.1R = 100\epsilon$.

Stage m, m = 0, 1, ... Inputs. Two complex points $x_{j(m,1)}^*, x_{j(m,2)}^*$ lying in the square $S(Y, R)$, positive values $r(m) = |x_{j(m,2)}^* - x_{j(m,1)}^*|/4$, $r = 2r(m)/(1 + \Delta)$, $\Delta = 0.1$, and natural $k(m)$ such that

$$1 \leqslant k(m) \leqslant k/2^m, \tag{62}$$

$$\sum_{g=1}^{2} i(p(x), D_{m,g}^*) \leqslant k(m) \tag{63}$$

$$|x_{j(m,g)}^* - x_{j(m,g)}| < r(m)/25, \tag{64}$$

where $D_{m,g}^*$ denotes the discs $D(x_{j(m,g)}^*, 2r(m))$ and where $x_{j(m,g)}$ denote two zeros of $p(x)$ for $g = 1, 2$. (Stage 0 was devised so that its outputs satisfy those requirements for $m = 0$, and the subsequent stages m will be devised so to maintain them. Note that conditions (62) bound the overall number of stages by $\log_2 k$; we will assure all the requirements of Proposition 10.1 and will end the computations when $k(m)$ decreases to 1 or earlier. The two inequalities (64) show that the distances from $x_{j(m,g)}^*$ to the nearest zeros of $p(x)$ for $g = 1, 2$ are within 1% of $|x_{j(m,2)}^* - x_{j(m,1)}^*|$.)

Computations. For the cost $O_A(\log n \log\log n, n)$ solve two Tasks r of Section 4 (where $\Delta = 0.1$ and r is defined above) at the two points $x_{j(m,g)}^*$ for $g = 1, 2$ (see Proposition 4.1). Denote the output integers s_1 and s_2. Then $n - s_g \leqslant i(p(x), D_{m,g}^*)$ for $g = 1, 2$ (by the definition of Task r), so $2n - s_1 - s_2 \leqslant k(m)$, due to condition (63). Fix $g = g(m) = 1$ if $n - s_1 \leqslant k(m)/2$; fix $g = g(m) = 2$ otherwise, and denote $k(m + 1) = n - s_g$, so that

$$i(p(x), D_{m,g}^*) \leqslant k(m + 1) \leqslant k(m)/2, \tag{65}$$

and condition (62) is extended if we replace m by $m + 1$. In fact relation (65) will also imply the similar extension of condition (63) because we will choose the two discs $D_{m+1,g}^*$ (for $g = 1, 2$) lying well inside the disc $D_{m,g}^*$. Apply Algorithm 3.1 (Turan's test) at the Q points $z_{i,m} = x_{j(m,g)} + r(m)\omega^i$ for $i = 0, 1, \ldots, Q - 1$, where g is fixed above, ω is a primitive Q-root of 1, and, say $Q = N = 64$. Of the Q output values r of the Q Turan's tests, choose the smallest. If that value exceeds $0.8r(m)$ [which, in particular, must be the case if $m + 1 \geqslant \log_2 n$ for then $k(m + 1) = 1$, see condition (62)], then the disc $D(x_{j(m,g)}^*, 0.22r(m))$ has isolation ratio > 8 for it is surrounded by a wide enough annulus free of zeros of $p(x)$. Note that $1 \leqslant i(p(x), D(x_{j(m,g)}^*, 0.22r(m))) \leqslant k(m + 1)$, so in that case we will satisfy all the requirements of Proposition 10.1 if we denote $M = m + 1$, output $x = x_{j(M,g)}^*$, $\rho = 0.22r(M)\sqrt{2}$, $k(M)$, and end the computations. Otherwise consider the latter Turan's test as the current iteration of Algorithm 3.2 and perform two more iterations of that algorithm with $Q = N = 128$, so the output approximation to a zero of $p(x)$ is computed with an absolute error less than $\epsilon = 0.001r(m)$ and remains at the distance $> 0.19r(m)$ from $x_{j(m,g)}^*$. Denote that approximation $x_{j(m+1,1)}^*$. Apply Lehmer–Turan's algorithm with $Q = N = 32$, $\epsilon = 0.001r(m)$ at $x_{j(m,g)}^*$ [two iterations of that algorithm will suffice due to condition (64)], and let $x_{j(m+1,2)}^*$ denote the output approximation to a zero of $p(x)$. It is easy to verify that the distance $4r(m + 1)$ between the two points $x_{j(m+1,g)}^*$ for $g = 1, 2$ exceeds $0.1r(m) = 100\epsilon$, that is, condition (64) is also extended [similarly to condition (62) and (63)] when m is replaced by $m + 1$. At this point, enter Stage $m + 1$. [The cost of performing Stage m is $O_A(\log n \log\log n, n)$, due to Theorem 3.2 and Proposition 4.1; this implies the cost bound of Proposition 10.1.] Q.E.D.

11. PRECISION OF COMPUTATIONS AND THE BOOLEAN CIRCUIT COMPLEXITY OF COMPUTING POLYNOMIAL ZEROS

Let some *worst case* upper bound ϵ on the output errors be prescribed and let us estimate the precision of computations B, measured by the number of floating point binary digits or bits in the mantissas (fractional parts) of the operands of our algorithms. It is well known that the dependence of the zeros of $p(x)$ on its coefficients is generally ill-conditioned [42, pp. 82–83], furthermore the results of Refs [2, 12 pp. 74–77; 23] suggest that B should be at least of an order of bn where b is defined by conditions (2) and (3). Our claim is that all our algorithms only require the precision of computations $B = O(bn)$.

