Some Polynomial and Toeplitz Matrix Computations

Victor Pan
Computer Sci. Dept.
SUNY Albany, Albany, NY 12222
Supported by NSF Grant DCR 85-07573)

and

John Reif
Computer Sci. Dept
Duke University, Durham, NC 27706
(Supported by NSF Grant DCR 85-03151

We will break our study of computations with polynomials and Toeplitz matrices into two parts, where we cover our papers [24] and [23] respectively.

Part 1. Approximate Evaluation of Polynomial Zeros

$O(n^2(\log^2 n+\log b))$ arithmetic operations or $O(n(\log^2 n+\log b))$ parallel steps, n processors suffice in order to approximate with absolute error $\leq 2^{m-b}$ to all the complex zeros of an n-th degree polynomial p(x) whose coefficients have moduli $\leq 2^m$. If we only need such an approximation to a single zero of p(x), then $O(n \log n(n+\log b))$ arithmetic operations or $O(\log n(\log^2 n+\log b))$ steps and $n+n^2/(\log^2 n+\log b)$ processors suffice (which places the latter problem in NC); furthermore if all the zeros are real, then we arrive at the bounds $O(n \log n(\log^3 n+\log b))$, $O(\log n(\log^3+\log b))$, and n, respectively. Those estimates are reached in computations with O(nb) binary bits where the polynomial has integer coefficients. This also implies a simple proof of the Boolean circuit complexity estimates for the approximation of all the complex zeros of p(x), announced in 1982 and partly proven by Schönhage. The computations rely on recursive application of Turan's proximity test of 1968, on its more recent extensions to root radii computations, and on contour integration via FFT within our modifications of the known geometric constructions for search and exclusion.

Part 2. Toeplitz Computations and Applications

We compute the inverse, determinant, and characteristic polynomial of an $n \times n$ Toeplitz matrix T using $O(\log^2 n)$ parallel arithmetic steps, n^2 processors, and the precision of computation of $O(n \log(n\|T\|))$ binary bits. Our new algorithm substantially improves the known parallel methods for all the above problems having polylogarithmic parallel time; the known processor bounds are decreased by a factor of n; even applying our parallel algorithm as a sequential algorithm for computing a characteristic polynomial of a Toeplitz matrix, we stay at the level of the current best sequential time bound. We also slightly improve the known sequential algorithms for the exact evaluation of the inverse of integer Toeplitz matrices (via Newton-Hensel's lifting algorithm) and extend our method to computations with block Toeplitz and some other structured matrices. As an example of several immediate applications of our results, we evaluate the greatest common divisor of m polynomials of degrees at most n with integer coefficients in the range from -t to t using $O(\log^2 n \log(m+n))$ parallel arithmetic steps and mn^2 processors and computing with the precision of $O(n \log(mnt))$ binary bits. The techniques that we use include: i) the reduction of matrix powering to matrix inversion and FFT, ii) a combination of Newton's iteration with some displacement structure theorems, iii) the variable diagonal technique; and iv) Csanky-LeVerrier's parallel algorithm with an improved solution of the system of Newton's identities.

1. Introduction to Parts 1 and 2

In the vast bibliography on the evaluation of complex polynomial zeros, only few works specifically address the complexity issue, see [22, 28] for surveys. The complexity estimates in those papers have been obtained via several different algorithms, relying on Newton's iterations and on the various techniques of computing power sums and contour integrals. Those three basic approaches and techniques have also been manipulated with in different ways and combined with each other and with some geometrical constructions for search and exclusion in complex domains, due to Weyl, Lehmer, Runge. In particular Smale and then Shub and Smale, proved that Newton's method is highly effective in the average case, see [28], while Turan's power sum method of 1968 and its recent extensions turned out to be good and reliable in the worst case, leading to the record worst case complexity bounds of $O(n^2 \log n(n \log n+\log(1/\epsilon)))$ arithmetic operations for approximating to all the complex zeros $x_1,...,x_n$ of an n-th degree polynomial p(x) with *relative* errors $< \epsilon$ and of $O(n \log n \log(1/\epsilon))$ for similarly approximating to a single zero, see [20, 22, 29]. Later on Renegar in [25] approximated all the zeros of p(x) with absolute errors $\leq \epsilon$ using $O(n^2 \log n(n+\log \log(1/\epsilon)))$ arithmetic operations, provided that p(x) has been scaled so to make all its zeros lie in a unit circle. This improved the above estimate of [22] for approximating all the zeros of p(x) (although the improvement is minor in the cases where $|\log \epsilon| = O(n \log n)$). Our new algorithms of this paper improve the estimates of [22] and [25] for the arithmetic computational complexity of computing all the zeros of p(x) roughly by a factor of n/log n, establishing a new asymptotic record bound. Furthermore, computing a single zero of a polynomial whose all zeros are real, we reach the upper bound $O(\log n(\log^3 n+\log \log(1/\epsilon)))$, which substantially decreases the bound of [22] in the cases of larger ϵ and which decreases the respective bound of [25] roughly n^2 times. This brings us quite close to the known lower bound $\Omega(n+\log \log(1/\epsilon))$. (At least n/2 arithmetic operations are needed already in order to process n input coefficients of p(x); the lower bound $\Omega(\log \log(1/\epsilon))$ follows from the stronger results of [15], rediscovered in a weaker form in [25]). Besides, our overhead constants are small; in particular they substantially improve over ones of [25]. It is interesting to compare the techniques of this paper with ones of [25]. As this is emphatically proclaimed in the summary and throughout the paper [25], the algorithm of [25] "is built around Newton's method and Schur-Cohn algorithm". In our paper we share Weyl's geometric construction with [25], but we rely on completely distinct algebraic and analytical tools, that is, on numerical integration (successfully replacing Newton's iterations) and power sum computations, particularly on Turan's proximity test and on its extension in [27] to root radius computation; nowhere we need to apply Newton's method or Schur-Cohn test. (In April 1987 a draft of this paper was sent to J. Renegar, he then used some of our observations in order to improve his original (but not our new!) complexity bounds for the price of abandoning only Schur-Cohn's test, but not Newton's method.)

Preprint of paper appearing in 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, October 1987, (IEEE Computer Society Press) pp. 173-184.

Our new algorithms are also effective for parallel evaluation of a single zero and of all the zeros of p(x), see Table 1.1 below. For parallel computations we assume the customary machine model, where in each step each processor performs at most one arithmetic operation, that is, +, -, *, ÷, or the evaluation of an N-th root of a positive number for a natural N. (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 ≥ 1).

Table 1.1. Arithmetic complexity of approximating polynomial zeros in unit circle with absolute errors $\leq 2^{-b}$.

	Sequential time	Parallel time,Processors
All zeros	$O(n^2(\log^2 n+\log b))$	$O(n(\log^2 n+\log b),n)$
Single zero (general case)	$O(n \log n(n+\log b))$	$O(\log n(\log^2 n+\log b),$ $n+n^2/(\log^2 n \log b))$
Single zero (all zeros are real)	$O(n \log n(\log^3 n+\log b))$	$O(\log n(\log^3 n+\log b),n)$

Table 1.1 shows in particular that the parallel evaluation of a single zero of p(x) requires only polylogarithmic parallel time and less than n^2 processors, so we add a new problem to NC, [2, 6]. In Section 11 we show that O(bn) bit-precision of computations suffices in order to support the arithmetic complexity estimates of Table 1.1 where $b=q+m$, $\epsilon=2^{-q}$, and the coefficients $c_0,c_1,..,c_n$ of p(x) are integers such that $|c_j| < 2^m$. (The study of perturbation of polynomial zeros due to variation of coefficients, see [16], pp. 74-77; [18, 27], suggests that the precision of computations must be at least of an order of bn in the worst case, but on the other hand, computing with the precision O(b) usually suffices in practice in order to evaluate the non-clustered complex zeros of many input polynomials p(x) with errors $\leq 2^{-b}$ relative to the coefficient size and also to factor an *arbitrary* univariate polynomial numerically in the complex domain.)

Since O(h log h log log h) Boolean operations or $O(\log^2 h)$ Boolean parallel steps, h log h log log h processors suffice for an arithmetic operation with integers modulo 2^h, [1, 5, 17, 26], and since in our case h = O(bn), we may immediately extend our estimates of Table 1.1 to the case of the Boolean circuit complexity model. In particular we arrive (within a logarithmic factor) at the Boolean sequential time bound on the complexity of computing all the zeros of p(x) stated in [27], Sect. 19 but only partly proven so far. (Schönhage [27] needs to use n^3 arithmetic operations already in his n stages of numerical integration, which exceeds the bound of Table 1.1 about n times, but he computes with lower precision in those stages and with the precision O(bn) in other stages.) His complete proof of the Boolean complexity bound seems to be very much involved and has not appeared yet. There seems to be two difficulties with that proof. One difficulty is due to the intention of 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 certainly a good example where multiple precision is required. Another difficulty is due to the approach of [27], where major efforts are spent to separate the contours lines (used for numerical integration) from polynomial zeros. Those efforts, supported by elaborate and intricate error analysis, lead to quite favorable asymptotic estimates for the Boolean complexity of computing polynomial zeros (provided that the proofs will be completed), but the overhead constants substantially grow. (Those constants are hidden in the "O" notation of the asymptotic estimates of [27].)

