Approximating Complex Polynomial Zeros: Modified Weyl's Quadtree Construction and Improved Newton's Iteration *

Victor Y. Pan
Mathematics and Computer
Science Department
Lehman College, CUNY
Bronx, NY 10468
email address: vpan@lcvax.lehman.cuny.edu
(Supported by NSF Grants CCR 9020690 and CCR 9625344
and PSC-CUNY Awards Nos. 662478, 664334 and 667340)

Abstract

We propose a new algorithm for the classical and still practically important problem of approximating the zeros z_j of an n-th degree polynomial $p(x)$ within the error bound $2^{-b} \max_j |z_j|$. The algorithm uses $O((n^2 \log n) \log(bn))$ arithmetic operations and comparisons for approximating all the n zeros and $O((kn \log n) \log(bn))$ for approximating the k zeros lying in a fixed domain (disc or square) isolated from the other zeros. Unlike the previous fast algorithms of this kind, the new algorithm has its simple elementary description, is convenient for practical implementation, and in the process of computing allows the users to adapt the computational precision to the current level of approximation and ultimately to the requirements on the output precision for each zero of $p(x)$. The algorithm relies on the novel versions of Weyl's quadtree construction and Newton's iteration.

Key words: Complex polynomial zeros, algorithms, computational complexity.
1991 Mathematics Subject Classification: 65H05, 68Q25, 68Q40, 65B99, 26C10, 30C15.

*The results of this paper have been presented at the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, Virginia, January 1994.

1 Introduction

1.1 The Problem and Our Results

Our subject is the classical problem of the approximation of the zeros of a polynomial

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - z_j), \quad p_n \neq 0 \ , \tag{1.1}$$

within the error bound $\epsilon \max_j |z_j|$ for a fixed positive ϵ. We will refer to this problem as **Problem 1.1**. The problem is stated numerically, its solution treats equally the clusters of the zeros and the multiple zeros of $p(x)$. Numerical truncation of the input coefficients routinely turns the multiple zeros into the clusters, so the clusters are not a rare phenomenon in computational practice. Problem 1.1, also known as the problem of solving a polynomial equation $p(x) = 0$, has history of over four millennia but still remains a major subject of practical significance and active research, attracting dozens of research publications every year [MN93], [P97].

Most important are applications to computer algebra, where the solution of multivariate polynomial systems of equations typically requires to approximate the zeros of high degree univariate polynomials with high precision in the presence of clusters of zeros. There are also important areas of computing where application of advanced polynomial rootfinders promises to be highly effective. In particular such is a major subject of approximating matrix eigenvalues on which we comment in the appendix.

Polynomial rootfinding is also a well established research problem in computer science or, more precisely, a set of problems, depending on various special restrictions on the input and requirements to the output. For example, the users may wish to solve **Problem 1.2** of approximating only those $k \leq n$ zeros of $p(x)$ that lie in a fixed disc D. In another typical example, the users may wish to approximate the ill-conditioned (clustered) zeros of $p(x)$ with a higher precision. (To see the motivation, examine a dramatic jump of the zero of the polynomial $p(x) = (x - 5/7)^{50}$ when the x-free term changes slightly.) Formally, we may include this requirement by restating Problems 1.1 and 1.2 as **Problems 1.3 and 1.4** where each output approximation is to be given by the pair of a disc D_h of a small radius $\epsilon_h \leq \epsilon$ and the index i_h showing the number of the zeros of $p(x)$ in D_h and where all discs D_h are disjoint and ϵ_h decreases as i_h grows. Surely, the users also wish to have numerically stable algorithms, easily accessible by a programmer and adaptive to various further requirements.

The main result of our paper is a new algorithm that solves Problem 1.2 for $\epsilon = 2^{-b}$ by using $O((kn \log n) \log(bn))$ arithmetic operations and comparisons and $O(n \log(bn))$ evaluations of the h-th roots of positive numbers for natural $h = O(n)$. (Hereafter, we will refer to all these elementary operations as to **ops** and will write $\mathbf{O_A}(f(n))$ to denote $O(f(n))$ ops.) We require that the input disc D be $(1 + c)$-*isolated* from the outside zeros for a fixed positive c, say, for $c = 0.01$ or $c = 0.01/n$, so that its $(1 + c)$-dilation should contain only the same k zeros of $p(x)$ as D. At the cost $\mathbf{O_A}((n \log n) \log \frac{\log n}{\log(1+c)})$, we may verify whether this assumption holds. If it does not hold, then within the same cost bound we may compute an integer k^+ satisfying $n \geq k^+ > k$ and a disc D_+, the δ-dilation of D for $1 < \delta < (1+2c)^{k^+-k}$, containing exactly k^+ zeros of $p(x)$ and satisfying the $(1+c)$-isolation assumption for D_+ and k^+ replacing D and k. Then we may solve Problem 1.2 for D_+ at the cost $\mathbf{O_A}((k^+ n \log n)\log(bn))$.

The same algorithm solves Problems 1.1, 1.3 and 1.4, and the same cost bounds apply except that we write $k = n$ for Problems 1.1 and 1.3 and $2^{-b} = \min_h \epsilon_h$ (rather than $2^{-b} = \epsilon$) for Problems 1.3 and 1.4 (see Remark 7.1 in section 7). Moreover, a simple extension of our algorithm works if the input disc is replaced by a square (see Remark 9.3 in section 9). Furthermore, the algorithm can be implemented with variable precision, which is set low initially and increases only when a computed approximation approaches a zero of $p(x)$ or a cluster of the zeros. The number of the more costly ops, which we have to perform with a higher precision, decreases in such an implementation (see Remark 7.1 in section 7).

The proofs of the correctness of the algorithm and of the cited complexity bounds are mostly elementary, although some of them are long and non-trivial. The description of the algorithm is much simpler and is supposed to be accessible easily by a programmer. The algorithm essentially amounts to recursive application of the auxiliary operations of the evaluation of $p(x)$ and $p'(x)$ at certain points x, the shift of the variable x (that is, computing the coefficients of the polynomial $t(x) = p(x+X)$ for a fixed complex value X), and approximation of the distance from X to the closest or the k-th closest zero of $p(x)$ for a fixed k.

The cited asymptotic complexity estimates ultimately rely on performing the two latter operations via fast polynomial multiplication, based on using FFT. In practical implementation, particularly for polynomials $p(x)$ of small and moderately large degrees, application of the classical algorithm or the one of [KO63] for polynomial multiplication can be more effective.

Our algorithm can be performed on p processors, with the acceleration by roughly the factor p, provided that $p \leq n$.

The algorithm can be effectively combined with other known algorithms for polynomial zeros to improve their performance (see our comments in subsections 1.2 and 1.4).

With some simplifications, the algorithm can be applied to approximating only the real zeros of $p(x)$ (in a fixed line interval) [PKSHZ96]. For approximating a single non-isolated zero of $p(x)$, the coefficient k^+ above may generally be large so that better complexity estimates can be achieved based on an algorithm of [P87], whose geometric construction incorporates the one of [L61].

Our techniques can be extended to approximating a finite set of the zeros of an analytic function in a disc if these zeros are isolated from all other zeros of the function and if effective algorithms known for proximity test and root radii computation for polynomials are extended respectively.

On an extension to approximating the eigenvalues of a symmetric tridiagonal matrix we refer the reader to [P93] and [PL93], on an extension to the unsymmetric eigenvalue computation see [P95a] and the appendix to this paper, where we point out some substantial potential advantages of shifting to polynomial rootfinders from the customary QR approach.

Technically, our algorithm amounts to an iterative process, which recursively invokes its two main blocks, one after another. The two blocks are Weyl's quadtree construction and Newton's iteration. We modified both of them and coordinated their combined recursive application based on our special concepts of the *isolation and rigidity ratios.*

Our analysis of the resulting algorithm (which gave us the stated complexity extimates) also largely relies on using the latter powerful concepts. In particular this applies to our proof of the quadratic convergence of our modification of Newton's iteration. Quadratic convergence is proved under a certain lower bound on the initial isolation ratio, and we also prove that our modification of Weyl's construction insures such a lower bound after a few steps. These results may be of independent interest for the study of Newton's iteration and Weyl's construction.

Our construction and analysis also include some more minor novel features, such as application of the theory of analytic functions and conformal maps to the analysis of Newton's iteration (cf. the proof of Lemma 10.1), our one-sided version of Turan's proximity test (cf. Remarks 3.2 and 6.4), our modification of Schönhage's root-radii algorithm (cf. Algorithm 4.3), and our rectangle superscription technique (cf. Algorithm 2.1).

To place our results into proper prospective, we will briefly recall two other major approaches to polynomial rootfinding as well as some history of Weyl's classical approach.

1.2 The Functional Iteration Approach

Practical approximation of the zeros of $p(x)$ presently relies on iterative algorithms based on the *Newton, Laguerre, and Jenkins-Traub* iteration processes [M73], [MR75], [HPR77], [F81], [JT70], [JT72], [IMSL87]. The most celebrated example is Newton's iteration,

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

which starts with some initial x_0 and is supposed to converge to a zero z_j of $p(x)$. If it does, the same process can be recursively applied to the deflated polynomial $p(x)/(x - z_j)$.

Theoretically, the weak point of these algorithms is their heuristic character. In spite of long and intensive effort of the researchers, the proof of the global convergence (that is, convergence right from the start) for the worst case input and the estimates for the convergence rate are hopelessly missing (unless already the initial point is close to a zero or another similar condition is satisfied). Nonetheless, the cited algorithms work satisfactory as heuristic algorithms for input polynomials of smaller degree (say, less than 50) having no clusters of their zeros, but fail for the larger degree polynomials $p(x)$, particularly for ones with multiple and/or clustered zeros [Go94].

The need for better algorithms has been accentuated by major applications to the thriving modern area of algebraic and symbolic computing, where dealing with a polynomial equation of degree over 200 (say) with many clusters of the zeros is a typical case.

Most of the numerous algorithms proposed for polynomial rootfinding are, like Newton's, functional iterations of the form

$$x_{i+1} = f(x_i), \quad i = 0, 1, \ldots, \tag{1.2}$$

for a fixed function $f(x)$ and an initial x_0, where x_i can be scalars or n-tuples. The functional iterations usually share the heuristic character of their special cases cited above (due to Newton, Laguerre, and Jenkins and Traub). We wish to mention the celebrated subclass based on the *Weierstrass method* [W903], also called Durand-Kerner's and sometimes Dochev's method and having many variations (cf. [P97], [BP,a]). This method is a multivariate version of Newton's iteration, which converges simultaneously to all the n zeros of $p(x)$ by using order of n^2 ops in each iteration. According to numerical experiments, the average case convergence of the algorithms based on this method is good, but the averaging misses the important case of polynomials with clustered zeros. The many attempts to prove sufficiently fast convergence for the worst case input have failed so far, leaving little hope for obtaining such proofs in the future. The desired proofs of quadratic convergence are available, however, in the case where the initial approximations are already close to the zeros of $p(x)$. Such initial approximations can be effectively supplied by our algorithm, which makes it a natural ally of the functional iterations.

For the sake of completeness, we will cite the so-called *path following version* of Newton's iteration of [KS94]. The paper [KS94] presents an algorithm, its convergence proof, and the worst case complexity estimate $\mathbf{O_A}(n^2 b \log^2 n)$ for computing approximations to all the n zeros of $p(x)$. In spite of the inferior complexity bound, the algorithm of [KS94] is technically interesting, and the path following methods are quite successful for multivariate polynomial computations [RS92], [SS93], [SS93a], [SS93b], [SS94], [SS96]. Strictly speaking, the algorithm of [KS94] is not a pure functional iteration of the form (1.2), but it amounts to recursive iterative application of such an iteration.

If we restrict our attention to other effective algorithms supported by the proofs of the *worst case estimates* for their computational complexity, it suffices to consider only two other major approaches, that is, *Weyl's and the divide-and-conquer approaches*, both combining *computational geometry constructions* for search and exclusion on the complex plane with *analytical tools,* such as Newton's iteration and numerical integration [Sc82], [P85], [P87], [R87], [BFKT86/89], [P89], [B-OT90], [BP91], [P95], [P96]. We will briefly recall these two approaches in the next two subsections.

1.3 Weyl's Quadtree Approach

H. Weyl proposed his ingenious algorithm in [We24] as an iterative process of search and exclusion on the complex plane directed towards simultaneous approximation of all the n zeros of $p(x)$. The algorithm can be viewed as a 2-dimensional version of the customary bisection of a line interval. Under the name of the *quadtree algorithm*, this algorithm has been successfully applied to various areas of practical importance, such as image processing, n-particle simulation, template matching, and computational geometry [Sa84], [Gre88], [Se94].

The elaboration of Weyl's approach in [GH72] leads to the solution of Problems 1.1 and 1.2 at the cost bounded by $\mathbf{O_A}(n^3 b \log n)$ and $\mathbf{O_A}(n^2 k^+ b \log n)$, respectively, assuming the incorporation of the modern fast polynomial arithmetic based on FFT. Lehmer, in [L61], modified the geometric search towards approximating a single zero. With deflation, this algorithm also solves Problem 1.1 at the above cost.

The algorithms of [Sc82] and [R87] supported the decrease by roughly factor $b/\log b$ of the above upper estimates, based on some analytic techniques. [R87] relied on Weyl's construction, whereas [Sc82] used a distinct (divide-and-conquer) approach. (The unpublished manuscript of [Sc82] incorporated many useful techniques, auxiliary algorithms, and asymptotic estimates for the Boolean complexity of these algorithms but has been remaining uncompleted since 1982.)

In [P87] Weyl's approach was pushed to its best, in terms of the record asymptotic complexity estimates, that is, $\mathbf{O_A}((n^2 \log n)(\log(bn))$ and $\mathbf{O_A}((k^+ n \log n)(\log(bn))$, which are asymptotically the same as the ones of our present paper. The algorithm of [P87], however, is non-practical because:

a) its analytic part relies on some sophisticated techniques of numerical integration in the complex domains, where the basic parameters are defined as a part of a recursive construction and are hardly accessible for their optimization by the user, and

b) the recursive merging of Weyl's geometric construction with the numerical integration stage of [P87] is complicated and hard to program on computers.

These difficulties are inherent in the approach of [P87]. To avoid them in our present paper, we had to develop a *distinct construction* with a different recursive process using modified Newton's iteration instead of numerical integration.

1.4 The Divide-and-Conquer Approach

The divide-and-conquer approach to approximating polynomial zeros relies on the recursive splitting of $p(x)$ into the product of smaller degree factors (ultimately linear), can be traced more than half-century back (cf. e.g. [SeS41]), includes many major technical contributions (see e.g. [Schr57], [DL67], [DH69], [Grau71], [Sc82]), and has recently culminated in the algorithms of [P95], [P96] that support optimal (up to a polylogarithmic factor) asymptotic complexity estimate $\mathbf{O_A}((\log^2 n + \log b) n \log^2 n)$. The algorithms, however, must not become practical methods of choice because they do not have simple structure, which complicates their implementation and indicates that a large overhead constant is hidden in the $\mathbf{O_A}$ notation above. Besides, the recursive splitting of $p(x)$ into factors requires a high precision of computing even where one seeks some well-conditioned zeros with a lower precision.

Practical efficacy of this approach can be enhanced in its combination with Weyl's construction: splitting algorithms can serve as an alternative to Newton's iteration where a high output precision is required and vice versa where a lower precision output suffices.

1.5 Organization of Our Paper

Sections 2−6 are devoted to some auxilary material. This includes several simple facts, basic definitions and complexity estimates for fast polynomial arithmetic in section 2, Turan's proximity test and the so-called Graeffe iteration in section 3, the root radii algorithms in section 4, the results on computing the number of the zeros of $p(x)$ in a fixed disc in section 5, and the classical version of Weyl's algorithm (incorporating Turan's test) in section 6. In section 7, we outline our iterative algorithm consisting of two blocks recursively invoked one after another. In section 8, we describe and analyze the block representing our modification of Weyl's construction. In section 9 we present our main algorithm, which combines our construction of section 8 and our modificaiton of Newton's iteration. In section 10 we analyze it and estimate the complexity of its performance. In section 11 we sketch another variant of Newton's iteration. In the appendix we comment on advantages of noncustomary application of polynomial rootfinders to approximating the eigenvalues of unsymmetric matrices.

Our presentation is mostly self-contained, though the following results and algorithms are cited with some source references but with no proofs or derivations:

(a) estimates for the cost of fast polynomial arithmetic (section 2),

(b) Turan's theorem (section 3),

(c) the reduction of the solution of a triangular Toeplitz linear system of equations to polynomial division (section 3),

(d) a proposition on winding number algorithms (section 5),

(e) a result from analytic function theory (section 10).