To support that claim, recall that all our algorithms are naturally subdivided into self-correcting stages, whose output errors are automatically corrected in the next stages; thus it is sufficient to estimate the precision required in each stage. Specifically, the entire error analysis of Turan–Weyl's isolation algorithm can be reduced to the error analysis of Algorithm 3.1 (Turan's test), of Algorithm 4.2 [where we may assume the inequalities (25) and $1 + \Delta \geqslant 2n$, see Proposition 4.3], of the solution of Task r (see Remark 5.2), and of the numerical integration (46) whose error bound is defined by conditions (48) and (49) with

$$g < 1/(8n^2) \tag{66}$$

(see Section 8). Turan–Weyl's isolation algorithm requires to solve Task r in order to compute $i(p(x), D)$ where i.r. $(D) \geqslant 8n^2$ for the disc D; then surely $1 + \Delta > 2n$ for Task r. [Here we assume that the indices of $p(x)$ are computed based on Remark 5.2, but alternatively we may ignore Task r and Remark 5.2 and use winding number algorithms of Ref. [13 pp. 239–241] or Ref. [11]; that option would not require to increase the precision of computations.] Apart from the stages of shifting the variable x by X, that is, of computing the coefficients $p_i(X)$ of condition (7), the error analysis is trivial for Algorithm 4.2 where $1 + \Delta \geqslant 2n$ and where condition (25) holds (see Proposition 4.3) and for solving Task r; it is rather simple for the numerical integration (46), assuming conditions (66) [re-examine conditions (47)–(50)]. In those stages of Turan–Weyl's isolation algorithm we do not require to increase the precision of computations more than by a constant factor. Let us focus on the errors of finite precision of computations in the stages (a) of the shift of the variable x by X and (b) of other computations of Turan's test, that is, of computing

$$\max_{g=1,\ldots,n} |s_{gN}/n|^{1/(gN)}].$$

We may truncate the mantissas of $p_0, \ldots, p_n$, scale $p(x)$, and assure that the input coefficients $p_0, \ldots, p_n$ of $p(x)$ are integers. Let

$$\max_j |p_j| < 2^m, \quad \epsilon = 2^{-h}, \quad b = m + h, \tag{67}$$

compare conditions (3), (4) and Theorem 1 of Ref. [10].

Stage (*a*). We will assume that the shift values X are such that both real and imaginary parts of $2^{h+2}X$ are integers. (Otherwise we would change X respectively, increasing the absolute output error bound less than by $\epsilon = 2^{-h}$; we will assume also that $|X| < 2^m$, see Theorem 1 of Ref. [10].) Then the coefficients of the polynomial $q(y)$ of equation (7) are integer multiples of $1/2^{(h+2)n}$ [that is, they take the form $H/2^{(h+2)n}$ where H is an integer], so it is sufficient to compute them with absolute errors less than $1/2^{(h+2)n+1}$ and to recover their exact values via rounding-off. Following Ref. [43], we reduce computing the shift of x to convolution of two vectors whose entries have absolute values $< 2^{bn}$ (compare condition (67) and Ref. [43] or equation (2.3) of Ref. [9]), so that $O(bn)$ bit-precision of computations will suffice. (This follows if we reduce the convolution to FFT whose error analysis is available, see Refs [30, p. 194; 44], or alternatively to integer multiplication [28, 29], whose error estimates are available in Ref. [27].)

Stage (*b*). Consider the evaluation of the power sums via the iterations (8) and via solving the system (10). O(1) iterations (8) suffice (we reduce them to convolutions and DFTs). That few iterations (8) may require to increase the precision not more than O(1) times (compare the error estimates from Refs [2, 27, 44], so we will only analyze the errors of the solution of the triangular Toeplitz systems (10). Let s denote an entry of the inverse of an $n \times n$ unit triangular Toeplitz matrix T. Then (see Ref. [30, Lemma 2, p. 192]) $|(\log|s| = O(n \log(1 + t))$, $\log(\log|s|^{1/n}) =$

$O(\log(1+t))$, where t denotes the maximum absolute value of an entry of T. In fact the diagonal entries of the coefficient matrix of the system (10) equal $p_{n,k} \neq 0$, which is the Nth power of the leading coefficient $p_0(X)$ of the polynomial $q(y) = y^n p(X + 1/y)$ of condition (7). We will keep our previous assumptions [see part (a) above] that the coefficients of $p(x)$ are m-bit integers between -2^m and 2^m and that X has real and imaginary parts of the form $j/2^{h+2}$ for an integer j. Then all the coefficients of $q(y)$ are integer multiples of $1/2^{(h+2)n}$, and it suffices to scale the system by $2^{(h+2)nN}$, which means the increase of the precision by $(h+2)nN = O(bn)$ binary bits. Thus, due to the estimate from Ref. [30], it suffices to compute with the precision of $O(bn)$ bits in order to control the errors in Stage (b). (We could arrive at a similar, and actually at even a more favorable, result if we used Cauchy's integrals (11) in order to compute $|s_{gN}|^{1/(gN)}$, compare conditions (48)–(50) and Ref. [2, p. 34]).

Summarizing we conclude that the computations with $O(bn)$ binary bit precision suffice to support the applications of Algorithm 3.1, as well as other steps of Turan–Weyl's isolation algorithm.

