# On the Complexity of Approximate Solution of Some Fundamental Problems of Algebraic Computations

by

Akimou Sadikou

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of requirements for the degree of Doctor of Philosophy, The City University of New York

1996.

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

8/1/96

Date 8/1/96

Victor Pan

Chair of Examining Committee

Prof. Victor Pan

Date

Executive Officer

Prof. Stanley Habib

Prof. V. Pan

Prof. S. Habib

Prof. E. Zachos

Prof. D. Bini

Dr. Myong-Hi Kim

Supervisory Committee

The City University of New York.

# Abstract

# On the Complexity of Approximate Solution of Some Fundamental Problems of Algebraic Computations

by

AKIMOU SADIKOU

Advisor: Professor Victor Pan

Our subject is the algorithms for approximate solution of some problems of univariate polynomial computations, namely, polynomial division, multipoint evaluation and interpolation, and solving a polynomial equation. We approach this subject by using some nonstandard techniques of algorithm design and analysis, which leads us to some improvement of the previously known estimates for the computational complexity, to some promise of improving practical performance of the known algorithms, and to some new insights into the existent approaches to the solution.

**Keywords:** parallel algorithms, computational complexity, structured matrices, polynomial division, multipoint evaluation and interpolation, complex polynomial zeros.

**Acknowledgement**

I will start this acknowledgement by thanking God of all his guidance during the period of this project.

I would like to express my deepest thanks and appreciation to my advisor Prof. Victor Pan. He has been an on-going source of encouragement and support of all kinds. He was also generous with his suggestions and comments.

To all members of the committee:

It was a pleasure and honor to have Prof. Dario Bini to serve on the committee. I have appreciated his helpful comments, in particular, his advice on the numerical aspects of the 4th chapter, including the choice of an appropriate proximity test satisfying the requirements of numerical stability. I especially appreciate his hospitality while I was visiting Pisa, in the summer of 1995.

My thank to Dr. Myong-Hi Kim for accepting being on the committee despite her busy schedule.

Professor Zachos was a kind mentor in the initial period of my study. He was a person who encouraged me to pursue a doctorat degree and directed me to Prof. Pan. My gratitude to you for all your advice since I met you in Brooklyn College while I was preparing my MS.

Professor Habib was always available for advice and help, and his assistant Mr. Joseph Driscoll was always super-cooperative.

Most of all, I owed tremendous debt of gratitudes to all my family. My wife-to-be, Affissatou Adegbindin and our son Mahfouz Sadikou have been special for their unconditional love, devotion, and understanding, during the difficult period of separation that was necessary for my work on the dissertation. My mother and my two sisters have my thanks for their guidance and their constant prayers.

# Contents

# List of Figures

# Chapter 1

# The Subjects of the Study

**Summary**

We study some fundamental computational problems with univariate polynomials, that is, their division, multipoint evaluation and interpolation, and solving a polynomial equation. We seek approximate solutions to these problems, within a given tolerance bound for the output approximation errors. All these computational problems are central in the field and have been intensively studied by the algorithm designers for a long time (see some relevant bibliography in [AHU], [BM], [Kn], [BP94], [Sc82], [Sc82a], [P87], [P94], [P95]). Nevertheless, we have found some distinct approaches that enable us to improve some of the previous bounds on the computational complexity of the solution, to give a new insight into some of the existent approaches and techniques, and to propose some amendments of the known algorithms that promise to improve their practical performance.

## 1.1 Model of Computation.

We will analyze both sequential behavior of our algorithms and their parallel implementations. We will use *the customary RAM model* for sequential computing and *EREW PRAM model* for parallel computing.

### 1.1.1 Introduction of the Models of Computing.

The RAM (random access machine) model is one of the customary models of sequential computation. It consists of a memory, a read only input tape, a write only output tape and a program (a code for the algorithm). The program cannot be modified. The input tape contains a sequence of the input characters (a set of variables and constants). The memory consists of an unbounded sequence of registers (spaces); each register can hold a single variable of the sequence of the input. One special register is the accumulator, where computations are performed. The instructions (to this special register) resemble the instructions found on an actual computer. In the RAM model we are allowed such instructions as load, store, read, write into memory location, execute operations (add, subtract, multiply, and divide), compare, jump and halt.

The PRAM model consists of a control unit, global memory, and an unbounded set of processors, each with its own private memory. Active processors usually execute identical instructions. Any processor can be enabled or disabled, and may access the global memory. Any processor (with its private memory ) may simulate a RAM model.

There are various PRAM models. They differ in the way they handle read or write conflict:

1. EREW (exclusive read exclusive write), read or write conflicts are not allowed.

2. CREW (concurrent read exclusive write), multiple processors may read from the same global memory location during the same instruction step, write conflicts are not allowed.

3. CRCW (concurrent read concurrent write), different policies are used to solve concurrent write to the global memory. There are 3 models; in the strongest one each processor is assigned priority, and the global memory location accepts the result only from the processor with the highest priority among all the processors in conflict.

### 1.1.2 The Computational Cost Estimates.

To each RAM instruction its time-cost is assigned, which is time for its execution. In particular, we may assume that all the instructions have unit time-cost. This is said to define the *uniform cost criterion*. We know that in practice, the cost may vary depending on the precision of the variable operands and on the kind of instruction involved (multiplication may cost more than addition). The overall sequential time cost is defined as the sum of the time-cost values assigned to all the instructions involved.

Deducing our computational complexity estimates, we will be allowing arithmetic operations (infinite precision), pairwise comparisons of positive numbers, and the evaluation of the $k$-th roots of complex numbers, for natural $k$, as unit cost operation.

For some of our algorithms we will also extend the study from infinite to the finite precision computations. Therefore, it is important to study the computation errors and the Boolean complexity of the computations. A simple and common way is to assign an appropriate precision values (number of digits) to the operands (data) and to the output of every arithmetic operation in the algorithm. Then one may deduce the Boolean complexity by combining the arithmetic estimates and the complexity (time-cost) for executing an operation with operands represented with such a precision.

A PRAM computation begins with the input stored in the global memory and a single active processing element. As we have already said, during each step of the computation, any active, enabled processor may simulate the RAM model, that is, it may read a value from a single private or global memory location, perform a single RAM operation, and write into one local or global memory location. In our case, all active, enabled processors will execute the same instruction, on different memory locations. The computation terminates when the last processor halts.

Among all the PRAM models, we will use the EREW PRAM model. Any algorithm using EREW PRAM model can be simulated by all the other PRAM models with the same complexity estimates.

### 1.1.3 Some Definitions.

We will use the notation $\mathbf{O}(t,p)$ for simultaneous bound $\mathbf{O}(t)$ on the parallel time and $\mathbf{O}(p)$ on the number of processors involved. We assume the *B-principle*, that is, a variant of Brent's scheduling principle, under which in time $\mathbf{O}(tp)$ a single processor may simulate the work performed by $p$ processors in time $t$. The product $W = tp$ of the time and processor bounds is called the *potential work* of a parallel algorithm [KR], [BP94], [PP]. The B-principle implies that for the potential work, $w$, of a parallel algorithm we always have $T = \mathbf{O}(w)$, where $T$ denotes the record sequential time upper bound for the solution of the given computational problem. (Indeed, under the B-principle, $\mathbf{O}(t,p)$ implies sequential time $\mathbf{O}(tp)$.) A parallel algorithm supporting the complexity bounds $\mathbf{O}(t,p)$ is called *work or processor optimal* if $tp = \mathbf{O}(T)$ and is called *work or processor efficient* if $tp = \mathbf{O}((\log T)^d T)$, where $d$ is some fixed constant and $T$ is defined above [KR], [BP94], [PP].

log denotes logarithms to the base 2.

## 1.2 The Fast Fourier Transform (FFT).

The discrete Fourier transform (DFT) is fundamental to our algorithms, and in fact the DFT also has many applications in science and engineering.

Hereafter, $n$ denotes a positive integer.

Given $n$, $\omega$ denotes a primitive $n$-th root of 1, that is,

$$\omega^n = 1, \quad \omega^h \neq 1, \quad h = 1, \ldots, n-1 ;$$

$\omega = \exp(2\pi\sqrt{-1}/n)$ is a primitive $n$-th root of 1. The set $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$ defines the set of $n$ Fourier points.

### 1.2.1 The Discrete Fourier Transform (DFT).

let $\Omega = [\omega_{i,j}]$ and $\bar{\Omega} = [\bar{\omega}_{i,j}]$ be $n \times n$ matrices, with $\omega_{i,j} = \omega^{ij}$, $\bar{\omega}_{i,j} = \omega^{-ij}$ for $i, j = 0, \ldots, n-1$. Then $\Omega^{-1} = \frac{1}{n}\bar{\Omega}$. Let $\mathbf{v} = [v_j]^T$ and $\mathbf{p} = [p_j]^T$ be $n$-dimensional (column) vectors.

**Definition 1.2.1.** *For given $n$, $\mathbf{v}$ and $\mathbf{p}$ as defined above, the forward DFT of $\mathbf{p}$ may be defined as the matrix-vector product $\Omega\mathbf{p}$, and the inverse DFT of $\mathbf{v}$ as $1/n$-th the matrix-vector product $\bar{\Omega}\mathbf{v}$.*

It is clear that if DFT of $\mathbf{p}$ is $\mathbf{v}$ (i.e $\Omega\mathbf{p} = \mathbf{v}$), then the inverse DFT of $\mathbf{v}$ is equal to $\mathbf{p}$ [i.e. $(1/n)(\bar{\Omega}\mathbf{v}) = \mathbf{p}$). We may also define the DFT as the evaluation of the polynomial with coefficients $p_0, \ldots, p_{n-1}$ at the set of $n$ Fourier points $\omega^i$, and the inverse DFT as the interpolation of the polynomial with value $v_i$ at the point $\omega^i$ for $i = 0, 1, \ldots, n-1$ (see Chapter 2).

### 1.2.2 The Fast Fourier Transform (FFT).

The straightforward implementation of DFT has time-complexity $\mathbf{O}(n^2)$. The fast Fourier transform is an effective algorithm for solving the DFT problem at a cost $\mathbf{O}(n \log n)$. It is a recursive algorithm based on the divide-and-conquer strategy, and it can be parallelized easily. Let us assume that $n$ is a power of 2, $T(n)$ denote the minimum number of operations sufficient for the DFT of the $n$-dimensional vector, and $y = x^2$. We have:

$$\begin{aligned} p(x) &= p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1} \\ &= (p_0 + p_2 x^2 + \ldots + p_{n-2} x^{n-2}) + x(p_1 + p_3 x^2 + \ldots + p_{n-1} x^{n-2}) \\ &= p_{(1)}(x^2) + x p_{(2)}(x^2) \\ &= p_{(1)}(y) + x p_{(2)}(y), \end{aligned}$$

where the degrees of $p_{(1)}(x)$ and $p_{(2)}(x)$ are $n/2$.

We may have:

**Lemma 1.2.1.** *If $n$ is even and positive, then the squares of the elements of the set of $n$-th Fourier points form the set of $n/2$-th Fourier points.*

Form Lemma 1.2.1 and the above expression of $p(x)$ via $q(x)$ and $s(x)$ we have:

$$T(n) \leq 2T(n/2) + 1.5(n-1) \text{ arithm. ops} .$$

The recursive application of the latter estimate for $n$ replaced by $n/2$, $n/4, \ldots$ give us the bound

$$T(n) \leq 1.5n \log n .$$

Therefore, $T(n) = \mathbf{O}(n \log n)$. This algorithm is amenable to fully effective parallelization supporting the parallel complexity estimate $\mathrm{O}(\log n, n)$.

## 1.3 Presentation of the Work.

In the second chapter, we will describe ours results on approximate polynomial division, in the third chapter, on multipoint polynomial evaluation and interpolation, and in chapter 4, on approximate solution of a polynomial equation. In each case, we will also present some background material.

# Chapter 2

# Polynomial Division

**Summary**

The evaluation-interpolation technique of A. Toom is applied to approximate polynomial division with a remainder. This elementary method leads to the same computation (except for simple power shifts), and to the same record asymptotic parallel complexity estimate for this problem, $O(\log m, m)$, as those supportint the much more involved methods of D. Bini and A. Schönhage. This approach also enables us to simplify the approximation error analysis.

## 2.1 Introduction.

Polynomial division problem is stated as follows:
for given coefficients of two polynomials

$$s(x) = \sum_{i=0}^{m} s_i x^i \ , \quad t(x) = \sum_{j=0}^{n} t_j x^j \ , \quad s_m t_n \neq 0 \ , m \geq n$$

(s(x) is the *dividend* and $t(x)$ the *divisor*),
compute, over a fixed field $\mathbf{F}$, the coefficients of the two polynomials, $q(x) = \sum_{j=0}^{m-n} q_j x^j$ (the quotient) and $r(x) = \sum_{j=0}^{n-1} r_j x^j$ (the remainder), such that

$$s(x) = t(x)q(x) + r(x) \ , \ \deg r(x) < \deg t(x) \ . \qquad (\ 2.1.1\ )$$

Here and hereafter, $\deg p(x)$ denotes the degree of a polynomial $p(x)$. Clearly, $\deg s(x) = \deg q(x) + \deg t(x)$, under ( 2.1.1 ).

The problem of polynomial division is a classical problem. There always exists unique solution $q(x)$ and $r(x)$ for given $s(x)$ and $t(x)$. The remainder $r(x)$ is also called the residue of $s(x)$ modulo $t(x)$. Polynomial division is a major block of many algorithms for polynomial computations, for instance, for polynomial evaluation and interpolation, for approximating polynomial zeros (cf. next chapters), and for computing the greatst common divisor (GCD) of two polynomials.

We concentrate on computing $q(x)$, and assume the computations over the field of complex numbers $\mathbf{C}$ under the EREW PRAM model or under any other model of parallel computing that supports the bounds $\mathbf{O}(\log K, K)$ [time, processors] on the complexity of the discrete Fourier transform at the $K$-th roots of 1 (defined in the previous section). Then the remainder $r(x)$ can be computed by means of multiplying the 2 polynomials, $q(x)$ and $t(x)$, and subtracting the result from $s(x)$. By applying fast algorithms (based on FFT) for polynomial multiplication, one may perform this algorithm by using $\mathbf{O}((m+n)\log(m+n))$ arithmetic operations.

The classical algorithm for " synthetic division" ([Kn], p. 402) relies on the recursive reduction of the degree of $s(x)$. It involves up to $(2n+1)(m-n+1)$ arithmetic operations. This algorithm does not involve any division operation if $t_n = 1$, and for any $t_n \neq 0$, we may replace $s(x)$ by $t_n^n s(x)$. Even a more effective solution, due to Sieveking and Kung [AHU], [BM], [BP94], reduces the evaluation of the coefficients of $q(x)$ to recursive computation of polynomial products. The straightforward parallelization of the Sieveking-Kung algorithm for this computation (using the B-principle) supports the complexity bounds $\mathbf{O}((\log N)^2, N/\log N)$, where $N = \deg q(x) = \deg s(x) - \deg t(x)$. This means that the algorithm is fast and work optimal. However, since polynomial division is a very fundamental operation, by all means one should try to obtain its further acceleration. This is indeed possible, and one may reach the bound $t = \mathbf{O}((\log N)\log^* N)$, by shifting to a faster but still work optimal algorithm [BP93], which supports the bounds

$\mathbf{O}((\log N)\log^* N, N/\log^* N)$, where $\log^* N = \min\{h, \log^{(h)} N \leq 1\}$ $\log^{(0)} N = \log N$, $\log^{(h)} N = \log\log^{(h-1)} N$, $h = 0, 1, \ldots, \log^* N$.

It is also known how to reach the time bound $\mathbf{O}(\log N)$, at the price of abandoning the work optimality property. In particular, one may reach the bound $\mathbf{O}(\log N, N\log\log N)$ [RT] and even $\mathbf{O}(h\log N, (N/h)(1 + 2^{-h}\log^{(h)} N))$ for any $h \leq \log^* N$ [BP93]. Devising a polynomial division algorithm that would have simultaneously optimized the time and processor bounds, that is, would have supported the optimal parallel arithmetic complexity bounds $\mathbf{O}(\log N, N)$, remains, however, a challenging open problem, unless one shifts to approximation version of polynomial division problem.

The approximation algorithm supporting the latter optimal bounds was proposed by D. Bini [B82, B84] and, independently, by A. Schönhage [Sc82a]. The 2 versions of this algorithm, by Bini and by Schönhage, look very different from each other and are based on some distinct advanced techniques. Bini's version appears in the form of solving a triangular Toeplitz linear system of equations (using diagonalization), whereas Schönhage's version relies on the Laurent expansion of the quotient of 2 polynomials into a power series and Cauchy integration. The comparison of these 2 algorithms made in [BP86], however, shows that they are computationally identical, that is, produce exactly the same intermediate (auxiliary) and, of course, output values.

We propose the third interpretation of the same approach, giving a new insight into it (see [PSL] and [PSL,a]). We believe that this is a conceptually simpler approach, and it well demonstrates the power of the general and effective *techniques of evaluation and interpolation*, due to Toom [To] and hereafter referred to as the *e.-i.t.*.

This technique is most widely known from its texbook application to polynomial multiplication ([AHU], [BM], [Baase]), and its further applications to some other polynomial computations can be found in [BP94]. It was, however, a novelty of [PSL], [PSL,a] in applying this technique to polynomial division, even though the basic idea of this application looks quite simple.

If $r(x) \equiv 0$, the computation of $q(x)$ using the e.-i.t. is straightforward (Algo-

rithm 2.2.1) and supports the record parallel arithmetic complexity estimate $\mathbf{O}(\log m, m)$. In this note, the e.-i.t. is extended to the general case, where $r(x) \not\equiv 0$ (Algorithm 2.2.2), arriving at the record complexity estimate, $\mathbf{O}(\log m, m)$, for computing *any precision approximation (a.p.a.)* to $q(x)$. The exact evaluation of $q(x)$ at the parallel cost $\mathbf{O}(\log m, m)$ remains an open problem (current record is given by the cited results of [BP93]).

Estimating the errors of the output approximations to the coefficients of $q(x)$ is not straightforward but is facilitated by some techniques from [BP86]. In particular, these techniques exploit the reduction of polynomial division to the solution of a triangular Toeplitz linear system of equations. The error analysis finally shows that the output errors based on our first, most straightforward algorithm are too large, not allowing us to obtain a desired high precision approximation of the coefficients of the quotient. Rather surprisingly, however, some additional rescaling techniques enable us to obtain a decisive refinement, so as to arrive at approximation within any given positive value of the tolerance to the output errors. These estimates indicate the effectiveness of the algorithm even if the extra cost of computation with long binary numbers involved is taken into account. Further analysis also shows that the algorithm is computationally equivalent to the well-known algorithms of [B82], [B84], and [Sc82], but we believe that our derivation of the algorithm is simpler and more illuminating. We will analyze the most important special case where the input polynomial $t(x)$ and the quotient $q(x)$ have integer coefficients.

## 2.2 Approximate Polynomial Division.

In this chapter, we fix a positive integer $K$ exceeding $m - n$. We choose $K = 2^k$, where $k = \lceil \log(m - n + 1) \rceil$, and then perform $DFT(K)$ by means of FFT.

First assume that $t(x)$ divides $s(x)$, so that $r(x) = 0$. Then application of the e.-i. technique immediately gives us $q(x)$, according to the following algorithm:

**Algorithm 2.2.1.**

**Input:** *The coefficients of 2 polynomials, $s(x)$ and $t(x)$, such that $t(x)$ divides $s(x)$.*

**Output:** *The polynomial $q(x)$ such that $q(x) = s(x)/t(x)$.*

**Computations:**

1. *Evaluate $s(x)$ and $t(x)$ on the set $S_h$ of all the $2^h$-th roots of 1 for $h = \lceil \log(m - n + 1) \rceil$. (By using FFT, this computation is performed in $1.5h2^h$ arithmetic operations.)*

2. *Evaluate $q(x) = s(x)/t(x)$ on the same set $S_h$ ($2^h$ divisions).*

3. *Apply inverse FFT to compute the coefficients of $q(x)$ by using $1.5h2^h + 2^h$ arithmetic operations.*

The overall sequential arithmetic computational cost is asymptotically the same as Sieveking-Kung's but has a lower overhead constant factor, which implies substantial advantages for its practical implementation. Moreover, this algorithm also allows its optimal parallelization, as does FFT.

In the case of general input polynomials $s(x)$ and $t(x)$, we extend this approach by exploiting the upper bound $n-1$ on the degree of the remainder $r(x)$. Due to this bound, the ratio $r(x)/t(x)$ converges to 0 as $x \longrightarrow \infty$, and its influence can be ignored if we approximate $q(x) = \frac{s(x)}{t(x)} - \frac{r(x)}{t(x)}$, where $|x|$ is large. This is what we do, by first scaling the variable $x$, that is, we make transition to a new variable $y = x/H$, $H$ being a large positive constant, and then we apply Algorithm 2.2.1 for $x$ replaced by $y$, $s(x)$ replaced by $S(y) = s(Hy)$ and $t(x)$ replaced by $T(y) = t(Hy)$.

Specifically, we will choose a large positive $H$ (to be specified later on) and proceed as follows:

**Algorithm 2.2.2.**

**Input:** *The coefficients of 2 polynomials, $s(x)$ and $t(x)$, and a positive integer $H$.*

**Output:** *Approximations $q_0^*, \ldots, q_{m-n}^*$ to the coefficients $q_0, \ldots, q_{m-n}$ of the polynomial*

$q(x) = \sum_{i=0}^{m-n} q_i x^i$ *such that* $\deg(s(x) - q(x)t(x)) < \deg t(x)$.