We also cited but have not reproduced the algorithms for computing the convex hull of a set on the plane and for approximating logarithms (section 4), as well as the correctness proof for Weyl's classical construction, but these subjects are not needed for our main algorithm and proofs. The result cited in (e) is needed for the correctness proof in section 10 but not for the presentation of our algorithms. Furthermore, by applying the algorithms of section 4, we could have avoided using the results cited in (b)−(d) at the price of the increase of our cost estimates by at most factor $\log \log n$.

Acknowledgements The present paper was submitted for publication in 1994 (cf. its proceedings version [P94]) and turned into a research report [P96a] in 1996 but very substantially revised in 1997 along the line suggested by the referee. Akimou Sadikou and Gabriel Dos Reis made several useful comments on the original draft of this paper. Frank Uhlig sent me reprint of [U97]. Olivier Devillers pointed me out the reference [Gra72].

2 Definitions and an Auxilliary Algorithm.

In this section, we will list some basic definitions and simple facts that are immediate consequences of these definitions. We will also specify a very simple (though apparently novel) algorithm that superscribes a rectangle (with its sides parallel to the coordinate axes) about a closed set on the complex plane.

As before, we will write $\mathbf{O_A}(t)$ to denote a total of $O(t)$ **ops**, that is, operations of the following classes: arithmetic operations with complex numbers, pairwise comparisons of positive numbers, and the evaluation of the h-th roots of positive numbers, for natural h.

log will denote logarithms to the base 2.

In complex domains we will usually count the polynomial zeros together with their multiplicities, not distinguishing between clustered and multiple zeros. "Computing a polynomial" will mean "computing its coefficients" (unless we explicitly specify otherwise).

$p(x)$ will denote a fixed polynomial of (1.1), of a degree n.

Definition 2.1 *$D = D(X, R)$ denotes the disc of a radius $\rho(D) = R$ with a center X on the complex plane; $S = S(X, R)$ denotes the square with the side length $2\rho(S) = 2R$ and with the vertices $X + R(1 + \sqrt{-1})$, $X - R(1 + \sqrt{-1})$, $X + R(1 - \sqrt{-1})$, $X - R(1 - \sqrt{-1})$.*

Remark 2.1 *Hereafter, we will only consider the rectangles and squares on the complex plane that have their sides parallel to the coordinate axes.*

Definition 2.2 *Two complex sets U and V are called* **equivalent** *if they contain exactly the same zeros of $p(x)$. Transformation of a set U into its equivalent subset 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 (see Figure 1): $r.r.(U) = inf(\rho(U^-)/\rho(U))$, $i.r.(U) = sup(\rho(U^+)/\rho(U))$. Here, $\rho(U)$ is defined in Definition 2.1, and the infimum and the supremum are over all the squares (or discs) U^- and U^+ that are equivalent to the square (respectively, disc) U and such that $U^- \subseteq U \subseteq U^+$ and U^+ and U are concentric. A disc or a square U are* **f-isolated** *if $i.r.(U) \geq f$.*

Definition 2.3 $d(U) = \max_{z_g,z_h} |z_g - z_h|$, $\quad d^*(U) = \max_{z_g,z_h} \max\{|Re\ z_g - Re\ z_h|, |Im\ z_g - Im\ z_h|\}$,
where $\max_{z_g,z_h}$ denotes the maximum over all the pairs z_g, z_h of the zeros of $p(x)$ in a complex set U.

Algorithm 2.1 (superscription of the smallest rectangle about a closed set).
Input: a closed set U on the complex plane.
Output: the side lengths and the real and imaginary coordinates of the center X of the smallest rectangle containing the set U (and having its sides parallel to the coordinate axes).
Computations: Compute the maximum M and the minimum m of the real parts of the points of U. Output $M - m$ and $\frac{M+m}{2}$. Repeat the same computations for the set of the imaginary parts of the points of U.

The next proposition immediately follows by inspection.

Proposition 2.1 *The two half-sums in the output of Algorithm 2.1 equal the real and imaginary parts of the center X of the smallest rectangle containing the set U and having its sides parallel to the coordinate axes. The two differences equal the lengths l_- and l_+ of the sides of this rectangle. The larger length $l^+ \geq l^-$ is also the side length of a smallest square $S(X, l^+/2)$ (having its sides parallel to the coordinate axes) superscribing the rectangle, and $2\rho_{min} = ((l^-)^2 + (l^+)^2)^{1/2}$ is the diameter of the smallest disc superscribing this rectangle. Furthermore, $l^+ = d^*(U)$ for $d^*(U)$ of Definition 2.3.*

Proposition 2.2 *$r.r.(S) = d^*(S)/(2\rho(S))$ for a square S; $r.r.(D) = d^*(D)/(c\rho(D))$, $\sqrt{2} \leq c \leq 2$, for a disc D.*

Proof: Let U denote the input square S or disc D. Apply Algorithm 2.1 and Proposition 2.1 and deduce that $2\min_{U^-}(\rho(U^-)) = l^+$ if $U = S$, $2\min_{U^-}(\rho(U^-)) \leq 2\rho_{min} \leq l^+\sqrt{2}$ if $U = D$, for U^- of Definition 2.2 and for $l^+ = d^*(U)$ and $2\rho_{min}$ of Proposition 2.1. Furthermore, clearly $2\rho(U^-) \geq l^+$ if $U = D$. Summarizing, we have $2\min_{U^-} \rho(U^-) = d^*(U)$ if $U = S$, $d^*(U) \leq 2\min_{U^-} \rho(U^-) \leq d^*(U)\sqrt{2}$ if $U = D$. Proposition 2.2 now follows from the definition of $r.r.(U)$ (Definition 2.2). □

Definition 2.4 *For a complex X, for an $f > 1$ and for a non-negative ε, the disc $D(X, \varepsilon)$ is called* **an f-isolated ε-cover** *of a zero z_j of $p(x)$ if this disc contains z_j and is f-isolated.*

Definition 2.5 *$i(p(x), U)$, the* **index** *of $p(x)$ in U, is the number of the zeros of $p(x)$ lying in a complex set U and counted with their multiplicities.*

Definition 2.6 *The n distances $r_1(X) \geq r_2(X) \geq \cdots \geq r_n(X)$ from a point X to the n zeros of $p(x)$ are called the* **root radii** *of $p(x)$ at X; in particular $r_s(X)$ is called the* **s-th root radius** *of $p(x)$ at X, and we will write $r_s(X) = \infty$ for $s \leq 0$, $r_s(X) = 0$ for $s > n$.*

Proposition 2.3 *$1/r_s(0)$ for $p(x)$ equals the $(n + 1 - s)$-th root radius at 0 of the reverse polynomial $p_{rev}(x) = x^n p(1/x)$. $r_s(X)$ for $p(x)$ equals $r_s(0)$ for $t(y) = p(y + X)$. $r_s(0)$ for $p(x)$ equals $ar_s(0)$ for $p(x/a)$ for any scalar $a > 0$.*

Proof. Compare the zero z_j of $p(x)$ with the zeros $1/z_j$ of $p_{rev}(x)$, $z_j - X$ of $t(y)$, and az_j of $p(x/a)$. $\square$

In this paper, we will use the known algorithms for some basic operations with polynomials, which support the following known complexity estimates (cf. e.g. [BP94], sections 1.2 and 1.3):

Proposition 2.4 *Multiplication and division with a remainder of two polynomials of degrees at most n can be performed at the cost $\mathbf{O_A}(n \log n)$.*

Proposition 2.5 *A shift of the variable x for a polynomial $p(x)$ of (1.1), that is, the transition from the coefficients of $p(x)$ to the coefficients of the polynomial $t(y) = p(y+X)$ for a fixed complex X, can be performed at the cost $\mathbf{O_A}(n \log n)$.*

For completeness, we also recall two trivial transformations of $p(x)$.

Proposition 2.6 *Scaling of the variable for $p(x)$ of (1.1), that is, the transition to $q(x) = p(ax) = \sum_{i=0}^{n}(p_i a^i)x^i$, costs $\mathbf{O_A}(n)$, whereas the reversion (of the order of the the coefficients) of $p(x)$, that is, the transition to $x^n p(1/x) = \sum_{i=0}^{n} p_i x^{n-i}$, is cost-free.*

Remark 2.2 *Our results stated for the unit disc $D(0,1)$ can be immediately extended to the disc $D(X,r)$, for any complex X and positive r; indeed it suffices to shift and to scale the variable.*

3 Squaring Polynomial Zeros and Isolation Ratio. Turan's Proximity Test.

In this section, we recall Turan's proximity test, which, at the cost $\mathbf{O_A}((1+\log N)n \log n)$, enables us to compute the distance from a complex point X to the closest zero of $p(x)$, with a relative error at most $5^{1/N} - 1$. Turan's algorithm relies on the following theorem that he proved (see [Tu84], p. 299) by using some sophisticated tools from number theory:

Theorem 3.1 *Under the notation of (1.1), let s_k denote the power sums $\sum_{j=1}^{n} z_j^k$, $k = 0, 1, \ldots$. Then we have $1 \le r_1(0)/\max_{g=1,\ldots,n} |s_{gN}/n|^{1/(gN)} \le 5^{1/N}$ for all natural N.*

By Theorem 3.1, we may closely approximate $r_1(0)$ via the computation of the power sums s_{gN} for a large N. Proposition 2.3 enables us to extend this computation to the approximation of $r_n(X)$, that is, to a proximity test at a point X.

Algorithm 3.1 (Turan's proximity test)

Input: Natural N and n, the coefficients of the polynomial $p(x)$ of (1.1), and a complex number X that is not a zero of $p(x)$.
Output: Two positive numbers $r = r(X)$ and $r^* = r^*(X) = 5^{1/N} r$ such that

$$1 \le r^*/r_n(X) \le 5^{1/N},\ 1 \le r_n(X)/r \le 5^{1/N},\ r_n(X) = \min_{j=1,\dots,n} |z_j - X|. \tag{3.1}$$

Computations:
Stage 1. Compute the values of the power sums,

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

Stage 2. Compute and output $r^* = \left[\max_{g=1,\dots,n} |\frac{s_{gN}}{n}|^{1/(gN)}\right]^{-1}$ and $r = r^*/5^{1/N}$.

Correctness of Algorithm 3.1 follows from Theorem 3.1.

The error factor $5^{1/N}$ of approximation to $r_n(X)$ converges to 1 as $N \longrightarrow \infty$. For our purpose in this paper, it suffices to choose $N = 32$; then

$$1.051581 < 5^{1/N} < 1.051582. \tag{3.2}$$

Stage 2 is immediately performed at the cost $\mathbf{O_A}(n)$, and here is Turan's algorithm for Stage 1, where he lets $N = 2^h$ be a power of 2:
Subalgorithm 3.2.
Stage (a). *Shift the variable by setting $y = x - X$ and compute the coefficients of the n - th degree polynomial $q(y)$ with the zeros $y_j = 1/(z_j - X)$, $j = 1, \dots, n$, such that*

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

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

Stage (b), *(Dandelin-Lobachevsky-Graeffe, [H70]). Let $q_0(y) = q(y)/p_0(X)$ and successively compute the coefficients of the polynomials*

$$q_{i+1}(y) = (-1)^n q_i(\sqrt{y}) q_i(-\sqrt{y}),\ i = 0, 1, \dots, h-1; \quad h = \log N. \tag{3.3}$$

Stage (c). *Compute the power sums s_{gN} of the zeros of $q_h(y) = \sum_{i=0}^{n} q_{i,h} y^h$ for $g = 1, \dots, n$, by solving the triangular Toeplitz system of Newton's identities in the variables $s_N, s_{2N}, \dots, s_{nN}$, which relate the coefficients of $q_h(y)$ to the power sums s_{gN} (cf. [H70]) :*

$$q_{n,h} s_N + q_{n-1,h} = 0,$$

$$q_{n,h} s_{2N} + q_{n-1,h} s_N + 2 q_{n-2,h} = 0,$$

$$\vdots$$

$$q_{n,h}s_{nN} + q_{n-1,h}s_{(n-1)N} + \cdots + nq_{0,h} = 0.$$

At Stage (a), we shift the variable x and reverse the polynomial; every iteration i of Stage (b) is a polynomial multiplication; Stage (c) of solving a triangular Toeplitz system amounts to polynomial division (see e.g. [P92a]). Due to Propositions 2.4–2.6, the overall cost of performing Algorithm 3.1 is bounded by

$$\mathbf{O}_{\mathbf{A}}(n(1+h)\log n), \;\; h = \log N. \tag{3.4}$$

Remark 3.1 *The iterative algorithm (3.3) will be used also in the next sections. Each step i of (3.3) squares the zeros of the polynomial $q_i(y)$. Indeed, for $i = 0$, we have $q_0(y) = \prod_j (y - y_j)$,*

$$q_1(y) = (-1)^n \prod_j (\sqrt{y} - y_j)(-\sqrt{y} - y_j) = \prod_j (y - y_j^2),$$

and similarly for $i = 1, 2, \ldots$. It follows that

$$q_h(y) = \prod_{j=1}^{n} (y - y_j^N) = \sum_{i=0}^{n} q_{i,h} y^j, \;\; N = 2^h. \tag{3.5}$$

The logarithm of the ratio $|y_j^N / y_i^N|$ grows linearly in N if $|y_j| > |y_i|$. Furthermore, together with the zeros of $q_i(y)$ and the variable $\sqrt{y}$, each iteration step (3.3) squares i.r.$(D(0,r))$ for any positive r. This enables us to increase i.r.$(D(0,r))$ from $f > 1$ to f^{2^h} at the cost $\mathbf{O}_{\mathbf{A}}(hn\log n)$.

Remark 3.2 *The lower bound of Theorem 3.1 is trivial. Moreover, we have*

$$nr_1^g(0) \geq |s_g| \quad \text{for} \quad g = 1, 2, \ldots. \tag{3.6}$$

This leads us to a simple one-sided proximity test for any fixed $g \geq 1$ (or a sequence of g) defining an upper bound on $r_n(0)$ if we successively compute (a sequence of) $|s_g|$ for the polynomial $q(y) = y^n p(X + 1/y)$ or $|s_{gN}|$ for the polynomial $q_h(y)$ of Subalgorithm 3.2. The computation of $|s_g|$ or $|s_{gN}|$ for $g = 1, \ldots, l$ can be reduced to polynomial division modulo y^{l+1} (performed at the cost $\mathbf{O}_{\mathbf{A}}(l \log l)$) and involves at most $l+1$ trailing coefficients of $q(y)$ or $q_h(y)$. If for a fixed l and for $g = 1, \ldots, l$, the upper bound of (3.6) on $\min_j |z_j|$ is not satisfactory, then one may recursively double l and repeat the computations. By Theorem 3.1, we will obtain a quite tight bound as l reaches n, if not earlier. The bound (3.6) and, consequently, the above proximity tests can be refined if we know good approximations to some zeros z_j of $p(x)$ or if we know that their contribution to $|s_g|$ is negligible; in these cases we may replace the values s_g by approximations to the g-th power sums of the remaining zeros and decrease the factor n in (3.6) respectively.

4 Root Radii Computation

In this section, we will approximate the root radii $r_s(X)$, for $s = 1, \ldots, n$, by initially following the line of [Sc82], section 14, and at the end, in Proposition 4.4 and Algorithm 4.3, simplifying slightly the algorithms of [Sc82]. We will assume that $X = 0$ (otherwise, we would have shifted the variable by letting $y = x - X$) and will write $r_s = r_s(X)$,

$$r_0 = \infty, \ r_{n+1} = 0. \tag{4.1}$$

Consider the two following tasks (note some redundancy in their statements).
Task r**.** *Given positive* r *and* Δ*, find a (generally non-unique) integer* s *such that*

$$r_{s+1}/(1+\Delta) < r < (1+\Delta)r_s.$$

Task s**.** *Given a positive integer* s*,* $1 \le s \le n$*, and a positive* Δ*, find a positive* r *such that*

$$r/(1+\Delta) < r_s < (1+\Delta)r.$$

It is sufficient to solve Tasks r and s for $1 + \Delta = 2n$ or $1 + \Delta = \sqrt{2n}$ because the extension to an abritrary positive Δ is immediate, by means of at most

$$g = g(\Delta) = \lceil \log(\frac{\log(2n)}{\log(1+\Delta)}) \rceil \tag{4.2}$$

iteration steps (3.3). Indeed, such an iteration step amounts to squaring $1 + \Delta$ in the context of Tasks r and s (see Remark 3.1), and we have $(1+\Delta)^{2^g} \ge 2n$, under (4.2). The computational cost of performing these iteration steps (as well as the cost of shifting the variable x, if needed) should be added to the overall cost of the solution, of course. Note that

$$g(\Delta) = 0 \ if \ 1 + \Delta \ge 2n; \ g(\Delta) = O(\log\log n) \ if \ 1/\Delta = O(1),$$

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

Most frequently, we will need to solve Task s for $s = n$, and we have the following corollary of Theorem 3.1 [compare (3.4)]:

Corollary 4.1 *(a) For* $s = 1$ *and* $s = n$*, Task* s *can be solved by means of the application of Algorithm 3.1 at the cost* $\mathbf{O_A}((1+g)n\log n)$*, where* g *is defined by (4.2). (b) Moreover, the cost decreases to* $\mathbf{O_A}(n \log n)$ *if* $1/\Delta \le O(1)$*.*

For part (a), we have a simpler alternative proof (compare [He74], pp. 451, 452 and 457, and [Sc82]). Recall the well-known inequalities:

$$t_1^*/n \le r_1 < 2t_1^*, \ t_1^* = \max_{k>0} |p_{n-k}/p_n|^{1/k}. \tag{4.3}$$

Apply Proposition 2.3 for $X = 0$ and extend the bounds (4.3) as follows:

$$t_n^*/2 < r_n \le nt_n^*, \ t_n^* = \min_{k>0} |p_0/p_k|^{1/k}. \tag{4.4}$$

Here, the minimization process ignores those k for which $p_k = 0$.