Similarly we may deduce that Lehmer–Turan's algorithms of Sections 3 and 10 and Turan–Weyl's algorithm of Section 5 only require computations with $O(bn)$ bit-precision. In the case of Lehmer–Turan's isolation algorithm an order of $\log\log n$ successive iterations (8) may be needed to raise the zeros of $p(x)$ to the powers of an order of $O(n)$ while solving Task r of Section 4; the required estimates for the errors and for the precision of such computations can be found in Ref. [2].

We may combine our arithmetic complexity estimates with the known bounds on the Boolean circuit complexity of arithmetic operations over integers mod 2^B [that is, with the estimates $O_B(t(B))$ for the Boolean sequential time and $O_B(\log^2 B, t(B))$ for the Boolean parallel cost where $t(B)$ is defined by equation (1)]. We apply those bounds with $B = O(bn)$, multiply the entries of Tables 1 and 2 by $t(B)$ or by $\log^2 B$, respectively, and arrive at the Boolean circuit complexity estimates for the problems of computing polynomial zeros. The previous works on polynomial zeros do not present such estimates explicitly, but actually all of them imply inferior bounds (exceeding ours at least by a factor of n or so). The only exception is the estimates of Ref. [2], stated in Theorem 19.2 for the sequential complexity of approximating to all the zeros of $p(x)$ on the Boolean circuits and implicit in the case of a single zero of $p(x)$. The proof of those bounds in Ref. [2] is long, involved, and so far remains uncompleted. We will outline another way of deducing and slightly refining those estimates in Section 12.3, but already in the present section we almost reached the bounds of Ref. [2] [up to within a factor of $\log n \log(bn)$ or less, depending on the value b)]; the proof of those slightly inferior estimates is much simpler than the proof of the bounds of Ref. [2].

Remark 11.1

We use the worst case estimate $B = O(bn)$ for the precision of computations supporting our arithmetic cost bounds; for many input polynomials $p(x)$, the precision bound of an order of bn is overly pessimistic; it can be actually decreased by a factor of n. Even in the worst case, we may compute with precision lower roughly by a factor of n (so that the sequential and parallel Boolean complexity estimates decrease respectively) if we apply the *same algorithms* not in order to compute the zeros of $p(x)$ but in order *to factor* $p(x)$ *numerically*, that is, to compute complex u_j and v_j such that all the coefficients of the polynomial

$$p(x) - \prod_{j=1}^{n} (u_j x + v_j)$$

have absolute values $< \epsilon$, compare Theorems 2.1 and 19.2 of Ref. [2].

12. SOME ALTERNATIVES, IMPROVEMENTS AND COMMENTS ON THE TECHNIQUES USED AND OPEN PROBLEMS

We may modify our algorithms in a number of ways and arrive at the same overall estimates for the asymptotic arithmetic and Boolean complexity (within polylogarithmic factors). Let us consider some of such modifications.

12.1. Some alternatives to Turan's test

The solution of Task s of Section 4 for $s = n$ via Turan's test and Theorem 3.1 can be replaced by the solution relying on relation (18). This would slightly increase the arithmetic cost of the solution, to $O_A(\log n \log\log n, n)$, but the proof of relation (18) is simpler than the proof of Theorem 3.1. In the applications of Section 6, we can see some other alternatives to (but not improvement of) Turan's Algorithm 3.1; in paritcular, winding number algorithms can replace Turan's test in some applications where the input square has larger isolation ratio; Schur–Cohn test [13, pp. 508–509], can replace Turan's in Section 9, although this would increase both sequential and parallel arithmetic time bounds by a factor of $n/\log n$.

12.2. Approximating to the zeros of a polynomial via its recursive splitting into factors

In this section we will modify our algorithms via recursive splitting of $p(x)$ into factors and then (see also Section 12.3) will (slightly) improve our arithmetic and Boolean cost bounds.

Let a disc D contain exactly k zeros of $p(x)$, that is, $x_{j(1)}, \ldots, x_{j(k)}$, where $0 < k < n$. Let

$$f(x) = \prod_{h=1}^{k} (x - x_{j(h)}), \quad g(x) = p(x)/f(x). \tag{68}$$

We will call such a factorization of $p(x)$ its *splitting over the disc D*. Suppose that we recursively split at first $p(x)$, then its factors $f(x)$ and $g(x)$ [or only one of them of the minimum degree if we only need to compute a single zero of $p(x)$], and so on (each time we split the polynomials over appropriately chosen discs), until linear factors give us the desired zeros of $p(x)$. Extending the earlier study of such an approach (see Refs [45, pp. 295–320; 46], Schönhage in Ref. [2 Chapt. 3], suggests some effective algorithms that approximately compute the coefficients of the factors f and g of $p(x)$ over a disc $D = D(0, R)$ with an error bound ϵ^*/n^n. The cost of those algorithms is

$$c_A(n) = O_A(\log n \log(bn), n \log b/\log(bn)) \tag{69}$$

provided that $bn = \log_2(R/\epsilon^*)$ and that, say i.r. $(D) \geqslant 2$, see the end of this section. Furthermore, Ref. [2] proves that the above bound ϵ^*/n^n on the errors of the coefficients of f and g suffices in order to assure the output error bound ϵ^* on the computed approximations to the coefficients of the *linear* factors of $p(x)$ and then on the desired output error ϵ [such that $b = O(\log(R/\epsilon))$] of the resulting approximations to the zeros of $p(x)$.