**Computations.**

*Compute*

1. *the coefficients* $S_i(H)$ *and* $T_j(H)$ *of the polynomials* $S(H,y) = s(Hy)$ *and* $T(H,y) = t(Hy)$ *[by means of the parallel prefix algorithm (compare [KR]), at the cost* $\mathbf{O}(\log m, m/\log m)$*], for* $i = 0, 1, \ldots, m$ *and* $j = 0, 1, \ldots, n$,

2. $S(H,\omega^i)$ *and* $T(H,\omega^i)$, *for* $i = 0, 1, \ldots, K-1$ *[*$\mathbf{O}(m)$ *additions and two* $DFT(K)$*],*

3. $s(H\omega^i)/t(H\omega^i)$, *for* $i = 0, 1, \ldots, K-1$ *[*$K$ *concurrent divisions],*

4. *the coefficients* $Q_j^*(H)$ *of the polynomial*

$$Q^*(H,y) = \sum_{j=0}^{m-n} Q_j^*(H) y^j \ ,$$

*such that* $Q^*(H,\omega^i) = s(H\omega^i)/t(H\omega^i)$ *[an inverse* $DFT(K)$*],*

5. *the coefficients of polynomial*

$$q^*(x) = \sum_{j=0}^{m-n} q_j^* x^j \ ,$$

*with* $q_j^* = Q_j^*(H)/H^j$

*[by means of the parallel prefix algorithm, at the arithmetic cost* $\mathbf{O}(\log(m-n), (m-n)/\log(m-n))$*].*

*Output* $q_0^*, \ldots, q_{m-n}^*$, *the computed approximations to* $q_0, \ldots, q_{m-n}$.

Clearly, the computational cost of performing this algorithm is bounded by $\mathbf{O}(\log m, m)$.

**Remark 2.2.1.** *We shall choose* $H$ *sufficiently large, such that* $t(H\omega^\ell) = 0$ *for no* $\ell$.

**Remark 2.2.2.** *If* $m \leq 2n$, *then replacing*

$$s(x) \ \text{ by } \ \hat{s}(x) = \sum_{\ell=n}^{m} s_\ell x^{\ell-(2n-m)} \ \text{ and } \ t(x) \ \text{ by } \ \hat{t}(x) = \sum_{\ell=2n-m}^{n} t_\ell x^{\ell-(2n-m)}$$

*leaves* $q(x)$ *invariant* ($\deg \hat{s} = 2 \deg \hat{t} = 2(m-n)$), *so that we may assume* $m \geq 2n$, *and thus* $m = \mathbf{O}(K)$.

## 2.3 Estimating the Approximation Error.

**Proposition 2.3.1.** *Let*

$$H^n > 2\sum_{\ell=0}^{n-1} |t_\ell| H^\ell \ . \qquad (2.3.1)$$

*Then*

$$|q_j^* - q_j| \le \frac{2H^{-j}}{H-1} \max_{0\le \ell<n} |r_\ell| \ . \qquad (2.3.2)$$

**Proof.** $[Q_j^*(H) - q_j H^j]^T$ is the image of the inverse $DFT(K)$ applied to $[r(H\omega^i)/t(H\omega^i)]^T$, thus

$$\max_{0\le j\le m-n} |Q_j^*(H) - q_j H^j| \le \frac{1}{K}\sum_{i=0}^{K-1} \left|\frac{r(H\omega^i)}{t(H\omega^i)}\right| \le \frac{1}{K}\sum_{i=0}^{K-1} \frac{|r(H\omega^i)|}{\min_{0\le j<K}|t(H\omega^j)|} \le \sum_{\ell=0}^{n-1} |r_\ell| \frac{H^\ell}{\min_{0\le j<K}|t(H\omega^j)|} \ .$$

Since $t_n = 1$,

$$|t(H\omega^j)| \ge H^n - |\sum_{\ell=0}^{n-1} t_\ell H^\ell \omega^{j\ell}| \ge H^n - \sum_{\ell=0}^{n-1} |t_\ell| H^\ell \ .$$

Thus from ( 2.3.1 ), $|t(H\omega^j)| \ge \frac{1}{2}H^n$. Therefore,

$$\min_{0\le j<K} |t(H\omega^j)| \ge \frac{1}{2}H^n \ . \qquad (2.3.3)$$

Additionally,

$$\sum_{\ell=0}^{n-1} |r_\ell| H^\ell \le \left(\max_{0\le \ell<n} |r_\ell|\right) \sum_{\ell=0}^{n-1} H^\ell = \max_{0\le \ell<n} |r_\ell| \frac{H^n - 1}{H-1} \ .$$

Therefore,

$$\max_{0\le j\le m-n} |Q_j^*(H) - q_j H^j| \le \frac{2}{H-1} \max_{0\le \ell<n} |r_\ell| \ ,$$

and since $Q_j^*(H) = q_j^* H^j$, the proposition follows. □

**Remark 2.3.1.** *$H \ge 3\max_{0\le \ell<n} |t_\ell|^{1/(n-\ell)}$, for example, will satisfy inequality ( 2.3.1 ).*

**Remark 2.3.2.** *[BP86] provides the bound*

$$\begin{aligned} \sum_{j=0}^{m-n} |q_j| &\le (1+t')^{m-n} \sum_{i=n}^{m} |s_i| \ , \\ t' &= \max\{|t_n|, |t_{n-1}|, \ldots, |t_{2n-m}|\} \ , (t_g = 0 \text{ if } g < 0) \ , \end{aligned}$$

*which leads to the estimate*

$$\max_{0\le \ell<n} |r_\ell| \le \max_{0\le g<n} |t_g| \sum_{g=0}^{m-n} |q_g| + \max_{0\le i<n} |s_i| \ ,$$

*and thus we obtain*

$$|q_j^* - q_j| \le \frac{2H^{-j}}{H-1}\left(\max_{0\le \ell\le n} |t_\ell|\left((1+t')^{m-n}\sum_{\ell=n}^{m}|s_\ell|\right) + \max_{0\le \ell\le m}|s_\ell|\right) \ , j=0,\ldots,m-n \ . \tag{2.3.4}$$

## 2.4 Estimating the Computation Error.

Provided that Algorithm 2.2.2 has been applied with a sufficiently high floating point precision, the lemma and the propositions in this section show that the norm of the output error vector is negligible.

Recall that for a vector $v=[v_\ell]$, we know $\|v\|_1 = \sum_\ell |v_\ell| \ge \|v\|_2 = \sqrt{\sum_\ell |v_\ell|^2}$. Denote the floating point binary representation of a number $x$ truncated to $d$ bits by $\mathrm{fl}(x)=\tilde{x}$, and write $\mathrm{fl}(v)=[\mathrm{fl}(v_\ell)]$.

**Proposition 2.4.1** *(compare [Atk]). For a given pair of d-bit binary values $x$ and $y$, we have*

$$\mathrm{fl}(x\circ y) = (x\circ y)(1+\delta),\ |\delta|<2^{-d} \ ,$$

*where $\circ$ denotes any arithmetic operation, $+$, $-$, $*$ or $\div$.*

In the next proposition, and throughout this section, $\phi(K)=8.5K^2\log K$.

**Proposition 2.4.2.** *[GS]. Let $d$ and $k$ be positive integers, $K=2^k$, let $v$ and $z$ be a pair of vectors such that $v$ is the forward DFT of $z$ or the inverse DFT of $Kz$, computed by using FFT with the precision of $d$ floating point bits. Then $\mathrm{fl}(v)=v+\mathcal{F}(z)$, where $\|\mathcal{F}(z)\|_2 \le 2^{-d}\phi(K)\|z\|_2$.*

In the proofs below, the following notation is used :

- define vectors $u = [u_i]^T$ and $v = [v_i]^T$, where $u_i = S(H, \omega^i)$ and $v_i = T(H, \omega^i)$ (refer to Algorithm 2.2.2, step (a), for definition of $S$ and $T$).

- for any integer $\ell$, define a diagonal matrix $D_{H,\ell} = \text{diag}\ (1, H, H^2, H^3, \ldots, H^\ell)$, so that $u$ is the forward $DFT(K)$ of $D_{H,m}s$ and $v$ is the forward $DFT(K)$ of $D_{H,n}t$.

- $\sigma = \mathcal{F}(D_{H,m}s) = [\sigma_i]^T$ and $\tau = \mathcal{F}(D_{H,n}t) = [\tau_i]^T$, where $\mathcal{F}$ is defined in Proposition 2.4.2.

- $\tilde{u} = u + \sigma$ and $\tilde{v} = v + \tau$, so that $\tilde{u}$ and $\tilde{v}$ are the approximations to the vectors $u$ and $v$, computed with $d$-bit precision by Algorithm 2.2.2.

- $p = [p_i]^T$, with $p_i = u_i/v_i$, $\xi = [\xi_i]^T$, with $\xi_i = \text{fl}(\tilde{u}_i/\tilde{v}_i) - p_i$, and $\tilde{p} = p + \xi$.

Let $q^*$ and $\tilde{q}^*$ denote the coefficient vectors of the output polynomial of Algorithm 2.2.2 performed with infinite precision and with $d$-bit precision, respectively. Then (for $\bar{\Omega}$ defined in Section 1.2), we have

$$q^* = \frac{D_{H,K-1}^{-1}\bar{\Omega}p}{K}\ , \quad \tilde{q}^* = \frac{D_{H,K-1}^{-1}\left(\bar{\Omega}\tilde{p} + \mathcal{F}(\tilde{p})\right)}{K}\ ,$$

and the computation error vector $e = \tilde{q}^* - q^*$ can be represented as:

$$e = \frac{D_{H,K-1}^{-1}\left(\bar{\Omega}\xi + \mathcal{F}(\tilde{p})\right)}{K}\ . \qquad (\ 2.4.1\ )$$

Deduce from Proposition 2.4.1 that

$$\xi_i = \frac{(u_i + \sigma_i)\delta_i + \sigma_i}{v_i + \tau_i} - \frac{u_i\tau_i}{v_i(v_i + \tau_i)}\ , \quad |\delta_i| < 2^{-d}. \qquad (\ 2.4.2\ )$$

Assume for the rest of this section that $H$ satisfies ( 2.3.1 ) and

$$d > \log(6\phi(K))\ . \qquad (\ 2.4.3\ )$$

**Lemma 2.4.1.**

a) $\|u\|_2 = \sqrt{K}\|D_{H,m}s\|_2 \leq \sqrt{K}H^m\|s\|_2$,

**b)** $\|D_{H,n}t\|_2 < \frac{3}{2}H^n$,

**c)** $|v_i| \geq \frac{1}{2}H^n$,

**d)** $|\tilde{v}_i| > \frac{1}{4}H^n$.

**Proof.** a) Follows from the definition of $u$ and $D_{H,m}$.

b) $\|D_{H,n}t\|_2 \leq \|D_{H,n}t\|_1 = H^n + \sum_{\ell=0}^{n-1} |t_\ell| H^\ell$ and ( 2.3.1 ) suffice.

c) A consequence of ( 2.3.2 ).

d) Observe that $|\tau_i| \leq \|\tau\|_2 = \|\mathcal{F}(D_{H,n}t)\|_2$, then apply Proposition 2.4.2 and obtain $|\tau_i| \leq 2^{-d}\phi(K)\|D_{H,n}t\|_2$. Substitute part b) and obtain $|\tau_i| \leq 2^{-d-2}6\phi(K)H^n$. Finally, combine the latter inequality with ( 2.4.3 ) rewritten as $2^d > 6\phi(K)$, and apply part c). □

**Proposition 2.4.3.** *Let ( 2.3.1 ) and ( 2.4.3 ) hold. Then*

$$\|\xi\|_2 \leq 2^{-d}H^{K-1}\|s\|_2 \left(4\sqrt{K} + (5 + 12\sqrt{K})\phi(K)\right).$$

**Proof.** Rewrite ( 2.4.2 ) as

$$\xi_i = \frac{\delta_i u_i + (1+\delta_i)\sigma_i}{\tilde{v}_i} - \frac{u_i\tau_i}{v_i\tilde{v}_i}.$$

Combine parts c) and d) of Lemma 2.4.1 and obtain

$$\|\xi\|_2 \leq 2^{2-d}H^{-n}\|u\|_2 + 4H^{-n}(1+2^{-d})\|\sigma\|_2 + 8H^{-2n}\|u\|_2\ \|\tau\|_2\ .$$

Apply Propostion 2.4.2 and arrive at the bound

$$\|\xi\|_2 \leq H^{-n}2^{2-d}(\|u\|_2 + (1+2^{-d})\phi(K)\|D_{H,m}s\|_2 + 2H^{-n}\phi(K)\|u\|_2\|D_{H,n}t\|_2)\ .$$

Apply parts a) and b) of Lemma 2.4.1 a) and deduce

$$\|\xi\|_2 \leq 2^{2-d}H^{m-n}\|s\|_2 \left(\sqrt{K} + (1 + 2^{-d} + 3\sqrt{K})\phi(K)\right).$$

The proposition follows since $2^{2-d} < 1$, that is, $4(1+2^{-d}) < 5$ under ( 2.4.3 ). □

**Proposition 2.4.4.** *Let ( 2.3.1 ) and ( 2.4.3 ) hold. Then*

$$\|e\|_2 \le 2^{-d} H^{K-1} \|s\|_2 \, (5 + 22\phi(K)) \ .$$

**Proof.** Recall ( 2.4.1 ) and deduce that $K\|e\|_2 \le \|\bar{\Omega}\xi + \mathcal{F}(\tilde{p})\|_2 \le \sqrt{K}\|\xi\|_2 + \|\mathcal{F}(\tilde{p})\|_2$ . Next apply Proposition 2.4.2, substitute the vector equation $\tilde{p} = p + \xi$, and deduce that

$$K\|e\|_2 \le \sqrt{K}\ \|\xi\|_2 + 2^{-d}\phi(K)\|\tilde{p}\|_2 \le \left(\sqrt{K} + 2^{-d}\phi(K)\right)\|\xi\|_2 + 2^{-d}\phi(K)\|p\|_2 \ .$$

Apply Proposition 2.4.3, substitute equation $p_i = u_i/v_i$, apply Lemma 2.4.1 c) and obtain

$$K\|e\|_2 \le \left(\sqrt{K} + 2^{-d}\phi(K)\right) 2^{-d} H^{K-1}\|s\|_2 \left(4\sqrt{K} + (5 + 12\sqrt{K})\phi(K)\right) + 2^{-d}\phi(K)(2H^{-n})\|u\|_2 \ .$$

Now apply Lemma 2.4.1 a) and deduce that

$$K\|e\|_2 \le 2^{-d} H^{K-1}\|s\|_2 \left(\left(\sqrt{K} + 2^{-d}\phi(K)\right)\left(4\sqrt{K} + (5 + 12\sqrt{K})\phi(K)\right) + 2\sqrt{K}\phi(K)\right) \ .$$

Divide both sides by $K$, apply bound $2^{-d} < 1/(6\,\phi(K))$ from ( 2.4.3 ) and complete the proof. □

**Corollary 2.4.1.** *If* $d > 2\log\left(H^{K-1}\|s\|_2\,(5 + 22\phi(K))\right)$ *then* $\|e\|_2 < 1/4$.

## 2.5 Boolean Complexity Estimates.

In this section, we will assume that $t(x)$ is monic and the polynomials $s(x)$ and $t(x)$ have integer coefficients. Therefore, the quotient $q(x)$ is a polynomial with integer coefficients.

We also know that if

$$|\tilde{q}_j^* - q_j| < 1/2 \text{ for } j = 0, \ldots, m - n, \qquad (\ 2.5.1\ )$$

we may recover the integer coefficients $q_j$ from $\tilde{q}_j^*$ by means of rounding-off.

Due to Corollary 2.4.1, ( 2.5.1 ) holds if we have $|q_j^* - q_j| < 1/4$. To ensure this latter bound, we need to decrease the right sides of the equations ( 2.3.2 ) and ( 2.3.4 ) below $1/4$. This is immediately achieved for large $j$. Thus, we may decrease the output error, if

we apply our algorithm to the polynomials $x^b s(x)$ and $t(x)$ for $b \geq m-n+1$. Then, the original quotient $q(x)$ will turn into $\tilde{q}(x) = x^b q(x) + \mathbf{O}(x^{b-1})$, and we will only need to use the output approximations to the $m-n+1$ leading coefficients of this new quotient $\tilde{q}(x)$, whose approximation errors are bounded according to ( 2.3.2 ) for $j \geq b$.

In particular, let $b = m-n+1$, let $H$ satisfy ( 2.3.1 ), and let

$$H^b > 2a \max_{\ell} r_\ell \, . \qquad (\ 2.5.2\ )$$

Then,

$$|q_j^* - q_j| < \frac{1}{a(H-1)} \text{ for all } j > b \, ,$$

due to the bound ( 2.5.2 ).

Therefore, for $H > 2$ and $a \geq 4$ we have $|q_j^* - q_j| < \frac{1}{4}$, for all $j > b$, and then the exact values of the $q_j$ can be recovered by rouding-off $\tilde{q}_j^*$ to the closest integers.

Now rewrite ( 2.5.2 ) so as to express the right-hand side by using only the values of the coefficients of $s(x)$ and $t(x)$.

Due to Remark 2.3.2, for $b = m-n+1$, we may choose any $H$ satisfying

$$H > \left[ 2a \left( (1+t')^{m-n} \sum_{i=n}^{m} |s_i| \max_{0 \leq \ell \leq n} |t_\ell| \right) + \max_{0 \leq \ell \leq m} |s_\ell| \right]^{1/b} . \qquad (\ 2.5.3\ )$$

Due to Remark 2.3.1 and ( 2.5.3 ), we may choose any $H$ exceeding the maximum value of the right sides of the inequalities ( 2.5.2 ) and ( 2.5.3 ); then we call Algorithm 2.2.2. The rouding-off of the output values to the closest integers will give us the exact coefficients of $q(x)$.

Also, from Corollary 2.4.1 we have

$$d = \mathbf{O}(K \log H + \log \|s\|_2) \, . \qquad (\ 2.5.4\ )$$

Assume that the Boolean complexity of an arithmetic operation performed with 2 integers modulo $2^d$ is $\mathbf{O}(\log d, P(d))$. THerefore, we will have the Boolean estimate of our algorithm of order $\mathbf{O}(\log m \log d, mP(d))$ where $d$ satisfied ( 2.5.4 ).

Furthermore, we may subsitute the value of $H$ in ( 2.5.4 ) [which can be obatained from ( 2.5.3 )]. We notice that our algorithm supports the same Boolean complexity estimates as those of [B84, Sc82a].

# Chapter 3

# Multipoint Polynomial Evaluation and Interpolation

**Summary**

The known record sequential time-complexity estimates for the problems of polynomial evaluation and multipoint interpolationthe algorithms that are devastatingly unstable numerically because of their recursive use of polynomial division. By applying a distinct approach to computing approximate solutions to both problems, we arrive at algorithms that are faster or equally fast and have a promise of improving numerical stability. Our approach relies on new techniques, so far not used in this area: we reduce the problems to Vandermonde matrix computations and then exploit some recent methods for improving computations with structured matrices.

## 3.1 Introduction.

Multipoint evaluation and interpolation of a polynomial are 2 major problems of numerical and algebraic computing. They are converse to each other and can be considered as 2 instances of a more general problem of converting one representation of a polynomial into another (compare [Gathen86]).

Formally, these 2 problems can be stated as follows:

**Problem 3.1.1**, *multipoint evaluation.*

**Input**: *The coefficients of a polynomial*

$$p(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_n x^n \ ,$$

*and a set of points* $\{u_i : \ i = 0, \ldots, m\}$.

**Output**: *The values of* $p(x)$ *at the points* $u_i$ *for* $i = 0, \ldots, m$ *(i.e. the values* $v_i = p(u_i)$ *for* $i = 0, \ldots, m$*).*

**Problem 3.1.2**, *interpolation.*

**Input**: *Two sets of scalars:*

**i)** $\{u_i : \ i = 0, \ldots, n; \ u_i \neq u_j$ for $i \neq j\}$ *(nodes of interpolation),*

**ii)** $\{v_i : \ i = 0, \ldots, n\}$ *(values at the nodes).*

**Output**: *The coefficients* $p_0, p_1, \ldots, p_n$ *of the polynomial* $p(x)$ *such that*

$$p(u_i) = v_i \ , \quad i = 0, 1, \ldots, n \ .$$

Let us recall some known algorithms and computational estimates for these 2 problems.

At first, consider the evaluation of a polynomial $p(x)$ of a degree $n$ at a single point.

Horner's method solves this problem by using $n$ multiplications and $n$ additions, which both are optimal bounds [AHU], [BM], [Kn], unless one allows to precondition the coefficients of $p(x)$. Even with preconditioning, however, one may only save less than $n/2$ multiplications.

We may solve Problem 1 by applying Horner's method $m$ times. This will involve $mn$ multiplications and as many additions, but much faster algorithms can be devised

based on the recursive use of fast polynomial division. These methods yield the solution by using $\mathbf{O}((m+n)(\log(m+n)^2)$ arithmetic operations [AHU], [BM], [BP94].

Likewise, Problem 2, of interpolation, can be solved by means of classical methods by using order $n^2$ arithmetic operations [CdB], [Wer84], but more recent techniques reduce the problem to a sequence of polynomial multiplications and divisions so as to yield the solution by using $\mathbf{O}(n(\log n)^2)$ arithmetic operations [AHU], [BM], [BP94].

There are also lower bounds of order $\Omega(n \log n)$ for Problems 1 (for $m \geq n$) and 2 [BM], [Ben-Or83], [PrepSham].

All these results are about evaluation of the exact solution of Problems 1 and 2.

It is also known that these fast algorithms, running in time $\mathbf{O}((m+n)(\log(m+n))^2)$ for Problem 1 and $\mathbf{O}(n(\log n)^2)$ for Problem 2, are not convenient for their numerical implementation, due to the recursive use of polynomial divisions, which recursively magnifies the approximation errors, so that for many input instances the output can be completely corrupted unless the computations are performed symbolically, with no errors, or numerically, with an extremely high precision (thus, restricting their application to the case of computers performing exact rational arithmetic).

This leads to an important and challenging problem of numerical multipoint evaluation and interpolation by using $\mathbf{o}((m+n)^2)$ and $\mathbf{o}(n^2)$ arithmetic operations, respectively.