Therefore, if $1 + \Delta > \sqrt{2n}$, then $r = t_1^* \sqrt{2/n}$ is a solution to Task s, for $s = 1$, whereas $r = t_n^* \sqrt{n/2}$ is a solution to Task s for $s = n$. At the cost $\mathbf{O_A}(ng \log n)$ of performing g iteration steps (3.3), the solution can be extended to the solution of Task s for $s = 1$ and $s = n$, for an arbitrary positive Δ, and for g of (4.2). This implies the cost bound of part (a) of Corollary 4.1. $\square$

We will also need to solve Task s for $1 < s < n$ (see Remark 7.2 and Stage 4 of Algorithm 9.1) and Task r (see Remark 5.1). Next, we will show solution algorithms relying on the following useful and elegant result:

Theorem 4.1 *[He74], pp. 458-462; [Sc82]. If* $1 \le m \le n$ *and if* $|p_{n+1-m-i}/p_{n+1-m}| \le av^i$ *for* $i = 1, \ldots, n+1-m$, *then* $r_m < m(a+1)v$.

Proof. Due to Van Vleck's theorem ([He74] , p. 459), we have

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

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

$$1 \le ax^{n+1-m}\binom{n}{m-1} + ax^{n-m}\binom{n-1}{m-1} + \cdots + ax^2\binom{m+1}{m-1} + ax\binom{m}{m-1}. \tag{4.5}$$

If $x \ge 1$, then $r_m \le v$, and Theorem 4.1 follows. Otherwise (4.5) implies that

$$1 + a < a(1 + x + x^2 + x^3 + \cdots)^m = a/(1-x)^m.$$

By applying the Lagrange formula to y^d for $y = 1 + a$, we obtain that $y^d - a^d = du^{d-1}$ for some u, $a \le u \le y$. Substitute this expression and deduce that

$$1/x < \frac{(a+1)^d}{(a+1)^d - a^d}, \ d = 1/m,$$

so that $r_m/v = 1/x < (a+1)/d$, and then again Theorem 4.1 follows. $\square$

Theorem 4.1 and Proposition 2.3 also imply a similar upper bound on $1/r_m$, which is the $(n+1-m)$-th root radius of the reverse polynomial $x^n p(1/x)$. The latter bound and one of Theorem 4.1 together immediately imply the solution of Task r for $r = 1$ and $1 + \Delta = 2n$. Indeed, compute a natural m such that $|p_{n+1-m}| = \max_{0 \le i \le n} |p_i|$. If $m = n + 1$, then $t_n^* \ge 1$, and, consequenty, $r_n > 1/2$ by (4.4), whereas $r_{n+1} = 0$ by (4.1). Therefore, $s = n$ is a desired solution to Task r for $r = 1$ and any $\Delta \ge 1$. Otherwise, $1 \le m \le n$. Then we apply Theorem 4.1 (for $a = v = 1$) to $p(x)$ and $q(x) = x^n p(1/x)$ and deduce that $\frac{1}{2(n+1-m)} < r_m < 2m$. It follows that $1/(2n) < r_m \le r_{m-1}$ and $r_m < 2n$. Therefore, $s = m - 1$ is a solution to Task r where $r = 1$ and $1 + \Delta = 2n$ (take into account (4.1) where $m = 1$, $s = 0$). The extension to arbitrary r is by means of scaling the variable x and to arbitrary Δ is by means of the iteration (3.3). We arrive at

Proposition 4.1 *Task r can be solved at the cost $\mathbf{O_A}(n(1+g)\log n)$ where g is defined by (4.2); the cost bound can be decreased to $\mathbf{O_A}(n)$ if $1+\Delta \geq 2n$.*

We could have solved Task s by recursively applying Proposition 4.1 in a binary search algorithm, but we will prefer a more direct approach outlined in [Sc82]. We will start with a high level description of this approach.

Pre-algorithm 4.1 *Given the coefficients $p_0, \dots, p_n$ of the polynomial $p(x)$ of (1.1) and an integer s, $1 \leq s \leq n$, choose two integers t and h that satisfy the following inequalities:*

$$t < n+1-s \leq t+h \leq n \tag{4.6}$$

and the following convexity property: *there exists no integer u in the range from 0 to n such that the point $(u, w(u))$ on the plane $\{(u,w)\}$ lies above the straight line passing through the two points $(t, w(t))$ and $(t+h, w(t+h))$, where $w(u)$ denotes* $\log|p_u|$, $\log 0 = -\infty$, *and we assume that no point $(u, -\infty)$ lie above any straight line on the plane $\{(u,w)\}$, thus discarding the points $(u, w(u))$ where $p_u = 0$. Compute and output $1/r$ where*

$$r = |p_{t+h}/p_t|^{1/h}. \tag{4.7}$$

For r of (4.7) the relations (4.6) and the above convexity property combined can be equivalently rewritten as follows:

$$|p_{t+g}/p_t| \leq r^g, \quad \text{for} \quad g = 1, \dots, n-t, \tag{4.8}$$

$$|p_{t+h-g}/p_{t+h}| \leq 1/r^g, \quad \text{for} \quad g = 1, \dots, t+h. \tag{4.9}$$

Proposition 4.2 *The output $1/r$ of Pre-algorithm 4.1 is a solution to Task s for $1+\Delta = 2n$.*

Proof. Due to inequalities (4.8) and (4.9), we may apply Theorem 4.1 to $p(y)$ with $a=1$, $v=1/r$, $i=g$, $m=n+1-t-h$ and to $y^n p(1/y)$ with $a=1$, $v=r$, $i=g$, $m=t+1$, respectively, and arrive at the desired bounds,

$$r_{n+1-t-h} < 2(n+1-t-h)/r, \;\; 1/r_{n-t} < 2(t+1)r.$$

Due to (4.6) and the bounds $1 \leq s \leq n$, it follows that

$$1/(2nr) < r_{n-t} \leq r_s \leq r_{n+1-t-h} < 2n/r. \tag{4.10}$$

□

Let us further specify the computations by Pre-algorithm 4.1 as follows:

Algorithm 4.2 *Compute the values* $\log|p_u|$, $u = 0, 1, \dots, n$, *with a prescribed precision; then compute the convex hull CH of the set $\{(u, \log|p_u|),\; u = 0, 1, \dots, n\}$ on the two-dimensional real plane, and then find the edge in the upper part of the* *boundary of CH whose orthogonal projection onto the u-axis is an interval including the point $n+1-s$. Choose t and $t+h$ to be the end points of this interval and, finally, compute r by using (4.7).*

Note that we compute the convex hull CH of the same set when we solve Task s for all s. Due to Graham's algorithm, the cost of this computation is $\mathbf{O_A}(n)$ (see [Gra72] or [PS85], pages 100–104).

By using the known fast algorithms (cf. [Al85]), we may rapidly approximate $\log|p_u|$, within the error bound $\delta > 0$ at the cost $\mathbf{O_A}(\log\log(1/\delta))$; for our purpose of solving Task s with $1+\Delta \geq 2n$, it suffices to choose δ satisfying $\log(1/\delta) = O(n)$, due to the well known perturbation theorems by Ostrowski and Schönhage (cf. [BP,a]), which bound the dependence of r on the coefficients of $p(x)$. It follows that the cost of computing sufficiently close approximation of $\log|p_u|$ for all u is $\mathbf{O_A}(n \log n)$.

The next proposition summarizes our estimates.

Proposition 4.3 *Task s for all s can be solved at the cost $\mathbf{O_A}(gn\log n)$, $g = g(\Delta)$ defined by (4.2).*

Next, we will simplify slightly the solution of Task s for a fixed s, for $1+\Delta \geq 2n$, and for a disc D, with the center 0, under the additional assumption that

$$i(p(x), D) = n+1-s, \;\; i.r.(D) \geq (1+\Delta)^2, \tag{4.11}$$

which will be satisfied in the applications to our Algorithms 9.1 and 11.1. In this case, we do not need to approximate the logarithms or compute the convex hull, and the overall cost of the solution is $\mathbf{O_A}(n)$ with a small overhead constant.

Proposition 4.4 *Let the relations (4.11) hold for a fixed integer s (such that $0 < s \leq n$), for $1+\Delta \geq 2n$, and for a disc D having the center 0 and an unknown radius. Then Task s for $1+\Delta \geq 2n$ can be solved at the cost $\mathbf{O_A}(n)$.*

Proof. Let t and h denote the two integers defined in Pre-algorithm 4.1 and thus satisfying (4.6) and (4.10). (4.11) implies that $r_{s-1}/r_s \geq (1+\Delta)^2 \geq 4n^2$. On the other hand, the first inequality of (4.6) implies that $r_s \geq r_{n-t}$. Consequently $r_{s-1}/r_{n-t} \geq (1+\Delta)^2 \geq 4n^2$. It follows that either $r_{s-1} > r_{n+1-t-h}$, and then $n+1-t-h > s-1$, $t+h \leq n+1-s$, or, otherwise, $r_{n+1-t-h}/r_{n-t} \geq r_{s-1}/r_{n-t} \geq (1+\Delta)^2 \geq 4n^2$. The latter inequalities imply that $r_{n+1-t-h} \geq 4n^2 r_{n-t}$, which contradicts (4.10). Consequently, $t+h \leq n+1-s$. Therefore, (4.6) implies that $t+h = n+1-s$. Since $t+h$ has been defined, it remains to choose an integer h, satisfying $n+1-s \geq h \geq 1$ and such that the convexity property of Pre-algorithm 4.1 holds or, equivalently, relations (4.7)-(4.9) hold. By (4.7) and (4.9), the values $(1/g)\log|\frac{p_{t+h}}{p_{t+h-g}}| = \log(|\frac{p_{t+h}}{p_{t+h-g}}|^{1/g})$ and, therefore, also $|\frac{p_{t+h}}{p_{t+h-g}}|^{1/g}$ reach their maximums where $g = h$, provided that g is the integer parameter ranging from 1 to $n+1-s$. If such a maximum h is defined uniquely, then (4.8) and the convexity property must also hold. Otherwise infinitesimal perturbation of the coefficients of $p(x)$ insures the uniqueness of the maximum h, and then again (4.8) and the desired convexity property follow. Since $t+h$ is fixed, the integer $g=h$ maximizing $|\frac{p_{t+h}}{p_{t+h-g}}|^{1/g}$ for $g=1,\dots,n+1-s$ and the value r of (4.7) can be computed at the cost $\mathbf{O_A}(n)$, and we arrive at Proposition 4.4. □

To extend Proposition 4.4 to any disc D and any positive Δ, we shift the variable, apply iteration (3.3), and recall Proposition 2.5 and Remark 3.1. The resulting algorithm will be referred to as **Algorithm 4.3**. The computational costs of its performance is estimated in the next corollary, which includes the bound $\mathbf{O_A}(n)$ of Proposition 4.4, the cost of the shift of the variable (if needed) and the cost of performing iteration (3.3) to lift the $i.r.(D)$ and $1+\Delta$ to or above $4n^2$ and $2n$, respectively.

Corollary 4.2 *Let a disc D contain exactly k zeros of $p(x)$, that is, $i(p(x),D)=k$, and let I be f-isolated for $f>1$. Then Task s for a disc D, a positive Δ, and $s=n+1-k$ can be solved at the cost $\mathbf{O_A}(n(1+(g+\delta)\log n))$, where*

$$g=\log(\frac{\log(2n)}{\log\min\{f,1+\Delta\}}),$$

$\delta=0$ if the disc D has center 0 and $\delta=1$ otherwise. In particular the cost bound is $\mathbf{O_A}(n\log n)$ if $1/\log\min\{f,1+\Delta\}=O(1/\log n)$ and $\mathbf{O_A}((n\log n)\log\log n)$ if $1/\log\min\{f,1+\Delta\}=O(1)$.

5 Computing the Number of Polynomial Zeros in a Disc.

Proposition 5.1 *Let $i.r.(D)=f$ for a given $f>1$ and for a disc $D=D(X,r)$. Then the index $i(p(x),D)$, showing the number of the zeros of $p(x)$ in D (cf. Definition 2.5), can be computed at the cost $\mathbf{O_A}(hn\log rn)$, where*

$$h=1+\left\lceil\log\lceil\frac{\log 9}{\log f}\rceil\right\rceil. \tag{5.1}$$

Proof. The well known *winding number algorithms* (see [He74], pp. 239–241, or [R87]) compute the index $i(p(x),D)$ of $p(x)$ in a disc D at the cost $\mathbf{O_A}(n\log n)$, provided that all the zeros of $p(x)$ lie far enough from the boundary of such a disc; namely, it suffices if $f=i.r.(D)\geq 9$ [R87]. If $f<9$, we first shift the variable x to transform the input disc into the disc $D(0,r)$, this changes neither the index nor the isolation ratio. Then we apply $h-1$ iteration steps (3.3) to increase the isolation ratio to or above 9 (cf. Remark 3.1) and then apply the winding number algorithms. The claimed cost bound follows from Proposition 2.5 and the estimate of Remark 3.1.
□

Remark 5.1 *Given a disc $D = D(0, \rho)$ such that i.r.$(D) = f \geq (1+v)^2$, 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 $\Delta \leq v$. The cost of the solution is $\mathbf{O_A}(n(1+g)\log n)$, where g is defined by (4.2), so that $g = O(h + \log\log n)$, [compare (4.2) with (5.1)]. This implies an alternative proof of Proposition 5.1, with the cost bound in its statement changed into $\mathbf{O_A}(gn\log n)$.*

6 Weyl's Exclusion Algorithm (with Turan's Proximity Test).

In this section, we will recall Weyl's exclusion algorithm ([He74], pp. 517–521) for approximating the zeros of $p(x)$. In the next sections, we will modify this algorithm to isolate the (clusters of the) zeros from each other.

Algorithm 6.1 (Weyl's quadtree algorithm) *(see Figure 2).*
Input: *positive integers k, G, $n \geq k$, and $N \geq 32$, the coefficients of the polynomial $p(x)$ of (1.1), a complex X and a positive R such that the square $S(X, R)$ contains exactly k (not necessarily distinct) zeros of $p(x)$ and is f-isolated, for $f > (1 + 5^{1/N}\sqrt{2})/2$ (cf. (3.1), (3.2), and Remark 6.3).*
Output: *a positive integer $H \leq 4n$ and complex values X_h, $h = 1, \ldots, H$, such that the union $\bigcup_{h=1}^{H} S(X_h, R/2^G)$ lies in $S(X, R)$ and contains the k zeros of $p(x)$.*
Stage 0 (initialization): *Call the square $S(X, R)$* **suspect in Stage 0** *or simply the* 0**-square**.
Computations:
Stage g, $g = 1, \ldots, G$. *At the beginning of Stage g, a set of squares* **suspect in Stage** $g - 1$ *(simply called* $(g-1)$**-squares** *and having sides of length $2R/2^{g-1}$) is available, supplied by Stage $g - 1$. At Stage g, divide each $(g-1)$-square into four subsquares, with the side length $2R/2^g$. Then test each subsquare for containing a zero of $p(x)$ as follows: within the factor $5^{1/N}$ [which is less than 1.052 for $N \geq 32$, by (3.2)], approximate the distance from the center of the subsquare to a closest zero of $p(x)$. Do this by applying Algorithm 3.1 at the center of the subsquare, which plays the role of the input value X of Algorithm 3.1, whose other input values, N, n, $p_0, \ldots, p_n$, are shared with Algorithm 6.1. Call the subsquare* **suspect in Stage** g, *or simply a* g**-square**, *unless the latter proximity test proves that the square contains no zeros of $p(x)$. Output the centers X_h of all the G-squares $S(X_h, R/2^G)$ and their overall number $H \leq 4k$.*

Correctness proof for this well-known algorithm is simple and can be found e.g. in [He74] or [R87]. The two key-observations are that the side-length of the suspect squares decreases by factor 2 in each stage of their recursive subdivision and that the union of all squares that are suspect in each recursive stage contains all the k zeros of $p(x)$ lying in $S(X, R)$, so that the centers of all the G-squares may serve as approximations to all the k zeros of $p(x)$ in $S(X, R)$ within the error bound $R/2^{G-0.5}$.