The inequality i.r. $(D) \geqslant 2$ holds for a disc $D = D(Y, R\sqrt{2})$ if i.r. $(S(Y, R) \geqslant 4$. The algorithms of our paper recursively supply squares $S(Y, R)$ with isolation ratios $\geqslant 4$, so we may incorporate such recursive splittings there. If we ignore the cost of computing all the splittings, that modification would surely decrease the cost of the resulting computations, for we will deal with polynomials whose degrees decrease in each iteration of our algorithms; furthermore Algorithm 7.1 will be greatly simplified: it will be essentially reduced to the case $k = n$, where the center of gravity M of equation (36) is given by the ratio of the two leading coefficients of $p(x)$ divided by the degree, compare Remark 7.1. Then, as before, we will compute the first (for $n + 1 - k = 1$ in this case) root radius at M and apply Subalgorithm 6.1. Already the outputs of the first application of Subalgorithm 6.1 will satisfy relations (30) and (35), due to Proposition 7.2. The cost of such an iteration (including the cost of the evaluation of M and of the root radius, as well as the cost of the application of Subalgorithm 6.1) is only $O_A(\log n, n)$ and is dominated by $c_A(n)$. In the result, the overall cost of the algorithm will decrease, although (as we will see) only slightly if we count also the cost of the recursive splitting itself.

For similar reasons, the contractions of the output squares $S(X, \rho)$ of Subalgorithm 10.1 in Lehmer–Turan's isolation algorithm of Section 10 will be greatly simplified, so the overall cost of all such contractions will be dominated by the cost of the applications of Subalgorithm 10.1, supporting Proposition 10.1. Therefore the overall cost bound for Lehmer–Turan's isolation algorithm will equal the sum of the cost bounds of Proposition 10.1 and of the bounds on the cost of all the splittings required. There can be at most $\log_2 n$ of them, due to the decrease of the degrees by at least 50% in each splitting. Slowing down parallel computations as in the proof of Theorem 9.2, we obtain that all the applications of Subalgorithm 10.1 contribute

$$c_A^*(n) = O_A(\log^2 n \log\log n, n/\log n) \tag{70}$$

to the overall cost bound. The cost of all the $H \leqslant \log_2 n$ splittings is

$$c_A^H = \sum_{h=0}^{H} c_A(2^h),$$

see equation (69). Slowing down the computations, we replace $c_A(2^h) = O_A(h(h + \log\ b), 2^h \log\ b/(\log\ b + h))$ by $O_A(h2^h/n)\log\ n(\log\ b + h), (n/\log\ n)\log\ b/(\log\ b + h))$ for $h = 0, 1, \ldots, H$. Then c_A^H is replaced by $O_A(\log^2 n\ \log(bn), (n/\log\ n)\log\ b/\log(bn))$. The latter bound dominates equation (70).

Similarly we deduce that the cost bounds for the $O(n)$ Turan–Weyl's iterations with splitting are $O_A(n \log\ n, n)$ at the stages of computing all the discs D and $O_A(n \log\ n \log\ (bn), n \log\ b/\log(bn))$ at the stages of recursive splitting of $p(x)$ given all the discs D. The latter cost bound dominates the former one.

Summarizing we arrive at the following estimates, slightly improving the bounds of Theorems 9.1 and 10.1, compare Remark 9.1.

Theorem 12.1

Let positive ϵ, R and b satisfy conditions (3) and (4). The isolated ϵ-approximations to all the zeros of $p(x)$ can be computed for the parallel cost $O_A(n \log\ n \log\ (bn), n \log\ b/\log\ (bn)))$ and for the sequential cost $O_A(n^2 \log\ b \log\ n)$, while an isolated ϵ-approximation to a single zero of $p(x)$ can be computed for the parallel cost $O_A(\log^2 n\ \log\ (bn)), (n/\log\ n)\log\ (bn))$ and for the sequential cost $O_A(n \log\ b \log\ n)$. These bounds are compatible with $O(bn)$ bit-precision of computations with respective implication on the bounds on the Boolean circuit complexity of computing all the zeros and a single zero of $p(x)$.

In the remainder of this section we will trace how equation (69) can be deduced from Ref. [2, Chapt. 3], and will indicate the single modification of the construction of Ref. [2] required for that. We will assume that we split $p(x)$ itself over a disc D, to use our previous notation. Splitting of $p(x)$ begins with computing the power sums s_K for $K = 1, 2, \ldots, n + 1$ via numerical integration (51), say along the boundary Γ of the disc $D^* = D(Y, \sqrt{2}r)$ [which should replace the disc $D(X, R)$ of expressions (51) and (52)]. Here $D = D(Y, r)$ and $\text{i.r.}\,(D) \geqslant 2$. [Then $\text{i.r.}\,(D^*) \geqslant \sqrt{2}$, $\text{r.r.}\,(D^*) \geqslant \sqrt{2}$, $D^* \supset D$.] After replacing the integrals (11) by the integral sums (51) with Q equally spaced nodes on the circumference Γ and after shifting and scaling the variable x, we can reduce the numerical integration to three DFTs at Q points. Two DFTs suffice in order to compute $p(x)$ and $p'(x)$ at the Q nodes, which after shifting and scaling the variable can be represented as ω^g, $g = 0, 1, \ldots, Q - 1$, where $q(y)$ denotes the polynomial obtained from $p(x)$ via shifting and scaling the variable x. Q should be sufficiently large to guarantee the desired precision of approximation to the power sums. The error estimates for the numerical integration (51) in Ref. [2, Sect. 12], rely on the formula (52). Based on those estimates, Ref. [2] requires Q to be of an order of n^2, but this is only needed to handle the case where $\text{i.r.}\,(D^*) \geqslant 1 + 1/O(n)$, which corresponds to $g = 1 - 1/O(n)$ in equation (49). In our case $\text{i.r.}\,(D^*) \geqslant \sqrt{2}$; this corresponds to $g \leqslant \sqrt{2}/2 < 0.71$ in equation (49). In that case it suffices to choose $Q = O(n)$ using the same formula (52). Then the cost of the approximate evaluation of all the power sums s_K is $O_A(\log\ n, n)$.