In particular, Fast Fourier transform (FFT) enables us to solve both of these problems in $\mathbf{O}(n \log n)$ operations [AHU] with no numerical stability problems [GS], in the special case where all the $n$ nodes are $N$-th roots of unity, for $N > \max\{m,n\}$. Similarly, in some other special cases [P89a], where the input nodes are the Chebyshev points, the estimates are $\mathbf{O}((m + n \log n) \log n)$ or $\mathbf{O}((n \log n) \log m)$ for evaluation and $\mathbf{O}(n \log^2 n)$ interpolation with improved numerical stability.

[Rok88] applies approximation theory to devise desired approximation algorithms for Problem 1 (multipoint evaluation) with the time-complexity estimate of $\mathbf{O}((m + n) \log(1/\epsilon) + m(\log(1/\epsilon))^3)$, but only in the case where all nodes $x$ for the evaluation are from a fixed real line interval. [P95a] improves the latter bound to $\mathbf{O}(m \log^2 u +$

$n \min(u, \log n))$, $u = \log(1/\epsilon)$, but still under the assumption of confining all nodes $x$ to a fixed real line interval.

However, for a general set of nodes on the complex plane, the order of $n^2$ operations is required in all the known numerically stable solutions of Problems 1 and 2.

In this chapter, we propose a distinct approach which leads us to desired fast algorithms for approximate solution of Problems 1 and 2 for a large class of complex inputs. The algorithms compute solutions to both problems within a given tolerance $\kappa\,\epsilon$ to the error, $\kappa$ denoting the condition number of some auxiliary computational problem [see ( 3.4.1 ) in Section 3.4].

Both Problems 1 and 2 are restated in the equivalent form of operations with Vandermonde matrices, and the solution is obtained based on some known properties of these matrices and related structured matrices; this approach may be of independent technical interest.

The algoirhtm involves Toeplitz type linear systems, which for many inputs may still cause some numerical stability problems [Bu85], not as devastating, however, as ones caused by recursive polynomial divisions. In particular, we may shift to symmetrized systems which are still of Toeplitz type, and if they remain sufficiently well-conditioned after their symmetrization, then numerical stability problems are avoided [Bu85].

Our algorithms have a common feature of many approximation algorithms: their output errors and their running time decrease with the condition number of the auxiliary linear system (in our case of Toeplitz type) to which we reduce the solution. Furthermore, we may try to use a random transformation of some input parameters in order to improve the condition of the latter linear system (see Remark 3.3.1 and the derivation of the complexity estimates based on our first algorithm of Section 3.4).

It is known that the Vandermonde matrix (defining the interpolation problem) is ill-conditioned for a very large class of sets of interpolation nodes, and the parameter $\kappa$ of ( 3.4.1 ), defining the approximation error of our computations, tends to be large. Thus we cannot, as of now, recommend our algorithms for practical computations, except for

the special cases of node sets defining well-conditioned Vandermonde matrices.

From the theoretical point of view, however, the new algorithms may be of interest since they demonstrate some previously hidden correlations between computations with polynomials and with structured matrices. To be more specific, we represent the original computational problems of polynomial evaluation and interpolation in the form of operations with a Vandermonde matrix. Then we apply the techniques of [P90a] for computations with structured matrices and reduce the original problem with any set of nodes to the case of roots of unity as the nodes; in the latter case FFT applies. The reduction involves Toeplitz type computations (with matrices having displacement rank at most 3) and a single multiplication of a generalized Hilbert matrix by a vector, at which stage we apply the fast approximation algorithm of [Rok85], known to be effective and reliable (for large input class) in numerous computations, in particular, for integral equations and $n$-body mechanics. We present and analyze our algorithms in Section 3.4 after some preliminaries in Sections 3.2 (definitions) and 3.3 (auxiliary results).

The resulting algorithms yield desired approximate solutions by using $\mathbf{O}(((\log n)^2 + \log(\kappa/\varepsilon))n)$ arithmetic operations assuming that $m = n$ for problem 1 and that $\kappa$ denotes the condition number of an auxiliary structured matrix (see Appendix C represented by [PSLT]). No recursive polynomial division is involved, and the algorithms promise to be efficient enough for numerical solution of Problems 1 and 2. If $\log(\kappa/\varepsilon) = \mathbf{o}(n)$, the algorithms run in time $\mathbf{o}(n^2)$. Moreover, if $\log(\kappa/\varepsilon) = \mathbf{o}((\log n)^2)$, say, then the solution time is $\mathbf{O}(n(\log n)^2)$.

On the other hand, the proposed direction, exploiting the known properties of associated structured matrices, promises some further improvement and seems to deserve further exploration.

## 3.2 Definitions.

Let $n$ be a positive integer, and $i$, $j$, $k$ integer parameters, ranging from 0 to $n-1$. Matrix rows and vector components are represented by $i$, columns by $j$. $W^T$ is the transpose of a matrix (vector) $W$; $W^H$ is the Hermitian transpose of $W$.

Complex vectors $\mathbf{e}$, $\mathbf{f}$, $\mathbf{g}$, $\mathbf{r}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{p}$, $\mathbf{x}$ and $\mathbf{y}$ are of the form $\mathbf{h} = [h_0, \ldots, h_{n-1}]^T$ (where $\mathbf{h}$ can represent any listed vector). $r_i = \omega^i$, where $\omega = \exp(2\pi\sqrt{-1}/n)$, is a primitive $n^{th}$ root of 1 (a Fourier point). Let the components $u_i$ of $\mathbf{u}$ be pairwise distinct and not equal to integer powers of $r$.

Matrices $I$, $J$, $V$, $H$, $T$ and $Z$ are of size $n \times n$.

The matrices

$$J = \begin{bmatrix} 0 & & & 1 \\ & & \cdot & \\ & \cdot & & \\ \cdot & & & \\ 1 & & & 0 \end{bmatrix} \text{ (reversion) and } Z = \begin{bmatrix} 0 & & & & 0 \\ 1 & \cdot & & & \\ & \cdot & \cdot & & \\ & & \cdot & \cdot & \\ 0 & & & 1 & 0 \end{bmatrix} \text{ (displacement)}$$

satisfy $J^2 = I$ (the identity matrix), $J\mathbf{u} = [u_{n-1}, \ldots, u_0]^T$, $Z\mathbf{u} = [0, u_0, \ldots, u_{n-2}]^T$.

$V(\mathbf{u}) = [u_i^j]$ is a Vandermonde matrix. $V(\mathbf{r}) = [r_i^j] = [\omega^{ij}]$ is the matrix of discrete Fourier transform (DFT) on $n$ points, so that $V(\mathbf{r})\mathbf{u} = \left[\sum_{i=0}^{n-1} r_i^j u_i\right]^T$ is the DFT of a vector $\mathbf{u}$.

$H(\mathbf{u}, \mathbf{v}) = [1/(u_i - v_j)]$ is a Cauchy (generalized Hilbert) matrix ($u_i \neq v_j$ for all $i$ and $j$).

$T = [t_{n-1+i-j}]$ is a Toeplitz matrix.

$$TJ = [t_{i+j}], \quad JT = [t_{2n-2-i-j}], \qquad (3.2.1)$$

are Hankel matrices.

[Toeplitz (Hankel) matrices have entries that are invariant under their shifts in the diagonal (antidiagonal) direction.]

## 3.3 Auxiliary Results.

We recall the following simple and/or well-known results (which we state by using definitions of Section 3.2):

$$H(\mathbf{u},\mathbf{v}) = -H^T(\mathbf{v},\mathbf{u}), \qquad (3.3.1)$$

$$V(\mathbf{r}) = V^T(\mathbf{r}), \quad V^H(\mathbf{r})\,V(\mathbf{r}) = n\,I. \qquad (3.3.2)$$

**Fact 3.3.1.** *The values of a polynomial* $p(x) = \sum_{i=0}^{n-1} p_i x^i$ *(with coefficient vector* $\mathbf{p}$*) on the set of points* $u_0, \ldots, u_{n-1}$ *are given by the vector*

$$\mathbf{v} = V(\mathbf{u})\,\mathbf{p}. \qquad (3.3.3)$$

Fact 3.3.1 defines the problem of interpolation and multipoint evaluation of a polynomial $p(x)$ in terms of a vector equation (compare Problems 3.4.1 and 3.4.2).

**Remark 3.3.1.** *It is simple to shift from polynomial* $p(x)$ *to* $t(x) = p(a\,x+b)$ *for any fixed complex numbers* $a$ *and* $b$*, and vice versa, at the cost of performing* $\mathbf{O}(n \log n)$ *operations [ASU]. Even simpler is the transition to the reverse polynomial* $p_{\mathrm{rev}}(x) = x^n\,p(1/x)$*. These transformations enable us to vary (to our convenience) the input matrix* $V(\mathbf{u})$ *of the problem of polynomial interpolation and multipoint evaluation. For evaluation, we may also vary the input node set, for instance, by partitioning it into two or several subsets, complementing each subset to* $n$ *points at our choice and solving two or several evaluation problems.*

**Fact 3.3.2** *[AHU]. Given vectors* $\mathbf{u}$ *and* $\mathbf{r}$*, it suffices to use* $\mathbf{O}(n \log n)$ *arithmetic operations to compute* $V(\mathbf{r})\,\mathbf{u}$ *and* $V^H(\mathbf{r})\,\mathbf{u}$ *(that is, to perform the forward and inverse DFT of a vector* $\mathbf{u}$*).*

**Fact 3.3.3.**

**a)** *[AHU] Given a vector* $\mathbf{v}$ *and a Toeplitz matrix* $T$*, it suffices to use* $\mathbf{O}(n \log n)$ *arithmetic operations to compute the vector* $T\,\mathbf{v}$*.*

b) *[AG, BA, BGY, Ho87, Mu] Furthermore,* $\mathbf{O}(n \log^2 n)$ *arithmetic operations suffice to compute* $T^{-1}\mathbf{v}$ *if* $T$ *is nonsingular.*

**Fact 3.3.4.** *The estimates of the previous fact hold even if the matrix* $T$ *is replaced by any matrix of the form* $V^T(\mathbf{u})\,V(\mathbf{u})$, $W(\mathbf{u},\mathbf{r}) = V^T(\mathbf{r})\,H(\mathbf{r},\mathbf{u})\,V(\mathbf{u})$ *or* $W^T(\mathbf{u},\mathbf{r})$.

**Proof** The extension of Fact 3.3.3 to the matrix $V^T(\mathbf{u})\,V(\mathbf{u}) = \left[\sum_{k=0}^{n-1} u_k^{i+j}\right]_{i,j}$ follows since this is a Hankel matrix [compare ( 3.2.1 )]. The extension of Fact 3.3.3 to $W(\mathbf{u},\mathbf{r})$ and $W^T(\mathbf{u},\mathbf{r})$ follows from [P90]. (Specifically, Proposition 6.1 of [P90] implies that $W(\mathbf{u},\mathbf{r})$, $W^T(\mathbf{u},\mathbf{r})$ are Toeplitz type matrices defined with their $n \times 3$ displacement generator matrices (see definitions in [P90, ChKLe]), and to such matrices both parts of Fact 3.3.3 can be extended [see [ChKLe] and/or [Mu] on the extension of part a) and [BA] or [Mu] on the extension of part b)]). □

**Proposition 3.3.1** *[Rok85]. Given a natural* $n$, *positive* $a$, $q$, $s$, *and* $\epsilon$ *and three complex vectors* $\mathbf{u}$, $\mathbf{v}$ *and* $\mathbf{y}$ *of dimension* $n$, *it suffices to use* $(5n-1)\,s$ *arithmetic operations to compute, within the error bound* $\epsilon$, *the values* $(H(\mathbf{v},\mathbf{u})\,\mathbf{y})_i = \sum_{k=0}^{n-1} y_k/(v_i - u_k)$, *for* $i = 0, 1, \ldots, n-1$, *provided that*

$$s \geq \frac{\log(a\,n) - \log((1-q)\,\epsilon)}{\log(1/q)}, \qquad (3.3.4)$$

$$a \geq \left|\frac{y_k}{u_k}\right|, \quad 1 > q > \left|\frac{v_i}{u_k}\right|, \quad \text{for all } i \text{ and } k. \qquad (3.3.5)$$

**Proof** (see appendix A, section A.1). □

**Remark 3.3.2.**

a) *If, say,* $q < 1/2$, $\log a = \mathbf{O}(\log n)$, $\log(1/\epsilon) = \mathbf{O}(\log n)$, *then ( 3.3.4 ) can be satisfied for* $s = \mathbf{O}(\log n)$.

b) *In our algorithms of the next section,* $\mathbf{v} = \mathbf{r}$, *and the vector* $\mathbf{u}$ *can be linearly transformed, according to Remark 3.3.1. This enables us to insure the last inequality of ( 3.3.5 ) for* $q = 1/2$ *and for all* $i$ *and* $k$.

## 3.4 Interpolation and Multipoint Evaluation.

**Algorithm 3.4.1. Interpolation.**

**Input:** *vectors* $\mathbf{u}$ *and* $\mathbf{v}$.

**Output:** *vector* $\mathbf{p}$ *satisfying ( 3.3.3 ).*

*Note that, by assumption about vectors* $\mathbf{u}$ *and* $\mathbf{r}$ *(in Section 3.2),* $V(\mathbf{u})$ *is non-singular, and* $H(\mathbf{r}, \mathbf{u})$ *can be defined. Assume (until the end of this section) that* $H(\mathbf{r}, \mathbf{u})$ *is non-singular.*

**Computations.** *Successively compute the three vectors:*

1. $\mathbf{f} = H(\mathbf{r}, \mathbf{u})\,\mathbf{v}$,

2. $\mathbf{g} = V^T(\mathbf{r})\,\mathbf{f} = V(\mathbf{r})\,\mathbf{f}$ *[see ( 3.3.2 )]*,

3. $\mathbf{p} = W^{-1}(\mathbf{u}, \mathbf{r})\,\mathbf{g}$.

The *correctness* of this algorithm follows since, for the computed vectors $\mathbf{p}$ and $\mathbf{g}$, we have:

$$W(\mathbf{u}, \mathbf{r})\,\mathbf{p} = W(\mathbf{u}, \mathbf{r})\,W^{-1}(\mathbf{u}, \mathbf{r})\,\mathbf{g} = \mathbf{g} = V^T(\mathbf{r})\,H(\mathbf{r}, \mathbf{u})\,\mathbf{v},$$

and ( 3.3.3 ) follows since $V(\mathbf{r})$ and $H(\mathbf{r}, \mathbf{u})$ are nonsingular.

We apply the algorithm for approximate evaluation of $\mathbf{p}$, so as to decrease the estimated *computational complexity*:

At Stage 1, approximating the components of $\mathbf{f}$ (within $\epsilon > 0$ ) requires $(5n-1)\,s$ operations, for $s$ defined by ( 3.3.4 );

Stage 2 requires $\mathbf{O}(n \log n)$ operations (see Fact 3.3.2);

Finally, at Stage 3, we need to compute the matrix $W(\mathbf{u}, \mathbf{r})$, of Toeplitz type, or more precisely, to compute its displacement generator of length (at most) 3. For this, we just need to compute the products $\mathbf{p}(\ell) = \left(W(\mathbf{u}, \mathbf{r}) - Z\,W(\mathbf{u}, \mathbf{r})\,Z^T\right)\mathbf{v}(\ell)$, $\ell = 1, 2, 3$, for three general vectors $\mathbf{v}(1)$, $\mathbf{v}(2)$, $\mathbf{v}(3)$.

Moreover, we may choose these vectors in the form
$\mathbf{v}(\ell) = \mathbf{b}(\ell) = [1, b(\ell), (b(\ell))^2, \ldots, (b(\ell))^{n-1}]^T$, where $b(\ell)$ is a random parameter, $\ell =$

1,2,3. [Indeed, the $n \times n$ matrix $B = [(b(i))^j]$ has rank $n$, if all the $b(i)$ are distinct numbers, $i, j = 0, 1, \ldots, n-1$ ; therefore, with a high probability (see [DeL], [Schw] and [Z79]), for a random choice of $b(i)$, the matrix $\left(W(\mathbf{u},\mathbf{r}) - Z\,W(\mathbf{u},\mathbf{r})\,Z^T\right) B$ has a rank at most 3, and so any of its random $n \times 3$ submatrices has a rank at most 3]. The vector $V(\mathbf{u})\,\mathbf{b}(\ell)$, $\ell = 1,2,3$, can be computed in $\mathbf{O}(n \log n)$ operations, since the evaluation of $V(\mathbf{u})\,\mathbf{b}(\ell)$ amounts to evaluation of the polynomial $[1 - (b(\ell)x)^n]/[1 - b(\ell)\,x] = \sum_{i=0}^{n-1}(b(\ell)\,x)^i$ at the points $u_0, \ldots, u_{n-1}$.

Therefore, the complexity of approximate evaluation of the vectors $\mathbf{p}(\ell)$, for $\ell = 1,2,3$, is still within the bounds of $\mathbf{O}(n\,(s + \log n))$, for $s$ defined by ( 3.3.4 ), provided that $\epsilon$ denotes the tolerance to the errors of the appproximate multiplications of $H(\mathbf{r},\mathbf{u})$ by vectors in the process of the evaluation of $\mathbf{p}(\ell)$.

Given $\mathbf{p}(\ell)$, $\ell = 1,2,3$, we can, by using the algorithms of Facts 3.3.3 or 3.3.4, find $W^{-1}\mathbf{g}$ at the cost $\mathbf{O}(n \log^2 n)$. Therefore, Stage 3 can be done at the cost of $\mathbf{O}(n\,(s + log^2 n))$, $s$ defined by ( 3.3.4 ).

The bound $\epsilon$ on the approximation errors of all the multiplications of vectors by the matrix $H(\mathbf{u},\mathbf{r})$ is not substantially magnified in the subsequent multiplications of the resulting vectors by the matrix $V(\mathbf{r})$ (having 2-norm equal to 1) but may be substantially increased in the evaluation of $W^{-1}\mathbf{g}$ unless

$$\kappa = \mathrm{cond}\, W \qquad (\ 3.4.1\ )$$

is small.

**Algorithm 3.4.2. Evaluation.**

**Input:** *vectors* $\mathbf{u}$ *and* $\mathbf{p}$.

**Output:** *vector* $\mathbf{v}$ *satisfying ( 3.3.3 ).*

**Computations.** *Successively compute*

*1.* $U(\mathbf{u}) = V^T(\mathbf{u})\,V(\mathbf{u})$,

*2.* $\mathbf{e} = U(\mathbf{u})\,\mathbf{p}$,

3. $\mathbf{x} = W^T(\mathbf{u}, \mathbf{r})^{-1}\,\mathbf{e}$,

4. $\mathbf{y} = V(\mathbf{r})\,\mathbf{x}$,

5. $\mathbf{v} = H^T(\mathbf{r}, \mathbf{u})\,\mathbf{y}$.

*Correctness.*

Premultiply both sides of ( 3.3.3 ) by $V^T(\mathbf{u})$ and obtain that

$$\mathbf{e} = V^T(\mathbf{u})\,V(\mathbf{u})\,\mathbf{p} = V^T(\mathbf{u})\,\mathbf{v}.$$

Then substitute $\mathbf{v} = H^T(\mathbf{r}, \mathbf{u})\,\mathbf{y} = H^T(\mathbf{r}, \mathbf{y})\,V(\mathbf{r})\,\mathbf{x}$, obtain that

$$W^T(\mathbf{u}, \mathbf{r})\,\mathbf{x} = \mathbf{e},$$

and thus verify the correctness of the above solution.

*Complexity*

Follow [CKL] in order to perform Stage 1 in $\mathbf{O}(n\,\log^2 n)$ operations.

Namely, first compute at this cost [AHU] the coefficients of the polynomial $\prod_{k=0}^{n}(u - u_k)$. Then use $\mathbf{O}(n\,\log n)$ operations in order to obtain the power sums $\sum_{k=0}^{n-1} u_k^s$ from the system of Newton's identities (see e.g., [P90], Appendix A).

We need $\mathbf{O}(n\,\log n)$ operations at Stage 2 and $\mathbf{O}(n\,(s+\log^2 n))$ at Stage 3, for $s$ defined in Proposition 3.3.1 [use Fact 3.3.4 and the algorithm for the evaluation of $W(\mathbf{u}, \mathbf{r})$ shown above], and we need $\mathbf{O}(n\,\log n)$ operations at Stage 4 (due to Fact 3.3.2).

At Stage 5, we approximate all the components of the vector $\mathbf{v}$ within the error bound $\epsilon$ at the cost $(5n - 1)\,s$, where $s$ is defined by Proposition 3.3.1.

# Chapter 4

# Polynomial Zeros

**Summary**

We propose an algorithm, for approximating polynomial zeros, relying on the combination of some modification of Weyl's geometric construction on the complex plane (which achieves the initial approximation to the zeros and/or their clusters as well their isolation from each other) and a variant of Newton's iteration for the refinement of the initial approximation. The algorithm has a complexity estimate of $\mathbf{O}(n^2 \log(bn) \log n \log\log n)$. The algorithm is simple for implementation and numerically stable.

## 4.1 Introduction.

The problem of approximating polynomial zeros is stated as follows:
Given the coefficients of a polynomial $p(x)$, where

$$\begin{aligned} p(x) &= p_0 + p_1 x + \ldots + p_{n-1}x^{n-1} + p_n x^n \\ &= p_n(x - z_1)(x - z_2)\ldots(x - z_n) \ , \end{aligned}$$

compute or approximate the zeros $z_1, z_2, \ldots, z_{n-1}, z_n$ of $p(x)$.

This problem has substantially influenced the development of mathematics throughout the centuries and has many important applications to the theory and practice of computing.

One of the early instances of this work was the resolution of the $2^{nd}$ degree equation, known already in the ancient Greece. The study of the special equation $x^2 + 1 = 0$ introduced the notion of the complex numbers.

