An Efficient Algorithm for the Complex Roots Problem

C. Andrew Neff * John H. Reif †

Abstract

Given a univariate polynomial $f(z)$ of degree n with complex coefficients, whose norms are less than 2^m in magnitude, the root problem *is to find all the roots of $f(z)$ up to specified precision $2^{-\mu}$. Assuming the arithmetic model for computation, we provide an algorithm which has complexity $O(n \log^5 n \log b)$, where $b = m + \mu$. This improves on the previous best known algorithm of Pan for the problem which has complexity $O(n^2 \log^2 n \log b)$. A remarkable property of our algorithm is that it does not require any assumptions about the root separation of f, which were either explicitly, or implicitly, required by previous algorithms. Moreover it also has a work efficient parallel implementation. We also show that both the sequential and parallel implementations of the algorithm work without modification in the Boolean model of arithmetic. In this case, it follows from root perturbation estimates that we need only specify $\theta = \lceil n(b + \log n + 3) \rceil$ bits of the binary representations of the real and imaginary parts of each of the coefficients of f.*

*Stratasys, Inc., 8 Skyline Drive, Hawthorne, NY 10532-2164. E-mail: andy@lego.ny.stratasys.com

†Department of Computer Science, Duke University, Durham, NC 27708-0129. Research also done in part at School of Computer Science, Carnegie Mellon University, Pittsburgh,PA 15213-2890. E-mail: reif@cs.duke.edu. Supported by NSF Grant NSF-IRI-91-00681, Rome Labs Contracts F30602-94-C-0037, ARPA/SISTO contracts N00014-91-J-1985, and N00014-92-C-0182 under subcontract KI-92-01-0182.

We also show that by appropriate rounding of intermediate values, we can bound the number of bits required to represent all complex numbers occurring as intermediate quantities in the computation. The result is that we can restrict the numbers we use in every basic arithmetic operation to those having real and imaginary parts with at most ϕ bits, where

$$\phi = \pi + 2n^2m + 3(n + \log n + 1) + n^2(1 + 2\log n) + \log\log n$$

and

$$\pi = \mu + 2n + nm + \log\log n + 5$$

Thus, in the Boolean model, the overall work complexity of the algorithm is only increased by a multiplicative factor of $M(\phi)$ (where $M(\psi) = O(\psi(\log\psi)\log\log\psi)$ is the bit complexity for multiplication of integers of length ψ)

The key result, on which the algorithm is based, is a new theorem of Coppersmith and Neff relating the geometric distribution of the zeros of a polynomial to the distribution of the zeros of its high order derivatives, and introduce several new techniques (splitting sets and 'centered' points) which hinge on it.

We also observe that our root finding algorithm can be efficiently parallelized to run in parallel time $O(\log^6 n \log b)$ using n processors.

1 Introduction

In this paper we shall consider the computational version of the ubiquitous *Fundamental Theorem of Algebra*, which of course says that for any polynomial $f \in \boldsymbol{C}[z]$ of degree n, there are exactly n complex numbers $z_1, \ldots, z_n$ such that

$$f(z) \;=\; a\prod_{i=1}^{n}(z - z_i)$$

where a is the leading coefficient of f. The problem which we solve in this paper – known as *The Complex Roots Problem* – is to find arbitrarily accurate approximations, w_i, to the actual roots, z_i, of f, using only elementary arithmetic operations and comparisons. Moreover, the goal is to do this as efficiently as possible, that is, to perform as few of the elementary operations (steps) as possible.

The number of steps will obviously depend on the nature of the input polynomial and on the precision required, so we need some measure of the "size" of a problem instance, and will then seek to put an upper bound on the number of steps required which is a function of the size. To that end, we fix our input form as follows.

Let $f(z) \in \boldsymbol{C}[z]$ be a *monic* univariate polynomial of degree n with coefficients over the complex numbers $\boldsymbol{C}$, written as

$$f(z) = z + \sum_{i=0}^{n-1} c_i z^i \ .$$

Let m be the smallest integer such that

$$|c_i| < 2^m$$

for all $0 \leq i \leq n-1$, and let $z_1, \ldots, z_n$ be the (unknown) complex roots of f. Let μ be the required precision, that is we require that our *output* have the form $[w_1, \ldots, w_n]$ such that for each $1 \leq i \leq n$

$$|z_i - w_i| \ \leq \ 2^{-\mu} \, .$$

(For convenience of notation, and for historical reasons, we also introduce the quantity $b = m + \mu$.)

In this paper, we will give an algorithm which solves the complex roots problem above, that is given the sequence of input coefficients $c_0, \ldots, c_{n-1}$ it outputs the complex numbers $w_1, \ldots, w_n$, and show that, assuming each elementary arithmetic operation $+, -, *, /$ and comparison can be performed

exactly, and in one step (the arithmetic computation model), then the number of steps taken by the algorithm is bounded above by a fixed absolute constant multiple (implicit in the paper, but for the sake of brevity, not explicitly evaluated) of the quantity $n \log^5 n \log b$. That is, the *arithmetic complexity* of the algorithm is $O(n \log^5 n \log b)$.

In the boolean model of computation, the number of steps required to compute each elementary arithmetic operation grows with the size of the operands, but we will show that the problem can be solved in its entire generality, by performing *approximate* arithmetic operations (i.e. compute, then round) on numbers whose bit lengths are bounded by ϕ, where

$$\phi = \pi + 2n^2 m + 3(n + \log n + 1) + n^2(1 + 2 \log n) + \log \log n$$

and

$$\pi = \mu + 2n + nm + \log \log n + 5$$

and hence the *boolean complexity* of the algorithm is only larger than its arithmetic complexity by a factor of $M(\phi)$, where $M(\psi) = O(\psi(\log \psi) \log \log \psi)$ is the complexity of multiplying two ψ bit integers.