Finally let us very briefly indicate how equation (69) follows from that development. For the cost $O_A(\log^2 n, n/\log\ n)$ the algorithm of Ref. [2, Sect. 13], recovers the approximations to the coefficients of the factors $f(x)$ from the approximations to the power sums; the errors of those approximations are kept sufficiently small in order to enable us to start the refinement of those approximations via Newton's iterations of Sections 10 and 11 of Ref. [2]. Each such an iteration is reduced to few multiplications and divisions of polynomials of degrees $\leqslant n$, so its cost is $O_A(\log\ n, n)$. $O(\log\ b)$ such iterations suffice to output the coefficients of $f(x)$ and $g(x)$ with error bound $\leqslant \epsilon$.

12.3. Decreasing the Boolean circuit bounds

Estimating the Boolean circuit complexity in Section 11 and in Theorem 12.1 we relied on the customary pattern of implementing each finite precision arithmetic operation on Boolean circuits and assumed the same finite precision $B = O(bn)$ for each arithmetic operation. In this section we will slightly improve those estimates for the Boolean circuit complexity (by factors varying from

$\log n$ to $\log^2 n$). We will rely on the algorithms already used in the previous sections and on their more clever implementation on Boolean circuits. A factor of $\log n$ improvement immediately follows if we apply the specialized algorithms of Refs [30, 33] in order to implement (on Boolean circuits) the operations of polynomial multiplication and division, DFT, and shift of the variable, which are used in our algorithms as the basis blocks. Those specialized algorithms ultimately rely on the customary reduction of each of the above operations to multiplication of two polynomials with integer coefficients [23–25] in turn reduced to a single multiplication of long integers (the latter step follows Refs [28, 29]).

Our arithmetic complexity estimates of the previous section suggest that for a further decrease of the Boolean cost of Turan–Weyl's isolation algorithm it would suffice to decrease the Boolean cost of its stages of recursive splitting over given discs D, because the asymptotic cost of computing the splitting discs D is already of a lower order than the overall cost.

Such a decrease indeed immediately follows if we complement Turan–Weyl's isolation algorithm by Theorem 12.1 of Ref. [2] and by its further improvement from Section 13 of Ref. [2]. The latter results supply upper estimates for the sequential Boolean cost $t_B(n)$ of approximating to the coefficients of $f(x)$ and $g(x)$ of equation (68), so that $\|p(x)-f(x)g(x)\|_1 < 2^{-B}\|p(x)\|_1$, where

$$\left\| \sum_{i=0}^{h} u_i x^i \right\|_1 \quad \text{denotes} \quad \sum_{i=0}^{h} |u_i|.$$

Specifically Ref. [2] proves that

$$t_B(n) = O_B(H(n)\log H(n)\log\log H(n)), \tag{71}$$

where

$$H(n) = k(B+k) + (1/\delta)(n + |\log_2 \mu|)^2 + nB;\ \mu = \min_{x\in\Gamma} |p(x)|,$$

Γ being the boundary of the disc D^* over which $p(x)$ is splitted [we may always turn D^* into $D(0, 1)$ via shifting and scaling the variable x], and $\delta = \ln m(D^*)$, $m(D^*) = \min$ {i.r. (D^*), r.r. (D^*)}. That bound has already incorporated an order of $\log n$ improvement of the Boolean cost bound due to using the specialized algorithms cited above. [The proof of the above refinement of Theorem 12.1 of Ref. [2] is much simpler than the entire proof of the main result of Ref. [2], which essentially amounts to the bound (74) below.) In our case $m(D^*) \geqslant \sqrt{2}$, $\delta \geqslant \ln\sqrt{2} > 0.3$, $\mu \geqslant (\ln\sqrt{2}/(2\sqrt{2}))^n > 0.1^n$, $k \leqslant n$, so $H(n) = O(n^2 + nB)$, $t_B(n) = O_B((n^2 + Bn)\log(Bn)\log\log(Bn))$. Substitute here $B = O(nb)$ [which will suffice to assure the final output error bound ϵ for the approximations to the zeros of $p(x)$] and arrive at the bound

$$t_B(n) = O_B(bn^2\log(bn)\log\log(bn)). \tag{72}$$

Let $T_B(n)$ denote the sequential Boolean time of Turan–Weyl's exclusion algorithm (with recursive splitting) applied to an nth degree polynomial, *excluding* the time required for computing the discs D needed for the recursive splittings. Then

$$T_B(n) \leqslant \max_{0<k<n} (T_B(k) + T_B(n-k) + t_B(n)).$$

Combine the latter bound on $T_B(n)$ with the bound (72), and deduce (by induction on n) the following estimate:

$$T_B(n) = O_B(bn^3\log(bn)\log\log(bn)). \tag{73}$$

In fact estimate (73) also includes the bound on the sequential Boolean time for computing all the splitting discs D. Indeed, recall the arithmetic cost bound $O_A(n\log n, n)$ of those computations, see Section 12.2. Using precision $B = O(bn)$ and the cited specialized algorithms for polynomial arithmetic, DFT, and the shift of the variable, we extend that bound to the Boolean bound of estimate (73).