In contrast to that, in this paper the zeros of p(x) are included into regions of regular shapes (circles or squares) and are evaluated using Weyl's construction, which naturally subdivides the computations into isolated and self-correcting stages. All this simplifies the complexity analysis and enables us to decrease the overhead constants, making our algorithms good candidates for practical implementation. Furthermore the arithmetic complexity bounds of the first two lines of Table 1.1 seem to be overly pessimistic in the case of many polynomials p(x). While 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 again such a systematic disbalance is certainly a rather exceptional case in practice.

Further extensions and applications of our results include, in particular, a) the evaluation of the eigenvalues of a matrix via the evaluation of its characteristic polynomial, see [22]; b) factorization of a polynomial in the complex domain via the same algorithms for computing all the zeros (with more favorable Boolean complexity bounds for factorization), compare [27], Theorems 2.1 and 19.2), and c) computing the greatest common divisor (gcd) of two or several polynomials. Numerical treatment of the latter problem can be reduced to numerical evaluation of all the zeros of the input polynomials.

That reduction, however, cannot be suggested for parallel computation, for the evaluation of polynomial zeros requires superlinear parallel time versus $O(\log^2 n)$ time for the polynomial gcd computations, [6]. That parallel time bound for the gcd, however, is supported by using a quite high processor bound. This leads us to our second major subject. We treat that subject (more broadly) as parallel computations with Toeplitz matrices $T = [t_{ij}]$, $t_{ij} = t_{i-j}$, i,j=0,1,...,n-1 (which includes the gcd computations as an important but very particular case). Fast and processor efficient evaluation of the inverse T^{-1}, the determinant det(T), and the characteristic polynomial $\det(\lambda I-T)$ of T should have important applications to time series analysis, image processing, control theory, statistics, solution of integral and partial differential equations, Padé approximations, rational interpolation, partial fraction decomposition, computations with polynomials, and with matrix polynomials, see [7, 8, 30, 31]. Our main result is a parallel algorithm for the above evaluations for an n×n Toeplitz matrix T. That algorithm runs in $O(\log^2 n)$ parallel arithmetic time and requires only $O(n^2)$ processors. The precision of computation of O(n log(n||T||)) binary bits suffices, where $||T|| = \max_j \sum_i |t_{ij}|$. Our arithmetic complexity bounds and our precision bounds immediately lead to a respective substantial improvement of the known bounds on the parallel Boolean circuit complexity.

For comparison the previous parallel Toeplitz algorithms either run in linear arithmetic time (of order n) or otherwise required at best M(n) processors and the precision of an order of n log(n||T||) binary bits, the same as for the general matrix computations, see [19-21]. Here $M(n) = O(n^\omega)$ denotes the sequential time required for n×n matrix multiplication, currently in theory $\omega \leq 2.375...$ with huge overhead constants, and in practical computations $\omega = 3$ with small overhead. Thus our improvement means saving an order of n processors (if we refer to the practical bound $\omega = 3$). Furthermore our parallel algorithm for

computing $\det(\lambda I - T)$ has sequential time $O(n^2 \log^2 n)$ arithmetic operations, which stays on the level of the current best sequential time estimates for that computational problem. The current best arithmetic sequential time for the inversion of T, however, is about n times less than the sequential time of our algorithm, $O(n \log^2 n)$, see [4, 7]. Furthermore, to invert a Toeplitz matrix T, we need to compute the coefficients of its characteristic polynomial, which may be fairly large, while some of the cited sequential algorithms for *numerical* inversion of T involve large numbers only where T is ill-conditioned, see [8].

On the other hand, many important computations (such as rational interpolation and computing the greatest common divisor of two or several polynomials) are reduced to solving ill-conditioned Toeplitz linear systems, and then even the sequential version of our algorithms can be made competitive with (and even slightly superior to) the current best algorithms, via the techniques used in [20, 21]. Namely, we may assume that the entries of the input Toeplitz matrix T are integers (the real or complex binary entries can be truncated to finite precision binary numbers, and then the matrix T can be scaled to turn those binary numbers into integers) and evaluate $\det(T)$ and $\mathrm{adj}(T) = T^{-1}\det(T)$ modulo a prime power p^k, that is, we compute with $\lceil k \log_2 p \rceil$ binary bits. We choose p and k such that $k \log_2 p$ (slightly) exceeds the number of bits required to represent each output value ($\lceil n \log_2(n\|T\|) \rceil + 1$ bits would suffice for any T; for some T even fewer bits suffice). Then we easily recover all the exact integer output values (that is, $\det(T)$ and the entries of $\mathrm{adj}(T)$) from their values mod p^k and arrive at the desired exact solution, including also $T^{-1} = \mathrm{adj}(T)/\det T$, compare [20, 21]. The precision of those computations (measured by the number of binary bits used in order to represent the operands) is within one binary bit from the number of bits already required in order to represent the absolutely largest output value (assuming that we compute the outputs exactly, which is a reasonable assumption in the case of ill-conditioned linear systems). Surprisingly though, we may compute with only $\lceil \log_2 p \rceil$ binary bits throughout, except for some stages involving $O(n \log^2 n)$ arithmetic operations. We will do this via Hensel-Newton's lifting algorithm and will arrive at a *sequential* algorithm for the *exact inversion* of Toeplitz matrices, whose overall asymptotic Boolean cost is even lower than the cost of the current best algorithms, see [23].

We hope that the techniques that we used are of some independent interest. Our algorithm relies on the reduction of the auxiliary matrix powering (which destroys the Toeplitz structure) to matrix inversion (which does not). That reduction is akin to the *variable diagonal technique* of [20, 21]. For matrix inversion we use Newton's iterations along the line of [19], but we had to modify those iterations in order to preserve the lower *displacement rank* of Toeplitz matrices, which characterizes their structure, compare [12]. Also we exploit the Csanky-LeVerrier construction of [10], with an improvement in the stage of solving a system of Newton's identities.

Our present main objective is to demonstrate the power of the concepts and techniques used in our algorithm and to deduce the desired estimates for parallel complexity. With more work and further elaboration our method should lead to more effective parallel algorithms more suitable for implementation. We can see and foresee many further applications of our results to computations with Toeplitz and other structured matrices (such as Hankel, block Toeplitz and block Hankel matrices, matrices and block matrices having small displacement ranks). In particular the greatest common divisor of m polynomials of degree n with integer coefficients ranging from -t to t can be computed using $O(\log^2 n \log(m+n))$ parallel arithmetic steps, mn^2 processors, and the bit-precision of computations $O(n \log(mnt))$. This immediately leads to improvements of the known parallel algorithms for several other polynomial and rational computations, such as rational (Cauchy and Hermite) interpolation, Padè approximation, computing elementary symmetric functions, Taylor expansion, Chinese remainder algorithm, partial fraction decomposition, square free decomposition of a polynomial, computing the least common multiple of two or several polynomials, computing polynomial resultants, and so on, compare [7, 30, 31].

Part 1. Approximate Evaluation of Polynomial Zeros

2. Definitions and An Auxiliary Algorithm

Definition 2.1, compare [6]. $O_A(t,p)$ means t parallel arithmetic steps and p processors. $O_A(T) = O_A(T,1)$ denotes the sequential arithmetic time, that is, the number of arithmetic operations used. Replacing arithmetic operations by Boolean ones we arrive at the Boolean model of computations with the notation $O_B(t,p)$, $O_B(T)$.

Let a monic polynomial p(x) of degree n be fixed,

$$p(x) = \sum_{i=0}^{n} c_i x^i = \prod_{j=1}^{n} (x - x_j),\ c_n = 1. \qquad (2.1)$$

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, X-R, X+R$\sqrt{-1}$, X-R $\sqrt{-1}$; A = A(X,R,r), R > r, denotes the annulus D(X,R) - D(X,r). We will write $\rho(S) = \rho(D) = R$, $\rho(A) = R - r$.

Remark 2.1. Hereafter in different sections each of the characters R, X, r, x, Y, X_g, R_g, r_g, Y_g, and so on 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) denote 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, a disc, or an annulus, we define its *rigidity ratio* r.r.(U), and its *isolation ratio*, i.r.(U), as follows,

$$\text{r.r.}(U) = \text{minimum } (\rho(U^-) \,/\, \rho(U)),$$
$$\text{i.r.}(U) = 1/\text{minimum } (\rho(U) \,/\, \rho(U^+)),$$

Here the minimization is 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) $\geq d(S)/(2\sqrt{2}\rho(S))$ *for a square* S; r.r.(D) $\geq d(D)/(2\rho(D))$ *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 polynomial p(x) in a complex domain U, counted with their *multiplicities*, is called the *index* of p(x) in U and denoted i(p(x),U).