To put this result in proper context, we mention some of the history of solutions to this problem. The early algorithm of [GH 72] had arithmetic complexity $O(n^3 b)$. Although the algorithm of Schönhage [S 82] was not analyzed in the arithmetic model, it can be seen to require $O(n^3 \log^{O(1)}(bn))$ arithmetic operations. Renegar [R 87] gave a $O((n+\log b)n^2 \log n)$ arithmetic time algorithm, which is $O(n^3 \log n)$ in the case of precision $b \leq n^{O(1)}$, but is an improvement in the case of extremely large (super-polynomial) precision. Pan [P 87] gave an $O(n^2 \log b \log n)$ arithmetic time algorithm, which has been the best known bound for the arithmetic complexity of the Complex Roots Problem.

We do not suggest that the algorithm in this paper, at least in its current form, presents an immediate replacement for any of the best numerical routines currently used in practical implementations (see, for example [JT 70]), however, it does take a big step towards unifying theory and practice in the area of complex root finding. There seems some hope that future simplifications of this presentation, or perhaps, some of the techniques herein, may lead to improvements in actual implementations.

1.1 Organization of the Paper

The Introduction is given in Section 1. Section 2 defines splitting sets and centered points used in our polynomial factorizations. Section 3 describes algorithms for approximate splitting of a polynomial. Section 4 describes how to find an isolated balanced factorization, and Section 5 gives an analysis of the full root algorithm. Section 6 briefly discusses the parallelization of our root finding algorithm. Section 7 concludes the paper with a discussion of related work and open problems.

2 Notation

In order to keep careful track of the errors introduced by rounding and approximation, we need to introduce some notation.

Definition 2.1 *Let z be a complex number. We denote by $\eta(z) \geq 0$ the minimum integer with the property that $2^{\eta(z)}z$ is a complex integer – that is, has real and imaginary parts which are both integers. If no such integer exists, we write $\eta(z) = \infty$.*

Definition 2.2 *For t and l positive integers, let $\boldsymbol{Q}_t$ be the subset of complex numbers z such that $\eta(z) < t$. Let $\boldsymbol{Q}_t^l$ be the subset of $\boldsymbol{Q}_t$ consisting of those z satisfying $|z| < 2^l$.*

Definition 2.3 *Let $\mathcal{P}$ be the set of monic polynomials with complex coefficients. With t and l as above, let $\mathcal{P}_t$ subset of $\mathcal{P}$ having all coefficients in $\boldsymbol{Q}_t$, and let $\mathcal{P}_t^l$ be the subset of polynomials having all coefficients in $\boldsymbol{Q}_t^l$.*

Definition 2.4 *Let z_i be the roots of f. We let*

$$\rho_u(f) = \max_i\{|z_i|\}$$

and

$$\rho_l(f) = \min_i\{|z_i|\}$$

and call each, respectively, the upper and lower root radius of f.

Definition 2.5 *Let $\mathcal{R}_t : \boldsymbol{Q} \to \boldsymbol{Q}_t$ be the rounding operator, and extend the domain to $\mathcal{P}$ coefficientwise.*

We will use the l^∞ coefficient norm on our spaces of polynomials, that is

$$|f| = \max_i\{|c_i|\}$$

where the c_i are the coefficients of f.

One easily computable measure of distance between two polynomials f and g, is then $|f - g|$. For us, the more salient measure of distance is the *root perturbation*, and hence we are led to

Definition 2.6 *The root distance, $\Delta(f, g)$ between two polynomials f and g is given by*

$$\Delta(f,g) = \begin{cases} \infty & \text{if } \deg(f) \neq \deg(g) \\ \min_{\pi\in\Sigma_n}\left\{\max_i\{|z_i - w_{\pi(i)}|\}\right\} & \text{if } \deg(f) = \deg(g) = n \end{cases}$$

where z_i are the roots of f, w_j are the roots of g, and Σ_n is the set of permutations on $\{1, \ldots, n\}$.

Our algorithm will work by computing a polynomial, g, which is explicitly presented as a product of linear factors. In order to know that the (immediately available) roots of g provide approximations of the desired precision to the roots of f, we rely on the following corollary to the theorem of Ostrowski [N 94].

Theorem 2.1 *Let f and g be monic polynomials of the same degree n, and let $\rho_u(f) < 2^l$. If $|f - g| < 2^{-\kappa}$, then*

$$\Delta(f, g) \; < \; 2^{l+\log n+3-\kappa/n} \; .$$

Proof: This follows immediately from [N 94], Theorem 4.5, by estimating the size of the coefficients of f as symmetric functions of its roots, and then using the condition on $|f - g|$ and the triangle inequality to estimate the size of the coefficients of g as well. ∎

We will keep the l^∞ errors introduced in each factorization of a polynomial into polynomials of lower degree small enough so that, by using the triangle inequality to estimate the total error, and the previous theorem, we will guarantee the precision we require.

An essential part of the algorithm is to translate coordinates to an origin which is "good" for factoring f. Thus we introduce

Definition 2.7 *For $c \in \boldsymbol{C}$, define the translated polynomial $\tau_c(f)$ by*

$$\tau_c(g)(z) \; = \; g(z + c) \, .$$

So, in particular, $\tau_c(g)(0) = g(c)$.