Estimate (73) is close to the bound of the main theorem of Ref. [2]

$$T_B(n) = O_B((b + \log n)\, n^3\log(bn)\log\log(bn)). \tag{74}$$

Our derivation of bound (73) is simpler than Schönhage's proof of bound (74). To our advantage, we use Turan–Weyl's construction, which immediately supplies the discs D^* for splitting polynomials such that i.r. $(D^*) \geqslant \sqrt{2}$, while in the construction of Ref. [2] the splitting discs D^* are computed in a more complicated way, and only the lower bound $1 + 1/O(n)$ on their isolation ratios is assured. With such a lower bound (lower than our $\sqrt{2}$) further substantial complications (implying also the increase of the overhead constants) follow where a certain lower bound on μ is required for the transition from equation (71) to equation (74).

In the case of computing a single zero of $p(x)$ via Lehmer–Turan's construction with recursive splitting, the degrees of the polynomials to be splitted decrease by at least 50% in each splitting stage. Therefore equation (72) implies the upper bound $2t_B(n)$ on the sequential Boolean time required for all the splittings within Lehmer–Turan's construction. Taking into account the decrease of the degree in each splitting by at least 50%, we also arrive at the bound $t_B^*(n) = O(t_A^*(n)\, bn \log(bn) \log\log(bn)/\log n)$ on the sequential Boolean time required for all the applications of Subalgorithm 10.1. Here $t_A^*(n) = O(n \log n \log\log n)$ denotes the arithmetic sequential time required for all those splittings and defined as the product of the time and processor bounds in the estimate (70) for $c_A^*(n)$ in Section 12.2; the factor $O(bn \log(bn) \log(bn)/\log n)$ represents the average Boolean time for an arithmetic operation in the cited specialized algorithms for polynomial multiplication and division, shift of the variable, and DFT. Summarizing, we arrive at the following bound on the sequential Boolean time required in order to approximate to a zero of $p(x)$ with absolute error $\leqslant \epsilon$ (using Lehmer–Turan's isolation algorithm with splitting)

$$T^*_B(n) = O_B(bn^2 \log \log n \log(bn) \log\log(bn)). \tag{75}$$

For comparison, here is the bound implicit in Ref. [2] [and slightly inferior to equation (75)]

$$T^*_B(n) = O_B((b+n)n^2 \log n \log(bn) \log\log(bn)).$$

As a challenge to the reader, we leave the problems of possible further improvement of the bounds (73) and (75) [say by a factor of $\log\log n$ in bound (75)] and of their rather simple extensions to the parallel Boolean circuit complexity bounds.

12.4. A zero of higher order derivative vs the center of gravity

Algorithm 7.1 can be modified if we replace the center of gravity M by the (unique) zero z of the $(k-1)$th derivative of $p(x)$ lying in the input disc D, provided that i.r. $(D) \geqslant 75n^5$. The respective extension of Proposition 7.2, combined with the result on the rapid convergence of Newton's iterations to z, is supplied in the following nontrivial theorem.

Theorem 12.2 [11]

(a) Let $p(x)$ be a polynomial of a positive degree n and let k, r, R, X be four numbers such that $R \geqslant 15\, n^3 r > 0$ and each of the discs $D(X, r)$ and $D(X, R)$ contains exactly k zeros of $p(X)$, counted with their multiplicities. Then $p^{k-1}(x)$, the $(k-1)$th order derivative of $p(x)$, has unique zero z in the discs $D(X, 3nr/2)$ and $D(X, R/(10n^2))$. (b) Furthermore, if the disc $D(X, 5n^2 R)$ also contains only the same k zeros of $p(x)$, then Newton's iterations

$$x_{i+1} = x_i - g(x_i)/g'(x_i), \quad i = 0, 1, \ldots,$$

applied to the function $g(x) = p^{(k-1)}(x)$ with the initial point $x_0 = X$, converge to z, and $|x_i - z| \leqslant 2^{3-2^i} |X - z|$ for all positive i.

Comparing with the cost of computing the center of gravity M of equation (36), this means the same sequential time bound but a slightly larger (by a factor of $\log n$) parallel time bound. Note also a higher isolation ratio of disc D required in Theorem 12.2.

The proof of part (b) of Theorem 12.2 in Ref. [11] relies on the result of Ref. [47] and Ref. [5] that guarantees the quadratic convergence of Newton's iterations to a zero of $p(x)$ assuming certain relations for the high order derivatives of $p(x)$. The proof of part (a) relies on the following well known formula

$$p^{(k-1)}(x)/p(x) = \sum \prod_{h=1}^{k-1} 1/(x - x_{j(h)}), \tag{76}$$

see Ref. [13]. Here the summation is over all the ordered k tuples of distinct integers from 1 to n.

In the case where $k = n$, we have that $p^{(n-1)}(z) = p_n n!z + p_{n-1}(n-1)!$, so $z = M = -p_{n-1}/(np_n)$ (compare Remark 7.1), and computing z and M becomes equally simple. [This will be the case also if the recursive splitting of $p(x)$ over the discs is applied.] We will conclude with a simple proof of a weaker version of part (a) of Theorem 12.1, which will demonstrate some ideas used in the proof of part (a) of that theorem.