Another important aspect of this problem is the fundamental theorem of algebra, which states the existence of the complex solutions . In 1799, C. F. Gauss provided the proof for the fundamental theorem of algebra; unfortunately, it had a substantial flaw, which was filled up by A. M. Ostrowski in 1920.

In $18^{th}$ and $19^{th}$ centuries, the focus was on finding a formula for the solution to the problem. This motivation had a very profound influence on mathematics because of the failure of all the attempts for a such formula for polynomials of degree greater than 4. A. Ruffini in 1813 and H. N. Abel in 1827, proved the theorem on the nonexistence of such formula. E. Galois has substantially extended this theorem developing his celebrated theory in 1833.

Due to the insolvability of the equation in closed form, the motivation was shifted to the iterative approach, which became most exciting with the development of computers. Hundreds of papers on this subject have appeared, and new ones still continue to appear.

Among all the numerous algorithms proposed for approximating complex polynomial zeros, the lowest computational cost estimates are yielded based on 2 approaches. Both of them combine some geometric constructions on the complex plane [in order to ensure a certain degree isolation of a zero $z$ or of a subset (cluster) $Z$ of the zeros of a given polynomial $p(x)$ from all its other zeros] with some analytic techniques (such as Newton's iteration and/or numerical integration based on FFT) applied in order either to achieve rapid approximation of the isolated zero $z$ or a cluster $Z$ of the zeros of $p(x)$ or to split $p(x)$ numerically into 2 factors, one of which is $x - z$ or $\prod_{z \in Z} (x - z)$, respectively. The latter splitting approach can be called divide-and-conquer method. It has been developped in several papers [Schr], [DL], [DH], [Sc82], [BFKT], [P89], [BT], [BP91], [Kir], [NR], [P95]. In particular, in [P95] optimal bounds of $n$ arithmetic operations and $bn^2$ Boolean operations have been reached (up to polylogarithmic factors) for approximating the $n$

zeros of $p(x)$ within absolute error bound $2^{-b}$, provided that $n = \mathbf{O}(b)$ and that the variable $x$ has been scaled so as to bring all the zeros of $p(x)$ inside the unit disc $D(0,1) = \{x: |x| \leq 1\}$.

The lower bound $n$ on the number of arithmetic operations immediately follows since the output consists of $n$ generally distinct values, but even for approximating a single zero of $p(x)$, $\lceil \frac{n+1}{2} \rceil$ arithmetic operations are necessary since the input depends on the $n+1$ generally distinct coefficients of $p(x)$ and since each arithmetic operation has only 2 operands.

The Boolean cost has a lower bound $0.25bn^2$ (of [P95]) because the input perturbation by adding $2^{-2b/n}$ to any of the $\lceil n/2 \rceil$ trailing coefficients of the polynomial $(x - 7/9)^n$ causes a perturbation of its single multiple zero by at least $2^{-b}$. It follows that one has to keep at least $bn/2$ bits in the representation of each of $n/2$ trailing coefficients of $p(x)$. Processing these coefficients will then require at least $0.25bn^2$ bit-operations (Boolean operations).

Even though the algoritms of [P95] are nearly optimal, for practical implementation other approaches and algorithms may still be interesting. In this chapter, we will study one of such approaches, which can be viewed as the second best approach, according to the associated complexity estimates, and which exploits and extends Weyl's celebrated construction of search and exclusion on the complex plane.

This construction, first proposed in [W24] for approximating polynomial zeros, has been effectively used in many other areas of computer science, under the name of quadtree algorithms (see [Ga82], [HS79], [Sa81], [Sa84], [Se94] on several application of the quadtree algorithms).

In [W24], H. Weyl presented his algorithm with a proof of its convergence for any input polynomial $p(x)$ but with no computational complexity estimates. [HG] presented an improved modification using $\mathbf{O}(n^3 b)$ arithmetic operations. Further modification yielded the bounds $\mathbf{O}(n^3 + n^2 \log b)$ [R87] and $\mathbf{O}(n^2 \log(bn) \log n)$ [P87] and [P94]. Though even the latter bound is inferior to one of [P95], the algorithms of [P87] and [P94] may turn out

to be more attractive for practical application since they use more primitive geometric costruction than one of [P95] (and, therefore, promise to decrease the overhead constant factors, hidden in the "O" complexity estimates) and these former constructions seem to be easier for coding. Besides, the analytic technique of Newton's iteration used in [P94] is also very simple for coding. In particular, this technique is substantially simpler to program than the sophisticated techniques used for splitting in the divide-and-conquer algorithms of [Sc82], [BFKT], [BT], [Kir], [NR] and [P95].

Our main objective in this chapter is the specification of the algorithm of [P94], its refinement [in particular, by replacing Turan's proximity test by an algorithm, whose asymptotic time bound may be slightly larger (by $\log\log n$ factor) but which is more numerically stable], and supplying some omitted details.

In our future work, we plan a series of numerical experiments for this algorithm.

The algorithms presented in this chapter can be implemented on $p$ processors, with their acceleration by roughly the factor $p$, provided that $p \leq n$. We also note that the estimated computational time (in both sequential and parallel implementations) can be decreased by roughly the factor $n/\log n$ if we only need to approximate a single zero of $p(x)$ rather than all the $n$ zeros [P87].

The presented techniques and results can be extended to approximating the zeros of an analytic function in a square, if these zeros are isolated from all other zeros of the function.

For an extension to approximating the eigenvalues of a symmetric tridiagonal matrix, see [P93], [PL], and [P].

In our exposition we will follow and extend [P94]. We will present the results in the following order. Section 4.2 is devoted to some preliminaries. In section 4.4 we recall Weyl's exclusion algorithm (quadtree algorithm), incorporating the proximity test, which we specify in section 4.3. In section 4.5 we define the main algorithm in the flowchart form, as a recursive process of isolation and contraction. Section 4.6 describes the geometric construction used at the isolation stages. In section 4.7, we present some

analytic techniques needed at the contraction stages. Sections 4.3 contains also several auxiliary results needed in sections 4.4–4.7. Section 4.8 contains figures. In appendix B we analyze the correctness of the algorithms.

**Remark 4.1.1.** *In this chapter, we do not cite numerous papers that proposed various techniques for approximating all the complex zeros of a polynomial but do not supply the competitive worst case estimates for the computational complexity. Among these techniques, we wish to single out some practically successful methods: the Jenkins-Traub method [JT], a modification of Newton's iteration due to [M73], Laguerre's method [HPR] [F81], and Durand-Kerner method and its various extensions and modifications, such as ones by Aberth, Maehly, Werner, Pasquini-Trigiante, Farmer-Loizou and by others (see [BP,a]), which are most popular in practice among all the known approaches to the problem of approximating all the complex zeros of a polynomial. These methods also require to obtain an initial set of approximations to the n zeros of $p(x)$, for which purpose one may apply the techniques of our section 4.6.*

**Remark 4.1.2.** *In some papers, notably in [Sc82] and [KS], the error of the approximations to the zeros of $p(x)$ is measured by the maximum magnitude $E(p)$ of the coefficients of the error polynomial $p(x) - p_n \prod_{j=1}^{n}(x - z_j^*)$, $z_j^*$ denoting the computed approximations to the zero $z_j$ of $p(x)$ and $p_n$ being the leading coefficient of $p(x)$. In this case the problem is sometimes called complex factorization [versus approximating the zeros of $p(x)$, where the output error is measured by $E_z(p) = \max_j |z_j^* - z_j|$]. Recall from the example of the polynomial $(x^n - 7/9) - 2^{-bn}$ that one needs to have $E(p) = \mathbf{O}(2^{-bn})$ in order to ensure that $E_z(p) \leq 2^{-b}$ in the worst case, and the algorithms of [Sc82] and [KS] support substantially smaller upper bounds on the complexity of the complex factorization than of approximating the zeros. In particular, Schönhage in [Sc82] reached the sequential Boolean complexity bounds $\mathbf{O}((b + n\log n)n^2 \log(bn)\log\log(bn))$ for the problem of complex factorization, with $E(p) \leq 2^{-b}$, and $\mathbf{O}((b + \log n)n^3 \log(bn)\log\log(bn))$ for the problem of approximating the zeros, with $E_z(p) \leq 2^{-b}$. A distinct approach of [KS] uses $\mathbf{O}((b+n)n\log^2 n)$ and $\mathbf{O}(bn^2\log^2 n)$ arithmetic operations and comparisons in*

*order to solve the complex factorization problem and the approximating zeros problem, respectively [with the error bounds in terms of $E(p)$ and $E_z(p)$, respectively] provided that $n = \mathbf{O}(b)$, $\log E(p) = -b$. To reach these bounds, multipoint polynomial evaluation had to be involved in the algorithm of [KS]. Application of the known fast algorithms for this operation [AHU], [BM] would have led to numerical stability problems, so one is motivated to devise numerically stable polynomial evaluation algorithms (compare our chapter 3). Technically, the algorithm of [KS] is very interesting since it cleverly exploits and effectively develops (for the univariate case) the path lifting method, originated in [S81], [S85] and more recently developed into a highly effective algorithm for a system of polynomial equations [SS], [SS,a], [SS,b], [SS,c], [SS,d].*

## 4.2 Some Preliminaries and Definitions.

We will count the polynomial zeros with their multiplicities, not distinguishing between approximating $m$ clustered zeros and a multiple zero of multiplicity $m$. By saying "computing a polynomial" we will mean "computing its coefficients" (unless we explicitly specify otherwise). $p(x)$ denotes a fixed polynomial of degree $n$,

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - z_j), \ p_n \neq 0. \qquad (4.2.1)$$

**Definition 4.2.1.** *$D = D(X, R)$ denotes the disc of radius $\rho(D) = R$ with the 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 $X+R(1+\sqrt{-1})$, $X-R(1+\sqrt{-1})$, $X+R(1-\sqrt{-1})$, $X-R(1-\sqrt{-1})$.*

**Definition 4.2.2.** *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 4.1): $r.r.(U) = inf(\rho(U^-)/\rho(U))$, $i.r.(U) = sup(\rho(U^+)/\rho(U))$, where $\rho(U)$ is defined in Definition 4.2.1 and where 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 $U^+$ and $U$ are concentric, $U^- \subseteq U \subseteq U^+$. A disc or a square $U$ and every its equivalent subset are* **f-isolated** *if $i.r.(U) \geq f$.*

**Definition 4.2.3.** *$d(U) = \max_{z_g, z_h} |z_g - z_h|$ for a complex set $U$,*
*$d^*(U) = \max_{z_g, z_h} \max\{|Re\ z_g - Re\ z_h|, |Im\ z_g - Im\ z_h|\}$,*
*where $\max_{z_g, z_h}$ denotes the maximum over all the pairs $z_g$, $z_h$ of the zeros of $p(x)$ in $U$ and we use the customary notation $c = Re\ c + \sqrt{-1}\ Im\ c$, with real $Re\ c$ and $Im\ c$, for any complex $c$.*

**Proposition 4.2.1.** *$r.r.(S) = d^*(S)/(2\rho(S))$ for a square $S$; $r.r.(D) = d^*(D)/(c\rho(D))$, $\sqrt{2} \leq c \leq 2$, for a disc $D$.*

**Definition 4.2.4.** *For a complex $X$, for an $f > 1$ and for a nonnegative $\varepsilon$, the disc $D(X, \varepsilon)$ is called* **an $f$-isolated** *$\varepsilon$-neighborhood of a zero $z_j$ of $p(x)$ if this disc contains $z_j$ and is $f$-isolated.*

**Definition 4.2.5.** *$i(p(x), U)$, 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 4.2.6.** *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 set $r_s(X) = \infty$ for $s \leq 0$, $r_s(X) = 0$ for $s > n$.*

**Proposition 4.2.2.** *$1/r_s(X)$ equals the $(n + 1 - s)$-th root radius at $X$ of the (shifted and reversed) polynomial $(x - X)^n p(X + \frac{1}{x - X})$.*

**Definition 4.2.7.** *For fixed positive $\overline{R}$ and $\varepsilon$, write*

$$b = \log(\overline{R}/\varepsilon), \qquad (4.2.2)$$

*provided that $\overline{R}$ is an upper bound on the moduli of all the zeros of $p(x)$,*

$$\overline{R} \geq |z_j|, \ for\ j = 1, \ldots, n. \qquad (4.2.3)$$

*In particular(see [He], section 6.4a), the relations ( 4.2.3 ) hold for*

$$\overline{R} = 2 \max_{h=1,\dots,n} |p_{n-h}/p_n|^{1/h}. \qquad (4.2.4)$$

**Remark 4.2.1.** *Hereafter, all the rectangles and squares on the complex plane have their sides parallel to the coordinate axes.*

**Algorithm 4.2.1.**

**Input:** *a closed set $U$ on the complex plane.*

**Output:** *the side lengths and the real and imaginary coordinates of the center of the smallest rectangle containing the set $U$ (and having its sides parallel to the coordinate axes).*

**Computation:** *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 $U$.*

**Proposition 4.2.3.** *The two half-sums in the output of Algorithm 4.2.1 equal the real and imaginary parts of the center $X$ of the minimum rectangle containing the set $U$. The two differences equal the lengths of the sides of this rectangle. The center $X$ and the length $2r$ of the longer side define the smallest square $S(X, r)$ containing the set $U$.*

In this chapter, we will use the known algorithms for the basic operations with polynomials [such as their multiplication and division with a remainder, discrete Fourier transform (DFT), scaling and the shift of the variable]; the arithmetic cost of these computations is $\mathbf{O}(n \log n)$ (see [AHU], [P92a], [BP94]).

**Remark 4.2.2.** *Our results stated for the unit disc $D(0, 1)$ immediately extend to the disc $D(X, r)$, for any complex $X$ and positive $r$: it suffices to shift and to scale the variable.*

## 4.3 Root Radii Computation, the Proximity Test, and the Computation of the Number of Polynomial Zeros in a Disc.

In this section, we will approximate the root radii $r_s(X)$, for $s = 1, \ldots, n$, by following and a little simplifying [Sc82], section 14. We will keep assuming that $X = 0$ (otherwise, we would shift the variable by letting $y = (x - X)$ and will denote $r_s = r_s(X)$,

$$r_0 = \infty, \ r_{n+1} = 0. \qquad (4.3.1)$$

We will also present the proximity test, which enables us to compute the distance from a complex point $X$ to the nearest zero of $p(x)$, with a relative error $\Delta$, at the cost $\mathbf{O}(n(1+g)\log n)$, where $g$ is defined in ( 4.3.4 ). At the end of the section we will present an algorithm for computing the number of polynomial zeros that lie in a given disc.

### 4.3.1 Root Radii Computation.

Consider the two following tasks (note the redundancy in their statements):

**Task** $r$. *Given positive $r$ and $\Delta$, find a (generally non-unique) integer $s$ such that*

$$r_{s+1}/(1+\Delta) < r < (1+\Delta)r_s. \qquad (4.3.2)$$

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

$$r/(1+\Delta) < r_s < (1+\Delta)r. \qquad (4.3.3)$$

We will solve Tasks $r$ and $s$ for $1 + \Delta = 2n$. The extension to an abritrary positive $\Delta$ is immediate, by means of

$$g = g(\Delta) = \lceil \log(\frac{\log(2n)}{\log(1+\Delta)}) \rceil \qquad (4.3.4)$$

iteration steps ( 4.3.16 ); indeed, such an iteration step amounts to squaring $1+\Delta$ in the context of Tasks $r$ and $s$, and we have $(1+\Delta)^{2^g} \ge 2n$, under ( 4.3.4 ). 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

$$g(\Delta) = 0 \text{ if } 1 + \Delta \geq 2n;\ g(\Delta) = \mathbf{O}(\log\log n) \text{ if } 1/\Delta = \mathbf{O}(1), \qquad (4.3.5)$$

$$g(\Delta) = \mathbf{O}(\log n) \text{ if } 1/\Delta = n^{\mathbf{O}(1)}. \qquad (4.3.6)$$

Most frequently, we need to solve Task $s$ for $s = n$, and we have the following corollary:

**Corollary 4.3.1.** *For $s = 1$ and $s = n$, Task $s$ can be solved at the arithmetic cost $\mathbf{O}(n(1+g)\log n)$, where $g$ is defined by ( 4.3.4 )-( 4.3.6 ).*

Let us show a proof from [Sc82]. Recall the well-known inequalities (compare [He], pp.451, 452 and 457):

$$t_1^*/n \leq r_1 < 2t_1^*.\ t_1^* = \max_{k>0} |p_{n-k}/p_n|^{1/k}. \qquad (4.3.7)$$

Apply Proposition 4.2.2 for $X = 0$ and extend the bounds ( 4.3.7 ) as follows:

$$t_n^*/2 < r_n \leq nt_n^*,\ t_n^* = \min_{k>0} |p_0/p_k|^{1/k}. \qquad (4.3.8)$$

Therefore, $r = t_1^*\sqrt{2/n}$ is a solution to Task $s$ for $s = 1$, whereas $r = t_n^*\sqrt{n/2}$ is a solution to Task $s$ for $s = n$: in both cases, we may choose any $\Delta$ such that $1+\Delta > \sqrt{2n}$. This can be extended to the solution of Task $s$, for $s = 1$ and $s = n$ and for an arbitrary positive $\Delta$, at the arithmetic cost $\mathbf{O}(ng\log n)$ of $g$ iteration steps ( 4.3.16 ), where $g$ is defined by ( 4.3.4 )-( 4.3.6 ). This implies the cost bound of Corollary 4.3.1.

We also need to solve Task $s$ for $1 < s < n$ at Stage 10 of Algorithm 4.7.1 and Task $r$ in Remark 4.3.3. Next, we will show solution algorithms relying on the following useful and elegant result:

**Theorem 4.3.1.** *([He, pp. 458-462], [Sc82]). If $1 \leq m \leq n$ and if $|p_{n+1-m-i}/p_{n+1-m}| \leq av^i$ for $i = 1, \ldots, n+1-m$, then $r_m < m(a+1)v$.*

*Proof.* Due to Van Vleck's theorem ([He], 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 $|p_{n+1-m}|r^{n+1-m}$, apply the assumed bound on the ratios $|p_{n+1-m-i}/p_{n+1-m}|$, and deduce for $x = v/r_m$ that

$$1 \le ax^{n+1-m}\binom{n}{m-1} + ax^{n-m}\binom{n-1}{m-1} + \cdots + ax^2\binom{m+1}{m-1} + ax\binom{m}{m-1}. \quad (4.3.9)$$

If $x \ge 1$, then $r_m \le v$, and Theorem 4.3.1 follows. Otherwise ( 4.3.9 ) 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 4.3.1 follows. □

Theorem 4.3.1 and Proposition 4.2.2 also imply a similar upper bound on $1/r_m$, which is the $(n+1-m)$-th root radius of the reverse polynomial $x^n p(1/x)$.

Such bounds immediately imply the solution of Task $r$ for $r = 1$ and $1 + \Delta = 2n$. Indeed, compute the natural $m$ such that $|p_{n+1-m}| = \max_{0 \le i \le n}|p_i|$. If $m = n+1$, then $t_n^* \ge 1$, $n > r_n$, and $r_{n+1} = 0$ [see ( 4.3.1 ) and ( 4.3.8 )]. Moreover, for the reverse polynomial, $q(x) = x^n p(1/x)$, we have $t_1^* \le 1$, and, therefore, $r_1 < 2$ [see ( 4.3.7 )], which implies that $r_n > 1/2$ for $p(x)$. It follows that $s = n$ is a desired solution to Task $r$, for $r = 1$ and $1 + \Delta = n$, as well as for $r = \sqrt{n/2}$ and $1 + \Delta = \sqrt{2n}$. Otherwise, $1 \le m \le n$. Then we apply Theorem 4.3.1 (for $a = v = 1$) to $p(x)$ and $q(x) = x^n p(1/x)$ and deduce that $\frac{1}{2(n+1-m)} < r_m < 2m$, so that $1/(2n) < r_m < 2n$, and $s = m - 1$ is a solution to Task $r$ where $r = 1$ and $1 + \Delta = 2n$ [take into account ( 4.3.1 ) where $m = 1$, $s = 0$]. The extensions to arbitrary $r$ is by means of scaling the variable $x$ and to arbitrary $\Delta$ by means of the iteration ( 4.3.16 ). We arrive at

**Proposition 4.3.1.** *Task r can be solved at the arithmetic cost* $\mathbf{O}(n(1+g)\log n)$ *where* $g$ *is defined by ( 4.3.4 )-( 4.3.6 ); the cost bound can be decreased to* $\mathbf{O}(n)$ *if* $1+\Delta \geq 2n$.

We could have solved Task $s$ by recursively applying Proposition 4.3.1 in a binary search algorithm, but we will prefer a more direct algorithm outlined in [Sc82].