The asymptotic cost of performing Algorithm 6.1 is dominated by the cost of all applications of Algorithm 3.1 involved. The cost of the single application is $\mathbf{O_A}(n \log n)$, due to (3.4) for a fixed constant N. The total number of applications is at most $4kG$, due to Remark 6.1 below, and this yields the overall cost estimate $O(knG \log n)$, which turns into $O(knh \log n)$ if we require to bound by $\sqrt{2}R/2^h$ the errors of the approximations to the zeros of $p(x)$ by the centers of the output G-squares.

In the next sections, we will yield the cost bound $O(\log h)$ in terms of h, which is a substantial improvement for large h.

Remark 6.1 *Each zero of $p(x)$ may make at most four squares suspect in Stage g, for any g [note that our proximity test computes $r_n(X)$ within 6% error, due to relations (3.1), (3.2) and $N \geq 32$].*

Remark 6.2 *To apply Algorithm 6.1 to approximate all the n zeros of $p(x)$, we need an initial square $S(X, R)$ containing all these zeros. We may compute such a square for $X = 0$ either at the cost $\mathbf{O_A}(n)$, by applying the bound (4.3), or at the cost $\mathbf{O_A}(n \log n)$, by applying part (b) of Corollary 4.1.*

Remark 6.3 *The input lower bound on f insures that the zeros of $p(x)$ lying outside the input suspect square $S(X, R)$ do not influence the outcome of the proximity tests, at the first stage and consequently at all stages of Algorithm 6.1. If the bound on f were smaller, the influence of the outside zeros lying near $S(X, R)$ could have caused processing some extra suspect squares.*

Remark 6.4 *Instead of Algorithm 3.1, one may apply an alternative proximity test, based on (4.8) and iteration (3.3). Furthermore, in many cases, a simpler test may detect that a given square $S(X, R)$ contains a zero of $p(x)$ and, therefore, is suspect. In particular if*

$$R \geq r^{(1)} = n|p(X)/p'(X)| \text{ or, more generally, } R \geq r^{(l)} = n \min_{0<k\leq l} (k! p(X)/p^{(k)}(X)) \text{ for a fixed } l \leq n ,$$

then the square $S(X, R)$ and even the disc $D(X, R)$ contain a zero of $p(x)$ (cf. (4.4)). If $R < r^{(l)}$, then by this simple one-sided test, we cannot decide if the square $S(X, R)$ is suspect or should be discarded. The reader is referred to Remark 3.2 and to [He74] and [BP,a] on some other one-sided tests. The proximity tests are also important parts of some other algorithms for polynomial zeros (cf. [Ha87] on their major role in the Jenkins-Traub algorithm) and, therefore, are of some independent interest.

Remark 6.5 (G. Dos Reis). *In the four proximity tests from the centers $X + (R/2)(\pm 1 \pm \sqrt{-1})$ of the four subsquares of a g-square $S(X, R)$, we first perform two shifts of the variable x, by $\Delta_+ = (1 + \sqrt{-1})R/2$ and $\Delta_- = (1 - \sqrt{-1})R/2$, and then easily extend them to the two other shifts, by $-\Delta_+$ and $-\Delta_-$.*

7 Combining Approximation and Isolation of Polynomial Zeros.

Our next goal is the description of our main algorithm for polynomial zeros, which starts with an input disc $D = D(X_0, R)$ containing exactly k zeros of $p(x)$ and consists in recursive solution of the following problem:

Problem 7.1.

Input: two real constants $f \geq 10$ and $\varepsilon = 2^{-b} > 0$, complex coefficients of a polynomial $p(x)$ of (1.1), a natural k, $2 \leq k \leq n$, and an f-isolated disc $D = D(X_0, R)$ that contains exactly k zeros of $p(x)$.

Output: an integer k_0, $0 \leq k_0 \leq k$, f-isolated ε_h-covers $D_{0,h}$ of k_0 zeros $z_{j(h)}$ of $p(x)$ in S for $\varepsilon_h \leq \varepsilon$, $h = 1, \ldots, k_0$, an integer $l \geq 0$ ($l \geq 2$ if $k_0 = 0$), l discs $D_1, \ldots, D_l$ that are disjoint, f-isolated, and such that $D_j \subseteq D$, $i_j = i(p(x), D_j) \geq 1$, $j = 1, \ldots, l$ (cf. Definition 2.5), $\sum_{j=0}^{l} i_j = k$, and the indices of $p(x)$ in all discs $D_{0,h}$ and D_j, for $h = 1, 2, \ldots, k_0$; $j = 1, \ldots, l$.

The following algorithm computes the desired approximations to the k zeros in a fixed disc by recursively solving Problem 7.1.

Algorithm 7.1

Input: *as for Problem 7.1.*

Output: *f-isolated ε_h-covers of all the k zeros z_h of $p(x)$ in $D(X_0, R)$, where $\varepsilon_h \leq \varepsilon$, $h = 1, \ldots, k$, and the indices of $p(x)$ in all these ε_h-covers.*

Computations: *Initially solve Problem 7.1 for the input disc D. Then, for each output disc D_j, $j = 1, \ldots, l$, solve Problems 7.1 with D_j made its input disc. Recursively repeat this process until f-isolated ε_h-covers of all the zeros z_h of $p(x)$ in D have been computed together with the respective indices of $p(x)$, where $\varepsilon_h \leq \varepsilon$.*

Since $k_j < k$, for $j = 1, \ldots, l$, we will arrive at the desired ε_h-covers of all the k zeros of $p(x)$ lying in D in at most $k - 1$ recursive steps of solving Problem 7.1. It remains to specify the solution of Problem 7.1 and to estimate its complexity.

We will partition the solution of Problem 7.1 into two stages.

Problem 7.1a.

Input: *as for Problem 7.1 and, in addition, a value $e > 1$.*

Output: *either a common f-isolated ε_h-cover of all the k zeros z_h of $p(x)$ in the disc D for some $\varepsilon_h \leq \varepsilon/3$ or else an $(f/\sqrt{2})$-isolated square $S^* = S(X^*, R^*)$ that lies in the smallest square $S = S(X_0, R)$ superscribing the input disc D, is equivalent to D, and satisfies the inequality*

$$r.r.(S^*) \geq 1/e. \tag{7.1}$$

Problem 7.1 b.

Input: *f, ε, $p(x)$ and k as for Problem 7.1a and, in addition, the $(f/\sqrt{2})$-isolated output square S^* of Problem 7.1a, satisfying (7.1) for the fixed $e > 1$.*

Output: *as for Problem 7.1.*

We will consider Problem 7.1b in the next section and Problem 7.1a in sections 9-11.

Remark 7.1 *Our Algorithm 7.1 can be immediately extended to compute a solution to Problem 7.1 under the additional requirement that* $\epsilon_h \leq \epsilon_h^*$ *for a fixed set of positive* ϵ_h^*, $h = 1, \dots, k_0$. *In particular each* ϵ_h^* *can be defined as a fixed function in* ϵ *and in the index* $i(p(x), D_{0,h})$, *to adapt the output precision to the condition of the zeros. Also the precision of computing may (and should in practical implementations) be adapted respectively, to avoid the expense of performing excessively accurate intermediate computations. In particular performing the* g*-th step of Weyl's quadtree process for solving Problem 7.1b one should not drive to error bounds much below the level of* $2R^*/2^g$ *of the side length of the* g*-squares, whereas the computational precision used in the process of solving Problem 7.1a (by Newton's iteration) should be tuned to the level of the upper bound* $r^+_{n+1-k}(X)$ *on the* $(n+1-k)$*-th root radius at* X, *which we will compute in Stage 4 of Algorithm 9.1 of section 9.*

Remark 7.2 *If a real* $f \geq 10$ *and the input disc* $D = D(X, R)$ *for Problem 7.1 are given, but i.r.*(D) *is unknown, we may apply Algorithm 4.2 to define a maximal zero-free annulus* A *with the center* X *and the boundary circles of radii* R^- *and* R^+, $R^- \leq R^+$, *satisfying*

$$R^+/R^- \geq f, \qquad R^+ \geq R \ ,$$

for the smallest R^-. *(By a maximal annulus* A_s, *we mean one not contained in any larger annulus of the same class.) Then* $D^- = D(X, R^-)$, *the internal disc of* A, *would serve as an* f*-isolated input disc for Problem 7.1. If* $R^- \leq R$ *this disc is equivalent to* D*; otherwise*

$$k = i(p(x), D) < k^+ = i(p(x), D^-) \quad \text{(cf. Definition 2.5)} \ ,$$

and the complexity of approximating the k *zeros of* $p(x)$ *lying in* D *will grow by factor* k^+/k. *If we wish to avoid such a growth, here is the sketch of our tentative recipe. Linearly transform the variable to turn the disc* D *into the unit disc* $D(0,1)$ *and apply* l *steps (3.3), for the minimum* $l = l(f, D)$, *to make the disc* $D(0,1)$ f*-isolated. Approximate the zeros* $(a(z_j - X))^{2^l}$ *of the resulting polynomial. Recursively recover the values* $(a(z_j - X))^{2^{l-i}}$ *for* $i = 1, 2, \dots, l$ *and then obtain* $z_j - X$ *and* z_j. *In each descending step, from* $(a(z_j - X))^{2^{l-i+1}}$ *to* $(a(z_j - X))^{2^{l-i}}$, $j = 1, \dots, l$, *avoid ambiguity by applying Algorithms 3.1 or 4.2 at each of the* $2k$ *candidate points* $\pm a(a_j - X)^{2^{l-i}}$, *to discard the* k *extraneous candidate points.*

8 Isolation of the Zeros.

In this section we will specify **Algorithm 8.1**, which solves Problem 7.1b, and will estimate the computational cost of its applications within Agorithm 7.1. We will proceed as in Weyl's Algorithm 6.1 applied to the input square $S^* = S(X^*, R^*)$, except that we will add a block that verifies if for some g some set of g-squares forms a connected component which is sufficiently well isolated from all the zeros of $p(x)$ lying outside this component; if so, we will stop partitioning these g-squares, keep them invariant and output at the end of the computations.

Specifically, let m denote the number of all distinct zeros of $p(x)$ in S^*. Furthermore, for every g, let $w(g)$ denote the number of all g-squares processed by the algorithm, connect each pair of *adjacent g-squares*, that is, of g-squares sharing a common vertex, and partition their union into connected components, $C_1^{(g)}, \dots, C_{v(g)}^{(g)}$, where each component contains at least one zero of $p(x)$, so that $v(1) = 1$, $S^* = C_1^{(1)}$, $1 \le v(g) \le m$, $g = 1, 2, \dots$.

Let the $(1 + g_0)$-th stage be the **first separation stage** of Algorithm 8.1, that is, let

$$v(g) = 1 \text{ for } g = 1, 2, \dots, g_0;\ v(g_0 + 1) > 1. \tag{8.1}$$

For every $g > g_0$ and for each u, $u = 1, \dots, v(g)$, apply Algorithm 2.1 at Stage g to the set of all the vertices of all the g-squares of the component $C_u^{(g)}$, and arrive at the smallest rectangle covering $C_u^{(g)}$, as well as at the smallest disc $D_u^{(g)} = D(X_u^{(g)}, \tilde{R}_u^{(g)})$ and a smallest square $S_u^{(g)} = S(X_u^{(g)}, R_u^{(g)})$ covering this rectangle, where $R_u^{(g)} \le \tilde{R}_u^{(g)} \le R_u^{(g)}\sqrt{2}$ (compare Remark 2.1 and Proposition 2.1). For $u = 1, \dots, v(g)$, check if

$$d_u^{(g)} = \min_{v \ne u}(\max\{|Re(X_u^{(g)} - X_v^{(g)})|,\ |Im(X_u^{(g)} - X_v^{(g)})|\} - R_v^{(g)}) \ge f\tilde{R}_u^{(g)}, \tag{8.2}$$

that is, if the disc $D_u^{(g)}$ is f-isolated. If so, stop the recursive process of dividing the g-squares lying in this disc into 4-tuples of subsquares, call the component $C_u^{(g)}$ **invariant**, and add the disc $D_u^{(g)}$ to the list of the output discs of Problem 7.1b, either as $D_{0,h}$ (for an appropriate h) if

$$\tilde{R}_u^{(g)} \le \varepsilon \tag{8.3}$$

or as D_j (for an appropriate j) otherwise. Stop the computation by Algorithm 8.1 when all the components become invariant, compute and output their superscribing discs by using Algorithm 2.1, and apply the algorithm of section 5 to compute and to output the indices of $p(x)$ in these discs. This completes our description of the algorithm.

The remainder of the section will be devoted to the computational complexity analysis. We will start with some definitions and auxiliary results.

Hereafter $w(C_u^{(g)})$ denotes the number of the g-squares lying in the component $C_u^{(g)}$, so that

$$w(g) = \sum_{u=1}^{v(g)} w(C_u^{(g)}),\ g = 1, 2, \dots, G.$$

Also, for any fixed g, let $w_{in}(g)$ denote the number of g-squares each having at least one zero of $p(x)$ in its closure.

Lemma 8.1 $9w_{in}(g) \ge w(g)$ for all g.

Proof. By definition, if a g-square contains no zeros of $p(x)$, then at least one of its g-neighbors (that is, a g-square sharing one or two vertices with it) must contain a zero of $p(x)$. On the other hand, every g-square has at most eight g-neighbors, and Lemma 8.1 follows. $\square$

We will next define a tree T with $G+1-g_0$ levels of nodes, where g_0 is defined by (8.1). For $g = g_0, g_0+1, \dots,$ that is, starting with the first separation stage of the algorithm, the $v(g)$ components $C_1^{(g)}, \dots, C_{v(g)}^{(g)}$ are identified with the $v(g)$ nodes forming the $(g-g_0)$-th level of the tree. The component $C_1^{(g_0)}$ is identified with the root of the tree, which lies at the 0-th level. The leaves are identified with the invariant output components $C_u^{(g)}$, such that (8.2) holds for the values $X_u^{(g)}$, $R_u^{(g)}$, $\tilde{R}_u^{(g)}$. The edges of the tree go from each component $C_u^{(g)}$ to all the components [of the $(g+1-g_0)$-th level] formed by the $(g+1)$-squares that lie in $C_u^{(g)}$. (The tree does not reflect the connections between the adjacent g-squares in the same component $C_u^{(g)}$.)

Now let w denote $\sum w(C_u^{(g)})$, where the sum is over all the leaves $C_u^{(g)}$. Since $i.r.(D_u^{(g)}) \geq f \geq 10$ for every leaf-component $C_u^{(g)}$, all the g-squares lying in such a component are affected only by the zeros of $p(x)$ lying in this component. Therefore, by the argument used in Remark 6.1 we deduce that

$$w \leq 4m. \tag{8.4}$$

Next, we recall that the squares $S_u^{(g)}$ superscribing the components $C_u^{(g)}$ for $g \geq g_0$ are f-isolated only if these components are leaves, and then we deduce the following result.

Lemma 8.2 *Every component $C_u^{(g)}$, $g > g_0$ for g_0 of (8.1), has an ancestor-fork (that is, an ancestor with at least two children) at level g_1, $g_1 > g - g_0 - \log\ (f w(C_u^{(g)})) + \delta$, where f is the input value of Problem 7.1 and Algorithm 7.1, $\delta = 0$ if $C_u^{(g)}$ is a leaf (that is, an invariant output component), and $\delta = 1$ otherwise.*

Proof. Let $C_u^{(g)}$ have its closest ancestor-fork $C_v^{(g_1)}$ at some level g_1. Then the distance from $X_u^{(g)}$ to the closest zero of $p(x)$ lying outside $C_u^{(g)}$ is at least $R^*/2^g + 2R^*/2^{g_0+g_1}$, because such a zero of $p(x)$ is separated from $X_u^{(g)}$ by both a g-square lying in $C_u^{(g)}$ and, therefore, having side length $2R_g = 2R^*/2^g$ and some square $S(X, R^*/2^{g_0+g_1})$ discarded in Stage g_0+g_1 and, therefore, having side length $2R_{g_0+g_1} = 2R^*/2^{g_0+g_1}$. On the other hand, observe that $R_u^{(g)} \leq R_g w(C_u^{(g)})$, $R^* = 2^g R_g$. Consequently, $i.r.(S_u^{(g)}) \geq \frac{1+2^{g-g_0-g_1+1}}{w(C_u^{(g)})}$. Therefore, $\log(i.r.(S_u^{(g)}) w(C_u^{(g)})) > g - g_0 - g_1 + 1$. On the other hand, if $C_u^{(g)}$ is a leaf of T, then $i.r.(S_v^{(g-1)}) \leq f$ for the parent $C_v^{(g-1)}$ of $C_u^{(g)}$, and otherwise $i.r.(S_u^{(g)}) \leq f$. Combining the above relations completes the proof of the lemma. $\square$

Lemma 8.3 *Every component $C_u^{(g)}$, for $u = 1, \dots, v(g)$, $g = 3, \dots, G$, is covered by*

$$s(g,u) \leq 2 + \lfloor (w(C_u^{(g)}))/2 \rfloor$$

squares that are suspect in Stage $g-2$.