We also occasionally will scale coordinates

Definition 2.8 *If $c \in \boldsymbol{C} - \{0\}$, define*

$$\begin{aligned} \alpha_c(g)(z) \; &= \; c^{-n} g(cz) \\ \alpha^c(g)(z) \; &= \; c^n g(z/c) \end{aligned}$$

and multiply polynomials

Definition 2.9

$$mult(g_1, g_2) = g_1 g_2 \, .$$

There are methods for computing each of these polynomial transformations in $O(n \log n)$ steps (or fewer in the case of scaling). In the arithmetic model, we assume that these operations are performed exactly, but we still need to be concerned with errors since we deal with *approximate factors* throughout the algorithm. That is we need to bound $|o(g) - o(h)|$, or $|o(g_1, g_2) - o(h_1, h_2)|$ for each operation, o, that we use. Also, in the boolean model we need to round after computing the operation and this introduces another error, but it is actually small in comparison.

We collect these error bounds into the following easy lemma.

Lemma 2.1 *Assume that g_1, g_2, h_1, h_2 are all polynomials with l^∞ norms bounded by 2^l, and degrees bounded by n, and that $|c| < 2^l$. Then for each of the basic polynomial operations above, the error introduced by the computation followed by rounding to $\mathcal{P}_t$ is bounded by*

$$|o(g) - o(h)| \; < \; \max\left\{2^{2(l+\log n)}|g - h|, 2^{l-1}\right\}$$

for each of the unary operations, and bounded by

$$|o(g_1, g_2) - o(h_1, h_2)| \; < \; \max\left\{2^{2(l+\log n)} \max(|g_1 - h_1|, |g_2 - h_2|) \, , \, 2^{l-1}\right\} .$$

for the biniary operation mult.

2.1 Organization of the Algorithm

At a high level, the algorithm is exceedingly simple. It starts by translating f to a "good" coordinate system, one where there is a disk that divides the roots in a balanced way, and whose boundary is "not too close" to any of the roots. Then it factors f approximately into three polynomials f_1, f_2, and f_3 in the transformed coordinate system. If we were able to arrange that each of the degrees of the f_i is less than or equal to half the degree of f, (we call this a *balanced factorization* or *balanced splitting*) then we translate each of them back to the original coordinate system (in practice we wouldn't actually do this, we'd just keep a record of the accumulated translations for each factor, but it greatly simplifies our description), and then proceed recursively. It might not have been possible to arrange a balanced splitting though, so taking f_1 to be the unique factor of degree greater than half the degree of f, we translate it to another "good" coordinate system, and approximately factor it into at most four factors which we will be able to *guarantee* have degrees each less than or equal to half the degree of f. As before, we now translate back to original coordinates, leaving us with at most six factors f_{11}, f_{12}, f_{13}, f_{14} (the four factors of f_1), and f_2, and f_3. We now have a balanced factorization of f, and can recursively apply the algorithm to each of the factors.

Remark 2.1 *Each translation will always be of bounded size, $|c| < 2^m$, and hence, by standard root bound theorems, it will be straightforward to check that all coefficients of translated polynomials and factors will have upper root radii at most 2^{m+1}, and will have l^∞ norm less than $2^{n+n(m+1)} = 2^{2n+nm}$ ([N 94]).*

¿From here on, we will always round to $\pi = \mu + 2n + nm + \log\log n + 5$ bits (to the *right* of decimal point), Appealing to the previous lemma, and using the triangle inequality, we see that the l^∞ error introduced by each translation (including rounding in the Boolean case) is in total bounded by $\phi_1 = 2^{2n+nm-\pi-1} = 2^{-\mu-\log\log n-6}$.

The l^∞ error introduced by the factorization step, that is the step that factors the polynomials using the given coordinate system, must be broken into two

parts. First, there is the computational error ε_1, which would be introduced even if its input polynomial was *exactly* the polynomial we wished to factor. Second, there is the inherent stability error, ε_2, which is independent of the computational procedure, and is introduced because the input polynomial is already in error by as much as twice (since we may require two successive translations) the amount ϕ_1. Hence,

$$\varepsilon_2 \;<\; 2^{-\mu-\log\log n-5}$$

In the next section, we will bound ε_1 by

$$\varepsilon_1 \;<\; 2^{-\pi+2n+nm} \;=\; 2^{-\mu-\log\log n-5}\,, \tag{1}$$

and hence, finally, the total error introduced by a factorization step, including the necessary translation is bounded by

$$\varepsilon_3 \;<\; 2^{-\mu-\log\log n-4}.$$

A complete *balanced factorization*, as described at the beginning of this section, may require *five*, "translate and factor" steps (actually, only two translations, but for ease of notation, we can think of a factorization without translation as a "translate by 0 and factor"), as well as a final translation to bring the computed factors back to the original coordinate system, so the total error introduced by a balanced factorization step is bounded by

$$\varepsilon \;<\; 5\varepsilon_3 + 2^{2n+nm-\pi-1} \;<\; 2^{-\pi+4}\,.$$

Since there are at most $2\log n$ factorizations in the computation path to get to any approximate linear factor, we appeal to the triangle inequality to obtain

Lemma 2.2 *If f is the input polynomial, and g is the computed polynomial (which is expressed explicitly as a product of monic linear polynomials), then*

$$\Delta(f,g) \;<\; 2\log n 2^{-\pi+4} \;<\; 2^{-\mu}\,.$$

And hence, the (immediately available) roots of g provide the solution to the complex roots problem for f.

3 Approximate Factorization of a Polynomial

It has long been understood that the computation of an accurate approximate factorization of a polynomial f is closely tied to the geometry of its root set. This is because the most obvious approach depends on computing contour integrals via an FFT, and in order to get good convergence one needs to exploit the existence of a "large" *root free* annulus.

To make this more precise we introduce

Definition 3.1 *Fix $\delta > 0$. A disk $D = D(z_0; R)$ is called* **δ-isolated**

(or is said to have **isolation ratio δ** *) for a polynomial f if there are no roots of f in the annulus*

$$T_D = D(z_0; (1+\delta)R) - D(z_0; (1+\delta)^{-1}R)\,.$$

(This definition is essentially the same as the one that can be found in [P 87] and [P 89] except that we use the disk whose boundary is 'centered' in the root free annulus instead of on the inside boundary of the root free annulus. The choice is only a matter of notational convenience.)