**Algorithm 4.3.1.** *Given the coefficients* $p_u$ *of* $p(x)$ *and an integer* $s$, $1 \leq s \leq n$, *choose two integers* $t$ *and* $h$ *that satisfy the following inequalities:*

$$t < n+1-s \leq t+h \leq n \qquad (4.3.10)$$

*(so that* $h > 0$, $t < n$*), and the following convexity property: there exists no integer* $u$ *in the considered range 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* $\log|p_u|$. *Compute and output*

$$r = |p_{t+h}/p_t|^{1/h}. \qquad (4.3.11)$$

The relations ( 4.3.10 ) and ( 4.3.11 ) and the above convexity property immediately imply that

$$\begin{aligned} p_{t+g}/p_t &\leq r^g, \quad for \quad g = 1,\ldots,n-t, \\ p_{t+h-g}/p_{t+h} &\leq 1/r^g, \quad for \quad g = 1,\ldots,t+h. \end{aligned} \qquad (4.3.12)$$

**Proposition 4.3.2.** *The output* $r$ *of Algorithm 4.3.1 is a solution to Task* $s$ *for* $1+\Delta = 2n$.

*Proof.* Due to the first and the second inequalities of ( 4.3.12 ), we may apply Theorem 4.3.1 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,

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

Since $h > 0$, $t < n$, it follows that

$$1/(2nr) < r_{n-t} \leq r_s \leq r_{n+1-t-h} < 2n/r. \qquad (4.3.13)$$

□

Let us next specify the computations in Algorithm 4.3.1 as follows:

**Algorithm 4.3.2.** *Compute the values* $\log|p_u|$, $u = 0, 1, \ldots, n$, *with a prescribed precision; then compute the convex hull* $CH$ *of the set* $\{(u, \log|p_u|),\ u = 0, 1, \ldots, n\}$ *in the two-dimensional real space, 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 endpoints of this interval and, finally, compute* $r$ *by using ( 4.3.11 ).*

Note that we compute the convex hull $CH$ of the same single set when we solve Task $s$ for all $s$. Thus we arrive at the following result:

**Proposition 4.3.3.** *Task* $s$ *for all* $s$ *can be solved at the arithmetic cost of* $c_{\mathbf{A}}(CH) + \mathbf{O}(gn \log n)$ *where* $g$ *is defined by ( 4.3.4 )-( 4.3.6 ) and where* $c_{\mathbf{A}}(CH)$ *denotes the cost of computing the values* $\log|p_u|$ *for* $u = 0, 1, \ldots, n$ *and the convex hull of the set* $\{(u, \log|p_u|),\ u = 0, 1, \ldots, n\}$ *of* $n + 1$ *points on the plane.*

Let us next simplify the solution of Task $s$ for $1 + \Delta = 2n$ and for a disc $D$, with the center 0, under the additional assumption that

$$i(p(x), D) = n + 1 - s,\ i.r.(D) \geq (1 + \Delta)^2. \qquad (4.3.14)$$

In this case, the arithmetic cost of the solution is $\mathbf{O}(n)$ with a small overhead constant.

**Proposition 4.3.4.** *Let the relations ( 4.3.14 ) hold for a fixed integer* $s$ *(such that* $0 < s \leq n$*), for* $1 + \Delta = 2n$, *and for a disc* $D$ *having the center 0 and an unknown radius. Then Task* $s$ *for* $1 + \Delta = 2n$ *can be solved at the arithmetic cost* $\mathbf{O}(n)$.

*Proof.* Let $t$ and $h$ denote the two integers defined in Algorithm 4.3.1 and thus satisfying ( 4.3.10 ) and ( 4.3.13 ). ( 4.3.14 ) implies that $r_{s-1}/r_s \geq (1+\Delta)^2 = 4n^2$. On the other hand, the first inequality of ( 4.3.10 ) implies that $r_s \geq r_{n-t}$, so that $r_{s-1}/r_{n-t} \geq (1+\Delta)^2 = 4n^2$. 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, $r_{n+1-t-h}/r_{n-t} \geq r_{s-1}/r_{n-t} \geq (1+\Delta)^2 = 4n^2$. The latter inequalities imply that $r_{n+1-t-h} \geq 4n^2 r_{n-t}$, which contradicts ( 4.3.13 ), so that $t+h \leq n+1-s$. Therefore, ( 4.3.10 ) implies that $t+h = n+1-s$. Since $t+h$ has been defined, it remains to choose $h > 1$ such that the convexity property of Algorithm 4.3.1 holds, or, equivalently, such that 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 $n$. The integer $h$ can be computed at the arithmetic cost $\mathbf{O}(n)$, and we arrive at Proposition 4.3.4. □

We now observe that, for $g(\Delta)$ of ( 4.3.4 )–( 4.3.6 ), $g = g(\Delta)$ iteration steps ( 4.3.16 ) suffice to reduce the solution of Task $s$, for any fixed $1+\Delta > 1$ and for any disc $D$ satisfying ( 4.3.14 ), to the case where $1+\Delta = 2n$, and, by shifting the variable $X$, we may ensure that the disc $D$ has center 0.

**Corollary 4.3.2.** *Let the relations ( 4.3.14 ) hold for a fixed integer $s$, $0 < s \leq n$, for a fixed positive $\Delta$, and for a disc $D$ having the center 0 and an unknown radius. Then Task $s$ can be solved at the arithmetic cost bounded by $\mathbf{O}(n + g(\Delta)n\log n)$, where $g(\Delta)$ is defined by ( 4.3.4 ) [and satisfies ( 4.3.5 ) and ( 4.3.6 )].*

The algorithm supporting Corollary 4.3.2 is referred to as **Algorithm 4.3.2a.**

### 4.3.2 A Proximity Test.

Let us present an algorithm for proximity test having the arithmetic cost of Corollary 4.3.1.

**Algorithm 4.3.3.**

**Input:** *Naturals $n$ and $\Delta$, the coefficients of polynomial $p(x)$ of ( 4.2.1 ), having degree $n$, and a complex number $X$ that is not a zero of $p(x)$.*

**Output:** *A nonnegative number $r = r(X)$ such that*

$$r/(1+\Delta) \le r_n(X) \le r(1+\Delta),\ r_n(X) = \min_{j=1,\dots,n} |z_j - X|. \qquad (4.3.15)$$

**Computations.**

**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 $y_j = (z_j - X)$, $j = 1,\dots,n$, such that*

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

$$q(y) = p_n(X) \prod_{j=1}^{n} (y - y_j).$$

**Stage b),** *(Dandelin-Lobachevsky-Graeffe, [H]). Let $q_0(y) = q(y)/p_0(X)$ and $g = g(\Delta)$ is defined by ( 4.3.4 ), and successively compute the coefficients of the polynomials*

$$q_{i+1}(y) = (-1)^n q_i(\sqrt{y}) q_i(-\sqrt{y}),\ i = 0, 1, \dots, g-1. \qquad (4.3.16)$$

**Stage c).** *Compute $t_n^* = \min_{k>0} |q_0/q_k|^{1/k}$, set $r = [t_n^* \sqrt{\frac{n}{2}}]^{1/N}$ [see ( 4.3.8 )].*

We note that each step i of Stage b) squares the zeros of the polynomial $q_i(y)$, so that

$$q_h(y) = \prod_{j=1}^{n} (y - y_j^N) = \sum_{i=0}^{n} q_{i,h} y^j,\ N = 2^g. \qquad (4.3.17)$$

This makes the iteration ( 4.3.16 ) a powerful tool for separating the zeros of $p(x)$ from each other, because the logarithm of the ratio $|y_j/y_i|$ grows linearly in $N$, if $|y_j| > |y_i|$; furthermore, higher powers of the absolutely and strictly largest zero of $p(x)$ dominate in the respective power sums of all the zeros.

At Stage a), we just shift the variable $x$; every iteration $i$ of Stage b) is a polynomial multiplication; at Stage c) we find the maximum or the minimum in a set having cardinality at most $n$. Thus the overall cost of performing Algorithm 4.3.3 is

$$O(n(1+g)\log n)\ , \qquad (4.3.18)$$

where $g = g(\Delta)$.

**Remark 4.3.1.** *The algorithm of [Tu] (compare [P87], [P94]) computes $r = r(X)$ supporting an arithmetic complexity bound improved by factor $g(\Delta)$ but leads to numerical stability problems, since it generally requires to increase by factor $n$ the precision of the computations. This is, clearly, undesirable in practice and leads to a respective increase of the Boolean complexity estimates.*

**Remark 4.3.2.** *Each iteration step ( 4.3.16 ) squares $i.r.(D(0,r))$ for any positive $r$. This enables us to increase an $i.r.(D)$ from $f > 1$ to $f^{2^h}$ at the cost $\mathbf{O}((h+\delta)n\log n)$, where $\delta = 0$ if the disc $D$ has center 0.*

### 4.3.3 Computing the Number of Polynomial Zeros in a Disc.

**Proposition 4.3.5.** *Let $i.r.(D) = f > 1$ for a disc $D = D(X,r)$. Then the index $i(p(x),D)$ of $p(x)$ in $D$ (defined in Definition 4.2.5) can be computed at the arithmetic cost $\mathbf{O}(hn\log n)$, where*

$$h = 1 + \left\lceil \log\lceil \frac{\log 9}{\log f} \rceil \right\rceil . \qquad (4.3.19)$$

*Proof* (compare Remark 4.3.3). The well known *winding number algorithms* (see [He], pp. 239–241; [R87]) compute the index $i(p(x),D)$ of $p(x)$ in a disc $D$ at the cost $\mathbf{O}(n\log n)$, provided that all the zeros of $p(x)$ lie far enough from the boundary of such a disc; specifically, it suffices if $i.r.(D) \geq 9$ ([R87]). Proposition 4.3.5 now follows due to Remark 4.3.2. □

**Remark 4.3.3.** *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 4.3 for $r = (1+v)\rho$ and for any $\Delta \leq v$. The arithmetic cost of the solution is $\mathbf{O}(n(1+g)\log n)$, where $g$ is defined by ( 4.3.4 )-( 4.3.6 ), so that $g = \mathbf{O}(h + \log\log n)$, [compare ( 4.3.4 ) with ( 4.3.19 )]. This implies an alternative proof of Proposition 4.3.5, provided that the cost bound in its statement changes into $\mathbf{O}(gn\log n)$.*

## 4.4 Weyl's Exclusion Algorithm.

In this section, we will present Weyl's exclusion algorithm ([W24], [He], pp. 517–521; [R87], [P87], [P94]). Later on, we will use the resulting combination of the algorithms in order to isolate the clusters of the zeros of $p(x)$, although historically such an algorithm has been first introduced for approximating the zeros.

**Algorithm 4.4.1***[Weyl](see Figure 4.2).*

**Input:** *positive integers $k$, $G$, $n \geq k$, and $f > 1$, the coefficients of the polynomial $p(x)$ of ( 4.2.1 ), 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.*

**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 $k$ zeros of $p(x)$.*

**Initialization:** *Call the square $S(X, R)$ suspect in Stage 0.*

**Stage** $g$, $g = 1, \ldots, G$. *Divide each square that is suspect in Stage $g < G$ (such a square has sides of length $R/2^g$) into four subsquares, with the side length $R/2^{g+1}$. Then, approximate the distances from the centers of the subsquares to the nearest zeros of $p(x)$. [Do this by applying Algorithm 4.3.3 of section 4.3 at the four centers of the four subsquares, where each center plays the role of the input value $X$ of Algorithm 4.3.3, whose other input values, $N$, $n$, $p_0, \ldots, p_n$, are shared with Algorithm 4.4.1.] Call a square suspect in Stage $g$ unless the latter proximity test proves that the square contains no zeros of $p(x)$. Output the centers $X_h$ of all the squares $S(X_h, r)$ that are suspect in Stage $G$ (where $r = R/2^G$). Output $H$, the overall number of the output squares. (The next remark shows that $H \leq 4n$.)*

**Remark 4.4.1.** *Each zero of $p(x)$ may make at most 4 squares suspect in Stage $g$, for any $g$ [note that our proximity test computes $r_n(X)$ within a fixed error bound, due to ( 4.3.15 )].*

## 4.5 Combining Approximation and Isolation of Polynomial Zeros.

Our algorithm for polynomial zeros will rely on the solution of the following problem:

**Problem 4.5.1.**

**Input:** *two real constants* $f > 1$ *and* $\varepsilon = 2^{-b} > 0$, *complex coefficients of a polynomial* $p(x)$ *of ( 4.2.1 ), a natural* $k$, $2 \le k \le n$, *and an* $f$*-isolated square* $S$ *that contains exactly* $k$ *zeros of* $p(x)$.

**Output:** *an integer* $k_0$, $0 \le k_0 \le k$, $f$*-isolated* $\varepsilon_h$*-neighborhoods of* $k_0$ *zeros* $z_h$ *of* $p(x)$ *in* $S$ *for* $\varepsilon_h \le \varepsilon$, $h = 1, \ldots, k_0$, *an integer* $u \ge 0$ *(*$u \ge 2$ *if* $k_0 = 0$*) and* $u$ *squares* $S_1, \ldots, S_u$ *that are disjoint,* $f$*-isolated, and such that* $S_g \subseteq S$, $k_g = i(p(x), S_g) \ge 1$, $g = 1, \ldots, u$, $\sum_{g=0}^{u} k_g = k$.

The computation of the indices of $p(x)$ in each square $S_g$, $g = 1, \ldots, u$, can be based on the well-known winding number algorithms [He], pp. 239–241 (see section 4.3); [R87] or on the algorithms for approximating the distances from any fixed complex point to all the zeros of $p(x)$ [Sc82], [P87], [P94].

**Algorithm 4.5.1.** *To compute* $f$*-isolated* $\varepsilon_h$*-neighborhoods of all the zeros* $z_h$ *of* $p(x)$, *proceed as follows: Initially set* $k = n$, $S = S(0, \overline{R})$, *for* $\overline{R}$ *of ( 4.2.4 ), and solve Problem 4.5.1 for such a square* $S$. *Then recursively, for each output square* $S_g$, $g = 1, \ldots, u$, *set* $S = S_g$ *and solve Problems 4.5.1 (for* $S = S_g$ *playing the role of the input square). Recursively repeat this process until the desired* $f$*-isolated* $\varepsilon_h$*-neighborhoods of all the zeros of* $p(x)$ *have been computed.*

Since $k_g < k$, for $g = 1, \ldots, u$, we will arrive at the desired $\varepsilon_h$-neighborhoods of all the $n$ zeros of $p(x)$ in $S$ in at most $n - 1$ recursive steps of solving Problem 4.5.1. It remains to specify the solution of Problem 4.5.1 and to estimate its complexity.

We will partition the solution of Problem 4.5.1 into two stages:

**Problem 4.5.1a.**

**Input:** *as for Problem 4.5.1.*

**Output:** *either a common f-isolated $\varepsilon_h$-neighborhood of all the k zeros $z_h$ of $p(x)$ in S for some $\varepsilon_h \leq \varepsilon$ or, else, an equivalent subsquare $S^* = S(X^*, R^*)$ of S, such that*

$$e \geq 1/r.r.(S^*) \qquad (4.5.1)$$

*for a fixed positive e.*

**Problem 4.5.1b.**

**Input:** *the output subsquare $S^*$ of Problem 4.1a, satisfying ( 4.5.1 ).*

**Output:** *as for Problem 4.5.1.*

We will consider Problem 4.5.1b in the next section and Problem 4.5.1a in section 4.7.

## 4.6 Isolation of the Zeros.

In this section we will assume available a black box algorithm **(Algorithm 4.7.1)** of section 4.7 that solves Problem 4.5.1a and will specify **Algorithm 4.6.1**, that is, the blocks of Algorithm 4.5.1 where Problem 4.5.1b is solved. We will proceed as in (Weyl's) Algorithm 4.4.1 applied to the input square $S^* = S(X^*, R^*)$ of Algorithm 4.6.1 (which is the output square of Algorithm 4.7.1), so that for every $i$ we define the $i$-th stage of Algorithm 4.6.1 as identical to the $i$-th stage of Algorithm 4.4.1, except that we now add some blocks for the verification if the requirements to the output of Algorithm 4.6.1 have already been satisfied for some squares that are suspect in this stage; if so, we stop partitioning these squares and keep them invariant until the end of the computations, when we output these invariant squares. Namely, for every $g$, we will partition the union of all the $w(g)$ squares that are suspect in Stage $g$ into $v(g)$ maximal connected components, $C_1^{(g)}, \ldots, C_{v(g)}^{(g)}$, each component contains at least one zero of $p(x)$, so that $1 \leq v(g) \leq k$, $g = 1, 2, \ldots$; $v(1) = 1$, $S^* = C_1^1$. Let the $(1 + g_0)$th stage be the separation

stage of Algorithm 4.6.1, that is,

$$v(g) = 1 \text{ for } g = 1, 2, \ldots, g_0;\ v(g_0 + 1) > 1. \qquad (4.6.1)$$

In every Stage $g$, for $g > g_0$, apply Algorithm 4.2.1 to the set of all the vertices of all the suspect squares of the component $C_u^{(g)}$, for $u = 1, \ldots, v(g)$, and arrive at the minimum square, $S_u^{(g)} = S(X_u^{(g)}, R_u^{(g)})$, covering $C_u^{(g)}$ (compare Remark 4.2.1 and Proposition 4.2.3). Call $C_u^{(g)}$ an *f-isolated component* if the square $S_u^{(g)}$ is equivalent to $C_u^{(g)}$ and if

$$i.r.(S_u^{(g)}) \geq f.$$

Rewrite the latter assumption as follows:

$$d_u^{(g)} = \min_{v \neq u}(\max\{|ReX_u^{(g)} - ReX_v^{(g)}|, |ImX_u^{(g)} - ImX_v^{(g)}|\} - R_v^{(g)}) \geq fR_u^{(g)}. \qquad (4.6.2)$$

For each $u$, $u = 1, \ldots, v(g)$, test if

$$\sqrt{2}R_u^{(g)} \leq \min\{\varepsilon, d_u^{(g)}/f\}. \qquad (4.6.3)$$

If so, output the disc $D(X_u^{(g)}, R_u^{(g)}\sqrt{2})$ as an $f$-isolated $(\sqrt{2}R_u^{(g)})$-neighborhood of the zeros of $p(x)$ lying in the component $C_u^{(g)}$ and define $C_v^{(g+i)} = S(X_u^{(g)}, R_u^{(g)}) = S_v^{(g+i)} = S(X_v^{(g+i)}, R_v^{(g+i)})$, for an appropriate $v = v(g+i)$, as an invariant component that remains unchanged in all Stages $g + i$ of Algorithm 4.6.1 for $i > 0$. Otherwise test if ( 4.6.2 ) holds. If so, output the square $S_u^{(g)}$ as an input square $S$ for Algorithm 4.7.1 and define $C_v^{(g+i)} = S(X_u^{(g)}, R_u^{(g)}) = S_v^{(g+i)} = S(X_v^{(g+i)}, R_v^{(g+i)})$ for appropriate $v = v(g + i)$ as an invariant component that remains unchanged in all Stages $g + i$ of Algorithm 4.6.1 for $i > 0$. Apply Stage $g + 1$ of Algorithm 4.4.1 to all the squares that lie in the remaining noninvariant components and that are suspect in Stage $g$. Stop the computation by Algorithm 4.6.1 when all the components have become invariant.

Let us next analyze Algorithm 4.6.1. Hereafter, $w(C_u^{(g)})$ denotes the number of squares that are suspect in Stage $g$ and lie in the component $C_u^{(g)}$, so that

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

Our next goal is to deduce that

$$W = \sum_{g=g_0+1}^{G} w(g) = \mathsf{O}((1 + \log f)k). \qquad (\ 4.6.4\ )$$

We define a tree $T$ with $G + 1 - g_0$ levels of nodes, where $g_0$ is defined by ( 4.6.1 ). The $v(g)$ components $C_1^{(g)}, \ldots, C_{v(g)}^{(g)}$ 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 level 0. The leaves are identified with the invariant components $C_u^{(g)}$, such that ( 4.6.2 ) and/or ( 4.6.3 ) hold for the values $X_u^{(g)}, R_u^{(g)}$. 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 squares that are suspect in Stage $g + 1$ and lie in $C_u^{(g)}$. We let $w$ denote $\sum w(C_u^{(g)})$ where the sum is over all the leaves $C_u^{(g)}$; we deduce from Remark 4.4.1 that

$$w \leq 4k. \qquad (\ 4.6.5\ )$$

Hereafter, we will assume that every zero of $p(x)$ that makes two or several squares suspect in Stage $g$ shall lie in or coincide with the intersection of the boundaries of these squares. [Clearly, we may always move the zeros of $p(x)$ on the complex plane, so as to satisfy this assumption without any decrease of the value $W$ of ( 4.6.4 ).] Then, each node $C_u^{(g)}$ that is not a leaf contains not more suspect squares than its children do, that is,

$$w(C_u^{(g)}) \leq \sum_{r(u)} w(C_{r(u)}^{(g+1)}), \qquad (\ 4.6.6\ )$$

where the summation is over all the children $C_{r(u)}^{(g+1)}$ of the node $C_u^{(g)}$. We now recall that, besides the leaves, none of the components $C_u^{(g)}$ for $g \geq g_0$ is $f$-isolated and deduce the following proposition:

**Proposition 4.6.1.** *Every component $C_u^{(g)}$, $g > g_0$, has an ancestor-fork (that is, an ancestor with at least two children) at level $g_1$, $g_1 = g - g_0 - \log((f - 1)w(C_u^{(g)})) + y + \delta$ for some $y > 0$, where $\delta = 0$ if $C_u^{(g)}$ is an invariant output component, $\delta = 1$ otherwise.*

*Proof.* Observe that $R_u^{(g)} \leq R^* w(C_u^{(g)})/2^g$, whereas the distance from $X_u^{(g)}$ to the nearest zero of $p(x)$ lying outside $C_u^{(g)}$ is at least $R_u^{(g)} + 2R^*/2^{g_1+g_0}$, because such a zero

of $p(x)$ must be separated from $X_u^{(g)}$ by both the square $S(X_u^{(g)}, R_u^{(g)})$ and some square $S(X, R^*/2^{g_0+g_1})$ discarded in Stage $g_0 + g_1$. Consequently, $i.r.(S_u^{(g)}) \geq 1 + \frac{2^{g-g_1-g_0+1}}{w(C_u^{(g)})}$. Therefore, $i.r.(S_u^{(g)}) \geq f$ if $(f-1)w(C_u^{(g)}) \leq 2^{g-g_1-g_0+1}$ or, equivalently, if $\log((f-1)w(C_u^{(g)})) \leq g - g_1 - g_0 + 1$. On the other hand, $i.r.(S_u^{(g)}) < f$ unless $C_u^{(g)}$ is a leaf of $T$, and in the latter case $i.r.(S_v^{(g-1)}) < f$ for the parent $C_v^{(g-1)}$ of $C_u^{(g)}$. By combining the above relations we complete the proof. □

To derive ( 4.6.4 ), we will also use the following estimates:

**Fact 4.6.1.** *Every component $C_u^{(g)}$, for $u = 1, \ldots, v(g)$, $g = 3, \ldots, G$, is covered by*

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

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

The proof is rather straightforward (although tedious): it appeals, on one hand, to the fact that a square that is suspect at Stage $g-2$ has the 4-fold increase of the side length versus the side length of a square that is suspect at Stage $g$ and, on the other hand, to the connectivity of the component $C_u^{(g)}$ consisting of the chain of $w(C_u^{(g)})$ smaller suspect squares. To complete the proof, it essentially remains to classify all possible configurations of these $w(C_u^{(g)})$ squares (the worst case configuration is shown in Figure 4.3). We will omit further details but will note that for the proof of the asymptotic bound ( 4.6.4 ), it actually suffices to prove a weaker version of Fact 4.6.1, where Stage $g-2$ is replaced by Stage $g - \mathbf{O}(1)$ and the right side of the inequality of Fact 4.6.1 is turned into $\mathbf{O}(1) + a\, w(C_u^{(g)})$ for some fixed $a < 1$. The latter version of Fact 4.6.1 immediately follows from the 2 above observations (about the 4-fold increase of the side length and the component connectivity).