Definition 2.7. The distances $r_0(X) \geq r_1(X) \geq \ldots \geq r_{n-1}(X)$ from a 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 *s-th root radius* of p(x) at X.

Definition 2.8. For fixed positive R and ϵ,

$$b = \log_2(R/\epsilon). \tag{2.2}$$

Algorithm 2.1. Superscription of a square about a given set of points. Input. A (finite) set H of complex points. **Computation.** Compute the maximum M and the minimum 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 2.2, compare 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 z and the half-length* r *of the longer side define a square* S(x,r) *containing the set* H. *Moreover* $r \le \rho(S)$ *for any square* S *containing* H.

In the sequel we will use the known algorithms for some basis operations with polynomials (such as their multiplication and division with a remainder, discrete Fourier transform (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 [1, 5, 17, 22].

3. Turan's Proximity Test

Algorithm 3.1 (Turan's proximity test).

Inputs. A degree n polynomial p(x) of (2.1) 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 (shifted inverse) power sums,

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

Stage 2. Compute and output $r = \max_{g=1,...,n} |s_{gN}/n|^{1/(gN)}$ and $r^* = 5^{1/N}r$.

Theorem 3.1, [29], p. 299. *For the output* r *of Algorithm 3.1*

$$1 \le \min_{g=1,...,n} |X - x_j| / r \le 5^{1/N},$$

that is, $i(p(x),D) = 0$, $i.r.(D) \le 5^{1/N}$ *where* $D = D(X,r)$, *see Definitions 2.3, 2.6.*

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

$$1.051581 < 5^{1/N} < 1.051582. \tag{3.1.}$$

[29] performs Stage 1 for $N = 2^h$ as follows.

Subalgorithm 3.1.

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

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

$$q(y) = y^n p(X + 1/y) = \sum_{i=0}^{n} c_i(X)\, y^{n-i} = c_0(X) \prod_{j=1}^{n} (y-y_j). \tag{3.2}$$

Stage b). Let $q_0(y) = q(y)/c_0(X)$ and successively compute the coefficients of the polynomials

$$q_{i+1}(y) = q_i(\sqrt{y})q_i(-\sqrt{y}),\ i=0,1,...,h-1. \tag{3.3}$$

Iteration i *squares* the zeros of $q_i(y)$ (*Dandelin, Graeffe,* [14, 16]), so

$$q_h(y) = (-1)^{hn} \prod_{j=1}^{n}(y-y_j^N) = \sum_{i=1}^{n} c_{i,h}y^i,\ N = 2^h. \tag{3.4}$$

Stage c). Compute the (shifted inverse) power sums s_{gN} for g=1,...,n from the following triangular Toeplitz system of *Newton's identities,* [16], p. 36,

$$\begin{aligned} &c_{n,h}s_N + c_{n-1,h} = 0,\\ &c_{n,h}s_{2N} + c_{n-1,h}s_N + 2c_{n-2,h} = 0,\\ &\cdots\cdots\cdots\cdots\cdots\cdots\\ &c_{n,h}s_{nN} + c_{n-1,h}s_{(n-1)N} + \ldots + nc_{0,k} = 0. \end{aligned} \tag{3.5}$$

The cost of Algorithm 3.1 is $O_A(\log n,n)$ for Stages a), b), c) can be immediately reduced to O(1) applications of DFT at O(n) points, [22], Sect. 4.

Remark 3.1. Subalgorithm 3.1 can be replaced by numerical integration, [14],

$$s_K = \frac{1}{2\pi i} \int_{y \in \Gamma} (y^K q'(y)/q(y))dy. \tag{3.6}$$

4. Computing the Number of Polynomial Zeros in an Isolated Disc

The well known *winding number algorithms* (whose cost is $O_A(\log n,n)$), [14], pp. 239-241; [25], 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. [25] proves that the bound $i.r.(D) \ge 9$ already suffices to assure the latter assumption, see Remark 4.2 in [24] for an alternative solution having the same cost. Let us reduce the case of arbitrary disc D with $i.r.(D) > 1$ to the case $i.r.(D) \ge 9$.

Algorithm 4.1. Inputs. Complex X, positive r and ν, and polynomial p(x) of (2.1) such that $i.r.(D(X,r)) \ge 1+\nu > 1$, compare Definition 2.3.

Stage 1. Compute the coefficients of the monic polynomial $q_0(y) = p(X+ry)/r^n$, $i(p(x),D(X,r)) = i(q_0(y),D(0,1))$.

Stage 2. Apply h iterations (3.3) where

$$h = \lceil \log_2(\log 9/\log(1+\nu)) \rceil = O(\log(1/\nu)) \text{ as } \nu \to 0. \tag{4.1}$$

Stage 1 of the algorithm transforms the discs D(X,r) and $D(X,(1+\nu)r)$ into the discs D(0,1) and $D(0,(1+\nu))$, respectively. Stage 2 transforms the disc D(0,1) into itself and the disc $D(0,(1+\nu))$ into the disc $D(0,(1+\nu)^{2^h})$, where $\nu^{2^h} \ge 9$ due to (4.1), 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 $i(q_h(y),D(0,1)) = i(p(x),D(X,r))$.

Proposition 4.1. *Let* $i.r.(D) \ge 1+\nu > 1$ *for a disc* D. *Then the index* i(p(x),D) *can be computed for the cost* $O_A(h \log n,n)$ *where* h *is defined by (4.1);*

Corollary 4.1. *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 4.1 with $r=\epsilon$, $\nu=1/n$. Q.E.D.

Remark 4.1. The results of this section would not change if we set q(y) = p(X+y) in Stage 1 of Algorithm 4.1.

5. Turan-Weyl's Exclusion Algorithm

Algorithm 5.1 (Turan-Weyl).

Inputs. Polynomial p(x) of (2.1), 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 $\geq \sqrt{2}$. 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* if Algorithm 3.1 outputs $r \leq R/2^{j-0.5}$. In Stage J output the centers of all 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* $\leq R/2^{j-0.5}$, *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.

Proposition 5.2. *Let* ϵ *be a positive constant,* $b = \log_2(R/\epsilon)$, *compare (2.2). 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*-approximations to a zero of* p(x) *for* $\epsilon_s \leq \epsilon$. ϵ_s *for all the suspect squares can be computed for the cost* $O_A(\log k, k^2)$, *so isolated* ϵ_s*-approximations to all the* k *zeros of* p(x) *in* S(X,R) *where* $\epsilon_s \leq \epsilon$ *for all* s *can be computed for the cost* $O_A((b+\log n)\log n, kn)$.

Proof. 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

$$r < \epsilon/((12n+1)\sqrt{2}). \tag{5.1}$$

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

$$r_0 = r\sqrt{2},\ r_{i+1} = r_i + 3r_0 = (3i+1)r_0 \text{ for } i=0,1,2,... \tag{5.2}$$

Successively (for i=0,1,...) check if any suspect square does not lie in the disc $D(X_s,r_i)$, but lies in or intersects the disc $D(X_s,r_i(n+1)/n)$ and therefore lies inside the disc $D(X_s,r_i)$. Since there are at most 4k suspect squares, checking step i will give answer "no" for some $i = i(s) \leq 4k$. Then $r_i = (3i+1)r_0 \leq (12k+1)r_0 < \epsilon$, and $\text{i.r.}(D(X_s,r_i)) \geq 1+1/n$, 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. Q.E.D.

Remark 5.1. The index of p(x) in the ϵ_s–neighborhood of each isolated ϵ_s–approximation to a zero of p(x) can be computed by Algorithm 4.1.

6. How to Contract a Square Region

For the cost $O_A(\log n, n)$, the next algorithm will contract a square S(Y,R) having isolation ratio ≥ 3 into its subsquare S(Z,r) such that either

$$r < 0.8R \tag{6.1}$$

or

$$\text{r.r.}(S(Z,r)) > 0.1 \text{ and } d(S(Z,r)) > 0.3R \geq 0.3r, \tag{6.2}$$

compare Definitions 2.3 and 2.4.

Subalgorithm 6.1. Inputs. Polynomial p(x) of (2.1), complex Y, and positive R, such that $\text{i.r.}(S(Y,R)) \geq 3$.

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 $d^*(1,1) \geq d^*(1,-1)$ and h=-1 otherwise. Due to Theorem 3.1,

$$\text{r.r.}(D_g) \geq 1/5^{1/N} \text{ for } g=1,2. \tag{6.3}$$

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 $V = \{v_1,...,v_q\}$, $q \leq 10$, 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 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)). Output Z and r.