Previously, in the case $\delta = 1/2$, the techniques of contour integration and Newton iteration were effectively used by Schönhage [S 82] and later by Pan [P 87] to approximately factor the polynomial f.

Lemma 3.1 *Suppose we are given a disk D which has isolation ratio $\delta = 1/2$ for polynomial f. Then there is a $O(n \log^2 n \log b)$ (arithmetic) algorithm for computing a factorization of f into approximate factors, f_1 and f_2, corresponding to roots inside and outside D, respectively.*

For $i = 1, 2$ if $\tilde{f}_i$ are the two exact factors of f, then

$$|f_i - \tilde{f}_i| < 2^{2n+nm)-\pi}. \tag{2}$$

Moreover, the bit precision required by the algorithm is exactly $\pi + (2n + nm)$ when implemented in the Boolean model.

Definition 3.2 *In the rest of the paper, we shall say that an approximate factorization of a polynomial f is a* **full precision factorization** *if the factors satisfy equation 2. We also call the approximate factors, f_i,* **full precision factors** *of f.*

(Actually the construction works for any fixed constant $\delta > 0$, but the number of steps required increases as δ goes to zero, and hence the method needs modification when the degree n is allowed to get large, since then there may not exist non-trivial δ-isolated disks.)

We need to generalize lemma 3.1 to the case where δ is not fixed independent of the degree n. The main result of this section then is

Theorem 3.1 *Suppose that $|f| < 2^{2n+nm}$ and that we are given any disk $D = D(0; R)$ centered at 0, which has known isolation ratio $\delta > 0$ for f. Then we can compute, to the same precision as in lemma 3.1, the approximate factors f_1 and f_2 in $O(n \log^2 n \log b \log \delta^{-1})$ arithmetic operations.*

Moreover, making no assumptions on minimum root separation, the bit precision required by the algorithm, when implemented in the Boolean model, is exactly

$$\pi + -\log\delta + 2n^2m + 3n + \log n + 3 + n^2(-\log\delta + 1) + \log\log n\,.$$

Let $\mathcal{P} = \mathcal{P}_{\pi}^{\in\backslash+\backslash\Updownarrow}$ be the set of monic polynomials defined in the previous section. Of course, during the recursive factorization, the degrees of the polynomials we encounter can only decrease, so we will implicitly assume that all degrees are bounded by n.

3.1 Approximate Splitting of Polynomials with Isolation Ratio δ

In the case that we need to factor a polynomial according to a disk D with isolation ratio, δ, smaller than $1/2$, we will use a combination of established techniques (polynomial powering, contour integration, and Newton iteration) described in the Appendix to efficiently and accurately split f into approximate factors corresponding to roots inside and outside D, respectively.

Our basic approach will be to first scale coordinates so that the isolation disk D can be taken to be the unit disk. Then we apply the Graeff's Method for $k = \lceil \log\delta^{-1} \rceil - 1$ stages. Each iteration of this computation produces a new polynomial whose roots are the *squares* of the roots of the previous polynomial. We need to keep all polynomials within our coefficient class, so we also check at each stage to see if our new polynomial has any roots with magnitude larger than 2^n or less than 2^{-n}. If such roots exist, we factor the polynomial using lemma 3.1, and proceed to apply Graeff's Method only to the factor with moderate sized roots. This will, in k stages, produce a polynomial f_k with the property that the unit disk has isolation ratio $1/2$.

We then apply Lemma 3.1 to obtain a split of f_k into factors with roots corresponding to the roots of f_k inside and outside D, respectively. Then we will apply a series of k stages of a partial GCD computation, to reconstruct the factors of f from these factors of f_k. These factors of f will then have roots corresponding to roots of f inside and outside D.

The following observation allows us to assume that throughout the powering process, all roots of our polynomial are less than 2^n and greater than 2^{-n}

in magnitude. If any roots are greater than 2^n in magnitude, or less than 2^{-n} in magnitude, there must be a disk with isolation ratio $1/2$ within this range that we can use to factor f by the basic factorization technique of Lemma 3.1. Finding such a disk is easily done using well established root radii estimation techniques [P 87, R 93b].

Remark 3.1 *It is important to keep the roots of the successive polynomials from getting too large even in the arithmetic model of the algorithm. The reason for this is that theorem 2.1 depends on the upper root radius. If we allow this to get larger than $O(2^n)$, the precision required would be too large to effectively apply lemma 3.1 at stage k. Moreover, the error of each Partial GCD computation would also be too large.*

We now make the preceding discussion more precise with

Algorithm FACTORPOL

Let $k = \lceil \log \delta^{-1} \rceil - 1$.

[0] Scale coordinates so that D is the unit disk. By lemma 2.1, this introduces an error of at most $2^{2n+nm-\pi-1}$ in the l^∞-norm.

[1] **For** $i = 1, 2, \ldots, k$ **do**

[1.1] Apply Graeffe's Method to $f_0 = f$, computing (symbolically)

$$f_i(z) = f_{i-1}(\sqrt{z}) f_{i-1}(-\sqrt{z})$$

[1.2] Apply lemma 3.1 to get a splitting of f into polynomials h_i, f_i, H_i (where h_i and H_i may have degree 0) and where (recall definition 2.4)

(i) $\rho_u(h_i) \leq 2^{-n}$

(ii) $\rho_l(H_i) \geq 2^n$

(iii) $2^{-n} < \rho_l(f_i) \leq \rho_u(f_i) < 2^n$.