Proof. Let the component $C_u^{(g)}$ consist of g-squares $S_{1,u}^{(g)},\ldots,S_{L,u}^{(g)}$ (with side length $2R^*/2^g$), $L=w(C_u^{(g)})$, and let it be covered by $(g-2)$-squares $S_{1,u}^{(g-2)},\ldots,S_{\ell,u}^{(g-2)}$ (with side length $8R^*/2^g$), where each $(g-2)$-square $S_{i,u}^{(g-2)}$ contains at least one g-square $S_{j(i),u}^{(g)}$, so that $\ell = s(g,u)\le L$. Furthermore, $\ell=L$ is possible only for $L\le 4$. Indeed, otherwise the connected component $C_u^{(g)}=\bigcup_i S_{i,u}^{(g)}$ must contain some pair of g-squares that lie in a pair of $(g-2)$-squares having no common vertices. Any chain of g-squares connecting such a pair of g-squares must contain a pair of g-squares adjacent to each other and lying in the same $(g-2)$-square, which implies that $L>l$. To relate L and ℓ more precisely, let us assume that the squares $S_{j,u}^{(g)}$ have their upper and right edges deleted, let us represent each g-square by its southwestern vertex, and let a line interval connect each pair of such vertices representing a pair of adjacent squares. Then the edges of the resulting graph $\hat{G}$ should cross or at least touch all the squares $S_{j,u}^{(g-2)}$, $j=1,\ldots,\ell$. Four edges of $\hat{G}$ are sufficient to cross or to touch four squares $S_{j,u}^{(g-2)}$ that form a single $(g-3)$-square but at least 4 edges are needed to meet any other $(g-2)$-square or any pair of such squares, respectively (see Figures 3 and 4). This gives us the desired estimates, $l\le 4+\lfloor (L-4)/2\rfloor = 2+\lfloor L/2\rfloor$. $\square$

Corollary 8.1 *$s(g,u)\le 5w(C_u^{(g)})/7$ if $w(C_u^{(g)})>6$.*

Our next goal is to prove the following result.

Proposition 8.1

$$W_1=\sum_{g=g_0+1}^{G} w(g) = O((1+\log f)m). \tag{8.5}$$

(Later on, we will deduce a similar estimate for $W_0=\sum_{g=1}^{g_0} w(g)$.)

Proof. Due to Lemma 8.1, it suffices to prove Proposition 8.1 under the assumption that for any g the closure of any g-square contains a zero of $p(x)$. In particular this assumption implies that each node $C_u^{(g)}$ that is not a leaf represents a component containing not more suspect squares than all its children do together, that is,

$$w(C_u^{(g)})\le \sum_{r(u)} w(C_{r(u)}^{(g+1)}), \tag{8.6}$$

where the summation is over all the children $C_{r(u)}^{(g+1)}$ of the node $C_u^{(g)}$.

Write $w_6=\sum_{u,g} w(C_u^{(g)})$, $w_6^*=\sum_{u,g}^* w(C_u^{(g)})$, where $\sum$ and $\sum^*$ denote the sums in all the pairs g and u such that $g>g_0$, $1\le u\le v(g)$, $w(C_u^{(g)})\le 6$, provided that $\sum^*$ is the sum where $C_u^{(g)}$ are forks and leaves, that is, includes no pairs (g,u) such that the node $C_u^{(g)}$ of the tree T has exactly one child.

Ignoring the components $C_u^{(g)}$ with $w(C_u^g)\le 6$, we obtain that, like the leaves themselves, all their parents together contain at most w suspect squares (by(8.6)), all the grandparents of all the leaves together contain at most $(5/7)w$ (by Corollary 8.1), and so on, so that

$$W_1 - w_6 < 2w \sum_{i=0}^{\infty} (5/7)^i = 7w \leq 28m \tag{8.7}$$

(cf. (8.4)), and it remains to estimate w_6 in order to prove (8.5).

Lemma 8.2 implies that in the tree T every node $C_u^{(g)}$ such that $w(C_u^{(g)}) \leq 6$ has less than $\log(6f)$ its successive predecessors with a single child. Due to the bound (8.6), each of these predecessors contributes at most $w(C_u^{(g)})$ to the sum w_5. It follows that

$$w_5 < (1 + \log(6f))w_6^* = w_6^* \log(12f). \tag{8.8}$$

Let us estimate w_6^* to complete the proof of Proposition 8.1. We first observe that the tree T has at most m leaves; therefore, it has at most $m - 1$ forks (the nodes with more than one child). The set of all the forks and leaves has a cardinality of at most $2m - 1$ and has a subset of nodes $C_u^{(g)}$ such that $w(C_u^{(g)}) \leq 6$. This subset is formed by exactly w_6^* squares, each of which is suspect in at least one of Stages $g_0 + 1, \ldots, G$. Consequently,

$$w_6^* \leq 6(2m - 1) = 12m - 6.$$

Combining this bound on w_6^* with (8.7) and (8.8) implies bound (8.5) of Proposition 8.1. □

Proposition 8.2

$$W_0 = \sum_{g=1}^{g_0} w(g) = O(m + \log e).$$

Proof. Let us first estimate the overall number $\overline{w}_6$ of the g-squares in the components $C_1^{(g)}$ where

$$w(C_1^{(g)}) \leq 6, \quad g \leq g_0. \tag{8.9}$$

We combine (7.1) and (8.9) to deduce that

$$\overline{w}_6 \leq 6(\log(6e) + 1). \tag{8.10}$$

Indeed, as g increases, $r.r.(S_1^{(g)})$ never decreases, whereas it grows by factor at least $2^h/6$ in h successive steps. This follows because $S_1^{(g)}$ and $S_1^{(g+h)}$ are equivalent to each other, $R_1^{(g)} \geq 2^h R_1^{(g+h)}/w(g+h)$ and $w(g+h) \leq 6$ by (8.9). Now the bound $2^h/6 \leq e$, or equivalently, $h \leq \log(5e)$, follows since $r.r.(S_1^{(g)})$ never exceeds 1 and since we initially have (7.1). The latter inequality and (8.9) combined imply (8.10).

Now, similarly to (8.4), we obtain that

$$w(g) \leq 4m, \; g = 1, 2, \ldots, G. \tag{8.11}$$

Combine the latter bound, (8.10), Lemma 8.3, and Corollary 8.1 and deduce Proposition 8.2. □

We have deduced Propositions 8.1 and 8.2 for a single application of Algorithm 8.1, which we actually invoke $O(k)$ times in the process of performing Algorithm 7.1. Let us extend these propositions to bound the sum $W = \sum_{g=1}^{G} w(g)$, considered over all applications of Algorithm 8.1. (In fact, there are at most $k-1$ such applications.) In every application every output disc $D_u^{(g)}$ satisfying the relation (8.2) but not (8.3) (that is, f-isolated but not embedded into an equivalent and f-isolated disc of a radius at most ε) serves as the input disc in the subsequent application of Algorithm 9.1 of the next section. This algorithm solves Problem 7.1a and, in turn, outputs either a final $(\epsilon/3)$-cover of all the zeros of $p(x)$ lying in $D_u^{(u)}$ or an input square for Problem 7.1b and Algorithm 8.1. In the latter case we substitute the new tree generated in the latter application of Algorithm 8.1 for the respective leaf of the tree T defined in the preceding application of Algorithm 8.1. In this way, we construct a single tree associated with all the $O(k)$ applications of Algorithm 8.1 within Algorithm 7.1 and having at most k leaves. (8.11) and Proposition 8.1 (with m replaced by k) are immediately extended to the entire computational process represented by the latter tree, whereas the bound (8.10) is applied $k-1$ times, so that the bound $O(k+k\log e)$ replaces $O(m+\log e)$ of Proposition 8.2. Summarizing, we arrive at the following result, which bounds the overall number W of the g-squares in all components $C_u^{(g)}$ for u and g.

Proposition 8.3 *Let (7.1) hold for a fixed $e > 1$ and let W denote the overall number of the g-squares processed in all applications of Weyl's Algorithm 6.1 within Algorithm 7.1. Then*

$$W = O(k + k\log(ef)).$$

We now recall that each proximity test in Algorithm 6.1 (one for each suspect square) is performed by means of Algorithm 3.1 at the cost bounded by $\mathbf{O_A}(n\log n)$. Therefore, Proposition 8.3 implies the following estimate.

Proposition 8.4 *The computational cost of all applications of Algorithm 8.1 within Algorithm 7.1 is bounded by $\mathbf{O_A}((1+\log(ef))kn\log n)$. In particular, for $\log(ef) = O(\log n)$ the latter bound turns into $\mathbf{O_A}((\log n)^2 kn)$, whereas for $e = O(1)$ and $f = O(1)$ it turns into $\mathbf{O_A}(kn\log n)$.*

9 Contraction of a Complex Square.

Our next Algorithm 9.1 solves Problem 7.1a, but we will start with three remarks on some cases where the solution can be obtained immediately and on the possibility to start with an input square rather than a disc.

Remark 9.1 *The input disc for our Algorithm 9.1 for Problem 7.1a is assumed to be either given from outside (initially) or supplied by the solution of Problem 7.1b* *produced by Algorithm 8.1. Application of Algorithm 9.1 to the invariant output components $C_u^{(g)}$ of Algorithm 8.1, however, is motivated only if the side length $2R_g^* = R^*/2^{g-1}$ of a g-square is at least as large as the half-length $R_u^{(g)}$ of a longer side of the smallest rectangle superscribing $C_u^{(g)}$, that is, if*

$$2R_g^* \geq R_u^{(g)}. \tag{9.1}$$

Indeed, otherwise the component $C_u^{(g)}$ contains a pair of g-squares $S(X_1, R_g^*)$ and $S(X_2, R_g^*)$ with $|X_1 - X_2| \geq 4R_g^*$. Now recall the definitions of $d^*(U)$ and $\rho(S)$ (cf. Definitions 2.1 and 2.3) and, as before, let $S_u^{(g)} = S(X_u^{(g)}, R_u^{(g)})$ denote a smallest square superscribing $C_u^{(g)}$. Then, by Proposition 2.1, by the definition of $S_u^{(g)}$ and g-squares, and by (3.1), we have

$$d^*(S_u^{(g)}) = d^*(C_u^{(g)}) \geq |X_1 - X_2| - 5^{1/N} 2\sqrt{2} R_g^* \geq (4 - 5^{1/N} 2\sqrt{2}) R_g^*,$$

$$d^*(S_u^{(g)}) \geq 2\rho(S_u^{(g)}) - (2R_g^*)(1 + 5^{1/N}\sqrt{2}\ ).$$

It follows that

$$2\rho(S_u^{(g)})/d^*(S_u^{(g)}) \leq 1 + (1 + 5^{1/N}\sqrt{2})/(2 - 5^{1/N}\sqrt{2}) = \frac{3}{2 - 5^{1/N}\sqrt{2}}.$$

Consequently, r.r.$(S_u^{(g)}) = \frac{d^*(S_u^{(g)})}{2\rho(S_u^{(g)})} \geq \frac{2 - 5^{1/N}\sqrt{2}}{3} > 0.17$ for $N \geq 32$ (cf. Proposition 2.2 and (3.2)), which means that the requirements to the solution of Problem 7.1a are satisfied for the square $S^* = S_u^{(g)}$ and for $e = 1/0.17$. Thus, we may simply continue the recursive process of partitioning all the g-squares that lie in the component $C_u^{(g)}$, instead of application of Algorithm 9.1. Furhtermore, since a few steps of Algorithm 8.1 are usually less costly than the application of Algorithm 9.1, we may extend Algorithm 8.1 by applying a few such additional steps g and shift to Algorithm 9.1 only if (9.1) holds in all these steps.

Remark 9.2 *If $k = n$, then we do not have to apply Algorithm 9.1, since we may first compute and output $X = -p_{n-1}/(np_n) = \sum_{j=1}^{n} z_j/n$ and then apply a simple modification of Algorithm 3.1 to compute and to output a desired approximation R^* to $r_{n+1-k}(X) = r_1(X)$ from above.*

Remark 9.3 *Suppose that initially one is given an f-isolated square $S = S(X_0, R)$ (rather than disc D) containing k zeros of $p(x)$. Then one may apply two stages of Algorithm 8.1. If the union of all 2-squares is covered by a single square $S(W, R/2)$ with side length R, then $|W - X_0| = R/\sqrt{2}$, and the smallest disc covering the latter square is equivalent to S, has a radius at most $R/\sqrt{2}$, and, therefore, is f-isolated since $|W - z_j| \geq |X_0 - z_j| - R/\sqrt{2} \geq (f - 1/\sqrt{2})R$ for all the zeros z_j of $p(x)$ lying outside the input square S and since $fR/\sqrt{2} \leq (f - 1/\sqrt{2})R$ for $f \geq 10$. In this case we may apply Algorithm 9.1 with such an input disc. Otherwise, based* *on the argument of Remark 9.1, we may define a square S^* equivalent to S with r.r.$(S^*) > 0.17$. Then S^* satisfies the requirements to the input square of Algorithm 8.1 for any $e > 5.9$, and we may apply this algorithm to S^* or, even simpler, we may just continue its recursive application to the available 2-squares.*

We will next describe the algorithm, estimate the computational cost of its performance, and show its extenstion. Its correctness proof in the next section will exploit the relations (9.2)-(9.4) below imposed on the input. The algorithm consists of Newton's iteration (see (9.5) below) and application of the techniques of sections 2−4 in order to check after each iteration step if the requirements to the solution of Problem 7.1a can already be satisfied.

Algorithm 9.1

Input *and* **Output** *as for Problem 7.1a, plus two input values, $\Delta > 0$ and an integer $N \geq 32$, where e, f, Δ, N satisfy the following relations:*

$$e \geq 4\sqrt{2}(1+\Delta)^2/(2-5^{1/N}) \ , \tag{9.2}$$

$$(f-2)k/(n-k) > 6, \ f \geq max\{10, (1+\Delta)^2)\}, \tag{9.3}$$

$$1/\theta = (1-2/e)((f-2)k/(n-k)-3) \geq 18(1+4(1+\Delta)^2). \tag{9.4}$$

Initialization: *Let σ denote a set of complex numbers available from the previous application of Algorithm 9.1 and chosen to be empty at the first application of this algorithm. For the f-isolated input disc $D = D(X_0, R)$, compute the complex value $Y = X_0 + 2R$.*
Computations:

1. *Compute the value $p'(Y)$. If $p'(Y) \neq 0$, set $t = Y$. Otherwise add the complex point Y to the set σ and choose a point t such that $|t - X_0| = 2R$ and $p'(t) \neq 0$. (It suffices to test at most $n - |\sigma|$ candidate points $t \notin \sigma$ to find one for which $p'(t) \neq 0$, where $|\sigma|$ denotes the cardinality of the set σ. Each candidate point t for which $p'(t) = 0$ is added to the set σ.)*

2. *Compute the value*
$$u = t - \frac{kp(t)}{p'(t)}. \tag{9.5}$$

3. *Write $X = u$. If $p(u) = 0$, write $r = r(X) = r^* = r^*(X) = 0$. Otherwise, fix a natural $N \geq 32$ and apply Algorithm 3.1 (Turan's proximity test), which outputs $r = r(X)$ and $r^* = r^*(X) = 5^{1/N}r$. (Note that $r^*(X) \geq r_n(X) \geq r(X)$.)*

4. *Apply Algorithm 4.3 to compute the values r, $r_s^-(X) = r/(1+\Delta)$, $r_s^+(X) = \min\{r(1+\Delta), l(u,D)+2R\}$, where $l(u,D)$ is the distance from $u = X$ to the input disc D, $l(u,D) = 0$ if $u = X \in D$, $r_s^-(X) \leq r_s(X) \leq r_s^+(X)$,*

$s = n - k + 1$, Δ is an input value of the algorithm, and $r_s(X)$ is the s-th root radius of $p(x)$ at X (cf. Definition 2.6).