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); $\text{i.r.}(S) \geq 3$; *ii)* $d(S) \geq d^*$, *and iii)*

$$2\sqrt{2}R - d^* \leq (4R-2r)5^{1/N}. \tag{6.4}$$

Proof. $S(Y,R) - \bigcup_{g,h} D(Y(g,h),r(g,h)) \subseteq S \subseteq S(Y,R)$ 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 i). On the other hand, each disc $D(Y(g,h),r^*(g,h))$ must contain at least one zero of p(x), and this implies ii). To prove (6.4), 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

$$\max\{d(1),d(-1)\} \geq \sqrt{2}\,r. \tag{6.5}$$

Next prove for h=1 that

$$d(h) = 2\sqrt{2}R - (2R\sqrt{2} - d(1,h))/\sqrt{2}) \tag{6.6}$$

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},E_1,F_1,F_{-1} denote four successive points of the diagonal of the square S(Y,R) passing through Y(-1,1) and Y(1,1), namely, the four intersection points of that diagonal with the four lines: L(-1,-1), boundaries of the two discs D(Y(g,g)),r(g,g)) for g=-1 and g=1, and 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 (6.6) for h=1; similarly (6.6) can be proven for h=-1. Recall (6.5) and deduce that $(2\sqrt{2}R-d) \le 4R-2r$, where $d = \max\{d(1,1),d(1,-1)\}$. Now (6.4) follows because $2\sqrt{2}R-d^* = (2\sqrt{2}R-d)5^{1/N}$. Q.E.D.

Corollary 6.1. *If* $R \ge r \ge 0.8R$ *then*

$$d(S(Z,r)) \ge \text{dist}(D_1,D_2) = d^* > (2\sqrt{2}-(2.4)5^{1/N})R. \tag{6.7}$$

Otherwise (6.1) holds.

Next let N=32, apply (3.1) and Proposition 2.1, and deduce (6.2) from (6.7). Since i.r.(S(Z,r)) ≥ 3, 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. *For a positive* E, *a square* S(Y,R) *having isolation ratio* ≥ 3 *can be contracted for the cost* $O_A(g \log n,n)$, $g = \log(R/E)$, *into a square* S(Z,r) *such that* $r < E$ *or else the relations (6.2) are satisfied.*

7. Accelerated Shrinking of an Isolated Square

In this section we will rapidly contract a square S(Y,R) having isolation ratio at least 3 either i) into a disc of radius $\le \epsilon$ (so its center approximates to all the zeros of p(x) lying in S(Y,R) with errors $\le \epsilon$) or ii) into a square output by Subalgorithm 6.1 and satisfying the relations (6.2). At first we contract the square S(Y,R) into a disc having isolation ratio, say $\ge 16n$, applying Proposition 6.2. Then we will need some auxiliary results. Denote

$$M = \sum_{h=1}^{k} x_{j(h)} / k. \tag{7.1}$$

If k=n, then the value $M = c_{n-1}/n$ is immediately available, see (2.1). 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) $\ge (1+\nu)^2$, $\nu > 0$, 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* $\le (1+\nu)^{-\bar{c}n^c}r$ *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* $d(D) \ge r_{n-k}(M)k/(k-1)$, compare Definitions 2.4, 2.7.

Proof. Let $x_{j(1)},\dots,x_{j(k)}$ denote the k zeros of p(x) lying in the disc D and let $|M - x_{j(1)}| = r_{n-k}(M)$. Without loss of generality, assume that M=0, $x_{j(1)}$ is negative. Then $\sum_{h=1}^{k} \text{Re } x_{j(h)} = 0$. Therefore $\sum_{h=2}^{k} \text{Re } x_{j(h)} = -x_{j(1)} = r_{n-k}(M)$, $\max_{h=2,\dots,k} \text{Re } x_{j(h)} \ge r_{n-k}(M)/(k-1)$, so $d(D) \ge r_{n-k}(M)k/(k-1)$. Q.E.D.

Corollary 7.1. *Let under the assumptions of Proposition 7.2,* $|M^* - M|/r_{n-k}(M) = \alpha$ *for some complex* M^*, *and let* D^* *denote the disc* $D(M^*,(1+\alpha)r_{n-k}(M))$. *Then* $D^* \supseteq D(M,r_{n-k}(M))$, $r_{n-k}(M^*) \le (1+\alpha)r_{n-k}(M)$,

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

The next algorithm rapidly contracts a disc, as desired.

Algorithm 7.1.

Inputs. Polynomial p(x) of (2.1), complex X, natural k, and positive r and ν such that

$$i(p(x),D(X,r))=k,\ \text{i.r.}(D(X,r))=(1+\nu)^2>\max\{4,(256n^2)^{1/n}\}. \tag{7.2}$$

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, $\nu_0=\nu$, $D_0=D(M_0,r_0)$.

Stage j, j=0,1,..., 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

$$\epsilon_{j+1} = (1+\nu_j)^{-4n}r_j \tag{7.3}$$

(apply Proposition 7.1 where c=1, $\bar{c}$=4, X = M_j, r=r_j, $\nu=\nu_j$, compare also Proposition 7.3 below). Then compute an approximation r^*_{j+1} to the (n-k)-th root radius $r_{n-k}(M_{j+1})$ of p(x) at M_{j+1} with relative error at most 8n. (The cost is again $O_A(\log n,n)$, see [24].) Denote

$$r_{j+1}=8n\ r^*_{j+1},\ D_{j+1}=D(M_{j+1},r_{j+1}),\ (1+\nu_{j+1})^2=(1+\nu_j)^{2n}. \tag{7.4}$$

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

$$\epsilon \le r_{j+1} < 128n^2(1+\nu_j)^{-4n}r_j \tag{7.5}$$

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 (6.2) are satisfied).

Let us analyze Algorithm 7.1. At first consider the case where (7.5) holds for all j. Then recursively apply the last equation of (7.4) and obtain that $(1+\nu_j)^2 = (1+\nu)^{2n^j}$, j=0,1,..,J-1. Substitute this into (7.5), apply the resulting relation recursively, and deduce that in this case

$$\log_2(r/r_j) > 4n^j\log_2(1+\nu),$$

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

We need, however, the following result in order to apply Proposition 7.1 when we deduce (7.3).

Proposition 7.3. *If (7.5) holds for* j=0,1,..,g, *then* $(1+\nu_j)^2 \le$ i.r.(D_j) *for* j=0,1,...,g.

Proof. (7.4) implies that all the discs D_j are equivalent to D_0 and to each other. Therefore $|M_{j+1} - M_j| \le r_j+r_{j+1}$ for all j. On the other hand,

$$(\text{i.r.}(D_{j+1}))r_{j+1} = r_{n-k-1}(M_{j+1}) \ge$$

$$r_{n-k-1}(M_j) - |M_{j+1} - M_j| = (\text{i.r.}(D_j))r_j - |M_{j+1} - M_j|$$

for all j. It follows that

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

for all j. Apply (7.5) and deduce that

$$(\text{i.r.}(D_{j+1}) + 1)/(\text{i.r.}(D_j) - 1) \ge (1+\nu_j)^{4n}/(128n^2) \tag{7.6}$$

for all j. Recall that $(1+\nu_0) =$ i.r.(D_0) and observe that i.r.(D_j) grows more rapidly than $(1+\nu_j)^2$ as j grows. (The latter fact follows from the comparison of (7.6) with the last equation of (7.4) for j=0,1,... and from the inequality of (7.2).) Q.E.D.

Next assume that (7.5) does not hold for some j. Note that

$$i(p(x), D_{j+1}) = k,\ r_{n-k}(M_{j+1}) < r_{j+1} < 64n^2r_{n-k}(M_{j+1}). \tag{7.7}$$

(7.7) and Corollary 7.1 together imply the following estimate,

$$d(D_{j+1}) \ge (r_{n-k}(M_{j+1}) / (1+\alpha_{j+1}))(k/(k-1)),$$

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

Therefore

$$d(D_{j+1}) \geq (r_{n-k}(M_{j+1}) - \epsilon_{j+1})k/(k-1). \tag{7.8}$$

If (7.5) does not hold for some j, then

$$128n^2\epsilon_{j+1} = 128n^2(1+\nu_j)^{-4n}r_j \leq r_{j+1} < 64n^2 r_{n-k}(M_{j+1}),$$

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

Then (7.7) and (7.8) imply that

$$d(D_{j+1}) \geq (0.5)r_{n-k}(M_{j+1})k/(k-1) \geq (r_{j+1}/(128\,n^2))(k/(k-1)). \tag{7.9}$$

On the other hand, unless (7.5) 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 will satisfy the relations (6.2) in O(log n) recursive applications of Subalgorithm 6.1 (due to (7.9)), and then the computations of Algorithm 7.1 will end. Summarizing we arrive at the following result.