Comment: After k stages, this results in a degree n' polynomial (where $n' \leq n$) $f_k(z)$ which has roots which are the 2^kth powers of the roots of $f_0(z)$. Note that the unit radius disk D is 1/2-isolated for $f_k(z)$.

[2] Apply Lemma 3.1 to get an approximate factorization of f_k into polynomials F_{k+1} and G_{k+1} satisfying $\rho_u(G_{k+1}) < 1$ and $\rho_l(F_{k+1}) > 1$. Let $G_k = h_k G_{k+1}$ and $F_k = H_k F_{k+1}$. The lemma allows us to assume that the precision of the approximate factors is high.

Comment: At this point, we have constructed a factorization of a "blown-up" version of f. Now we proceed back down the chain of Graeffe polynomials, constructing a corresponding factorization of each polynomial from the factorization of the one "above" it.

[3] **For** $i = k, k-1, \ldots, 1$ **do**

Let

$$F_{i-1}(z) = H_{i-1} PGCD_{\deg(F_i)}(f_{i-1}(z), F_i(z^2)) \tag{3}$$

and

$$G_{i-1}(z) = h_{i-1} PGCD_{\deg(G_i)}(f_{i-1}(z), G_i(z^2)). \tag{4}$$

The $PGCD_l$ operator finds the monic polynomial of degree l in the standard quotient remainder sequence for the two polynomials which are its arguments. In general, this could be the zero polynomial, but we will know otherwise in our particular application.

The motivation for this step is best understood if we imagine that all computations, including the factorization in step [2], could have been carried out exactly. (In actuality, this is not true even in the arithmetic model of computation, due to the error introduced in step [2].) Then, the roots of $F_i(z)$ are exactly the squares of those roots of $f_{i-1}(z)$ which are greater than 1 in modulus. Also, the roots of $F_i(z^2)$ are *all* the square roots of the roots of $F_i(z)$. So, if $p_{i-1}(z)$ is the factor of f_{i-1} corresponding to all of its roots which are greater than 1 in modulus, then $F_i(z^2) = p_{i-1}(z)p_{i-1}(-z)$, *and*

$$PGCD_{\deg(F_i)}(f_{i-1}(z), F_i(z^2)) = GCD_{\deg(F_i)}(f_{i-1}(z), F_i(z^2)) = p_{i-1}(z)\,. \quad (5)$$

That is, we have computed the factor of f_{i-1} which we seek. The same, of course, holds for the second $PGCD$ computation, but with respect to roots smaller than 1 in modulus.

The difficulty we face is that we must show that the $PGCD$ computation in the presence of errors in the input polynomials still gives a good approximation to the result we are after.

[4] Apply $O(\log n + \log\log \delta^{-1})$ Newton iterations to F_{i-1} and G_{i-1} in order to guarantee that

$$\begin{aligned} |F_{i-1} - F_{i-1}^0| &< 2^{-\sigma} \\ |G_{i-1} - G_{i-1}^0| &< 2^{-\sigma} \end{aligned} \quad (6)$$

where F_{i-1}^0 and G_{i-1}^0 are the corresponding true factors of f_i. The value of σ will be discussed shortly.

[5] Now $i = 0$, and we've factored the scaled polynomial, so scale the factors to the original coordinate system.

OUTPUT F_0, G_0 which is an approximate factorization of $f_0 = f$.

The key to keeping the errors from growing is the Newton iteration phase. The problem with using it is that we must be sure that we have a good enough initial approximation polynomial. The initial approximation we have comes from the $PGCD$ computation, so we will need to analyze the errors there.

We begin with [P 89, P 94]

Lemma 3.2 *Let f be monic and let*

$$\eta = \eta(f) = \min_{|z|=1} |f(z)| ,$$

and let

$$\epsilon_f = \min \left\{ \eta/8 \, , \, \frac{(7\eta)^4}{n^2(7\eta + 9n)^2 2^{4n+2}} \right\} .$$

If g and h are the two (exact) monic factors of f corresponding to the roots of f inside and outside the unit disk, and if g_0 and h_0 are two approximations to g and h satisfying

$$|g_0 - g| \, < \, \epsilon_f$$

and

$$|h_0 - h| \, < \, \epsilon_f \, ,$$

and if g_i and h_i are the sequence of Newton iteration polynomials beginning with g_0 and h_0, then

$$|g_i - g| \, < \, \epsilon_f^{(1.5)^i}$$

$$|h_i - h| \, < \, \epsilon_f^{(1.5)^i} .$$

We now fix

$$\sigma = -\log \epsilon_f - 2n^2 \log \delta^{-1} + 2(2n + nm)$$

and suppose inductively that F_i and G_i satisfy equation 6. We need to show that we can compute F_{i-1} and G_{i-1} accurately enough to have an initial Newton approximation, that is with an l^∞ error no larger than ϵ_f. Other than the $PGCD$ computation, we only have two multiplications to compute

these polynomials, so it suffices to show that the l^∞ error at the end of the $PGCD$ computation is no more than $2^{\log \epsilon_f - 2(2n+nm)}$.

To do this, we begin by expressing the coefficients of $P_{i-1}(z) = PGCD_{\deg(F_i)}(f_{i-1}(z), F_i(z^2))$ explicitly as a quotient of two determinants. In order to greatly simplify the notation necessary, we will assume that

$$\deg(f_{i-1}) = 2\deg(F_i) = 2d = c$$

so that we are computing the $PGCD$ of two polynomials of equal degree. If this is not the case, we can multiply the smaller degree polynomial by the appropriate power of z. In practice, this would not be the optimal way to proceed, but it wouldn't increase the overall work by more than a factor of 2. We leave it to the reader to make the notational modifications to the estimates that follow, which would allow computation with smaller matrices.