(a) If $r_s^+(X) \le \epsilon/3$, output the disc $D_{\epsilon/3} = D(X, r_s^+(X))$ as a common f-isolated $r_s^+(X)$-cover of all the k zeros of $p(x)$ lying in the input disc D and stop.

(b) Otherwise if

$$r_s^-(X) - r^*(X) \ge 2\sqrt{2} r_s^+(X)/e \ , \tag{9.6}$$

define the intersection $I = S^+ \cap S$ of the square $S^+ = S(X, r_s^+(X))$ (which contains all the k zeros of $p(x)$ lying in the disc $D = D(X_0, R)$) with the square $S = S(X_0, R)$ (containing D). I is a rectangle (lying in S and equivalent to D). Output a smallest square S^* containing I and lying in S and go to Algorithm 8.1. (We will show in the next section that in this case S^* is an $(f/\sqrt{2})$-isolated square equivalent to D and satisfying (7.1).)

(c) Otherwise write

$$t = X + 2r_s^+(X) \tag{9.7}$$

and go to Stage 2.

Let us estimate the computational cost of recursive application of Algorithm 9.1, in combination with Algorithm 8.1, where all the $Q = O(n)$ input discs in all Q calls for Algorithm 9.1 are arranged in the form of a tree with Q nodes. In each recursive step, one may successively or concurrently apply Algorithm 9.1 to all the discs (nodes) of the current level of the tree starting with the root, at the first step.

The computational cost of performing Algorithm 9.1 is estimated as follows:

Stages 1, 2: $\mathbf{O_A}(n)$ [dominated by the cost of computing the values of $p(x)$ and $p'(x)$];

Stage 3: $\mathbf{O_A}(n \log n)$ (the cost of the applications of Algorithm 3.1);

Stage 4: $\mathbf{O_A}(n \log n)$, for shifting to the polynomial $q(y) = p(y + X)$ and for each single step of iteration (3.3) (if needed); $\mathbf{O_A}(n)$ for application of other steps of Algorithm 4.3.

According to Remark 10.1 of the next section, we need at most two steps (3.3) in each application of Algorithm 4.3 at Stage 4 of Algorithm 9.1. The cost of performing each iteration loop of Algorithm 9.1, consisting of its Stages 2-4, is thus $\mathbf{O_A}(n \log n)$. Together with the bound (10.12) of the next section on the number of loops, this gives us the estimate

$$\mathbf{O_A}((n \log n) \log\log((1 + \Delta)(R/\epsilon)))$$

for the overall computational cost of performing Algorithm 9.1. Substitute $b = \log(R/\epsilon)$, $\log(1 + \Delta) = O(\log n)$ (we will always choose Δ satisfying the latter relation) and obtain the overall bound $\mathbf{O_A}((n \log n) \log(b \log n))$.

In terms of b, the cost bound is cumulative since the size of the output square of Algorithm 9.1 bounds from above the size of its input discs in all subsequent calls for this algorithm (if they are needed). Thus the overall cost of performing Algorithm 9.1 within Algorithm 7.1 is bounded from above by $\mathbf{O_A}((n\log n)\log(b\log n))$ times the maximum number of its concurrent applications, which is at most k. The resulting estimate $\mathbf{O_A}((kn\log n)\log(b\log n))$ and the estimate of Proposition 8.4 together give us the overall cost bound for our solution of Problem 7.1. To specify this bound, it remains to specify the parameters Δ, e and f, so as to satisfy (9.2)-(9.4). (9.2)-(9.4) holds, if we choose, say, $e = 16n^2\sqrt{2}/(2-5^{1/N})$, $f = max\{4n^2, 2+(n-k)(3+18(1+16n^2)/(1-2/e))/k\}$, $1+\Delta = 2n$. Then Proposition 8.4 gives us the cost bound $\mathbf{O_A}((\log n)^2 n)$. Adding the cost bound for performing Algorithm 9.1, we arrive at the claimed estimate $\mathbf{O_A}((n\log n)\log(bn))$ for the overall cost of our solution of Problem 7.1.

Remark 9.4 *We may decrease the computational cost of performing Algorithm 8.1 by choosing the values* e *and* Δ *constant (that is, independent of* n*), say,* $\Delta = 1$ *and* $e = 16\sqrt{2}/(2-5^{1/N})$*. However, (9.3) for* $k=1$ *implies the bound* $f > 6n-4$*, and we still arrive at the same asymptotic cost bound for Problem 7.1.*

10 Analysis of Algorithm 9.1

We will next analyze Algorithm 3.1 and will prove its correctness. We will consider separately cases (a), (b) and (c) of Stage 4.

Case (a). Let $r_s^+(X) \le \epsilon/3$. Then the disc $D_{\epsilon/3} = D(X, r_s^+(X))$ is an $(\epsilon/3)$-neighborhood of all the k zeros of $p(x)$ lying in the input disc $D = D(X_0, R)$. It remains to prove that $i.r.(D_{\epsilon/3}) \ge f$. We have $R > \epsilon \ge 3r^+(X)$, for otherwise already the input disc D would have been an f-isolated ε-cover of all the k zeros of $p(x)$ lying in this disc, and then we would not have invoked Algorithm 9.1. On the other hand, the discs $D_{\epsilon/3}$ and D have common points (the k zeros of $p(x)$). Therefore,

$$|X_0 - X| \le r_s^+(X) + R \le 4R/3,$$

so that

$$\min_{j>k} |X - z_j| \ge (f - 4/3)R \ge (3f-4)r_s^+(X).$$

Consequently, $i.r.(D_{\epsilon/3}) \ge 3f-4 \ge f$ since $f \ge 10$ (cf. (9.3)). This proves correctness in case (a).

Case (b). We need to show that (7.1) holds for the input value e of Algorithm 9.1 and that the square S^* is $(f/\sqrt{2})$-isolated. By the definition of $r_s^+(X)$, the disc $D^+ = D(X, r_s^+(X))$ is equivalent to the input disc D, and we obtain that

$$d^*(S^+) = d^*(D) \ge (r_s^-(X) - r^*(X))/\sqrt{2}\ \ , \quad \rho(S^+) = r_s^+(X)$$

(cf. Definitions 2.1 and 2.3). The latter relations together with Proposition 2.2 imply that $r.r.(S^+) = d^*(S^+)/(2\rho(S^+)) \geq \frac{r_s^-(X)-r^*(X)}{2\sqrt{2}r_s^+(X)}$. Apply (9.6) and obtain that $r.r.(S^+) \geq 1/e$. On the other hand, $2\rho(S^*)$ is the length of the longer side of I, and $I \subseteq S^+$, so $\rho(S^+) \geq \rho(S^*)$, whereas $d^*(S^*) = d^*(S^+)$ since S^* and S^+ are equivalent. Therefore, $r.r.(S^*) \geq r.r.(S^+) \geq 1/e$, and (7.1) is verified. On the other hand, $S^* \subseteq S$, and consequently, $i.r.(S^*) \geq i.r.(S) \geq i.r.(D)/\sqrt{2} \geq f/\sqrt{2}$. This proves correctness in case (b).

Case (c). Correctness of the algorithm will follow when we prove quadratic convergence of Algorithm 9.1 under the following assumption.

Assumption 10.1. *In Algorithm 9.1, a series of J successive recursive applications of Stage 4 has not been interrupted by cases (a) or (b).*

Our first goal (to be reached in Lemma 10.6) is an upper estimate on $r_s(u)/R$ (in terms of $r_s(t)/R$) at each recursive application of Stage 4, where $u = X$ and t satisfy (9.5), $s = n + 1 - k$, and $r_s(x)$ denotes the distance from x to the k-th closest zero of $p(x)$, (cf. Definition 2.6). With no loss of generality, we will assume that $X_0 = 0$ and an f-isolated input disc $D = D(0, R)$ contains exactly k zeros of $p(x)$, $z_1, \ldots, z_k$, $1 \leq k \leq n$, R being an upper bound on $r_{n+1-k}(0)$. [Indeed, we may re-enumerate the zeros of $p(x)$ and shift from any input disc on the complex plane to the disc D by shifting the variable x.] Thus we have our initial relations:

$$|t| = 2R,\ |z_j| \leq R,\ j = 1, \ldots, k,$$

$$|z_j| \geq fR,\ j = k+1, \ldots, n. \tag{10.1}$$

We write

$$Q(x) = \sum_{j=1}^{k} \frac{1}{x - z_j},\ V(x) = \sum_{j=k+1}^{n} \frac{1}{x - z_j} \tag{10.2}$$

and deduce the equation

$$p'(x)/p(x) = Q(x) + V(x). \tag{10.3}$$

We will simplify our analysis by assuming (with no loss of generality) that $|t| \leq 2R$. This bound holds for our initial t and is maintained inductively, as we will prove later on in this section (after the proof of Corollary 10.1).

We let D_{min} denote the smallest disc that is equivalent to the disc D,

$$D_{min} = D(X_{min}, R_{min}),\ R_{min} = (r.r.(D))R,\ X_{min} \in D.$$

Let us also write

$$q(x) = x - k/Q(x),\ q = q(t), \tag{10.4}$$

$$\nabla(x) = (x - q(x))V(x)/k = V(x)/Q(x), \qquad \nabla = \nabla(t). \tag{10.5}$$

We will next prove some auxiliary results.

Lemma 10.1 *Let x lie outside the closed disc D_{min}. Then $Q(x) \neq 0$, and the complex point $q(x)$ of (10.4) lies in the closed disc D_{min}.*

Proof. With no loss of generality, let $X_{\min} = 0$ and let x be positive. Then the real parts of $x - z_j$ and $1/(x - z_j)$ are positive for all j. Therefore, $Q(x) \neq 0$. It remains to prove that $q(x) \in D_{\min}$. Write $y_x(z) = 1/(x - z)$ and let $y_x(D_{min})$ denote the image of the disc D_{min} in the map $z \to y_x(z)$, that is, $y_x(D_{min}) = \{y_x(z) : \ z \in D_{min}\}$. Since $y_x(z)$ is the reciprocal of a linear function in z and since $x \notin D_{min}$, we apply a known result from the theory of conformal maps and analytic functions [Ah79] and deduce that $y_x(D_{min})$ is a disc. On the other hand, $y_x(z_j) \in y_x(D_{min})$, for $j = 1, 2, \ldots, k$, since $z_j \in D_{min}$, for $j = 1, 2, \ldots, k$. Therefore, the average value $Q(x)/k = (1/k) \sum_{j=1}^{k} y_x(z_j)$ lies in $y_x(D_{min})$. The inverse map, $y_x^{-1}(w) = x - 1/w$, transforms $y_x(D_{min})$ into the disc D_{min}, and therefore, $q(x) = y_x^{-1}(Q(x)/k) \in D_{min}$. □

Lemma 10.2 *$u - q = (t - q)\nabla/(1 + \nabla)$, where u, q, t and ∇ satisfy the equations (9.5), (10.4) and (10.5).*

Proof. Due to (9.5) and (10.3) we have

$$u = u(t) = t - k/(Q(t) + V(t)).$$

Deduce from (10.4) that $Q(t)/k = 1/(t - q)$, substitute this expression into the above expression for u and deduce that

$$u = t - \frac{1}{1/(t-q) + V(t)/k} = t - \frac{t-q}{1 + (t-q)V(t)/k} \ .$$

Substitute (10.5) and obtain that

$$u = t - \frac{t-q}{1+\nabla} = \frac{t\nabla + q}{1+\nabla}.$$

Therefore,

$$u - q = \frac{t\nabla + q}{1+\nabla} - q =$$

$$\frac{t\nabla - q\nabla}{1+\nabla} = \frac{(t-q)\nabla}{1+\nabla}.$$

□

Lemma 10.3 *Under the assumptions of Lemma 10.2 we have*

$$|\frac{\nabla}{1+\nabla}| \leq \frac{r_s(t)|V(t)|}{|k - r_s(t)|V(t)|}.$$

Proof. Apply the equations (10.5) for $x = t$, recall that $|t - q| \leq r_s(t)$, and deduce that

$$|\nabla| = |t - q|\ |V(t)|/k \leq r_s(t)|V(t)|/k,$$

$$1/|1 + \nabla| \leq 1/|1 - |\nabla|| \leq 1/|1 - r_s(t)|V(t)|/k|.$$

Combine these bounds on $|\nabla|$ and $1/(1 + |\nabla|)$ and obtain Lemma 10.3. $\square$

Hereafter we will write

$$D_x = D(x, r_s(x)),\ for\ s = n + 1 - k, \quad x = t,\ x = u, \tag{10.6}$$

for $r_s(x)$ denoting the s-th root radius of $p(x)$ at x (cf. Definition 2.6).

Lemma 10.4 *Under the assumptions of Lemmas 10.2 and 10.3, we have* $|u| < 4R$.

Proof. Combine Lemmas 10.2 and 10.3 and obtain that

$$|u - q| \leq |t - q|/(k/\gamma(t) - 1),$$

where $\gamma(t) = r_s(t)|V(t)|$. The relations $|t| \leq 2R$, (10.1) and (10.2) together imply that $r_s(t) \leq 3R$,

$$\gamma(t) = r_s(t)|V(t)| \leq 3R|V(t)| \leq 3(n - k)/(f - 2).$$

Combine this bound with (9.3) and obtain that

$$k/\gamma(t) \geq \frac{(f - 2)k}{3(n - k)} > 2.$$

Therefore, $|u - q| < |t - q| \leq 3R$, and since $q \in D_{min}$, we have $|q| \leq R$, and consequently $|u| < 4R$. $\square$

Lemma 10.5 *Let* $t \notin D_{\min}$. *Then*

$$r_s(u)/R \leq 2R_{\min}/R + \frac{(r_s(t)/R)^2\mu}{|k - (r_s(t)/R)\mu|}, \tag{10.7}$$

where $s = n + 1 - k$ *and*

$$\mu = R|V(t)|. \tag{10.8}$$

Proof. Recall that $|t| \leq 2R \leq 4R,\ |u| < 4R, f \geq 10$. Therefore, by (10.1) and by the definition of D_x given in (10.6), we have $z_j \in D_x,\ j = 1, \ldots, k$, for $x = t$ and $x = u$. Consequently, for $x = t$ and $x = u$ we have $D_{min} \subseteq D_x$; furthermore, the discs D_x and D_{min} have a common point z_j for some $j \leq k$ on their boundary circles. Therefore, $r_s(x) - 2R_{min} \leq |x - a| \leq r_s(x)$ if $a \in D_{min}$ for $x = t$ and $x = u$. On the other hand, $q \in D_{min}$, due to Lemma 10.1 and since $q = q(t),\ t \notin D_{min}$ (cf. (10.4)). Therefore, $r_s(x) - 2R_{min} \leq |x - q| \leq r_s(x)$. Combine the former of these inequalities for $x = u$ with the equation of Lemma 10.2, then apply the latter of them for $x = t$ and obtain that

$$r_s(u) - 2R_{min} \leq |u - q| = |t - q| \; |\nabla/(1+\nabla)| \leq r_s(t)|\nabla/(1+\nabla)|.$$

Combine the latter inequality with one of Lemma 10.3 and obtain that

$$r_s(u) - 2R_{min} \leq \frac{(r_s(t))^2 |V(t)|}{|k - r_s(t)|V(t)||}.$$

This bound and (10.8) together immediately imply (10.7). $\square$

The next lemma extends Lemma 10.5.

Lemma 10.6 *Let $s = n + 1 - k$, $e > 2$, $t \notin D_{\min}$, and*

$$r_s(u) \geq eR_{min}. \tag{10.9}$$

Let f satisfy (9.3) and (9.4). Then

$$r_s(u)/R \leq (r_s(t)/R)^2 \theta, \tag{10.10}$$

for

$$\theta = \frac{1}{(1 - 2/e)((f-2)k/(n-k) - 3)}$$

of (9.4).