**Corollary 4.6.1.** $s(g,u) \leq 0.75 w(C_u^{(g)})$ *if* $w(C_u^{(g)}) > 6$.

Now let $w_6 = \sum_{u,g} w(C_u^{(g)})$, $w_6^* = \sum_{u,g}^* w(C_u^{(g)})$ where $\sum$ and $\sum^*$ denote the sums in all the pairs $g$ and $u$ such that $g > g_0$, $1 \leq u \leq v(g)$, $w(C_u^{(g)}) \leq 6$, provided that $\sum^*$

consists of 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.

By combining the bounds ( 4.6.5 ) and ( 4.6.6 ) with Corollary 4.6.1. we obtain that

$$W - w_G \leq 8w \leq 32k, \qquad (4.6.7)$$

and it remains to estimate $w_G$ in order to prove ( 4.6.4 ).

Proposition 4.6.1 implies that in the tree $T$ every node $C_u^{(g)}$ such that $w(C_u^{(g)}) \leq 6$ has at most $\log(6(f-1)) - 1$ its successive predecessors with a single child. Due to the bound ( 4.6.6 ), each of these predecessors contributes at most $w(C_u^{(g)})$ to the sum $w_G$. It follows that

$$w_G \leq \log(6(f-1))w_G^*. \qquad (4.6.8)$$

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

$$w_G^* \leq 6(2k-1) = 12k - 6.$$

Combining this bound on $w_G^*$ with ( 4.6.8 ) and ( 4.6.7 ), we arrive at ( 4.6.4 ).

We have deduced the bound ( 4.6.4 ) for a single application of Algorithm 4.6.1, which is actually called $\mathbf{O}(n)$ times in the process of performing Algorithm 4.5.1. In every application of Algorithm 4.6.1, its every output square $S_u^{(g)}$, satisfying the relation ( 4.6.2 ) but not ( 4.6.3 ) (that is, such a square is $f$-isolated but is not imbedded in an equivalent and $f$-isolated disc of a radius at most $\varepsilon$), serves as the input square in the subsequent application of Algorithm 4.7.1, which, in turn, generally outputs an input square for Problem 4.6.1 and Algorithm 4.6.1. We then use the tree $T$ generated in the latter application of Algorithm 4.6.1 to replace the respective leaf of the tree generated in the preceding application of Algorithm 4.6.1. In this way, we construct a single tree

associated with all the $\mathbf{O}(n)$ applications of Algorithm 4.6.1 within Algorithm 4.5.1 and having at most $n$ leaves. Proposition 4.6.1 and Fact 4.6.1 are also extended to the entire computational process, represented by the latter tree, and in this way we extend ( 4.6.4 ) to the same asymptotic estimate (with $k$ replaced by $n$) for the total number of suspect squares processed in all these calls for Algorithm 4.6.1, provided that in each call for Algorithm 4.6.1 we only count the suspect squares processed after the first splitting of the single input component into disjoint connected components.

Finally, we complement this estimate with the bound

$$W_0 = \mathbf{O}(n + n \log e), \qquad (4.6.9)$$

where $e$ is defined by ( 4.5.1 ) and $W_0$ denotes the overall number of squares that are suspect in all Stages $g$ such that $v(g) = 1$, $g \leq g_0$ [see ( 4.6.1 )], (that is, $C_1^{(g)}$ is the single connected component output in Stage $g$), in all the applications of Turan-Weyl's Algorithm 4.4.1, within Algorithm 4.5.1.

To deduce ( 4.6.9 ) we first consider a single application of Algorithm 4.6.1 and recall that the length of the sides of the squares that are suspect in Stage $g$ decreases by twice as $g$ grows by 1. It is sufficient to consider the cases, where

$$w(C_1^{(g)}) = w(g) \leq 6, \qquad (4.6.10)$$

since we may extend Fact 4.6.1 and Corollary 4.6.1 and also recall that

$$w(g) \leq 4k, \ g = 1, 2, \ldots, G. \qquad (4.6.11)$$

We now combine ( 4.6.10 ) and Proposition 4.2.1 and deduce that

$$w_G^* \leq 6\lceil \log e \rceil + 15. \qquad (4.6.12)$$

Indeed, due to ( 4.6.6 ) and ( 4.6.10 ), $w(C_1^{(g)})$ may grow with $g$ at most in 5 stages $g$, for $1 \leq g \leq g^*$, which contributes the term $1 + 2 + 3 + 4 + 5 = 15$ on the right side of ( 4.6.12 ). On the other hand, as $g$ increases by 1, $r.r.(S_1^{(g)})$ never decreases and grows by

twice, if $w(C_1^{(g)})$ stays invariant. This implies ( 4.6.12 ), since $r.r.(S_1^{(g)})$ never exceeds 1 and since we initially have ( 4.5.1 ).

Combine Fact 4.6.1 and Corollary 4.6.1 with ( 4.6.11 ) and ( 4.6.12 ) and deduce the bound

$$\sum_{g=1}^{g_0} w(g) = O(k + \log e),$$

for a single application of Algorithm 4.6.1, and similarly we arrive at ( 4.6.9 ) for all of at most $n-1$ applications of Algorithm 4.6.1 within Algorithm 4.5.1. Namely, in order to deduce ( 4.6.9 ), we observe that there are at most $n-1$ forks in the tree associated with the entire Algorithm 4.5.1 (since this tree has at most $n$ leaves), and that for each fork, we have to go through at most $15 + 6\lceil \log e \rceil$ its successive descendants before we reach either the next fork or a node $C$ with $w(C) > 6$. We exclude the suspect squares associated with the node $C$ such that $w(C) > 6$ since to them we may apply or extend Corollary 4.6.1 and may apply the bound ( 4.6.11 ). For other nodes, we arrive at ( 4.6.9 ), by $n-1$ times applying the bound ( 4.6.12 ).

We now recall that each proximity test in Algorithm 4.4.1 (one for each suspect square) is performed by means of Algorithm 4.3.3 at the cost bounded by $\mathbf{O}(ng(\Delta)\log n)$. Therefore, from ( 4.6.4 ) and ( 4.6.9 ), we deduce the bound

$$\mathbf{O}((1 + \log f + \log e)n^2 g(\Delta)\log n)$$

on the computational cost of all the applications of Algorithm 4.6.1 within Algorithm 4.5.1. In particular, for $e = \mathbf{O}(1)$, $\log f = \mathbf{O}(\log n)$, the latter cost bound turns into $\mathbf{O}((n\log n)^2 g(\Delta))$, whereas for $e = \mathbf{O}(1)$, $f = \mathbf{O}(1)$, this cost bound turns into $\mathbf{O}(n^2 g(\Delta)\log n)$.

## 4.7 Contraction of a Complex Square.

We will now present the following algorithm for Problem 4.5.1a, where we assume that $1/e \leq 0.24$, $f = f^*\sqrt{2}$, $f^* > 2 + 6n(e-1)/(e-2)$, and $f^* > \sqrt{50}$ (compare Remark 4.7.1

and the correctness proof in the full version of the thesis), so that it suffices to set $e = 5$, $f = 3 + 8n\sqrt{2}$, or $e = 8$, $f = 3 + 7n\sqrt{2}$, or $e = 26$, $f = 3 + (25/4)n\sqrt{2}$:

**Algorithm 4.7.1.**

**Initialization:** *Let $\sigma$ denote a set of complex numbers available from the previous application of Algorithm 4.7.1 and chosen to be empty at the first application of this algorithm. Compute the $f^*$-isolated disc $D = D(x_0, R)$ whose boundary circle passes through the four vertices of the input square $S$. 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 $\sigma$ and choose a point $t$ near $Y$ such that $p'(t) \neq 0$. (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 $\sigma$. Each candidate point $t$ for which $p'(t) = 0$ is added to the set $\sigma$.)*

2. *Compute the value*
$$u = t - \frac{kp(t)}{p'(t)}. \qquad (4.7.1)$$

3. *If $p(u) = 0$, set $X = u$ and go to Stage 10. Otherwise, apply Algorithm 4.3.3 (proximity test), for $X = u$, and appropriate $\Delta$, which outputs $r = r(u)$ and $r^*(u) = (\Delta + 1)r$.*

4. *Compute $v_h = v_h(u) = u + 8\delta_h r^*(u)$, for $\delta_h = (\sqrt{-1})^h$, $h = 0, 1, 2, 3$.*

5. *If $p(v_h) = 0$ for some $h$, $0 \leq h \leq 3$, set $X = v_h$ and go to Stage 10. Otherwise apply Algorithm 4.3.3 for $X = v_h$, and suitable $\Delta$, $h = 0, 1, 2, 3$; output $r(v_h)$, $r^*(v_h)$, $h = 0, 1, 2, 3$.*

6. *Compute $\tilde{r}_h = r^*(v_h) + r^*(v_{h+2})$, for $h = 0, 1$. If $\tilde{r}_h \leq 15r^*(u)$ for $h = 0$ or $h = 1$, then go to Stage 10 for $X = u$.*

7. *Otherwise, for every* $h$, $h = 0, 1, 2, 3$, *denote that* $D_h = D(v_h, r_n(v_h))$, *observe that the pair of the discs* $D_h$ *and* $D_{h+1 \bmod 4}$ *has two points* $\alpha_h$ *and* $\beta_h$ *of the intersection of their boundaries [this follows from the comparison of the distances*

$$|v_h - v_{h+1 \bmod 4}| = 8\sqrt{2}r^*(u), \ h = 0, 1, 2, 3, \qquad (4.7.2)$$

*with the estimates ( B.2.1 ) and ( B.2.3 ) of appendix 4.8], and choose one of these two points,* $\alpha_h$, *that lies closer to* $u$ *(see Figure 4.4).*

8. *Compute the complex* $c$ *and the nonnegative* $\rho$ *satisfying*

$$2Re\ c = \min_h Re\ \alpha_h + \max_h Re\ \alpha_h,$$

$$2Im\ c = \min_h Im\ \alpha_h + \max_h Im\ \alpha_h,$$

$$\rho = \max_h |c - \alpha_h|.$$

*Here and hereafter we use the notation* $\min_h$ *and* $\max_h$ *for the minimums and maximums in* $h$, *for* $h = 0, 1, 2, 3$, *and* $z = Re\ z + \sqrt{-1}\ Im\ z$ *for all complex* $z$. *[Note that the closed disc* $\tilde{D} = D(c, \rho)$ *contains* $\alpha_0$, $\alpha_1$, $\alpha_2$, $\alpha_3$ *and, consequently, at least one zero of* $p(x)$; *furthermore, it can be proved that the disc* $\tilde{D}$ *is* $f$*-isolated, for some fixed* $f => 1$, *in particular, it is* $f$*-isolated for* $f > 2.38$, *if* $\Delta$ *very small (see claim 4.8 of appendix 4.8).]*

9. *Apply the winding number algorithm (supporting proposition 4.3.5 of section 4.3) to compute* $i(p(x), \tilde{D})$, *the index of* $p(x)$ *in* $\tilde{D}$. *If* $i(p(x), \tilde{D}) < k$ *(that is, if the disc* $\tilde{D}$ *does not contain all the* $k$ *zeros of* $p(x)$ *lying in the input square* $S$*), go to Stage 10 for* $X = c$. *Otherwise, set* $Y = c + 2\rho$ *and go to Stage 1.*

10. *Shift to the polynomial* $q(y) = p(y + X)$, *set* $s = n + 1 - k$, $\Delta = 0.01$, *and apply algorithm 4.3.2a of section 4.3 to compute an upper bound* $\hat{r}$ *on* $r_s$ *for* $q(y)$, *such that* $r_s \leq \hat{r} \leq 1.01 r_s$. *Then*

    a) *if* $\hat{r} \leq \varepsilon$, *output the disc* $D(X, \hat{r})$ *as a common* $f$*-isolated* $\hat{r}$*-neighborhood of all the* $k$ *zeros of* $p(x)$ *lying in the input square* $S$ *and stop;*

**b)** *otherwise, set* $X^* = X$, $R^* = \hat{r}$, *output the square* $S^* = S(X^*, R^*)$ *[satisfying ( 4.5.1 ) for* $1/e < 0.24$, *that is, for* $e > 25/6$*] and go to Algorithm 4.6.1.*

In both cases the set $\sigma$ is saved for the next application of Algorithm 4.7.1.

Let us next show that $r.r.(D(X,\hat{r})) > 0.24 \geq 1/e$ in the case of the transition to Stage 10 from Stage 6. (Similary, the lower bound 0.24 follows in the case of the transition to Stage 10 from Stage 9; in the case of the transition to Stage 10 from Stages 3 and 5, a similar but simpler argument gives a stronger bound of $r.r.(D(X,\hat{r})) > 0.49$.) By the definition of $\tilde{r}_h$ (see Stage 6), the distance between the two zeros of $p(x)$ that are the closest to $v_h$ and $v_{h+2}$, respectively, is at least $|v_h - v_{h+2}| - \tilde{r}_h = 16r^*(u) - \tilde{r}_h$, which is not less than $r^*(u)$ if $\tilde{r}_h \leq 15r^*(u)$, that is, in the case of the transition from Stage 6 to Stage 10. Therefore, in this case, $d^*(D(X,\hat{r})) \geq r^*(u)$, $X = u$, so that $d^*(D(X,\hat{r})) \geq \hat{r}/2$ if $\hat{r}/2 \leq r^*(u)$. On the other hand, $p(x)$ has a zero $u + cr^*(u)$ for $|c| \leq 1$, so that $d^*(D(X,\hat{r})) \geq r_s - r^*(u) \geq \hat{r}/1.01 - r^*(u)$, for the $r_s$ defined at Stage 10. If $r^*(u) < \hat{r}/2$, then it follows that $d^*(D(X,\hat{r})) > \hat{r}(1/1.01 - 1/2) = \frac{99}{202}\hat{r}$. We have shown that this bound also holds if $r^*(u) \geq \hat{r}/2$. Therefore, $d^*(D(X,\hat{r})) > 0.24\hat{r}$, and hence, due to Proposition 4.2.1, $r.r.(D(X,\hat{r})) > 0.24$.

Correctness of the algorithm now follows from rapid (quadratic) convergence of the value $u$ of ( 4.7.1 ) to the smallest disc, $D_{min}$, equivalent to the input square; such convergence will be proved in appendix 4.8.

**Remark 4.7.1.** *If* $k = n$, *then we do not have to apply Algorithm 4.7.1, since we may first compute and output* $X = -p_{n-1}/(np_n) = \sum_{j=1}^{n} z_j/n$ *and then apply the algorithms of [P87], [P94] to compute and to output a desired approximation* $R^*$ *to* $r_{n+1-k}(X)$.

Algorithm 4.7.1 is recursively applied in its combination with Algorithm 4.6.1, so that all the input squares of Algorithm 4.7.1 can be arranged in the form of a tree with $\mathbf{O}(n)$ nodes. In each recursive step, one may concurrently apply Algorithm 4.7.1 to all the squares (nodes) of the current level of the tree starting with the root, at the first step.

The computational cost of performing Algorithm 4.7.1 is estimated as follows:

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

**Stages 3, 5:** $\mathbf{O}(ng(\Delta)\log n)$ (the cost of $\mathbf{O}(1)$ applications of Algorithm 4.3.3);

**Stages 4, 6, 7, 8:** $\mathbf{O}(1)$;

**Stages 9:** $\mathbf{O}(n\log n)$, since the disc $D$ computed at Stage 8 is $f$-isolated for $f > 2$;

**Stage 10:** $\mathbf{O}(n\log n)$ for an application of algorithms of [P87], [P94] since the input disc is $f$-isolated for $f > 2+6n$; the same estimate, $\mathbf{O}(n\log n)$, holds for the compexity of shifting to the polynomial $q(y) = p(y+X)$.

We have quadratic convergence of the value $u$ of ( 4.7.1 ) to the disc $D_{min}$; furthermore, the approximation to the zeros of $p(x)$ in $D_{min}$ is improved in the subsequent applications of Algorithms 4.6.1 and 4.7.1. Due to the quadratic convergence of Algorithm 4.7.1, we need $\mathbf{O}(\log\log(R/\varepsilon))$ recursive calls to its Stage 2. in order to decrease the approximation error bound from $R$ to $\varepsilon$. Based on this observation, we arrive at the overall cost bound $\mathbf{O}(n^2 g(\Delta)\log n\log\log(R/\varepsilon))$ at the stages of refining the initial approximation of all the zeros of $p(x)$, which turns into the bound $\mathbf{O}(n^2 g(\Delta)\log n\log b)$, for $b = \log(R/\varepsilon)$.

**Remark 4.7.2.** *By applying Remark 4.3.2 for $h = \mathbf{O}(\log\log n)$, we may increase the ratio $f = O(1)$ (say, $f = 4$) to $f \geq 2\sqrt{2} + 6n\sqrt{2}(e-1)/(e-2)$, at the overall cost bounded by $\mathbf{O}(n^2\log n\log\log n)$, for the computation performed simultaneously for several discs that together contain all the zeros of $p(x)$. Thus, remaining within this cost bound, we may always satisfy the assumptions required for the application of Algorithm 4.7.1. Then we may compute isolated $\varepsilon_h^*$-neighborhoods of the $d$-th powers of all the zeros of $p(x+u)$ for a pair or a triple of fixed shift values $u$, for $d = \mathbf{O}(\log n)$, and for appropriate positive $\varepsilon_h^*$. For every $u$ and for every $\varepsilon_h^*$-neighborhood of a zero $z_h^d$ of $q(x)$, we then compute $d$ candidate $\varepsilon_h^*$-neighborhoods of $z_h$, for $\varepsilon_h^* \leq \varepsilon$, of which we need to select a true $\varepsilon_h$-neighborhood of $z_h$. We choose sufficiently small $\varepsilon_h^*$ (they only need to be moderately small relative to $\varepsilon$) and appropriate shifts $u$, so as to identify unique $\varepsilon_h$-neighborhoods of*

*$z_h$, for all $h$, by intersecting the candidate $\varepsilon_h$-neighborhoods (for all fixed $h$ and for all the shift values $u$).*

By combining the bounds of sections 4.6 and 4.7, for $e > 25/6$, $f > 2\sqrt{2} + 6n\sqrt{2}(e-1)/(e-2)$, we arrive at the cost bound $\mathbf{O}(n^2 g(\Delta) \log n \log(bn))$, for approximating all the zeros of $p(x)$ within $\varepsilon = 2^{-b}$. The cost bound can be improved, to $\mathbf{O}(n^2 g(\Delta) \log n \log(b \log n))$, if we apply Algorithm 4.6.1, say, for $f = 8$, and then lift $f$ to $2\sqrt{2} + 6n\sqrt{2}(e-1)/(e-2)$, by means of the recipe of Remark 4.7.2.

To conclude this section, we will show a modification of Algorithm 4.7.1, where the expression ( 4.7.1 ), is replaced by the following one:

$$u = t - \frac{k}{q}, \quad q = \frac{p'(t)}{p(t)} + \dot{\sum_j} \frac{1}{z_j^* - t} \qquad (4.7.3)$$

and where $z_j^*$ denotes the current approximation to the zero $z_j$ of $p(x)$, whereas $\dot{\sum}$ denotes the sum in $j$ over all the natural $j$ corresponding to the zeros $z_j$ of $p(x)$ that lie outside the input square $S$ and the initial disc $D$. Instead of avoiding the points $x$ and $t$ that annihilate $p'(x)$, we shall now avoid the points $t$ that annihilate $q = q(t)$; we will use a strategy similar to one of Algorithm 4.7.1. The presented modification of Algorithm 4.7.1 will be called **Algorithm 4.7.1a** and will always be applied to the largest of the currently unprocessed suspect squares. [This choice should insure the greatest acceleration of the convergence to the disc $D_{min}$.] The analysis shows (see the full version of the thesis) that the quadratic convergence of Algorithm 4.7.1a can be achieved already for the input square $S$ with an isolation ratio $f = f^*\sqrt{2}$, $f^* > 2 + \sqrt{6n(e-1)/(e-2)}$, so that it would suffice to set $e = 5$, $f = 3 + \sqrt{16n}$, or $e = 8$, $f = 3 + \sqrt{14n}$, or $e = 26$, $f = 3 + \sqrt{12.5n}$. To arrive at such a smaller initial isolation ratio, we need roughly by twice fewer iterations of Algorithm 4.6.1. This saving should actually be weighted against the additional arithmetic operations required in order to compute the value $u$ via ( 4.7.3 ) [rather than via ( 4.7.1 )]; for each such an evaluation, we need $3(n-k)$ additional operations.