Lemma 3.3 *Let* $A(z) = z^c + a_{c-1}z^{c-1} + \cdots + a_0$ *and* $B(z) = z^c + b_{c-1}z^{c-1} + \cdots + b_0$. *Let* $S(A,B)$ *be the* $2c \times 2c$ *matrix*

$$S(A,B) = \begin{bmatrix} 1 & 0\cdots & 0 & 1 & 0\cdots & 0 \\ a_{c-1} & \ddots & & b_{c-1} & \ddots & \\ \vdots & \ddots & & \vdots & \ddots & \\ & & 1 & & & 1 \\ 0 & \cdots & a_{c-1} & 0 & \cdots & b_{c-1} \\ & & \vdots & & & \vdots \\ a_1 & 0\cdots & 0 & b_1 & 0\cdots & 0 \\ a_0 & \ddots & & b_0 & \ddots & \\ & \ddots & & & \ddots & \\ & & a_1 & & & b_1 \\ 0 & \cdots & a_0 & 0 & \cdots & b_0 \end{bmatrix}$$

and let $S_d(A,B)$ *be the* $2(c-d) \times 2(c-d)$ *submatrix of* $S(A,B)$ *obtained by removing the bottom* $2d$ *rows, columns* $c-d+1, \ldots, c$ *and columns* $2c-d+1, \ldots, 2c$. *For* $0 \le j \le d-1$, *also let* $S_j(A,B)$ *be the* $2(c-d) \times 2(c-d)$

submatrix of $S(A,B)$ which is identical to $S_d(A,B)$ in the first $2(c-d)-1$ rows, and whose last row consists of the entries of $S(A,B)$ taken from the $(2c-d-j)^{th}$ row and the same columns.

Then, if $\det(S_d(A,B)) \neq 0$, the degree d polynomial in the standard quotient remainder sequence (see [AHU 74]), $PGCD_d^*(A,B)$, is non-zero and is given by

$$PGCD_d^*(A,B)(z) \;=\; \sum_{j=0}^{d} \det(S_j(A,B))z^j. \tag{7}$$

If we had to compute the determinants in equation 7 explicitly, we would be in trouble in the Boolean model of computation. We only know that each coefficient is bounded in magnitude by $2^{O(n^2)}$ (by remark 3.1) and hence the size of the determinants could get as large as $2^{O(n^3)}$, which would be too large for the bit precision we are using. Fortunately, we only need to compute the *monic* form of $PGCD_d^*(A,B)$, which we shall denote by $PGCD_d(A,B)$, and hence only need to be concerned with the quotients

$$Q_j \;=\; \frac{\det(S_j(A,B))}{\det(S_d(A,B))} \tag{8}$$

for $0 \leq j \leq d-1$.

We need to show that these are not too large, and that we can compute them in such a way that the errors are not too large either.

In our application, $A(z) = f_{i-1}(z)$ and $B(z) = F_i(z^2)$. Let $\tau = |b_0|^{1/2}$. Then τ is the square root of the product of the moduli of all roots of $F_i(z)$. Let us write

$$S'_j(A,B) \;=\; \tau^{-1} S_j(A,B) \tag{9}$$

for $0 \leq j \leq d$.

Remark 3.2 *The quotients, Q_j are not changed if we replace the entries of S_j with the entries of S'_j.*

We begin with the following perturbation lemma.

Lemma 3.4 *Suppose that $P_1(z) = Q_1(z)R(z)$ and $P_2(z) = Q_2(z)R(z)$, where $\deg(Q_1) = \deg(Q_2) = d$, $\deg(R) \leq d$, and where P_i, Q_i and R are all monic. Suppose that $\rho_l(Q_1) > 1$, and $\rho_u(P_2) < 1$. (The asymmetry of the assumption with respect to the P_i is intentional.) Also suppose that $\rho_u(P_1) < 2^k$. Let*

$$B = \sup_{\{|z| \leq 2\}} |Q_1(z)| \tag{10}$$

and

$$b = \inf_{\{\zeta : Q_2(\zeta) = 0\}} |Q_1(\zeta)| \tag{11}$$

and

$$\Lambda = 3^d + \frac{d 2^d B^2}{b} \tag{12}$$

and

$$\kappa_0 = \lceil \log \Lambda \rceil + 2.$$

If

$$\kappa > \kappa_0 + 1$$

and $|F_i - P_i| < 2^{-\kappa}$ then

$$\begin{aligned} |PGCD_d(F_1, F_2) - PGCD_d(P_1, P_2)| &= |PGCD_d(F_1, F_2) - R| & (13) \\ &< 2^{\kappa_0 - \kappa + dk + d}. & (14) \end{aligned}$$

Proof : For simplicity, we shall assume $\deg(R) = d$. (To achieve this, we

can multiply both P_1 and P_2 by the monomial $z^{d-\deg(R)}$).

For each $i \geq 0$, let $\mathcal{V}_i$ denote the vector space of polynomials of degree less than or equal to i. The dimension of this space is, of course, $i+1$, and $\mathcal{V}_j \subset \mathcal{V}_i$ for $j \leq i$. Consider the linear map $\boldsymbol{L} : \mathcal{V}_{d-1} \oplus \mathcal{V}_{d-1} \rightarrow \mathcal{V}_{3d-1} / \mathcal{V}_{d-1}$ defined by

$$\boldsymbol{L}\left((a,b)\right) \;=\; aP_1 + bP_2\,.$$

The fact that this map is both well defined and an isomorphism, follows immediately from the GCD assumption on P_1 and P_2. In this setup, let m_j be the image of the monomial z^j under the vector space quotient map $\mathcal{V}_{3d-1} \rightarrow \mathcal{V}_{3d-1}/\mathcal{V}_{d-1}$. Then if $\boldsymbol{L}(A,B) = m_d$,

$$AP_1 + BP_2 \;=\; R\,. \tag{15}$$

So

$$(A,B) \;=\; \boldsymbol{L}^{-1}(R)\,.$$

Our result will then follow from standard theorems in Numerical Analysis if we can put an appropriate bound on the *condition number* of the linear map $\boldsymbol{L}$ with respect to the standard monomial basis for the $\mathcal{V}_j$.