Proposition 7.4. *Let* b *satisfy (2.2) for* $R \geq r$. *Then for the cost* $O_A(\log(bn),n)$ *Algorithm 7.1 contracts its input disc* D(X,r) *satisfying (7.2) into a disc of radius* $< \epsilon$ *or else (only for* $k>1$*) into a disc* D_{j+1} *satisfying (7.9). 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 satisfy the relation (6.2) (for the additional cost* $O_A(\log^2 n, n)$*).*

8. Computing the Center of Gravity of a Set of Polynomial Zeros

Extending (3.6) we arrive at the following formula, [14],

$$M = \frac{1}{2\pi k i} \int_{\Gamma} (x\, p'(x) / p(x))\, dx, \tag{8.1}$$

where $i = \sqrt{-1}$, the value M=M(U) is defined by (7.1), and the domain U bordered by the contour Γ contains exactly k zeros of p(x) (not necessarily distinct), that is, $x_{j(h)}$, h=1,...,k. Assuming that U=D(X,r) is a disc with isolation ratio $\geq (1+\nu)^2$, we will choose Γ being the boundary of the disc D(X,R), $R = (1+\nu)r$ and will approximate to the integral (8.1) using the integral sum

$$M^* = \frac{1}{2\pi k Q i} \sum_{q=0}^{Q-1} (X + R\,\omega^q)\, p'(X + R\omega^q) / p(X + R\omega^q). \tag{8.2}$$

Here ω is a primitive Q-th 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

$$p'(x)/p(x) = \sum_{m=0}^{\infty} s_m x^{1-m} - \sum_{m=1}^{\infty} S_m x^{m-1}, \tag{8.3}$$

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

$\{x_{j(1)}, \ldots, x_{j(n)}\}$ is the set of all the zeros of p(x) numbered such that $|X - x_{j(i)}| < R$ if and only if $i \leq k$, compare [27], Sect. 12. (8.1)-(8.3) immediately imply that

$$|M^* - M| \leq 2R(kg^{Q+1} + (n-k)g^{Q-1}) / (k(1-g^Q)), \tag{8.4}$$

$$g = \min_{1 \leq j \leq n} \min \{ |X - x_j| / R,\ R / |X - x_j| \}, \tag{8.5}$$

and $x_1, \ldots, x_n$ denote all the zeros of p(x). $g \leq 1/(1+\nu)$ since $R = (1+\nu)r$, i.r.(D(X,r)) $\geq (1+\nu)^2$. (8.4) and (8.5) imply that

$$|M^* - M| < 4Rn / (k(1+\nu)^{Q-1}) = 4nr/(k(1+\nu)^Q). \tag{8.6}$$

We will keep Q of an order c^*n 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 $|M^* - M| \leq (1+\nu)^{-2n^c}r$, as this is required in Proposition 7.1. Q.E.D.

9. Turan-Weyl's Isolation Algorithm for Computing All the Zeros of a Polynomial

Using the machinery of the previous sections we will now arrive at the bounds of Table 1.1 for computing all the zeros of p(x). We will proceed recursively; in each recursive step the set of the n zeros of p(x) will be covered with a square or with several squares of the same size.

Initially cover all the zeros of p(x) with the single square S(0,R),

$$R = 2\sqrt{2} \max_{h=1,\ldots,n} |c_{n-h}/c_n|^{1/h}, \tag{9.1}$$

so i.r.(S(0,R)) = ∞, [14], Sect. 6.4. We will maintain isolation ratio at least six for the output squares (and thus for the input squares) of each recursive step. This will enable us to partition each recursive step into parallel processes associated with each isolated square and performed independently of each other. Let S(Y,R) denote an input square of some recursive step such that i.r.(S(Y,R)) $\geq$ 6, i(p(x),S(Y(R)) = k. 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 $\leq \epsilon$ or will arrive at the case where Subalgorithm 6.1 has been applied and its outputs satisfy (6.2), in particular, in that case S(Y,R) shrinks to a square S such that r.r.(S) > 0.1.Then we will apply (Turan-Weyl's) Algorithm 5.1 stopping in

$$J = \lceil \log_2(nR/\epsilon) \rceil + 3$$

iterations. 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 now.

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

$$H(j) \leq k. \tag{9.2}$$

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. Note that

$$R_g^* \leq (k+2)R / 2^{j+1}. \tag{9.3}$$

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

$$|X_h - X_g| > \sqrt{2}\,(R_h^* + 6\,R_g^*) \text{ for all } h \neq g. \tag{9.4}$$

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

$$\text{i.r.}(S) \geq 6. \tag{9.5}$$

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 (9.2)-(9.4) we deduce that

$$j \leq \lceil -0.5 + \log_2 7 + \log_2((k+1)(k+2)) \rceil. \tag{9.6}$$

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 call that modification of Algorithm 5.1 *Turan-Weyl's isolation algorithm*, compare [25]. Its cost 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 (9.4). 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 h 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 $k \leq 5$ suspect squares, then the edge in the tree from that component in the direction to the root corresponds to at most $\lceil \log_2(3k) \rceil \leq 4$ 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,..,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 $i.r.(S_g) \geq 6$, see (9.5), and compute the indices of p(x) in all those squares (see Proposition 4.1). 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 (9.1), 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 4.1 and 7.1 (required O(n) times) and of Subalgorithm 6.1 (required O(n log n) times), the overall cost bound increases to $O_A(n(\log^2 n+\log b),n)$.

Theorem 9.1. *All the zeros of a polynomial* p(x) *of (2.1) can be computed with absolute errors at most* ϵ *for the cost* $O_A(n(\log^2 n+\log b),n)$ *where* b *satisfies (2.2) for a positive* R *that upper bounds the moduli of all the zeros of* p(x).

10. Computing a Single Zero of a Polynomial

Next we will compute a single zero of p(x) for the cost estimated in line 2 of Table 1.1.

The first two strongly isolated components C_1 and C_2 are computed in at most O(log n) iterations of Turan-Weyl's algorithm, see (9.1)-(9.6). Compute the indices i_1 and i_2 of p(x) in both of these components, see Proposition 4.1. $i_1+i_2 \leq n$, so $\min\{i_1,i_2\} \leq n/2$, say $i_1 = i(p(x),C_1) \leq n/2$. Apply our algorithms of Sections 6-9 to the component C_1 and repeat that process recursively, defining a sequence of strongly isolated components $C_{g(0)} \supseteq C_{g(1)} \supseteq \ldots$, such that $C_{g(0)} = C_1$,

$$i(p(x),C_{g(h)}) \leq n/2^h, \ h=0,1,\ldots, \tag{10.1}$$

(10.1) implies 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 log n, add the cost of O(log n) applications of algorithms of Sections 4,7, and 8, and arrive at the bounds of Table 1.1.

In the important case where all the zeros of p(x) *are real, the algorithm for computing a single zero of* p(x) *can be substantially improved to reach the cost bound* $O_A(\log n(\log b+\log^3 n),n)$, *see [24].*

11. Boolean Circuit Complexity of Computing Polynomial Zeros

In this section we will reexamine our algorithms assuming a finite precision of computations. We have reduced the evaluation of polynomial zeros to computing root radii via root powering (Algorithm 3.1) and via numerical integration, applied in (8.2) and in the winding number algorithms. All these computations are immediately reduced to polynomial multiplications (to which also the shifts of variable x can be reduced), polynomial divisions (to which also the solution of triangular Toeplitz systems can be reduced), and to discrete Fourier transforms. Using the known estimates for the errors of performing those basis operations with finite precision and taking into account that Turan-Weyl's algorithm is self-correcting (that is, the errors of each its iteration are corrected in the next iteration), we may estimate the output errors of finite precision computations by our algorithm and its Boolean (bit-operation) complexity. The required basis estimates in the case of polynomial multiplication and division and DFT can be taken from [3, 13], and [27]. The analysis of the errors of finite precision computations is simple or

readily available for numerical integration, see [14], pp. 239-241, [27], so we will focus on Turan's tests, specifically, on the solution of Toeplitz systems and on the shifts of the variable. Next we will show (relying on the known error estimates for those operations, [3, 13]) that the magnification of errors by Turan's tests may only require to increase the precision of computations about O(n) times comparing with the prescribed output precision, while such an increase is inevitable in any algorithm for computing zeros of an arbitrary polynomial, see [16], pp. 74-77, [18]. (It is even easier to show that the precision of computations required in our case for numerical integration needs not be higher than that.)

Let us assume that the input coefficients $c_0,...,c_n$ of p(x) are integers and that

$$\max_j |c_j| < 2^m, \epsilon = 2^{-q}, b = m + q, \qquad (11.1)$$

compare (2.2) and Theorem 1 of [2]. (We may arrive at that case via trancation of the mantissas of $c_0,...,c_n$ and scaling p(x).) Then it can be shown that the computations with the precision O(bn) binary bits will suffice. Let us verify this for both stages of Turan's test:

a) shift of the variable x (by X),

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

Stage a). We will choose the shift values X such that both real and imaginary parts of $2^{q+2}X$ are integers. (This way we still may assure the absolute output error bound $< \epsilon = 2^{-q}$; note also that $|X| < 2^m$.) Then the coefficients of the polynomial q(y) of (3.2) are integer multiples of $2^{(q+2)n}$ (that is, they take the form $h/2^{(q+2)n}$ where h is an integer), so it is sufficient to compute them with absolute errors less than $1/2^{(q+2)n+1}$ and to recover their exact values via rounding-off. We reduce computing the shift to convolution of two vectors whose entries have absolute values $< 2^{bn}$ (compare (11.1) and [22]), so that O(bn) bit-precision of computations will suffice. (Convolution can be reduced to FFT whose error analysis is available, see [13] or [3], p. 194, or alternatively to integer multiplication, [11], whose error estimates are available in [27].)