Proposition 12.1

Let X be a complex number, let $kR > (2n-k)r$, and let exactly k zeros of $p(x)$ lie in each of the two discs $D(X, R)$ and $D(X, r)$. Then exactly $k-1$ zeros of the derivative $p'(x)$ lie in each of the discs $D(X, r)$ and $D(X, R_1)$, where

$$nR_1 = kR - (n-k)r.$$

Proof. At first let us prove that the discs $D(X, r)$ and $D(X, R_1)$ contain the same number of zeros of $p'(x)$ or equivalently that $p'(w) \neq 0$ for an arbitrary point w that lies in the annulus

$$r < |X - w| \leqslant R_1. \tag{77}$$

Let w be a point in that annulus. Assume that $w \geqslant 0$, $X = R_1$. Otherwise shift and scale the variable x to assure those relations. Recall that

$$p'(x)/p(x) = \sum_{j=1}^{n} 1/(x - x_j), \tag{78}$$

see Ref. [13]. Denote $q = R_1 - r$. Then $\mathrm{Re}(1/(w - x_j)) > q/(R_1 + r)$ if $x_j \in D(X, r)$; $\mathrm{Re}(1/(w - x_j)) \leqslant q/(R - R_1)$ if $x_j \in D(X, R)$. Therefore $\mathrm{Re}(p'(w)/(p(w)) > kq/(R_1 + r) - (n-k)q/(R - R_1) \geqslant 0$. Thus $p'(w) \neq 0$ for any point w lying in the annulus (77). Apply homotopic transformation of $p(x)$ into the polynomial $p(x)$ having k-multiple zero at X and the same $n-k$ zeros outside of $D(X, R)$ as $p(x)$ has. Deduce that the derivatives of $p(x)$ and $q(x)$ have the same number of zeros in $D(X, r)$ (counted with their multiplicities), that is, $k-1$. Q.E.D.

Proposition 12.1 can be applied recursively to $p(x)$, $p'(x)$, $p''(x)$, and so on, yielding the following result, which can be improved basing on the (more involved) direct application of equation (76) rather than on the recursive application of equation (78).

Corollary 12.1

Under the assumptions of Theorem 12.1, each of the discs $D(X, r)$ and $D(X, R_j)$ contains exactly $k-j$ zeros of the jth order derivative of $p(x)$, where

$$R_0 = R, \quad (n-1)R_{i+1} = (k-i)R_i - (n-k)r, \quad i = 0, 1, \ldots, j-1; \quad j \leqslant k.$$

12.5. Comments on the techniques used

In Table 3 we compare the main algebraic and numerical techniques used in Refs [2, 11] and in this paper. Our geometric construction of Sections 6–9 has some common feature with Weyl's, Schönhage's and Renegar's geometric constructions, being, however, distinct from all of them. Our construction of Section 10 is novel, with only minor borrowing from Lehmer's.

Schönhage himself defines his approach in Ref. [2] as a "splitting circle method", which "has been described in Ref. [45, pp. 543–560] already, without *a priori* bounds, however,". Renegar emphasizes in the summary and throughout the paper [11] that his algorithm "is built around Newton's method and Schur–Cohn algorithm".

12.6. Some open problems

The major open problem is whether computing all the complex zeros of $p(x)$ is in NC, that is, whether it is possible to compute all the complex zeros of $p(x)$ for the parallel cost $O_B(\log bn)^{O(1)}$, $(bn)^{O(1)}$) or $O_A((\log(bn))^{O(1)}, (bn)^{O(1)})$, where b is defined by conditions (3) and (4). (Positive answer is given in Corollary 5.1 in the simple case where $b \leqslant (\log n)^{O(1)}$ and in Ref. [10] in the case where

Table 3

Techniques used	Ref. [2]	Ref. [11]	This paper
Turan's power sum	No	No	Yes
Numerical integration	Only for splitting	No	Yes
Recursive splitting of $p(x)$ (via integration and Newton's refinement)	Yes	No	Optional (for minor improvements of the results)
Newton's iterations (without splitting)	No	Yes	Optional (with a minor drawback)
Schur–Cohn test	No	Yes	No

all the zeros are real.) It is also interesting (at least theoretically) if the root radius estimate of Ref. [14] can be extended to cover also the approximation to the sth root radius of $p(x)$ for $1 < s < n$ and if the bounds of the last lines of Tables 1 and 2 on the arithmetic circuit complexity and the bounds (73) and (75) on the Boolean circuit complexity can be further decreased. Finally, is it possible to improve the lower bound on the arithmetic sequential time from $\Omega(n + \log b)$ to, say $\Omega(n \log b)$ or $\Omega(n \log (nb))$?

Acknowledgements—Supported by NSF Grant DCR 8507573. The author wishes to thank Sally Goodall for typing this paper.