Because all roots of P_1 and P_2 are assumed to be less than 2^k in magnitude, their coefficients are bounded in magnitude by 2^{k+2d}, and hence

$$\|\boldsymbol{L}\| \;<\; 2^{k+2d}\,. \tag{16}$$

So we focus our attention on $\|\boldsymbol{L}^{-1}\|$. To this end, define A_j and B_j ($d \leq j \leq 3d-1$) as the polynomials satisfying

$$\boldsymbol{L}(A_j, B_j) \;=\; m_j \;.$$

The coefficients of A_j, B_j then, by definition, form the entries of the coore-sponding row of the matrix of $\boldsymbol{L}^{-1}$, and we will succeed in bounding $\|\boldsymbol{L}^{-1}\|$ if we can bound all these coefficients for each $d \leq j \leq 3d-1$.

Now

$$A_j(z)P_1(z) \;+\; B_j(z)P_2(z) \;=\; z^j + r_j(z) \;,$$

where $r_j(z)$ is a polynomial (not necessarily monic) of degree strictly less than d (definition of vector space quotient). Of course, by assumption we have

$$z^j + r_j(z) \;=\; q_j(z)R(z)$$

for some monic polynomial q_j. Let $w_1, \ldots, w_d$ be the d roots of R. Let T_l, $0 \leq l \leq d-1$ be the successive divided difference operators (see [H 74], vol. 1)

$$\begin{aligned} (T_0 f)(z) &= f(z) && (17)\\ (T_l f)(z) &= ([w_1, \ldots, w_l]\, f)(z) \;. && (18) \end{aligned}$$

Computing the divided differences, we have

$$(T_l(q_j R))(w_{l+1}) \;=\; 0 \qquad (19)$$

and

$$\left|(T_l(z^j))(w_{l+1})\right| \;\leq\; 2^l$$

and hence

$$|(T_l(r_j))(w_{l+1})| \;\leq\; 2^l\;.$$

Since $\deg(r_j) < d$, we have the identity

$$r_j(z) \;=\; \sum_{l=0}^{d-1}\left((T_l(r_j))(w_{l+1})\prod_{s=0}^{l-1}(z-w_s)\right) \tag{20}$$

(where, as usual, the empty product in the first term of the summation is taken to be 1).

It is now easy to verify that if $z_1,\ldots,z_d$ are any complex numbers satisfying $|z_j| < 2^k$, and T_l are the operators above, now taken with respect to the sequence z_j instead of the sequence w_j, then

$$\left|(T_l(z^j + r_j))(z_{l+1})\right| \;<\; (2^k+1)2^{2d}\;. \tag{21}$$

So now let us apply the same divided difference scheme, but with respect to the sequence of roots of Q_1 and Q_2 instead of with respect to the roots of R. Let T_l^1 be the sequence of divided difference operators taken with respect to $\xi_1,\ldots,\xi_d$, the roots of Q_1, and let T_l^2 be the sequence of divided difference operators taken with respect to $\zeta_1,\ldots,\zeta_d$, the roots of Q_2.

Remark 3.3 *Notice that no particular order is assumed for the ξ_i or the ζ_i, and hence the bounds that follow for T_l^1 and T_l^2 are independent of the subset of roots considered.*

As with equation 19 we have

$$(T_l^2(B_j P_2))(\zeta_{l+1}) \;=\; 0$$

so that by equation 21

$$\left|(T_l^2(A_j P_1)(\zeta_{l+1})\right| \;<\; (2^k+1)2^{2d}\;. \tag{22}$$

Now, it follows from the Cauchy Integral Formula, the Mean Value Theorem, and the assumptions of the lemma

$$\left|(T_l^2(P_1))(\zeta_{l+1})\right| \;<\; B\;. \tag{23}$$

Furthermore, we can express the divided difference operator of a product of two functions in the following way

$$(T_l^2(A_j P_1))(\zeta_{l+1}) \;=\; \sum_{r=0}^{l} \left(U_{l-r}^2(P_1)\right)(\zeta_{l+1})\;\left(T_r^2(A_j)\right)(\zeta_{r+1}) \tag{24}$$

where U_{l-r} is the divided difference operator

$$(U_{l-r}f)(z) \;=\; ([\zeta_{r+1},\ldots,\zeta_l]\,f)(z)\;.$$

Combining equations 24 and 23, lemma assumption 11, and the remark above, we now have, for each l,

$$\left|(T_l^2(A_j))(\zeta_{l+1})\right| \;<\; b^{-1}\sum_{r=0}^{l-1} B\left|(T_r^2(A_j))(\zeta_{r+1})\right|$$

and hence, by simple induction it follows that

$$\left|(T_l^2(A_j))(\zeta_{l+1})\right| \;<\; \left(\frac{B}{b}\right)^{l+2} \;.$$