Stage b). Consider the evaluation of the power sums via the iterations (3.3) and via solving the system (3.5). Surely the few required iterations (3.3) (reduced to convolutions and DFTs) cannot blow-up the errors too much (compare the error estimates from [13] or [27]), so we will only analyze the errors of the solution of the triangular Toeplitz systems (3.5). For an entry s of the inverse of a (gN) × (gN) unit triangular Toeplitz matrix T, we have the following useful estimate from [3], Lemma 2, p. 192, $|(\log |s|)| = O(gN \log(1+t))$, so $\log(|s|^{1/(gN)}) = O(\log(1+t))$, where t denotes the maximum absolute value of an entry of T. In fact the diagonal entries of the coefficient matrix of the system (3.5) equal $c_{d,k} \neq 0$, which is the N-th power of the leading coefficient of the polynomial $q(y) = y^d p(X + 1/y)$, see (3.2). We will keep our previous assumptions (see part a) above) that the coefficients of p(x) are m-bit integers and that X has real and imaginary parts of the form $h/2^{q+2}$ for an integer h, so all the coefficients of q(y) are integer multiples of $1/2^{(q+2)n}$, and it suffices to scale the system by $2^{(q+2)nN}$, which means the increase of the precision by $(q+2)nN = O(bn)$ binary bits. Thus, due to the estimate from [3], 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 result if we used Cauchy's integrals (3.6) in order to compute $|s_{gN}|^{1/(gN)}$, compare [27], p. 34.) Summarizing we obtain that the *computations with* O(bn) *binary bit precision suffice to support the applications of Algorithm 3.1* (and in fact similarly to support all other steps of our algorithms).

The errors of finite precision computation of polynomial zeros are closely related to the errors due to the perturbations of the coefficients. The known bounds on the perturbation errors, see [16], pp. 74-77; [2], Theorem 4, [18, 27], suggest that our estimate O(bn) for the bit-precision cannot decrease if applied to all the polynomials p(x) of degree n. For many input polynomials, however, the perturbation bounds of order bn are overly pessimistic, exceeding the actual values by a factor of n, and that property is translated to our precision bounds. Similarly the precision of computations could be decreased by a factor of n if we only need *to factor* p(x) *numerically*, that is, to compute 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 $\leq \epsilon$, see [27].

Finally we combine our arithmetic complexity estimates with the known bounds on the Boolean circuit complexity of arithmetic operations over integers modulo 2^h (that is, with the estimates $O_B(t(h))$ for the Boolean sequential time and $O_B(\log^2 h, t(h))$ for the Boolean parallel cost where $t(h) = O(h \log h \log\log h)$, see [1, 5, 17, 26, 27]). We apply those bounds with $h = O(bn)$, multiply the entries of Table 1.1 by t(h) or by $\log^2 h$, respectively, and arrive at the Boolean circuit complexity estimates for the problems of computing polynomial zeros. The resulting Boolean complexity bounds coincide within (poly)logarithmic factors with (yet unproven) estimates stated in [27]. Incorporating the estimates of [27] for the Boolean complexity of polynomial multiplication, DFT, and shifts of the variable and of [3] for polynomial division would probably improve our estimates for the Boolean complexity of computing all the zeros of p(x) by (poly)logarithmic factors in n.

12. Alternatives, Improvements, and Open Problems

We may modify our algorithm in a number of ways, keeping the same overall estimates for the asymptotic arithmetic complexity (within logarithmic factors). Let us briefly list some of them.

1) Instead of the center of gravity M of k zeros of p(x) in a disc D, computed via (8.1), (8.2), we could compute the zero z of the (k-1)-th derivative of p(x) in D via Newton's iteration, provided that i.r.(D) $> 80n^3$; z may serve similarly to M; the cost of computing z is $O_A(\log^2 n, n/\log n)$, which is only slightly worse than the bound of Proposition 7.1, compare [24, 25].

2) Turan's test can be replaced by the root radius algorithm of [27], Sect. 15, whose arithmetic cost, is $O_A(\log n \log\log n, n)$, slightly higher than Turan's; the proof of that cost bound in [27] is elementary. In the applications of Section 6, we can see some other alternatives to (but not improvement of) Turan's Algorithm 3.1.

3) Using the algorithms of [27], we may effectively compute the coefficients of the factors $\prod_{h=1}^{k} (x - x_{j(h)})$ of p(x), given a square $S = S(Y,R)$ containing exactly those k zeros $x_{j(h)}$, h=1,...,k, provided that, say i.r.(S) ≥ 3. Such an isolation of those factors is obtained via numerical integration over the boundary of the disc D(Y,2R) circumscribing S and via subsequent Newton's iteration. The asymptotic cost seems to remain the same as in our algorithm. The latter approach can be extended to replace Turan-Weyl's algorithm: we may apply the integration already where all the zeros of p(x) are included into two or several discs having

isolation ratio say $\geq 1+1/n$; such discs exist and can be computed for a low cost where Subalgorithm 6.1 has been applied and has output discs D_1 and D_2 satisfying (6.2), see [24]. In that approach the error analysis becomes much harder, but apparently the Boolean circuit complexity bound (but not arithmetic bounds!) for computing a single complex zero of p(x) can be decreased by a factor of n or so, compare [27].

The major open problem is computing all the complex zeros of p(x) for the cost $O_A((\log n)^{O(1)},n^{O(1)})$. It is also interesting (at least theoretically) if the root radius estimate of [29] can be extended to cover also the approximation to the s-th root radius of p(x) for $s>1$.

Part 2. Toeplitz Computations and Applications

2. Some Basic Definitions

$[\mathbf{T}]$ denotes a set of all $n\times n$ Toeplitz matrices, containing the two subsets $[\mathbf{L}]$ and $[\mathbf{U}]$ of lower and upper triangular Toeplitz matrices, respectively; T, L, and U (with or without subscripts, superscripts, and so on) will be our notation for the members of the sets $[\mathbf{T}]$, $[\mathbf{L}]$ and $[\mathbf{U}]$, respectively. We will write $L = L(\bar{x})$, $U = U(\bar{y})$ if $\bar{x}$ is the first column of L, and if $\bar{y}^T$ is the first row of U. Here and hereafter the superscript T indicates transposition, while the superscript r will indicate the reversion of the order of the entries of a vector. $[\mathbf{LU}]_d$ denotes the set of $n\times n$ matrices that allow the following *displacement representation*, $[\mathbf{LU}]_d := \{\sum_{i=1}^{d} L_iU_i\}$, [9]. For a matrix $V = \sum_{i=1}^{d} L_i(\bar{x}^{(i)})\, U_i(\bar{y}^{(i)})$ where $L_i \in [L], U_i \in [U]$, the set B of the pairs of the vectors $(\bar{x}^{(i)},\bar{y}^{(i)})$ is called a (nonunique) LU-*displacement generator of length* d for V (d being an upper bound on the *displacement ranks* of V and W, compare [9, 12]). We will use the shorthand, LU-generator. A Toeplitz matrix $T = LI+IU$ lies in $[\mathbf{LU}]_2$. Here and hereafter I denotes the identity matrix, $I \in [L], I \in [U]$.

3. Auxiliary Results

Proposition 3.1. *Given an* LU-*generator of length* d *for a matrix* V, *the cost of computing any fixed column of* V *is* $O_A(\log(dn),dn)$.

Proposition 3.2, [9]. *Let two matrices* V_1 *and* V_2 *be given with their* LU-*generators* B_1 *and* B_2 *of lengths* d_1 *and* d_2, *respectively; let* c *be a constant. Then an* LU-*generator of* V_1V_2 *of length* $\leq d_1+d_2+1$ *can be computed for the cost of* $O_A(\log(d_1d_2n),d_1d_2n)$, *and (obviously)* $B_1 \cup (cB_2)$ *is an* LU-*generator of* V_1+cV_2 *of length* $\leq d_1+d_2$. *Here the set* cB_2 *is obtained from the set* B_2 *by multiplying all the z-vectors of* B_2 *by* c.