REFERENCES

1. S. Smale, The fundamental theorem of algebra and complexity theory, *Bull. AMS* **4**(1), 1–36 (1981).
2. A. Schönhage, The fundamental theorem of algebra in terms of computational complexity, Manuscript, Dept. of Math., University of Tübingen, Tübingen, West Germany (1982).
3. V. Pan, Fast and efficient algorithms for sequential and parallel evaluation of polynomial zeros and of matrix polynomials. *Proc. 26th Ann. IEEE Symp. on Foundation of Computer Sci.* 522–533, Portland, Oregon (1985).
4. S. Smale, On the efficiency of the algorithms of analysis, *Bull. Am. math. Soc.* **13**(2), 87–121 (1985).
5. S. Smale, Newton method estimates from data at one point, *Proc. Conf. in Honor of Gail Young*, Springer, New York (1986).
6. S. Smale, Algorithms for solving equations, *Proc. of Int. Congr. Math.*, Berkeley (1987).
7. M. Shub and S. Smale, Computational complexity: on the geometry of polynomials and a theory of cost, Part I, *Annls Scient. ecole norm. sup.* **4**(18), 107–142 (1985).
8. M. Shub and S. Smale, Computational complexity: on the geometry of polynomials and a theory of cost, Part II, *SIAM J. Computing* **15**, 145–161 (1986).
9. V. Pan, Algebraic complexity of computing polynomial zeros, *Comput. Math. Applic.* **14**(4), 285–304 (1987).
10. M. Ben-Or, E. Feig, D. Kozen and P. Tiwari, A fast parallel algorithm for determining all roots of a polynomial with real roots, *Proc. 18th Ann. ACM Symp. on Theory of Computing*, pp. 340–349 (1986).
11. J. Renegar, On the worst-case arithmetic complexity of approximating zeros of polynomials, *Proc. 2nd Symp. on the Complexity of Approximately Solved Problems*, Columbia Univ., New York (1987).
12. A. S. Householder, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill, New York (1970).
13. P. Henrici, *Applied and Computational Complex Analysis*, Wiley, New York (1974).
14. P. Turan, *On a New Method of Analysis and Its Applications*, Wiley, New York (1984).
15. P. Turan, On the approximate solution of algebraic equations, (in Hungarian), *Comm. Math. Phys. Class Hung. Acad.* **XVIII**, 223–236 (1968).
16. P. Turan, Power sum method and approximative solution of algebraic equations, *Math. Computation* **29**(129), 311–318 (1975).
17. I. Gargantini and P. Henrici, Circular arithmetic and the determination of polynomial zeros, *Num. Math.* **18**, 305–320 (1972), also in Ref. [45, pp. 77–114].
18. A. Borodin, S. Cook and N. Pippenger, Parallel computation for well-endowed rings and space-bounded probabilistic machines, *Inf. Control* **58**, 1–3, 113–136 (1983).
19. A. Borodin, J. von zur Gathen and J. Hopcroft, Fast parallel matrix and GCD computation, *Inf. Control* **53**(3), 241–256 (1982).
20. S. A. Cook, The classification of problems which have fast parallel algorithms, *Int. Conf. on Foundations of Computation Theory*, Borgholm (1983).
21. Jia Wei Hong, How fast the algebraic approximation can be? *Scientia sin.* **XXIXA**(8), 813–823 (1986).
22. M. Mignotte, Some inequalities about univariate polynomials, *Proc. 1981 ACM Symp. on Symbolic and Algebraic Computation*, pp. 195–199 (1981).
23. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1976).
24. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York (1975).
25. D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison-Wesley, Reading, Mass. (1981).

26. J. E. Savage, *The Complexity of Computing*, Wiley, New York (1976).
27. A. Schönhage, Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients, *Proc. EUROCAM*, Marseille (1982).
28. M. J. Fischer and M. S. Paterson, String matching and other products, *SIAM-AMS Proc.* **7**, 113–125 (1974).
29. V. Pan, The bit-operation complexity of the convolution of vectors and of the DFT, Technical Report 80-6, Computer Science Dept., SUNYA, Albany New York (1980). Also abstract in *Bull. EATCS*, **14**, 95 (1981).
30. D. Bini and V. Pan, Polynomial division and its computational complexity, *J. Complexity* **2**, 179–203 (1986).
31. M. A. Jenkins and J. F. Traub, A three-stage variable shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. *Num. Math.* **14**, 252–263, (1970).
32. V. Pan and J. Reif, Displacement structure and the processor efficiency of fast parallel Toeplitz computations, Technical Report 87–9, Computer Science Dept., SUNY Albany, New York (1987). [Short version in *Proc. of 28th Annual IEEE Symp. FOCS*, pp. 173–184 (1987).]
33. S. Schönhage, Quasi-gcd Computations, *J. Complexity* **1**, 118–137 (1985).
34. J. Renegar, On the worst case arithmetic complexity of approximating zeros of systems of polynomials, Technical Report 748, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York (1987).
35. B. L. Van der Waerden, *Modern Algebra, 2,* Frederick Ungar, New York (1950).
36. V. Pan, Complexity of parallel matrix computations, *Theor. Comput. Sci.* (in press).
37. W. Keller-Gehrig, Fast algorithms for characteristic polynomial, *Theoretical Computer Science* **36**, 309–317 (1985).
38. D. H. Lehmer, A Machine model for solving polynomial equations, *J. Ass. comput. Mach.* **8**, 151–162 (1961).
39. M. J. Atallah and M. T. Goodrich, Efficient parallel solution to some geometric problems, *J. Parallel Distributed Comput.* **3**, 492–507 (1986).
40. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C. Yap, Parallel Computational Geometry, *Proc. 25th Ann. IEEE Symp. Foundations of Computer Science*, pp. 468–477 (1985).
41. F. P. Preparata and M. I. Shamos, Computational geometry: an introduction, *Texts and Monographs in Computer Science*, Springer Verlag, New York (1985).
42. K. E. Atkinson, *An Introduction to Numerical Analysis*, Wiley, New York (1978).
43. A. V. Aho, K. Steiglitz and J. D. Ullman, Evaluating Polynomials at fixed set of points, *SIAM Jl Comput.* **4**, 533–539 (1975).
44. W. Gentleman and G. Sande, Fast Fourier transform for fun and profit, *Proc. Fall Joint Comput. Conf.* **29**, 563–578 (1966).
45. B. Dejon and P. Henrici (Eds) *Constructive Aspects of the Fundamental Theory of Algebra*, Wiley, London (1969).
46. D. Y. Y. Yun, Algebraic algorithms using p-adic constructions, *Proc. 1976 ACM Symp. Symbolic and Algebraic Computation*, pp. 248–259 (1976).
47. M. Kim, Ph.D. Thesis, Graduate School, City University of New York (1985).