## 4.8 Figures.

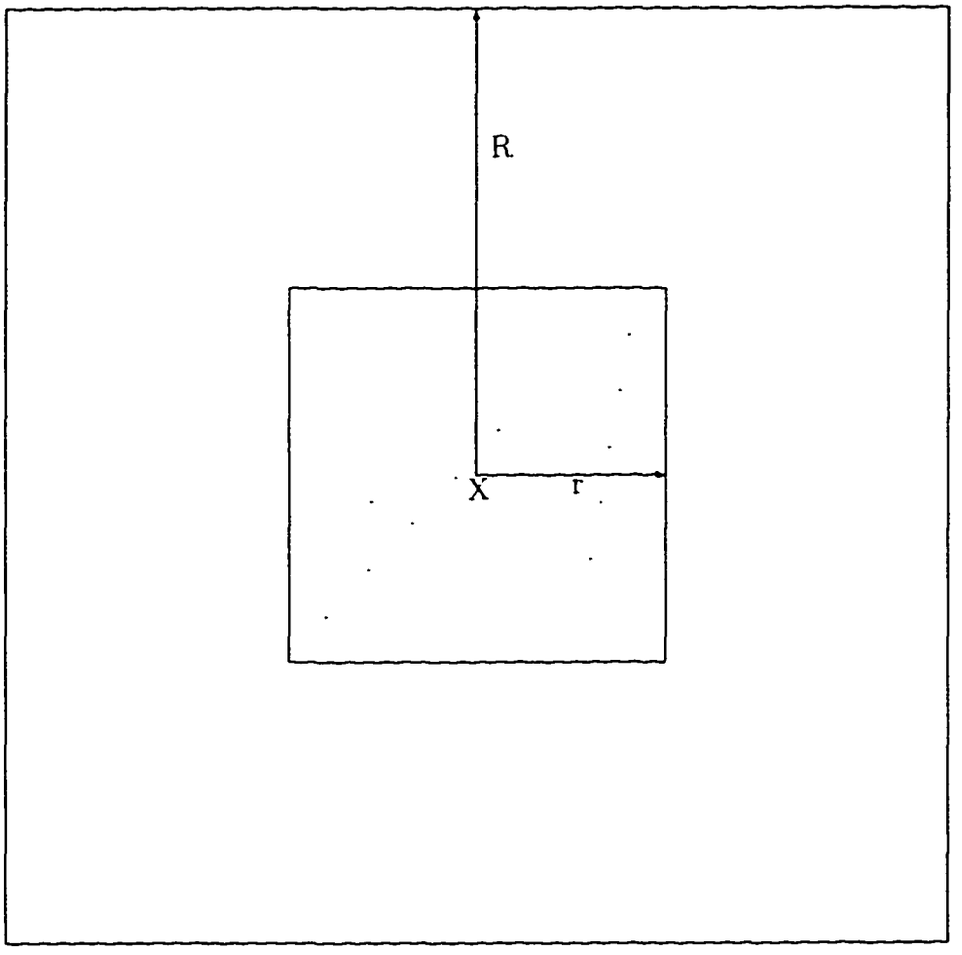


Figure 4.1: Isolation and rigidity ratios for squares.

$i.r.(S(X,r)) \geq R/r, \quad r.r.(S(X,R)) \geq r/R.$

The dots show all the zeros of $p(x)$ lying in the larger square. They also lie in the smaller square.

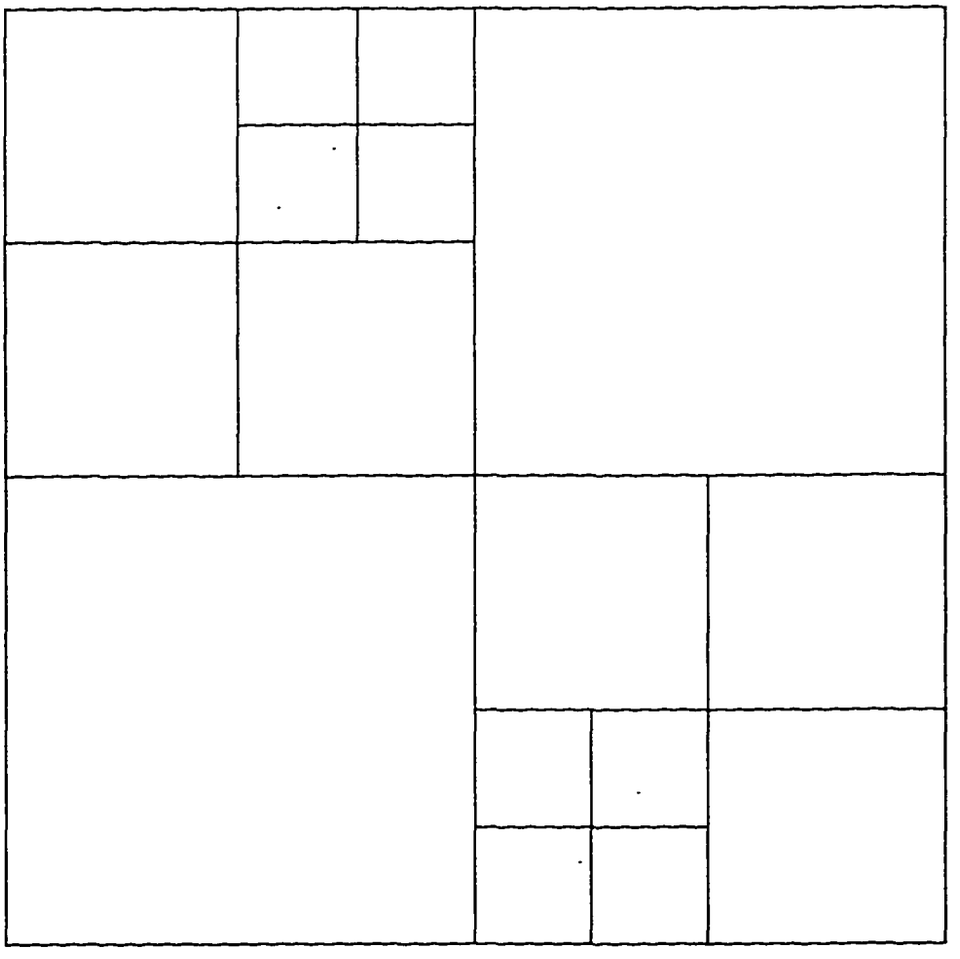

Figure 4.2: Weyl's algorithm.

The dots show all the zeros of $p(x)$ lying in the large square.

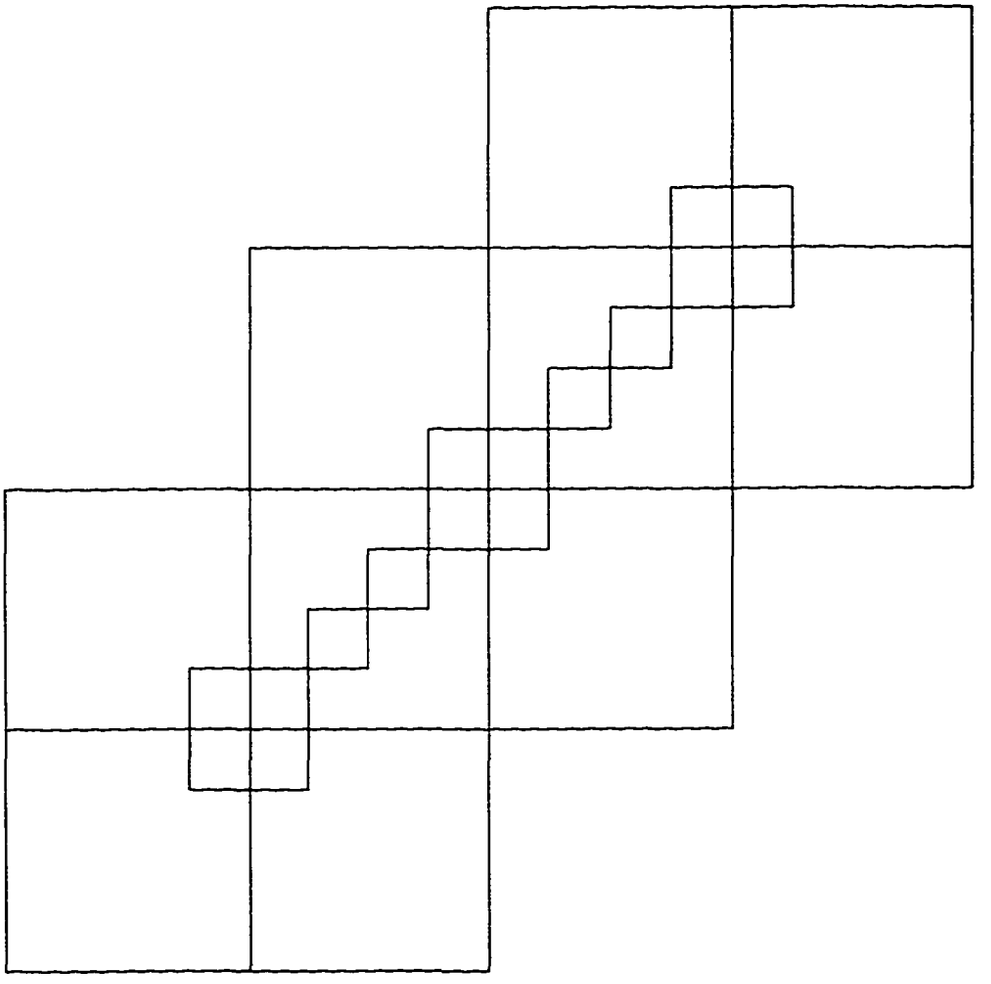

Figure 4.3: 16 smaller suspect squares at Stage $g + 2$ versus 10 larger ones at Stage $g$ of Weyl's algorithm.

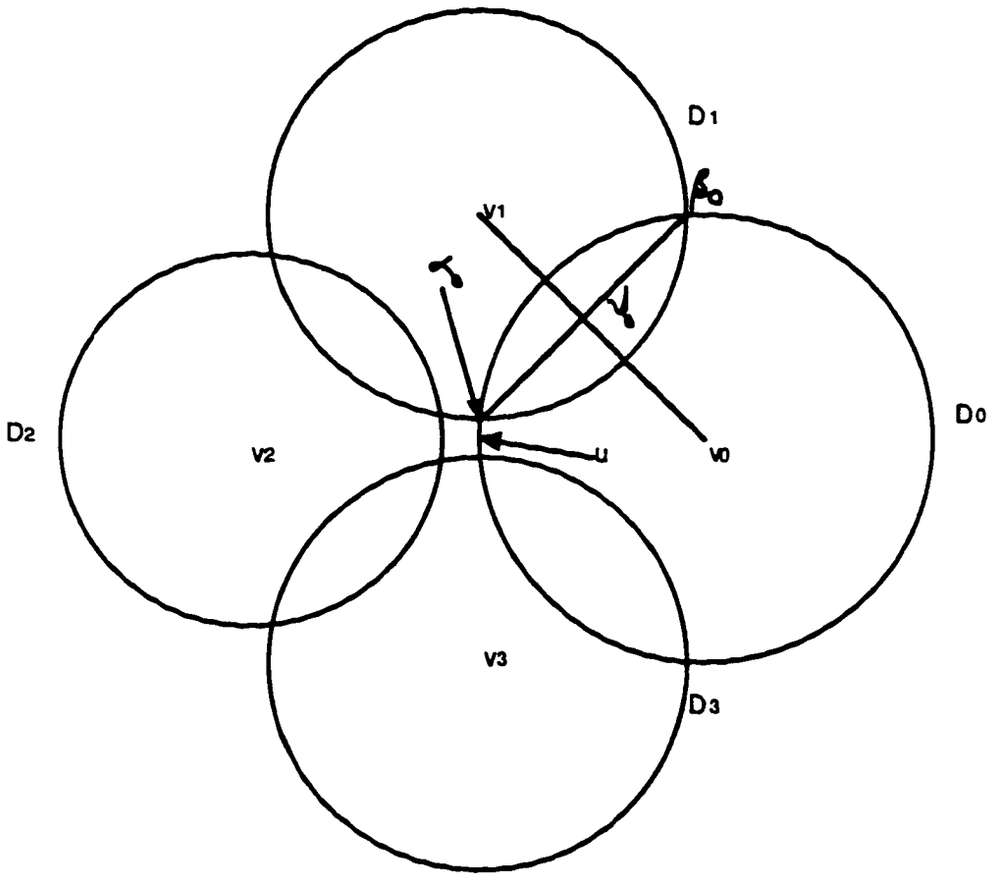


Figure 4.4: Algorithm 4.7.1: case where $r(v_h) + r(v_{h+2 \bmod 4}) > 15r(u)$ for $h = 0, 1$;

$D_h = D(v_h, r(v_h))$ for $h = 0, 1, 2, 3$;

$x_0 = |v_1 - \nu_0|, \quad y_0 = |\nu_0 - v_0|, \quad w_0 = |\alpha_0 - \nu_0| = |\nu_0 - \beta_0|.$

# Appendix A.

## A.1 Proof of Proposition 3.3.1.

Substitute the expressions

$$\frac{1}{v-u_k} = -\frac{1}{u_k}\sum_{h=0}^{\infty}(\frac{v}{u_k})^h$$

and obtain the power series representation

$$p(v) = \sum_{h=0}^{\infty} p_h v^h = \sum_{k=0}^{n-1}\frac{y_k}{v-u_k} = -\sum_{k=0}^{n-1}\frac{y_k}{u_k}\,\frac{1}{1-(v/u_k)}\,, \qquad (\text{A.1.1})$$

$$p_h = -\sum_{k=0}^{n-1}\frac{y_k}{u_k^{h+1}}\,, \quad h = 0,1,\ldots\,. \qquad (\text{A.1.2})$$

$p(v)$ converges when $|v|$ is small enough. Approximate $p(v)$ by the $s$-term partial sum and estimate the error in terms of $s$, $|v_i/u_k|$ and $a$. Due to ( 3.3.4 ) and ( 3.3.5 ), we obtain:

$$\left|p(v_i) - \sum_{h=0}^{s-1} p_h v_i^h\right| \le \frac{anq^s}{1-q} \le \epsilon\,, \quad \text{forall } i\,. \qquad (\text{A.1.3})$$

It remains to compute first $p_0,\ldots,p_{s-1}$ of ( A.1.2 ), by using $(3n-1)s$ operations, and then $\sum_{h=0}^{s-1} p_h v_i^h$ for $i = 0,1,\ldots,s-1$, by using $2ns$ operations. □

# Appendix B.

## B.1 Analysis of Algorithms 4.7.1 and 4.7.1a.

We will next analyze Algorithm 4.7.1 and will prove its correctness. With no loss of generality, we will assume that $x_0 = 0$ and that we are given an $f^*$-isolated disc $D = D(0,R)$ containing exactly $k$ zeros of $p(x)$, $z_1,\ldots,z_k$, $1 \le k \le n$, $R$ being an upper bound on $r_{n+1-k}(0)$. [Indeed, we may reenumerate the zeros of $p(x)$ and may shift from any disc on the complex plane to the disc $D$ by shifting the variable $x$.] Thus we have our initial relations:

$$Y = 2R, \quad |z_j| \le R, \quad j = 1,\ldots,k, \qquad (\text{B.1.1})$$

$$|z_j| \geq f^* R, \quad j = k+1, \ldots, n. \qquad (\text{B.1.2})$$

We now designate that

$$Q(x) = \sum_{j=1}^{k} \frac{1}{x - z_j}, \; V(x) = \sum_{j=k+1}^{n} \frac{1}{x - z_j} \qquad (\text{B.1.3})$$

and deduce the equation

$$p'(x)/p(x) = Q(x) + V(x). \qquad (\text{B.1.4})$$

Since we may choose the value $t$ arbitrarily close to $Y$, we will simplify our analysis by assuming that $t = Y = 2R$.

Now recall that $D_{min}$ denotes the minimum disc that is equivalent to the disc $D$ and contains the zeros $z_1, z_2, \ldots, z_k$ of $p(x)$, so that

$$D_{min} = D(X_{min}, R_{min}), \; R_{min} = (r.r.(D))R, \; X_{min} \in D. \qquad (\text{B.1.5})$$

Let us denote

$$q(x) = x - k/Q(x), \; q = q(t), \qquad (\text{B.1.6})$$

$$\Delta(x) = (x - q(x))V(x)/k, \; \Delta = \Delta(t). \qquad (\text{B.1.7})$$

We will next prove two auxiliary results.

**Lemma B.1.1.** *The complex point $q(x)$ of ( B.1.6 ) lies in the closed disc $D_{min}$ provided that $x$ lies outside $D_{min}$ and that $Q(x) \neq 0$.*

*Proof.* Set $y_x(z) = 1/(x - z)$ and let $D_x = y_x(D_{min})$ denote the image of the disc $D_{min}$, that is, $D_x = \{y_x(z) : \; z \in D_{min}\}$. 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 analytic function theory ([Ah]) and deduce that $D_x$ is a disc. On the other hand, $y_x(z_j) \in D_x$, 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 $D_x$. The inverse map, $y_x^{-1}(w) = x - 1/w$, transforms $D_x$ into the disc $D_{min}$, and therefore, $q(x) = y_x^{-1}(Q(x)/k) \in D_{min}$. □

**Lemma B.1.2.** *$u - q = (t - q)\Delta/(1 + \Delta)$, where $u$, $q$, $t$ and $\Delta$ satisfy the equations ( 4.7.1 ), ( B.1.6 ) and ( B.1.7 ).*

*Proof.* Due to ( 4.7.1 ) and ( B.1.4 ), we have

$$u = u(t) = t - k/(Q(t) + V(t)). \qquad (\text{B.1.8})$$

Deduce from ( B.1.6 ) that $Q(t)/k = 1/(t-q)$, substitute this expression into ( B.1.8 ) and obtain that

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

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

Therefore,

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

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

□

Hereafter denote that

$$R_x = \max_{j=1,\ldots,k} |x - z_j|, \; D_x = D(x, R_x), \; for \; x = t, \; x = u. \qquad (\text{B.1.9})$$

Next obtain the following estimate:

**Lemma B.1.3.** *Let $t \notin D_{min}$, $Q(t) \neq 0$. Then*

$$R_u/R \leq 2R_{min}/R + \frac{(R_t/R)^2\mu}{k - (R_t/R)\mu}, \qquad (\text{B.1.10})$$

*where*

$$\mu = R|V(t)|. \qquad (\text{B.1.11})$$

*Proof.* By definition of $D_x$ in ( B.1.9 ), $z_j \in D_x$, $j = 1, \ldots, k$, and, therefore, $D_{min} \subseteq D_x$, for $x = t$, $x = u$. Consequently, $R_x - 2R_{min} \leq |x - a| \leq R_x$ if $a \in D_{min}$. On the other hand, $q \in D_{min}$, due to Lemma 4.8 and since $q = q(t)$, $t \notin D_{min}$, $Q(t) \neq 0$. Therefore, $R_x - 2R_{min} \leq |x - q| \leq R_x$, for $x = t$ and $x = u$. Combine the former of these inequalities, for $x = u$, with the equation of Lemma 4.8, then apply the latter of them,

for $x = t$, and obtain that $R_u - 2R_{min} \leq |u - q| = |t - q|\ |\Delta/(1+\Delta)| \leq R_t|\Delta/(1+\Delta)|$. To estimate $|\Delta/(1+\Delta)|$ from above, set $x = t$, apply the equations ( B.1.7 ), recall that $|t - q| \leq R_t$, and deduce that

$$|\Delta| = |t - q|\ |V(t)|/k \leq R_t|V(t)|/k,$$

$$1/|1 + \Delta| \leq 1/(1 - |\Delta|) \leq 1/(1 - R_t|V(t)|/k).$$

Combine these bounds on $|\Delta|$ and $1/(1+|\Delta|)$ and obtain that $|\frac{\Delta}{1+\Delta}| \leq \frac{R_t|V(t)|}{k - R_t|V(t)|}$. Substitute the latter inequality into the upper bound on $R_u - 2R_{min}$ above and obtain that

$$R_u - 2R_{min} \leq \frac{R_t^2|V(t)|}{k - R_t|V(t)|},$$

which immediately implies ( B.1.10 ). □

The next lemma extends Lemma 4.8.

**Lemma B.1.4.** *For a fixed positive $e$, let*

$$\theta = \frac{1}{(1 - 2/e)((f^* - 2)k/(n-k) - 3)} \leq 1/3, \qquad (B.1.12)$$

$$R_u \geq eR_{min}. \qquad (B.1.13)$$

*Then*

$$R_u/R \leq \theta(R_t/R)^2 \leq R_t/R\ . \qquad (B.1.14)$$

*Proof.* From ( B.1.10 ) and ( B.1.13 ), obtain that

$$(1 - 2/e)R_u/R \leq \frac{(R_t/R)^2\mu}{k - R_t\mu/R} = \frac{(R_t/R)^2}{(k/\mu) - R_t/R}. \qquad (B.1.15)$$

Next obtain from the equations ( B.1.3 ) and ( B.1.11 ) 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 ( B.1.2 ) and $t = 2R$ imply that $|t - z_j| \geq (f^* - 2)R$, for $j > k$. Therefore,

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

Substitute this bound into ( B.1.15 ) and obtain that

$$(1-2/e)R_u/R \le \frac{(R_t/R)^2}{(f^*-2)k/(n-k)-(R_t/R)}.$$

We have $R_t \le 3R$, since $t = 2R$, and will choose $f^*$ such that $(f^*-2)k/(n-k) > 3$. Hence we obtain that

$$(1-2/e)R_u/R \le \frac{(R_t/R)^2}{(f^*-2)k/(n-k)-3}.$$

Substitute the expression for $\theta$ from ( B.1.12 ) and obtain that $R_u/R \le \theta(R_t/R)^2$, and, therefore, $R_u/R \le R_t/R$, since $\theta \le 1/3$ and $R_t \le 3R$. This completes the proof of ( B.1.14 ). □

Correctness of Algorithm 4.7.1 follows, and, moreover, $\mathbf{O}(\log\log(R/\varepsilon))$ its recursive steps suffice, since, for $f^* > 2+6n(e-1)/(e-2)$ and for $e > 2$, the inequality of ( B.1.12 ) holds, and we may recusively apply Lemma 4.8.