Theorem 3.1, [7]. *Let* $T \in [\mathbf{T}]$ *be an* $n\times n$ *uppermost principal submatrix of an* $(n+1)\times(n+1)$ *Toeplitz matrix* T^*, *both* T *and* T^* *be nonsingular;* $\bar{x} = [x_0,\dots,x_{n-1}]^T$ *and* $\bar{x}^* = [x_0^*,\dots,x_n^*]^T$ *be the first columns of* T^{-1} *and* $(T^*)^{-1}$, *while* $\bar{y}^r = [y_{n-1},\dots,y_0]^T$ *and* $(\bar{y}^*)^r = [y_n^*,\dots,y_0^*]^T$ *be the last columns of the same matrices. Let* $\bar{y} = [y_0,\dots,y_{n-1}]^T$, $\bar{u} = [0,x_{n-1},\dots,x_1]^T$, $\bar{v} = [0,y_{n-1},\dots,y_1]^T$, $\bar{u}^* = [x_n^*,\dots,x_1^*]^T$, $\bar{v}^* = [y_n^*,\dots,y_1^*]^T$. *Then*

$$x_0T^{-1} = L(\bar{x})U(\bar{y}) - L(\bar{v})U(\bar{u}), \tag{3.1}$$

$$x_0^*T^{-1} = L(\bar{x}^*)U(\bar{y}^*) - L(\bar{v}^*)U(\bar{u}^*), \tag{3.2}$$

$$x_0^* = y_0^* = \det T/\det T^*.$$

Definition 3.1. The pair of the vectors $\bar{x},\bar{y}$ of (3.1) if $x_0 \neq 0$ and the pair of the vectors $\bar{x}^*,\bar{y}^*$ of (3.2) if $x_0^* \neq 0$ will be called the *canonical generators* of T^{-1}.

Corollary 3.1. *Under the assumptions of Theorem 3.1, all the diagonal entries of the matrix* T^{-1} *can be computed from its canonical generator for the cost of* $O_A(\log n,n)$.

4. Inversion of Very Strongly Diagonally Dominant Toeplitz Matrices

Definition 4.1. $\|A\| = \max_j \sum_i |a_{ij}|$ for a matrix $A = [a_{ij}]$. A is called *strongly diagonally dominant* if for a constant q,

$$\|I - A\| < q < 1. \tag{4.1}$$

Proposition 4.1, see, e.g., [19]. *Let (4.1) hold and let Newton's iteration for computing* A^{-1} *be defined as follows,*

$$B_0^* = I, B_i^* = 2B_{i-1}^* - B_{i-1}^*A\,B_{i-1}^*, \ i=1,2,\dots \tag{4.2}$$

Then

$$\|I - B_i^*A\| \leq \|I - B_{i-1}^*A\|^2 < q^{-2^i},$$

$$\|B_i^* - A^{-1}\| * \|I-B_i^*A\| * \|A^{-1}\| < (1+q)q^{-2^i} \text{ for all i.}$$

Next assume that $A = T \in [\mathbf{T}]$ and modify Newton's iteration (4.2) as follows,

$$B_1 = 2I-T, \tilde{B}_i = 2B_{i-1}-B_{i-1}TB_{i-1}, \tag{4.3}$$

$$B_i = (L(\bar{x}^{(i)})U(\bar{y}^{(i)})-L(\bar{v}^{(i)})U(\bar{u}^{(i)}))/x_0^{(i)}, \ i=2,3,\dots$$

Here $\bar{x}^{(i)}$ and $(\bar{y}^{(i)})^r$ denote the first and the last columns of the matrix $\tilde{B}_i$ of (4.3), $\bar{y}^{(i)}$ denotes the reversed $(\bar{y}^{(i)})^r$, and $\bar{u}^{(i)}$ and $\bar{v}^{(i)}$ are defined by $\bar{x}^{(i)}$ and $\bar{y}^{(i)}$ the same way as $\bar{u}$ and $\bar{v}$ are defined by $\bar{x}$ and $\bar{y}$ in Theorem 3.1.

Proposition 4.2. *Let (4.1) and (4.3) hold for* $T = A$ *and* $q = 1/10$; *denote* $\|\tilde{B}_i - T^{-1}\| = a(i)$. *Then*

$$a(i+1) = \|\tilde{B}_{i+1} - T^{-1}\| < a^2(i)(10/9)\,(2.2(9a(i)+20)/(8-9a(i))^2.$$

Let $q = 1/10$. Note that $\tilde{B}_2$ of (4.3) and B_2 of (4.2) denote the same matrix if A=T. Therefore $a(2) < 1/9000$, see Proposition 4.1. If

$$a(i) < 1/9000, \tag{4.4}$$

then Proposition 4.2 implies that $a(i+1) = \|\tilde{B}_{i+1}-T^{-1}\|$ satisfies

$$a(i+1) < a^2(i)(10/9)(2.2*20.001/7.999)^2 < 34a^2(i). \tag{4.5}$$

For i=2, (4.4) and therefore (4.5) hold, which implies (4.4) and (4.5) for i=3 and then recursively for i=4,5,6,..., so that

$$34a(i) < 1/250^{2^{i-2}} \text{ for all } i \geq 2.$$

Consequently $\tilde{B}_i$ and therefore also B_i converge to T^{-1} quadratically as $i \to \infty$.

On the other hand, T and B_i for all i are in $[\mathbf{LU}]_2$, so the cost of each iteration of (4.3) is $O_A(\log n,n)$, see Propositions 3.1 and 3.2, and we arrive at the following result.

Theorem 4.1. *Let* c *be a constant,* A=T *be an* $n\times n$ *Toeplitz matrix satisfying (4.1) for* $q = 1/10$. *Then the canonical generator* $\bar{x}$, $\bar{y}$ *of (3.1) of the inverse matrix* T^{-1} *can be computed with error norm* $\leq 1/250^{n^c}$ *for the cost* $O_A(\log^2 n,n)$.

5. Computing Matrix Powers

Definition 5.1. The *canonical set of entries* of a square matrix is the set of all its entries in its first and its last columns and on its diagonal.

Next let us compute the canonical sets of entries of the powers T^k of a given $n\times n$ Toeplitz matrix T for $k=2,3,...,q$ where $q\geq n$; in this paper we will only use $q=n$ and (in Appendix) $q=2n$. Let $m=2q$, ω be a primitive $(m+1)$-th root of 1, h be such that

$$0 < h\,||T|| < 1/10,\ h_j = h\omega^j \text{ for } j=0,1,...,m. \tag{5.1}$$

Then (4.1) holds for $A=I-h_jT$ for each j, so we may apply the algorithm (4.3), Theorem 4.1, and Corollary 3.2 and compute the canonical set of entries of the matrix

$$(I-h_jT)^{-1} = \sum_{k=0}^{\infty} (h_jT)^k = \sum_{k=0}^{\infty} (hT)^k\omega^{jk} \tag{5.2}$$

with absolute error bound

$$E^* = (1/34)/250^{n^c} \tag{5.3}$$

(where c is an arbitrary constant) for the cost $O_A(\log^2 n, n)$.

Let $m=2q$, $S_j = \sum_{k=0}^{m} (hT)^k\omega^{jk}$. Then

$$\tilde{E} = ||(I-h_jT)^{-1}-S_j|| \leq (h||T||)^m < 1/10^m, \tag{5.4}$$

see (5.2). The entries of $(I-h_jT)^{-1}$ approximate to the entries of S_j with absolute errors $\leq E^*+\tilde{E}$, see (5.3), (5.4). The cost of computing all those approximations for all j is $O_A(\log^2 n, mn)$. Next apply inverse FFTs in order to compute the canonical set of entries of $(hT)^k$ for all $k \leq m$ (FFT is applied once for each entry of the canonical set), and then divide the results by h^k to obtain the desired entries of T^k. The divisions by h^k increase the errors h^k times, and $k\leq q=m/2$ in our case.

Theorem 5.1. *The canonical set of entries of the powers T^k of an $n\times n$ Toeplitz matrix for $k=2,3,...,q, q\geq n$, can be computed with absolute errors less than*

$$E = O(1/(250^{n^c}h^q)+(n^2\log n)/10^q) \tag{5.5}$$

for the cost $O_A(\log n \log(nq), nq)$ using the precision of computations of $O((q+n)\log(||T||+(1/h)))$ binary bits, where a positive h satisfies (5.1), $|\log h| = O(\log||T||)$, and c is an arbitrary constant. If all the entries of T are integers and if the error bound E of (5.5) is less than 1/2, then the exact values of the entries of the canonical sets of T^k can be recovered for all k via rounding-off the computed approximations to the nearest integers.

6. Csanky-LeVerrier Algorithm for Toeplitz Computations

Recall the following definitions and facts.

$$s_k = \text{trace}(T^k) = k=1,...,n, \tag{6.1}$$

$$\bar{c} = (D+L)^{-1}\bar{s}, \tag{6.2}$$

$D = \text{diag}(1,2,..,n)$, $L = L(\bar{p}) \in [L]$, $\bar{p} = [0,s_1,...,s_{n-1}]^T$;

$$\det(\lambda I-T)=\lambda^n-\sum_{i=1}^{n} c_i\lambda^i=\prod_{j=1}^{n} (\lambda-\lambda_j),\ c_0=(-1)^{n+1}\det(T);$$

$$T^{-1} = (T^{n-1} - \sum_{j=1}^{n-1} c_jT^{j-1})/c_0.$$

Algorithm 6.1 (Csanky-LeVerrier), [10]. **Input:** T^k, $k=1,2,...,n$.