Proof. From (10.7) and (10.9), obtain that

$$(1 - 2/e) r_s(u)/R \leq \frac{(r_s(t)/R)^2 \mu}{|k - r_s(t)\mu/R|} = \frac{|(r_s(t)/R)^2}{|(k/\mu) - r_s(t)/R|}. \tag{10.11}$$

Next obtain from the equations (10.2) and (10.8) that

$$\mu \leq R \sum_{j=k+1}^{n} \frac{1}{|t - z_j|} \leq (n-k)R/ \min_{j>k} |t - z_j|.$$

The relations (10.1) and $|t| \leq 2R$ imply that $|t - z_j| \geq (f-2)R$, for $j > k$. Therefore,

$$\mu \leq \frac{n-k}{f-2}.$$

Substitute this bound into (10.11) and obtain that

$$(1 - 2/e) r_s(u)/R \leq \frac{(r_s(t)/R)^2}{|(f-2)k/(n-k) - (r_s(t)/R)|}.$$

We have $r_s(t) \le 3R$, since $|t| \le 2R$. We also note that $(f-2)k/(n-k) > 3$ due to (9.3). Hence we deduce that

$$(1-2/e)r_s(u)/R \le \frac{(r_s(t)/R)^2}{(f-2)k/(n-k)-3}.$$

Substitute the expression for θ from (9.4) and obtain that $r_s(u)/R \le (r_s(t)/R)^2\theta$. This completes the proof of (10.10). $\square$

Lemma 10.7. *Suppose that (9.6) does not hold for $X = u$ at Stage 4 of Algorithm 9.1. Then*

$$r_s(u) > 0.5r_s^-(u) \ge 0.5r_s^+(u)/(1+\Delta)^2, \quad s = n+1-k.$$

Proof. Unless (9.6) hold for $X = u$, we have that

$$r_s^-(u) - r^*(u) < 2\sqrt{2}r_s^+(u)/e \le 2\sqrt{2}(1+\Delta)^2 r_s^-(u)/e.$$

Consequently,

$$r^*(u) > (1 - 2\sqrt{2}(1+\Delta)^2/e)r_s^-(u).$$

Recall that, $r^*(u) \le 5^{1/N} r_s(u)$. Therefore,

$$r_s(u) > (1 - 2\sqrt{2}(1+\Delta)^2/e)r_s^-(u)/5^{1/N}.$$

Substitute (9.2) and obtain that

$$r_s(u) > 0.5r_s^-(u) \ge 0.5r_s^+(u)/(1+\Delta)^2.$$

$\square$

Now recall Assumption 10.1 and let t_{j-1}, t_j and u_j denote the input value t and the output values t and u, respectively, at the j-th recursive applicaiton of Stage 4 of Algorithm 9.1 in the series of J uninterrupted recursive applications of this stage, $j = 1, \dots, J$. Note that in Lemma 10.6 we have $t = t_{j-1}$, $u = u_j$, whereas in (9.7) we have $t = t_j$, $u = X = u_j$.

Corollary 10.1. *Under Assumption 10.1 and for $s = n+1-k$, we have*

$$r_s(t_j)/R < (1+4(1+\Delta)^2)(r_s(t_{j-1})/R)^2\theta \le (r_s(t_{j-1})/R)^2/18, \quad j = 1, \dots, J.$$

Proof. By the virtue of (9.7) and since $X = u$ at Stage 4, we have

$$r_s(t) \le r_s(u) + 2r_s^+(u)\ , \quad s = n+1-k, \quad t = t_j, \quad u = u_j, \quad j = 1, \dots, J,$$

Substitute the bounds of Lemmas 10.6 and 10.7 and obtain that

$$r_s(t_j) < (1+4(1+\Delta)^2)r_s(u_j),$$

$$r_s(t_j)/R < (1+4(1+\Delta)^2)(r_s(u_j)/R) \le (1+4(1+\Delta)^2)(r_s(t_{j-1})/R)^2\theta.$$

Substitute the inequality of (9.4) to complete the proof of Corollary 10.1. □

At the first application of Stage 4, we have $|t_0| \leq 2R$, $r_s(t_0) \leq 3R$. Recursively substitute the upper estimates for $r_s(t_j)/R$, $j = 0, 1, \ldots, J$, on the right-hand side of the inequalities of Corollary 10.1 and obtain that

$$r_s(t_j)/R < 18/36^{2^{j-1}},\ j = 1, \ldots, J.$$

The latter bounds imply that $|t_j| \leq R_{min} + r_s(t_j) \leq (1 + 18/36^{2^{j-1}})R \leq 1.5R$ for $j = 1, \ldots, J$. Thus the initial bound $|t_j| \leq 2R$ for $j = 0$ has been inductively extended to all j. We also note that $\theta \leq 1/90$ under (9.4) and deduce from the above bound on $r_s(t_j)/R$ and Lemma 10.6 that $r_s(u_j)/R < 3.6/36^{2^{j-1}}$, $j = 1, \ldots, J$. Recall that $r_s^+(u) \leq (1+\Delta)^2 r_s(u)$ for all u and deduce the next result.

Proposition 10.1. *Under Assumption 10.1, we have*

$$r_s^+(u_j)/R < 3.6(1+\Delta)^2/36^{2^{j-1}},\quad s = n+1-k,\quad j = 1, \ldots, J.$$

It follows that unless Algorithm 9.1 stops earlier, its J-th iteration step for

$$J = 1 + \lceil \log((2\log(1+\Delta) + \log(10.8R/\epsilon))/\log 36) \rceil \qquad (10.12)$$

outputs $r_s^*(u) = r_s^+(X) < \epsilon/3$; then case (a) occurs, and the algorithm stops. This argument together with the next remark completes the proof of correctness and quadratic convergence of Algorithm 9.1. □

Remark 10.1. *The j-th successive applicaiton of Stage 4 of Algorithm 4.3 may have to include some steps of iteration (3.3) in order to lift the isolation ratios of the discs $D(t, r_s(y))$, $s = n-k+1$, $t = t_j$ to the level $(1+\Delta)^2$. By (9.3), we have $f \geq (1+\Delta)^2$. Since $|t_j| \leq 2R$ and $r_s(t_j) \leq 3R$ for all j, we have i.r.$(D(t, r_s(t)) \geq (f-2)/3$. By (9.3), $f \geq 10$, so that $((f-2)/3)^4 > f \geq (1+\Delta)^2$, and two steps (3.3) will always guarantee the desired lifting of the isolation ratio. If we assumed that $f \geq 13$, then even a single step (3.3) would have always sufficed.*

11 Another Variant of Newtion's Iteration

In this section, we will show a modification of Algorithm 9.1, where the expression (9.5) is replaced by the following one:

$$u = t - \frac{k}{q},\ q = \frac{p'(t)}{p(t)} + \sum_j^* \frac{1}{z_j^* - t}\ . \qquad (11.1)$$

Here z_j^* denotes the current approximation to the zero z_j of $p(x)$, whereas $\sum^*$ denotes the sum in j over all the natural j corresponding to the zeros z_j of $p(x)$ that lie outside the input disc D. Instead of avoiding the points x and t that annihilate $p(t)$ and $p'(x)$, we shall now similarly avoid the points t that annihilate $p(t)$ and $q = q(t)$. Such a modification of Algorithm 9.1 will be called **Algorithm 11.1** and at every

Stage g will be applied to the largest of the currently unprocessed g-squares. This choice should insure faster convergence to the disc D_{min}. Below we will show that the quadratic convergence of Algorithm 9.1 will be preserved for Algorithm 11.1 even if we replace the relations (9.3) and (9.4) by the following ones:

$$f > 2 + \sqrt{6(n-k)/k}, \ f \geq max\{10, \ (1+\Delta)^2\}, \tag{11.2}$$

$$1/\theta = (1-2/e)((f-2)^2 k/(n-k) - 3) \geq 18(1+4(1+\Delta)^2) \ . \tag{11.3}$$

If $n - k > k/3$, then these restrictions on f are weaker than (9.3) and (9.4). To arrive at such a smaller initial isolation ratio, we may need roughly by twice fewer iterations of Algorithm 8.1. This saving however, should be weighted against the additional arithmetic operations required in order to compute the value u via (11.1) [rather than via (9.5)]; for each such an evaluation, we need $3(n-k)$ additional ops.

Let us next briefly analyze the convergence of the computed values u to the disc D_{min} where we apply Algorithm 11.1. The transition from (9.5) to (11.1) amounts to substracting $V^*(x) = \sum_{j=k+1}^{n} \frac{1}{x_j^* - x}$ from both sides of the equation (10.3), which implies that

$$q(x) = p'(x)/p(x) - V^*(x) = Q(x) + V(x) - V^*(x).$$

The analysis of Algorithm 9.1 is extended, with the replacement of $V(x) = \sum_{j=k+1}^{n} \frac{1}{x - z_j}$ by

$$V(x) - V^*(x) = \sum_{j=k+1}^{n} (\frac{1}{x - z_j} - \frac{1}{x - z_j^*}) = \sum_{j=k+1}^{n} \frac{z_j - z_j^*}{(x - z_j)(x - z_j^*)}.$$

The resulting increase of the estimated convergence rate is quantitatively expressed by the following equation for the main parameter θ of Lemma 10.6:

$$\theta = \frac{1}{(1-2/e)((f-2)^2 k/(n-k) - 3)},$$

and the bound $(f-2)k/(n-k) \geq 6$ is replaced by $(f-2)^2 k/(n-k) \geq 6$. In other words, the statement of this basis lemma remains unchanged, except that in the expressions for θ the quantity $f - 2$ should be replaced by $(f-2)^2$. For $f > 3$, the replacement of $f - 2$ by $(f-2)^2$ decreases θ and, therefore, increases the convergence rate defined by (10.10). Furthermore, this replacement enables us to preserve the quadratic convergence of the algorithm provided that the upper bound on the initial isolation ratio f for the disc D satisfies the relations (11.2) and (11.3) instead of (9.3) and (9.4).

References

[Ah79] L. Ahlfors, *Complex Analysis*, McGraw-Hill, New York, 1979.

[Al85] H. Alt, Multiplication Is the Easiest Nontrivial Arithmetic Function, *Theoretical Computer Science*, 36 (1985), pp. 333–339.

[BFKT86/89] M. Ben-Or, E. Feig, D. Kozen, P. Tiwari, A Fast Parallel Algorithm for Determining All Roots of a Polynomial with Real Roots, *SIAM J. on Computing*, 17 (1989) 6, pp. 1081–92 (proceedings version in *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986), pp. 340–349, ACM Press, New York).

[B-OT90] M. Ben-Or, P. Tiwari, Simple Algorithm for Approximating All Roots of a Polynomial with Real Roots, *J. of Complexity*, 6 (1990) 4, pp. 417–442.

[BP91] D. Bini, V. Y. Pan, Parallel Complexity of Tridiagonal Symmetric Eigenvalue Problem, *Proc. 2-nd Ann. ACM-SIAM Symposium on Discrete Algorithms* (1991), pp. 384–393, ACM Press, New York.

[BP94] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, vol. 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[BP,a] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, vol. 2: Selected Topics*, Birkhäuser, Boston, to appear.

[DH69] B. Dejon, P. Henrici (editor), *Constructive Aspects of the Fundamental Theorem of Algebra*, Wiley, London, 1967.

[DL67] L.M. Delves and J.N. Lyness, A Numerical Method for Locating Zeros of an Analytic Function, *Math. Comp.*, 21 (1967), pp. 543-560.

[F81] L.V. Foster, Generalizations of Laguerre's Method: Higher Order Methods, *SIAM J. Numer. Anal.*, 18 (1981), pp. 1004-1018.

[GH72] I. Gargantini, P. Henrici, Circular Arithmetic and Determination of Polynomial Zeros, *Numerische Math.*, 18 (1972), pp. 305-320.

[GL96] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996.

[Go94] S. Goedecker, Remark on Algorithms to Find Roots of Polynomials, *SIAM J. Sci. Comput.*, 15 (1994), pp. 1059-1063.

[Gra72] R. L. Graham, An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set, *Information Processing Letters*, 1 (1972), pp. 132–133.

[Grau71] A. A. Grau, The Simultaneous Improvement of a Complete Set of Approximate Factors of a Polynomial, *J. Numer. Anal.*, 8 (1971), pp. 425–438.

[Gre88] L. Greengard, *Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, MA, 1988.

[H70] A. S. Householder, *The Numerical Treatement of a Single Nonlinear Equation*, McGraw-Hill, New York, 1970.

[Ha87] W. Hager, A Modified Fast Fourier Transform for Polynomial Evaluation and the Jenkins-Traub Algorithm, *Numerische Math.*, 50 (1987), pp. 253-261.

[He74] P. Henrici, *Applied and Computational Complex Analysis*, Wiley, New York, 1974.

[HPR77] E. Hansen, M. Patrick, J. Rusnack, Some Modifications of Laguerre's Method, *BIT*, 17 (1977), pp. 409-417.

[IMSL87] *IMSL User's Manual*, version 1.0, chapter 7, 1987.

[JT70] M. A. Jenkins, J. F. Traub, A Three-Stage Variable-Shift Iteration for Polynomial Zeros and Its Relation to Generalized Rayleigh Iteration, *Numer. Math.,* 14 (1970), pp. 252–263.

[JT72] M. A. Jenkins, J. F. Traub, Algorithm 419: Zeros of a Complex Polynomial, *Comm. ACM*, 15 (1972), pp. 97–99.

[KO63] A. Karatsuba, Yu. Ofman, Multiplication of Multidigit Numbers on Automata, *Soviet Physics Dokl*, 7, 595-596. 1963.

[KS94] M.-H. Kim, S. Sutherland, Polynomial Root-finding Algorithms and Branched Covers, *SIAM J. on Computing*, 23 (1994), 2, pp. 415–436.

[L61] D.H. Lehmer, A Machine Model for Solving Polynomial Equations, *J. ACM*, 8 (1961), pp.151-162.

[M73] K. Madsen, A Root-Finding Algorithm Based on Newton's Method, *BIT*, 13 (1973), pp. 71-75.

[MN93] J.M. McNamee, A Bibliography on Roots of Polynomials, *J. Comput. Appl. Math.*, 47 (1993), pp. 391-394.

[MR75] K. Madsen and J. Reid, Fortran Subroutines for Finding Polynomial Zeros, Report HL75/1172(C.13), *Computer Science and Systems Division, A. E. R. E. Harwell*, Oxford, 1975.

[P85] V. Y. 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 Science* (1985), pp. 522–531, IEEE Computer Science Press.

[P87] V. Y. Pan, Sequential and Parallel Complexity of Approximate Evaluation of Polynomial Zeros, *Computers and Math. (with Applications)*, 14 (1987) 8, pp. 591–622.

[P89] V. Y. Pan, Fast and Efficient Parallel Evaluation of the Zeros of a Polynomial Having Only Real Zeros, *Computers and Math (with Applications)*, 17 (1989) 11, pp. 1475–1480.

[P92] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, 34 (1992) 2, pp. 225–262.

[P93] V. Y. Pan, Accelerated Solution of the Symmetric Tridiagonal Eigenvalue Problem, Tech. Report TR 93–016, *Intern. Computer Science Institute*, Berkeley, CA, 1993.

[P94] V. Y. Pan, New Techniques for Approximating Complex Polynomial Zeros, *Proc. 5th Ann. ACM-SIAM Symp. on Discrete Algorithms* (1994), pp. 260-270, ACM Press, New York, and SIAM Publications, Philadelphia.

[P95] V. Y. Pan, Optimal (up to Polylog Factors) Sequential and Parallel Algorithms for Approximating Complex Polynomial Zeros, *Proc. 27th Ann. ACM Symp. on Theory of Computing* (1995), pp. 741-750, ACM Press, New York.

[P95a] V. Y. Pan, Weyl's Quadtree Algorithm for the Unsymmetric Eigenvalue Problem, *Applied Math. Letters*, 8 (1995) 5, pp. 87-88.

[P96] V. Y. Pan, Optimal and Nearly Optimal Algorithms for Approximating Polynomial Zeros, *Computers and Mathematics (with Applications)*, 31 (1996) 12, pp. 97-138.

[P96a] V. Y. Pan, Approximating Complex Polynomial Zeros: Modified Quadtree (Wey's) Construction and Improved Newton's Iteration, *Research Report 2894, INRIA Sophia-Antipolis*, France, 1996.

[P97] V. Y. Pan, Solving a Polynomial Equation: Some History and Recent Progress, *SIAM Review*, 39 (1997) 2, pp. 187-220.

[PKSHZ96] V. Y. Pan, M.-h. Kim, A. Sadikou, X. Huang, A. Zheng, On Isolation of Real and Nearly Real Zeros of a Univariate Plynomial and Its Splitting into Factors, *J. of Complexity*, 12 (1996), pp. 572-594.

[PL93] V. Y. Pan, E. Linzer, A New Approach to Bisection Acceleration for the Symmetric Eigenvalue Problem, manuscript, 1993.

[PS85] F. P. Preparata, M. I. Shamos, *Computational Geometry: An Introduction*, *Texts and Monographs in Computer Science*, Springer, New York, 1985.

[R87] J. Renegar, On the Worst-Case Arithmetic Complexity of Approximating Zeros of Polynomials, *J. of Complexity*, 3 (1987) 2, pp. 90–113.