Finally, we will analyze the convergence of the computed values $u$ to the disc $D_{min}$ provided that we shift from Algorithm 4.7.1 to Algorithm 6.1a, by replacing the equations ( 4.7.1 ) by ( 4.7.3 ). The transition from ( 4.7.1 ) to ( 4.7.3 ) amounts to substracting $V^*(x) = \sum_{j=k+1}^{n} \frac{1}{x_j^*-x}$ from both sides of the equation ( B.1.4 ), which gives us that

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

The analysis of Algorithm 4.7.1 is extended, with the replacement of $V(x) = \sum_{j=k+1}^{n} \frac{1}{x-z_j}$ by

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

The resulting increase of the estimated convergence rate is quantitatively expressed by the following relations for the main parameter $\theta$ of Lemma 4.8:

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

In other words, the statement of this basis lemma should remain unchanged, except that, in the expressions for $\theta$, the quantity $f^*-2$ should be replaced by $(f^*-2)^2$. This enables

us to preserve the quadratic convergence of the algorithm even where the upper bound on the initial isolation ratio $f^*$ for the disc $D$ decreases from $f^* > 2 + 6n(e-1)/(e-2)$, for Algorithm 4.7.1, to

$$f^* > 2 + \sqrt{6n(e-1)/(e-2)}, \qquad (\text{B.1.16})$$

for Algorithm 6.1a. This enables us to save about 50% of the preceeding iterations of Algorithm 5.1, for large $n$, at the expense of performing $3(n-k)$ additional operations, for each evaluation of $u$ via ( 4.7.3 ), rather than via ( 4.7.1 ).

## B.2 Isolation Ratio of an Auxiliary Disc.

**Claim B.2.1.** *The disc $\tilde{D}$ defined at Stage 8 of Algorithm 4.7.1 is $f$-isolated for $f > 2.38$ provided that $\Delta$ is chosen sufficiently small in the applications of Algorithm 4.4.1 to Algorithm 4.7.1.*

*Proof.* Note that, in the applications of Algorithm 4.3.3, we have $r^*(X) \longrightarrow r(X) = r_n(X)$, for any complex $X$, as $\Delta \longrightarrow 0$. Since $r^*(X)$ lies close to $r(X) = r_n(X)$, for any complex $X$ and for a sufficiently small $\Delta$, we will simplify our estimates and our notation by assuming that $r^*(X) = r(X)$, for all $X$, throughout the proof. In particular, we will write $r(u)$ instead of $r^*(u)$ and $r(v_h)$ instead of $r_n(v_h)$ and $r^*(v_h)$, $h = 0, 1, 2, 3$. We will now use these assumption and definitions in the following relations, basic for our proof:

$$r(v_h) \leq |v_h - u| + r(u) = 9r(u), \ h = 0, 1, 2, 3, \qquad (\text{B.2.1})$$

$$r(v_h) + r(v_{h+2 \bmod 4}) = \tilde{r}_h > 15r(u), \ h = 0, 1 \qquad (\text{B.2.2})$$

(see description of Stage 6 of Algorithm 4.7.1). Consequently, we have

$$r(v_h) \geq 6r(u), \ h = 0, 1, 2, 3. \qquad (\text{B.2.3})$$

Hereafter, for $h = 0, 1, 2, 3$, let $\nu_h$ denote the intersection point of the two line segments, $[\alpha_h, \beta_h]$ and $[v_h, v_{h+1 \bmod 4}]$ (see Figure 4). We will also designate that

$$x_h = |v_{h+1 \bmod 4} - \nu_h|,$$

$$y_h = |v_h - \nu_h|,$$

$$w_h = |\alpha_h - \nu_h| = |\beta_h - \nu_h|,$$

for $h = 0, 1, 2, 3$ (see Figure 4) and will let $\omega_h$ denote the angle between the lines passing through the pairs of points $(v_h, v_{h+1 \bmod 4})$ and $(v_{h+1 \bmod 4}, \alpha_h)$, so that $x_h = r(v_{h+1 \bmod 4}) \cos \omega_h$, $w_h = r(v_{h+1 \bmod 4}) \sin \omega_h$. Our next objective is to prove that

$$m_h \sqrt{2} = \min\{x_h + w_h, y_h + w_h\} > 5.951198575 r(u). \qquad (\text{B.2.4})$$

We first observe that

$$(x_h + w_h)^2 = x_h^2 + w_h^2 + 2x_h w_h = r^2(v_{h+1 \bmod 4})\,(1 + 2 \sin \omega_h \cos \omega_h) =$$

$$r^2(v_{h+1 \bmod 4})\,(1 + \sin(2\omega_h)) \le 36 r^2(u)\,(1 + \sin(2\omega_h))$$

[compare ( B.2.3 )]. We now note that $\sin(2\omega_h)$ takes on its minimum value where $r(v_h) = 6r(u)$, $r(v_{h+1 \bmod 4}) = 9r(u)$. Therefore, $m_h \sqrt{2}$ is bounded from below by $x_h + w_h$ provided that $x_h$, $y_h$ and $w_h$ satisfy

$$r^2(v_h) = x_h^2 + w_h^2 = 36 r^2(u),$$

$$r^2(v_{h+1 \bmod 4}) = y_h^2 + w_h^2 = 81 r^2(u).$$

Substracting these equations from each other gives us that

$$y_h^2 - x_h^2 = 45 r^2(u).$$

Recall ( 4.7.2 ) and obtain that

$$y_h + x_h = 8\sqrt{2} r(u), \quad y_h - x_h = 45 r^2(u)/(y_h + x_h) = 45 r(u)/(8\sqrt{2}).$$

From these two equations we have

$$x_h = (4\sqrt{2} - 45\sqrt{2}/32) r(u) = 83 r(u) \sqrt{2}/32.$$

Consequently,

$$w_h^2 = 36 r^2(u) - x_h^2 = (36 - (83)^2/2^9) r^2(u) = 11543 r^2(u)/2^9,$$

$$(w_h + x_h)/\sqrt{2} = (83 + \sqrt{11543})r(u)/32 > 5.951198575r(u),$$

which implies a lower bound of ( B.2.4 ) on $m_h$ for all $h$. On the other hand, for the radius $\rho$ of the disc $\tilde{D}$ defined at Stage 8 of Algorithm 4.7.1, we have

$$\rho \leq (8r(u) - \min_h m_h)\sqrt{2},$$

and therefore,

$$\rho < 2.8975r(u). \qquad (\text{B.2.5})$$

Now let $d$ denote the minimum distance from the disc $\tilde{D}$ to the points $\beta_0$, $\beta_1$, $\beta_2$ and $\beta_3$. Then

$$d \geq 2 \min_h w_h. \qquad (\text{B.2.6})$$

On the other hand, the distance from $\tilde{D}$ to the nearest zero of $p(x)$ lying outside $\tilde{D}$ is at least $d$ (see Figure 4). Therefore,

$$i.r.(\tilde{D}) \geq 1 + d/\rho. \qquad (\text{B.2.7})$$

Finally, we observe that $w_h$ reaches its minimum value, where

$$x_h = y_h = (4\sqrt{2})r(u),$$

$$r(v_h) = r(v_{h+1 \bmod 4}) = 6r(u)$$

(see Figure 4). In this case,

$$w_h^2 = r^2(v_{h+1 \bmod 4}) - x_h^2 = (36 - (4\sqrt{2})^2)r^2(u) = 4r^2(u), \; w_h = 2r(u).$$

By combining the latter lower bound on $w_h$ with ( B.2.5 )-( B.2.7 ), we obtain that

$$i.r.(\tilde{D}) \geq 1 + 4r(u)/\rho > 2.38,$$

which proves Claim 4.8. □

# Bibliography

[AG] G.S. Ammar, W.G. Gragg, Superfast solution of real positive definite Toeplitz systems, *SIAM J. on Matrix Analysis and Applications*, 9, pp.61–76, 1988.

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

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

[ASU] A. Aho, K. Steiglitz, J. Ullman, Evaluating polynomials at fixed sets of points, *SIAM J. on Computing*, 4, pp. 533–539, 1975.

[Atk] K. Atkinson, *Introduction to Numerical Analysis* Wiley, New York, 1978.

[B82] D. Bini, Parallel Solution of Certain Toeplitz Linear Systems, Tech. Report, *Department of Computer Science, University of Pisa*, Pisa, Italy, April 1982.

[B84] D. Bini, Parallel Solution of Certain Toeplitz Linear Systems, *SIAM J. Compit.*, 13, 2, pp. 268–276, 1984.

[BA] R.R. Bitmead, B.D.O. Anderson, Asymptotically fast solution of Toeplitz and related systems of linear equations, *Linear Algebra and Its Applications*, 34, pp. 103–116, 1980.

[Baase] A. Baase, *Computer Algorithms: Introduction to the Design & Analysis*, Addison-Wesley, Reading, Mass., 1988.

[Ben-Or83] M. Ben-Or, Lower Bounds for Algebraic Computation Trees, *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pp. 80–86, ACM Press, New York, 1983.

[BFKT] 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, 6, pp. 1081–92, 1989 (proceedings version in *Proc. 18th Ann. ACM Symp. on Theory of Computing*, pp. 340–349, ACM Press, New York, 1986).

[BGY] R.P. Brent, F.G. Gustavson, D.Y.Y. Yun, Fast solution of Toeplitz systems of equations and computations of Padé approximations, *J. of Algorithms*, 1, pp.259–295, 1980.

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

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

[BP86] D. Bini, V. Y. Pan, Polynomial Division and Its Computational Complexity, *J. of Complexity*, 2, pp. 179–203, 1986.

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

[BP93] D. Bini, V. Y. Pan, Improved Parallel Polynomial Division, *SIAM J. on Computing*, 22, 3, pp. 617–626, 1993.

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

[BP,a] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations*, volume 2, Birkhauser, Boston, to appear.

[BS] W. Baur, V. Strassen, On the Complexity of Partial Derivatives, *Theor. Computer Science*, 22, pp. 317-330, 1983.

[BT] W. S. Brown, J. F. Traub, On Euclid's Algorithm and the Theory of Subresultants, *J. of ACM*, 18, 4, pp. 505–514, 1971.

[Bu85] J.R. Bunch, Stability of methods for solving Toeplitz systems of equations, *SIAM J. on Scientific and Statist. Computing*, 6, pp. 349–364, 1985.

[CdB] C. D. Conte, C. de Boor, *Elementary Numerical Analisis: an Algorithmic Approach*, McGraw-Hill, New York. 1980.

[ChKLe] J. Chun, T. Kailath, H. Lev-Ari, Fast Parallel Algorithm for QR-Factorization of Structured Matrices, *SIAM J. of Scient. and Stat. Computing*, 8, pp.889-913, 1987.

[CKL] J.F. Canny, E. Kaltofen, Y. Lakshman, Solving systems of non-linear polynomial equations faster, *Proc. ACM-SIGSAM Int. Symp. on Symb. and Algebraic Computing*, pp. 121–128, 1989.

[DeL] R. A. Demillo, R. J. Lipton. A Probabilistic Remark on Algebraic Program Testing, *Information Process. Letters*, 7, 4, pp. 193–195, 1978.

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

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

[EG] D. Eppstein, Z. Galil, Parallel Algorithmic Techniques for Combinatorial Computation, *Annual Review of Computer Science*, 3, pp. 233–283, 1988.

[F81] L. V. Foster, Generalizations of Laguerre's Method: Higher Order Methods, *SIAM J. on Numer. Analysis*, 18, 1004–1018, 1981.

[Fi87] C. M. Fiduccia, A Rational View of the Fast Fourier Transform, *Proc. 25th Allerton Conf. Commun., Control and Computing*, 1987.

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

[G] L. Greengard, *Rapid Evaluation of Potential Fields in Particle Systems*, M.I.T. Press, Cambridge, 1988.

[Ga82] I. Gargantini, An Effective Way to Represent Quadtrees, *Communication of the ACM*, 25, 12, pp. 905–910, 1982.

[Gathen86] J. von zur Gathen, Representations and Parallel Computations for Rational Functions, *SIAM J. Comput.*, 15, 2, pp. 432–452, 1986.

[Gr] W. B. Gragg, The Padé Table and Its Relation to Certain Algorithms of Numerical Analysis, *SIAM Review*, 14, 1, pp. 1–62, 1972.

[GS] W. Gentleman, G. Sande, Fast Fourier Transforms for Fun and Profit, *Proc. AFIPS Fall Joint Comput. Conf.*, 29, pp. 563–578, 1966.

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

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

[HG] P. Henrici, I. Gargantini, Uniformly Convergent Algorithms for the Simultaneous Approximation of All Zeros of a Polynomial, 1969 (in *DH*).

[Ho87] H. de Hoog, On the solution of Toeplitz systems, *Linear Algebra and Its Applications*, 88/89, pp. 899-913, 1987.

[HPR] E. Hansen, M. Patrick, J. Rusnack, Some Modifications of Laguerre's Method, *BIT*, 17, 409–417, 1977.

[HS79] G. M. Hunter, K. Steiglitz, Operations on Images using Quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1, 2, pp. 145–153, 1979.

[JJ] J. Ja Ja, *An Introduction to Parallel Algorithms*, Addison-Wesley, Masschusetts, 1992.

[JT] 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, 252–263, 1970.

[Kir] P. Kirrinnis, Polynomial Factorization and Partial Fraction Decomposition by Simultaneous Newton's Iteration, extended abstract, 1994.

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

[KP] E. Kaltofen, V.Y. Pan, Processor Efficient Parallel Solution of Linear Systems over an Abstract Field, *Proc. 3rd Annual ACM Symp. on Parallel Algorithms and Architectures*, pp. 180-191, ACM Press, New York, 1991.

[KR] R. Karp, V. Ramachandran, A Survey of Parallel Algorithms for shared Memory Machines, *Handbook for Theoretical Computer Science (J. van Leeuwen Editor)*, pp. 869–941, Noth Holland, Amsterdam, 1990.

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

[L] S. Linnainmaa, Taylor Expansion of the Accumulated Rounding Errors, *BIT*, 16, pp. 146-160, 1976.

[M] M. Mignotte, An Inequality about Factors of Polynomials, *Math. of Computations*, 28, pp. 1153–1157, 1974.

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

[MN93] J. M. McNamee, A Bibliography on Roots of Polynomials, *J. of Computational and Applied Math.*, 47, 3, 391–394, 1993.

[Mu] B.R. Musicus, *Levinson and fast Choleski algorithms for Toeplitz and almost Toeplitz matrices* , Internal Report, M.I.T. Res. Lab. for Electronics, 1981.

[N] C. A. Neff, Specified Precision Polynomial Root Isolation Is In NC, *Journal of Computer and System Sciences*, 48, pp. 429–463, 1994.

[NR] C.A. Neff, J.H. Reif, An $O(n^{1+\epsilon} \log b)$ Algorithm for the Complex Root Problem, *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 540–547, IEEE Computer Society Press, 1994.

[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*, pp. 522–531, IEEE Computer Science Press, 1985.

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

[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, 11, pp. 1475–1480, 1989.

[P89a] V. Y. Pan, Fast Evaluation and Interpolation at the Chebyshev Set of Points, *Appli. Math. Lett.*, 2, 2, pp. 255–258, 1989.

[P90] V. Y. Pan, Parallel Least-Squares Solution of General and Toeplitz-like Linear Systems, *Proc. 2nd Ann. ACM Symp. on Parallel Algprithms and Architectures*, pp. 244–253, ACM Press, New York, 1990.

[P90a] V. Y. Pan, Computations with dense structured matrices, *Math. of Computations*, 55, pp. 179–190, 1990.

[P92a] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, 34, 2, pp. 225–262, 1992.

[P92b] V. Y. Pan, New Resultant Inequalities and Complex Polynomial Factorization, *Proc. ISTCS' 92, Springer's Lecture Notes in Computer Science*, 601, pp. 122–136, 1992 and *SIAM J. on Computing*, 23, 5, pp. 934–950, 1994.

[P92c] V. Y. Pan, Approximate evaluation of a polynomial on a set of real points, *Tech. Report, International Computer Science Institute, Berkeley, CA*, 1992.

[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*, pp. 260-270, ACM Press, New York, and SIAM Publications, Philadelphia, 1994.

[P95] V.Y. Pan, Optimal and Nearly Optimal Algorithms for Approximating Complex Polynomial Zeros, Computer and Math. (with Applics), to appear (Proceedings version in *Proc. 27th Ann. ACM Symp. on Theory of Computing*, ACM Press, New York, 1995).

[P95a] V.Y. Pan, An Algebraic Approach to Approximate Evaluation of a Polynomial on a Set of Real Points, *Advances in Computational Math.*, 3, pp. 41–58, 1995.

[P] V. Y. Pan, Weyl's Quadtree Algorithm for the Unsymmetric Eigenvalue Problem, *Applied Math. Letters*, to appear.

[PL] V. Y. Pan, E. Linzer, A New Approach to Bisection Acceleration for the Symmetric Eigenvalue Problem, preprint, 1995.

[PP] V.Y. Pan, F. P. Preparata, Work-Preserving Speed-up of Parallel Matrix Computations, *SIAM J. on Computing*, 24, 4, 1995 (Proc. version: Super-effective Slow-down of Parallel Computations, in *Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures*, pp. 402–409, ACM Press, NY, 1992).

[PR] V. Y. Pan, J. H. Reif, Some Polynomial and Toeplitz Matrix Computations, *Proc. 28th Ann. IEEE Symp. on Foundation of Computer Science*, pp. 173–184, IEEE Computer Science Press, 1987.

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

[PSL] V.Y. Pan, A. Sadikou, E. Landowne, Univariate Polynomial Division with a Remainder by Means of Evaluation and Interpolation, *Proc. of the 3rd IEEE Symp. on Parallel and Distributed Processing*, pp. 212–217, IEEE Computer Society Press, 1991.

[PSL,a] V.Y. Pan, A. Sadikou, E. Landowne, Polynomial Division with a Remainder by Means of Evaluation and Interpolation, *Information Process. Letters*, 44, pp. 149-153, 1992.

[PSLT] V.Y. Pan, A. Sadikou, E. Landowne, O. Tiga, A New Approach to Fast Polynomial Interpolation and Multipoint Evaluation, *Computers and Math. (with Applications)*, 25, 9, pp. 25–30, 1993.

[Q] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.

[Q94] M. J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, New York, 1994.

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

[Rok85] V. Rokhlin, Rapid solution of integral equations of classical potential theory, *J. of Comput. Physics*, 60, pp. 187–207, 1985.

[Rok88] V. Rokhlin, A Fast Algorithm for the Discrete Laplace Transformation, *J. of Complexity*, 4, pp. 12–32, 1988.

[RT] J. H. Reif, S. R. Tale, Dynamic Algebraic Algorithms, *Proc. 5th Ann. Ann. ACM-SIAM Symp. on Discrete Algorithms*, pp. 290–299, ACM Press, New York, and SIAM Publications, Philadelphia, Pennsylvania, 1994.

[S81] S. Smale, The Fundamental Theorem of Algebra and Complexity Theory, *Bull. of the Amer. Math. Soc.*, 4, 1, pp. 1–36, 1981.

[S85] S. Smale, On the Efficiency of Algorithms of Analysis, *Bull. of the Amer. Math. Soc.*, 13, 2, pp. 87–121, 1985.

[Sa81] H. Samet, Computing Perimeters of Regions in Images Represented by Quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3, 6, 1981.

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

[Sa] J. E. Savage, *The Complexity of Computing*, Wiley and Sons, New York, 1976.

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

[Sc82a] A. Schönhage, Asymptotically Fast Algorithms for the Numerical Multiplication and Division of Polynomials with Complex Coefficients, *Proc. EUROCAM (J. Calmet Edition)*, Marseille, 1982, *Lecture Notes in Computer Science*, 144, pp. 3–15, Springer, Berlin, 1982.

[Sc85] A. Schönhage, Quasi-GCD Computations, *J. of Complexity*, 1, pp. 118–137, 1985.

[Schw] J.T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities,*J. of ACM*, 27, pp. 701–717, 1980.

[Schr] J. Schröder, Uber Newtonsche Verfahren, *Arch. Rat. Mech. Anal.*, 1, pp. 154-180, 1957.

[Se94] H. Senoussi, A Quadtree Algorithm for Template Matching on Pyramid Computer, *Theoretical Computer Science*, 133, pp. 387–417, 1994.

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

[SS,a] 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, pp. 267–285, Birkhauser, 1993.

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

[SS,c] M. Shub, S. Smale, Complexity of Bezout's Theorem IV: Probability of Success, Extensions, Research Report RC 18921, *IBM T. J. Watson Research Center*, Yorktown Heights, New York, 1993.

[SS,d] M. Shub, S. Smale, Complexity of Bezout's Theorem V: Polynomial Time, Report No. 236, *Centre de Recerca Matemàtica, Institut d'Estudis Catalanas*, Barcelona, Spain, 1993.

[Str81] V. Strassen, Guassian Elimination is not Optimal, *Numer. Math.*, 13, pp. 354–356, 1969.

[Str81] V. Strassen, The Computational Complexity of Continued Fractions, *SIAM J. on Computing*, 12, pp. 1–27, 1981.

[To] A. L. Toom, The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers, *Soviet Math. Dokady*, 3, pp. 714–716, 1963.

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

[vdW] B.L. van der Waerden, *Modern Algebra*, Frederick Ungar Publisher, New York, 1953.

[W24] H. Weyl, Randbemerkungen zu Hauptproblemen der Mathematik, II, Fundamentalsatz der Algebra and Grundlagen der der Mathematik, *Math. Z.*, 20, pp. 131–151, 1924.

[Wer84] W. Werner, Polynomial Interpolation: Lagrange versus Newton, *Math. Comp.*, 43, pp. 205–217, 1984.

[Wilf] H. S. Wilf, A Global Bisection Algorithm for Computing the Zeros of Polynomials in the Complex Plane, *J. ACM*, 25, 415–420, 1978.

[Z79] R.E. Zippel, Probabilistic algorithms for sparse polynomials, *Springer Lecture Notes in Computer Science* 72, 216–226, 1979.