Successively compute the vectors $\bar{s} = [s_1,s_2,...,s_n]^T$ of (6.1) and $\bar{c} = [c_{n-1},c_{n-2},...,c_0]^T$ of (6.2) and then the matrix

$$\text{adj}(T) = \pm (T^{n-1} - \sum_{j=1}^{n-1} c_jT^{j-1}). \tag{6.3}$$

Then define T^{-1} as the pair (adj(T), det(T)) or if $c_0 \neq 0$, compute

$$T^{-1} = \text{adj}(T)/\det(T). \tag{6.4}$$

Applying Algorithm 6.1 in the case of a Toeplitz matrix T, we take the canonical sets of entries of T^k as the inputs and only compute the first and the last columns of adj(T) and of T^{-1} or, if $x_0 = 0$ in (3.1), we imbed T into the $(n+1)\times(n+1)$ matrix T^* of Theorem 3.1 (choosing the entries t^*_{n0} and t^*_{0n} of T^* such that T^* is nonsingular) and repeat the computation with T replaced by T^*. Computing the vector $\bar{c}$ that satisfies (6.2) can be reduced to the inversion of the triangular matrix D+L for the cost of $O_A(\log^2 n, M(n))$, but we use the elegant algorithm of [27], sect. 13, in that stage and arrive at the following cost bounds.

Table 6.1.

(6.1) and Theorem 5.1	(6.2)	(6.3) and (6.4)
$O_A(\log^2 n, n^2)$	$O_A(\log^2 n, n/\log n)$	$O_A(\log n, n^2/\log n)$

7. Extensions to Computations with Different Structured Matrices and to Computing the Greatest Common Divisor of Polynomials and the Rational Interpolation

Our algorithms and complexity bounds can be immediately extended to computations with many well structured matrices, such as all the matrices of the class (2.1) (in that case the asymptotic parallel time bounds do not change and the processor bounds increase d times), the matrices of similar classes, defined by Hankel type displacement generators or by some other similar operators, and so on, see [9, 12]. If a matrix V belongs to such a class of structured matrices, then so are the matrices V^T and V^TV, see [9] (which implies immediate extensions of our results to least-squares computations with structured matrices); furthermore our algorithm can be easily extended also to the case of structured block matrices. In fact, our approach works whenever Theorem 3.1 and Proposition 3.2 can be extended, and numerous extensions of those theorem and proposition are presented or implicit in [9, 12]. In particular this implies

Theorem 7, [23]. *The greatest common divisor of two given polynomials of degrees at most n with integer coefficients whose absolute values are less than a fixed value t can be computed for the cost of $O_A(\log^3 n, n^2)$ using the precision of computations of $O(n\log(nt))$ binary bits.*

Similarly our approach can be extended to several other rational and polynomial computations (such as the evaluation of the gcd of several polynomials, of the least common multiple of two or several polynomials, of the entries of the extended Euclidean scheme, of the squarefree decomposition of polynomials, of elementary symmetric functions, of partial fraction decomposition, of rational interpolation (Hermite or Cauchy) and of Padé approximation) basing on the known reduction of those problems to solving Toeplitz linear systems, see [7, 30, 31].

Appendix. Fast Sequential Inversion of Toeplitz and Almost Toeplitz Matrices

We may modify the method of Section 4 and extend it to p-adic computations for the inversion of integer matrices of the class $[LU]_d$, which will lead us to their effective sequential inversion (although its parallel Boolean time is at best linear). The algorithm can be extended to the inversion of other structured matrices.

Proposition A.1 *Let* S *be a nonsingular* $n \times n$ *integer matrix from the class* $[LU]_d$ *for a natural constant* d, k *be a positive integer,* $p > n$ *be a prime in the interval*

$$n^3||T||^{6/n^c} < p < n^{4+c}||T||^{6/n^c}$$

for a positive constant c,

$$p^N \geq (n||T||)^n > p^{N/2} \text{ for } N = 2^k.$$

Then the matrix S mod p *is nonsingular with probability converging to 1 as* $n \to \infty$, *and the matrix* S^{-1} *can be computed for the overall sequential Boolean cost* $O_B(n \log(nh)\mu(h))$.

References

[1]. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1976.

[2]. 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. 18-th Ann. ACM Symp. on Theory of Computing*, pp. 340-349, 1986.

[3]. D. Bini and V. Pan, "Polynomial Division and Its Computational Complexity," *Journal of Complexity*, vol. 2, pp. 179-203, 1986.

[4]. R.R. Bitmead and B.D.O. Anderson, "Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations," *Linear Algebra and Applications*, vol. 34, pp. 103-116, 1980.

[5]. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[6]. A. Borodin, J. von zur Gathen, and J. Hopcroft, "Fast Parallel Matrix and GCD Computation," *Information and Control*, vol. 52,3, pp. 241-256, 1982.

[7]. R.P. Brent, F.G. Gustavson, and D.Y.Y. Yun, "Fast Solution of Toeplitz Systems of Equations and Computation of Pade Approximations," *J. of Algorithms*, vol. 1, pp. 259-295, 1980.

[8]. J.R. Bunch, "Stability of Methods for Solving Toeplitz Systems of Equations," *SIAM J. Sci. Stat. Computing*, vol. 6,2, pp. 349-364, 1985.

[9]. J. Chun, T. Kailath, and H. Lev-Ari, "Fast Parallel Algorithm for QR-factorization of Structured Matrices," *Information Science Lab., Stanford University*, Stanford, CA, 1986.

[10]. L. Csanky, "Fast Parallel Matrix Inversion Algorithm," *SIAM J. Computing*, vol. 5,4, pp. 618-623, 1976.

[11]. M.J. Fischer and M.S. Paterson, "String Matching and Other Products," *SIAM-AMS Proc.*, vol. 7, pp. 113-125, 1974.

[12]. B. Friedlander, T. Kailath, M. Morf, and L. Ljung, "Extended Levinson and Chandrasekar Equations for General Discrete-Time Linear Estimation Problems," *IEEE Trans. Automatic Control*, vol. AC-23,4, pp. 653-659, 1978.

[13]. W. Gentleman and G. Sande, "Fast Fourier Transform for Fun and Profit," *Proc. Fall Joint Comput. Conf.*, vol. 29, pp. 563-578, 1966.

[14]. P. Henrici, *Applied and Computational Complex Analysis*, Wiley, N.Y., 1974.

[15]. Jia Wei Hong, "How Fast the Algebraic Approximation Can Be?," *Scientia Sinica* (series A), vol. XXIX,8, pp. 813-823, 1986.

[16]. A.S. Householder, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill, New York, 1970.

[17]. D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison-Wesley, 1981.

[18]. M. Mignotte, "Some Inequalities About Univariate Polynomials," *Proc. 1981 ACM Symp. on Symbolic and Algebraic Computation*, pp. 195-199, 1981.

[19]. V. Pan and J. Reif, "Efficient Parallel Solution of Linear Systems," *Proc. 17-th Ann. ACM Symp. on Theory of Computing*, pp. 143-152, Providence, R.I., 1985.

[20]. V. Pan, "Fast and Efficient Parallel Algorithms for Exact Inversion of Integer Matrices," *Proc. Fifth Conf. on FST & TCS*, vol. 206, pp. 504-521, Lecture Notes in Computer Science, Springer, 1985.

[21]. V. Pan, "Complexity of Parallel Matrix Computations," to appear in *Theoretical Computer Science*.

[22]. V. Pan, "Algebraic Complexity of Computing Polynomial Zeros," *to appear in Computers & Math. (with Applics.)*, 1987.

[23]. V. Pan and J. Reif, "Displacement Structure and the Processor Efficiency of Fast Parallel Toeplitz Computations," Technical Report 87-9, Computer Science Department, State University of New York at Albany, 1987.

[24]. V. Pan, "On Sequential and Parallel Complexity of Approximate Evaluation of Polynomial Zeros," to appear in *Computers & Math. (with Applics)*., 1987.

[25]. J. Renegar, "On the Worst-Case Arithmetic Complexity of Approximating Zeros of Polynomials," *Proc. 2-nd Symposium on the Complexity of Approximately Solved Problems*, 1987.

[26]. J.E. Savage, *The Complexity of Computing*, Wiley and Sons, N.Y., 1976.

[27]. A. Schönhage, "The Fundamental Theorem of Algebra in Terms of Computational Complexity," manuscript, Dept. of Math., University of Tübingen, Tübingen, West Germany, 1982.

[28]. S. Smale, "Algorithms for Solving Equations," *Proc. of Intern. Congress of Math.*, Berkeley, 1987.

[29]. P. Turan, *On a New Method of Analysis and Its Applications*, Wiley & Sons, New Jersey, 1984.

[30]. J. von zur Gathen, "Parallel Algorithms for Algebraic Problems," *SIAM J. on Comp.*, vol. 13,4, pp. 802-824, 1984.

[31]. J. von zur Gathen, "Representations and Parallel Computations for Rational Functions," *SIAM J. on Computing*, vol. 15,2, pp. 432-452, 1986.