[RS92] J. Renegar, M. Shub, Unified Complexity Analysis for Newton LP Methods, *Math. Programming*, 53 (1992), pp. 1-16.

[Sa84] H. Samet, The Quadtree and Related Hierarchical Data Structures, *Computing Surveys*, 16 (1984) 2, pp. 187-260.

[Sc82] A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, Manuscript, *Dept. of Math., Univ. of Tübingen*, Tübingen, Germany, 1982.

[Schr57] J. Schröder, Uber das Newtonsche Verfahren, *Arch. Rat. Mech. Anal.*, 1 (1957), 154-180.

[Se94] H. Senoussi, A Quadtree Algorithm for Template Matching on Pyramid Computer, *Theor. Comp. Science*, 136 (1994), pp. 387-417.

[SeS41] J. Sebastiao e Silva, Sur une Méthode d' Approximation Semblable à Celle de Graeffe, *Portugal Math.*, 2 (1941), pp. 271-279.

[SS93] M. Shub, S. Smale, Complexity of Bezout's Theorem I: Geometric Aspects, *J. of the Amer. Math. Soc.*, 6 (1993), pp. 459–501.

[SS93a] M. Shub, S. Smale, Complexity of Bezout's Theorem II: Volumes and Probabilities, *Computational Algebraic Geometry* (F. Eyssette, A. Galligo, Editors), *Progress in Mathematics*, 109 (1993), pp. 267–285, Birkhäuser.

[SS93b] M. Shub, S. Smale, Complexity of Bezout's Theorem III: Condition Number and Packing, *J. of Complexity*, 9 (1993), pp. 4–14.

[SS96] M. Shub, S. Smale, Complexity of Bezout's Theorem IV: Probability of Success, Extensions, *SIAM J. on Numer. Analysis*, 33 (1996), pp. 128–148.

[SS94] M. Shub, S. Smale, Complexity of Bezout's Theorem V: Polynomial Time, *Theoretical Computer Science*, 133 (1994), pp. 141–164.

[Tu84] P. Turan, *On a New Method of Analysis and Its Applications*, Wiley and Sons, New Jersey, 1984.

[U97] F. Uhlig, The DQR Algorithm, Basic Theory, Convergence and Conditional Stability, *Numerische Math.* 76 (1997), pp. 515–553.

[W903] K. Weierstrass, *Neuer Beweis des Fundamentalsatzes der Algebra,* Mathematische Werker, Tome III, Mayer und Mueller, Berlin, 1903, pp. 251-269.

[We24] H. Weyl, Randbemerkungen zu Hauptproblemen der Mathematik, II, Fundamentalsatz der Algebra and Grundlagen der Mathematik, *Math. Z.*, 20 (1924), pp. 131-151.

Appendix.

A. Application of Polynomial Rootfinders to the Unsymmetric Eigenvalue Problem

The eigenvalues of an $n \times n$ general matrix A equal the zeros of its characteristic polynomial, $p_A(x) = \det(xI - A)$. In principle, one may approximate the eigenvalues by first computing the coefficients of $p_A(x)$ and then applying the known polynomial rootfinders (say, ones of this paper). Practically, this approach is not recommended because the coefficients of $p_A(x)$ must be computed with an increased precision – their perturbation may cause increased perturbation of the zeros. We wish to argue, however, that the additional cost of the transition from A to $p_A(x)$ due to the increase of the computational precision (or even to the application of the infinite precision rational arithmetic at this stage) should not overweight the advantages of the subsequent application of effective polynomial rootfinders.

Indeed, practical computation of the eigenvalues of A starts with the orthogonal similarity transformation (at the cost $\mathbf{O_A}(n^3)$) of the matrix A to the Hessenberg from $H = (h_{i,j}) = UAU^{-1}$, $h_{i,j} = 0$ for $i > j + 1$, or to the tridiagonal form $T = (t_{i,j}) = QAQ^H$, $t_{i,j} = 0$ for $|i - j| > 1$, where W^H denotes the Hermitian transpose of a matrix W, $U^H U = I$ is the identity matrix, $Q^H Q = D$ is a diagonal matrix (cf. [GL96], [U97]). The transition from T to $p_T(x) = p_A(x)$ requires only $O(n)$ arithmetic operations (cf. e.g. [BP91], Appendix A), even performing them with higher precision still means a sufficiently low computational cost at this stage.

The practical method of choice for appoximating the eigenvalues of a Hessenberg or complex tridiagonal matrix is the QR algorithm. It requires orders of n^2 memory and of n^2 ops per iteraton [GL96]. For the number of iterations required to achieve single precision approximation of the n eigenvalues of A, there is a purely empirical estimate of order n on the "average case " input. This estimate, however, is "very approximate", according to [GL96], p. 359; furthermore, no convergence is insured in the case of multiple or clustered eigenvalues already for moderately large n (for $n > 50$ according to [Go94]). The recent DQR modification of [U97] yields certain saving in memory space and time cost per iteration at the price of certain deterioration of the convergence. Namely, possible breakdown of the iteration may force the DQR process to stop, and then the computation of the eigenvalues of T must resume from scratch.

Now we make our final comparison: the bounds of $O(n)$ on the memory space and from $O(n^2)$ to $O(n)$ (up to polylog factors) on the number of ops involved in the algorithms of this paper and of [P95], [P96], respectively, have been proved for the *worst case input* and *very high accuracy output*, that is, even in their most difficult cases these algorithms never have breakdown and approximate all the zeros of $p(x)$ within a required error bound, at the cost which is superior by at least from 1 to 2 orders of magnitudes (both in memory and ops estimates) over the QR algorithm in its easiest case.

B. Figures

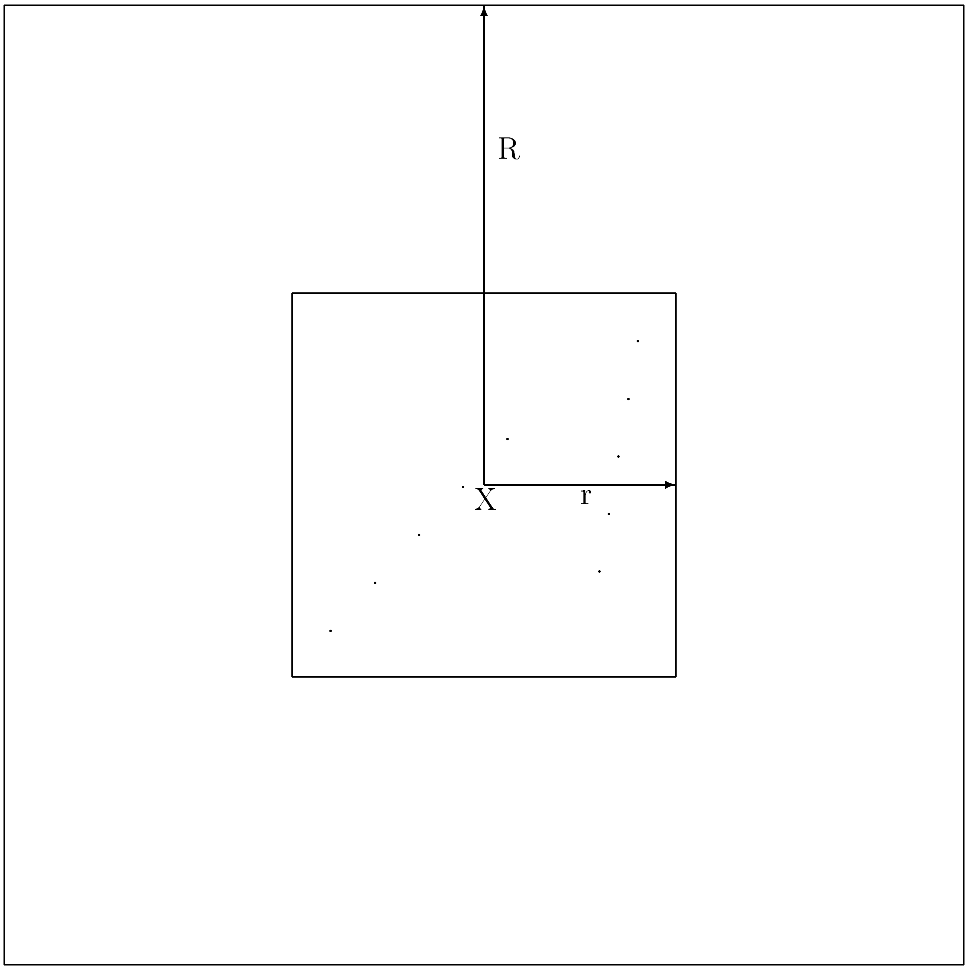

Figure 1: Isolation and rigidity ratios for squares.
$i.r.(S(X,r)) \geq R/r, \quad r.r.(S(X,R)) \geq r/R.$
The dots show all the zeros of $p(x)$ lying in the larger square. They also lie in the smaller square.

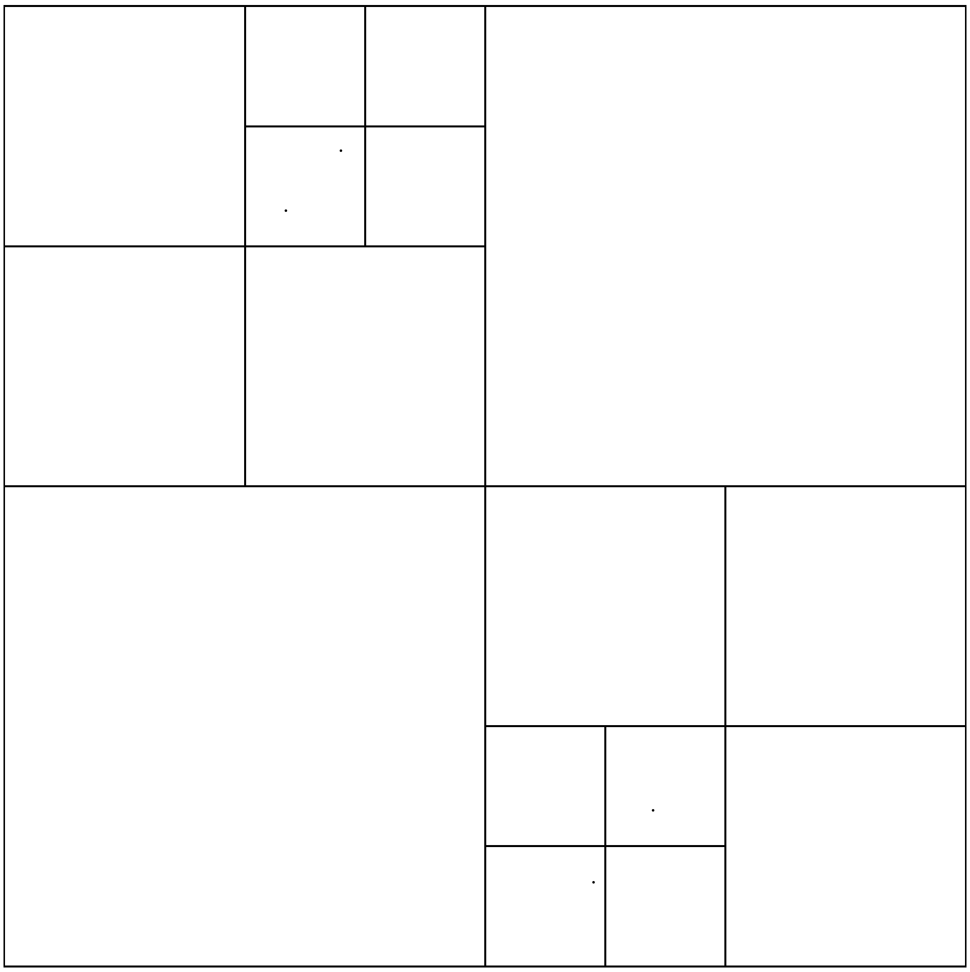

Figure 2: Weyl's algorithm.
The dots show all the zeros of $p(x)$ lying in the large square.

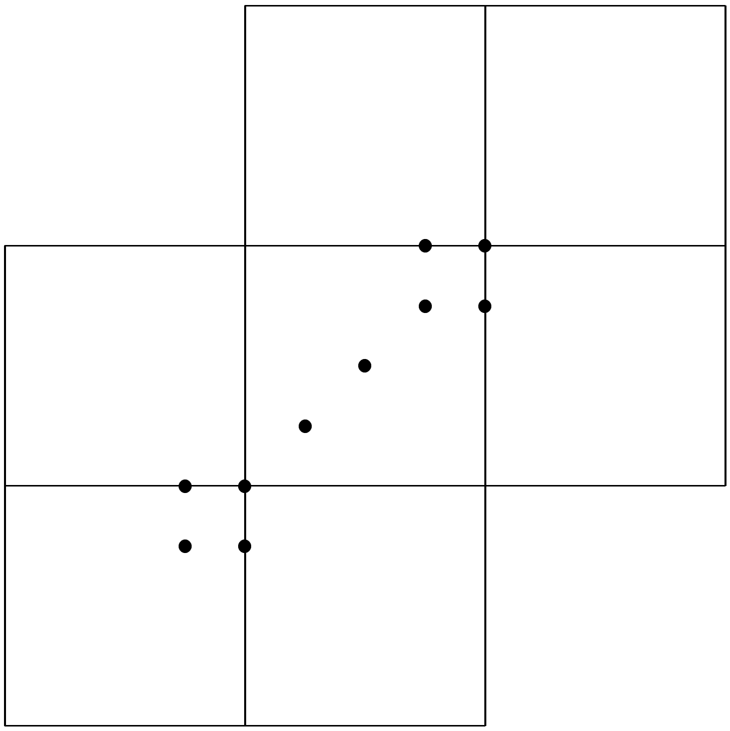

Figure 3:
14 smaller g-squares represented by their southwestern vertices (shown by black circles) are covered by 8 larger $(g-2)$-squares.

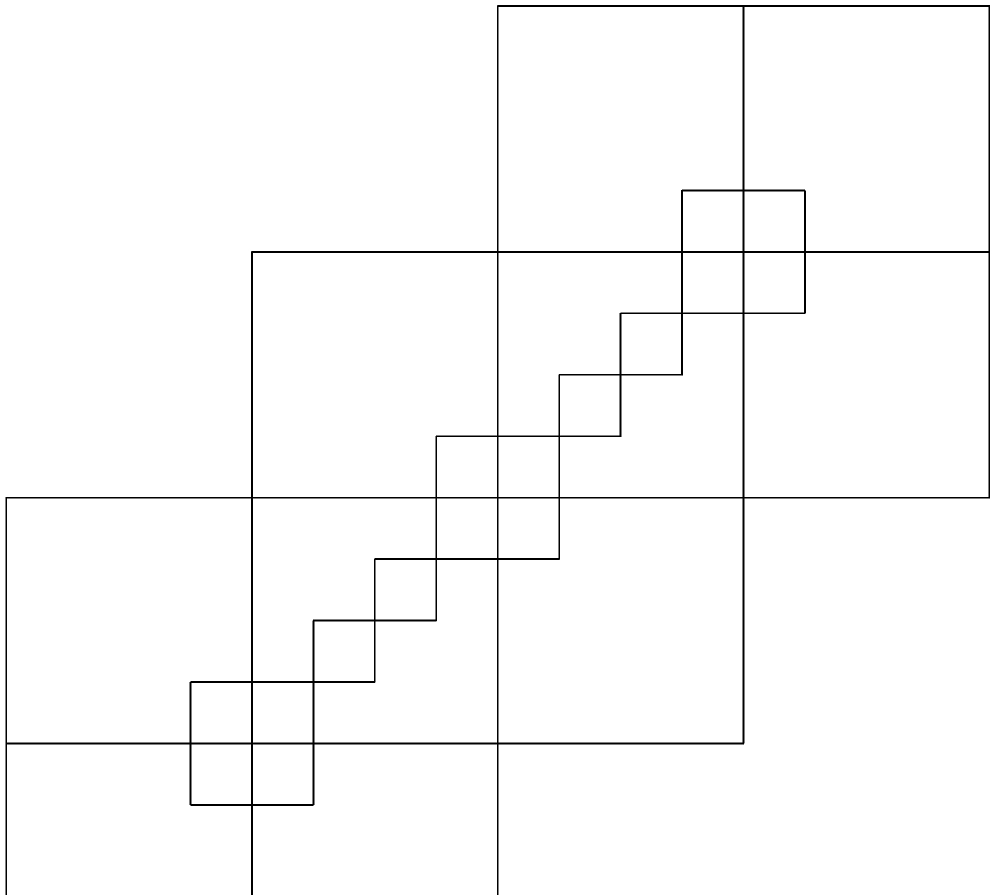

Figure 4:
16 smaller $(g+2)$-squares versus 10 larger g-squares of Weyl's algorithm, with diagonal connections allowed.