Since, $\deg(A_j) < d$, for all z

$$A_j(z) \;=\; \sum_{l=0}^{d-1}(T_l(A_j))(\zeta_{l+1}) \prod_{r=1}^{l}(z-\zeta_r)$$

and thus, since $|\zeta_r| < 1$ for all r,

$$\sup_{\{|z|=2\}} |A_j(z)| \;<\; d2^d\frac{B}{b} \;.$$

By the Maximum Modulus Theorem, we even have

$$\sup_{\{|z|\leq 2\}} |A_j(z)| \;<\; d2^d\frac{B}{b} \;. \tag{25}$$

So finally, by the Cauchy Estimates, each coefficient of A_j is bounded by $d2^dB/b < \Lambda$. Moreover, from equation 25 and the definition of B in the lemma, we must have

$$\sup_{\{|z|\leq 2\}} |A_j(z)P_1(z)| \;<\; d2^d\frac{B^2}{b} \;.$$

Also, since all roots of R and Q_2 are less than 1 in modulus, we have

$$\sup_{\{|z|\leq 2\}} |R(z)| \ < \ 3^d \tag{26}$$

and

$$\inf_{\{|z|=2\}} |P_2(z)| \ > \ 1 \ . \tag{27}$$

These last two equations, along with equation 15, give us

$$\sup_{\{|z|=2\}} |B_j(z)| \ < \ d2^d \frac{B^2}{b} + 3^d \ . \tag{28}$$

So, again using the Cauchy Estimates, each coefficient of B_j is bounded by Λ as well.

Hence, all entries in the matrix of the transformation $\boldsymbol{L}^{-1}$ are bounded by Λ. The results of the lemma now follow directly from standard matrix perturbation theorems in Numerical Analysis. ∎

The asymmetry in the statement of lemma 3.4 may seem puzzling, however, it becomes less so upon observing

Lemma 3.5 *If we modify the assumptions of lemma 3.4 so that instead*

$$\rho_u(Q_1) < 1$$

$$\rho_l(Q_2) > 1$$

$$\rho_u(Q_2) > 2^k$$

and

$$B = \sup_{\{|z|\geq 1/2\}} |Q_1(z)|$$

then the same perturbation bounds hold.

Proof :

Replace each polynomial p in the statement of lemma 3.4 with its "reverse" polynomial $\tilde{p}$. The roots of $\tilde{p}$ are the multiplicative inverses of the roots of p.

∎

Corollary 3.1 *Suppose that under the assumptions of lemma 3.4, or lemma 3.5, we also know that* $|\xi| > 1+\delta$ *for all roots,* ξ *of* Q_1*, and that* $|\zeta| < 1-\delta$ *for all roots,* ζ *of* Q_2*, and* $0 < \delta < 1/2$*. Then, taking* B *as above*

$$\begin{aligned} |PGCD_d(F_1,F_2) - PGCD_d(P_1,P_2)| &= |PGCD_d(F_1,F_2) - R| \quad (29) \\ &< (1+\frac{3}{2\delta})^d 2^{2dk+3d+\log d+1-\kappa} \quad (30) \end{aligned}$$

Proof : If $|z| \leq 2$, $Q_2(\zeta) = 0$, and $Q_1(\xi) = 0$, then

$$\left|\frac{(z-\xi)}{(\zeta-\xi)}\right| = \left|1 + \frac{(z-\zeta)}{(\zeta-\xi)}\right| < 1 + \frac{3}{2\delta}\,.$$

Hence

$$\left|\frac{Q_1(z)}{Q_1(\zeta)}\right| = \left|\prod_{Q_1(\xi)=0} \frac{(z-\xi)}{(\zeta-\xi)}\right| < \left(1+\frac{3}{2\delta}\right)^d . \quad (31)$$

Thus from lemma 3.4, and with notation as above,

$$\begin{aligned} |PGCD_d(F_1,F_2) - PGCD_d(P_1,P_2)| &< (3^d + \frac{d2^dB^2}{b})2^{dk+d-\kappa} \qquad (32)\\ &< 2^d(2^d + \frac{dB^2}{b})2^{dk+d-\kappa} \\ &< 2^d(2^d + dB(\frac{3}{2\delta})^d)2^{dk+d-\kappa} \\ &< 2^d(2^d + d2^{dk+d}(\frac{3}{2\delta})^d)2^{dk+d-\kappa} \\ &< 2^d(d2^{dk+d+1}(\frac{3}{2\delta})^d)2^{dk+d-\kappa} \\ &= (\frac{3}{2\delta})^d 2^{2dk+3d+\log d+1-\kappa} \end{aligned}$$

∎

Our intent is to bound errors in step [3] of algorithm **FACTORPOL**. To do this we apply lemma 3.4 to equation 4 and lemma 3.5 to equation 3. Since, in such application, we can bound d above by n, k above by nm, and δ above by $1/2$ (recall that $\delta > 1/2$ is the "trivial" case of constant isolation ratio) we have

Corollary 3.2 *The number of bits of precision lost due to inherent error (i.e. assuming exact computation) in each PGCD computation in step [3] of algorithm* **FACTORPOL** *is bounded above by*

$$\mathcal{H} = -\log\delta + 2n^2m + 3n + \log n + 3$$

We also need to analyze the computational error. Since the computation of the PGCD is done by determinant and then division by the leading coefficient (recall equation 8), it is *not* obvious how to do the computation in the boolean case without introducing much greater error than that introduced above. However, we are rescued by the following

Lemma 3.6 *With the same notation as in corollary 3.1*

$$\delta^{d^2} \; < \; \det\left(S'_d(P_1, P_2)\right) \; < 2^{d^2}$$

Proof :

We know that $S_d(P_1, P_2)$ is exactly $\mathcal{R}(Q_1, Q_2)$, the resultant of Q_1 and Q_2. If $\xi_1, \ldots, \xi_d$ are the roots of Q_1 and $\zeta_1, \ldots, \zeta_d$ are the roots of Q_2, then

$$\mathcal{R}(Q_1, Q_2) \; = \; \prod_{i=1}^{d} \prod_{j=1}^{d} (z_i - w_j).$$

Since

$$\left|\frac{\zeta_j}{\xi_i}\right| \; < \; 1 - \delta$$

it follows

$$\delta \; < \; \left|1 - \frac{\zeta_j}{\xi_i}\right| < \; 2\,.$$

Hence

$$\delta^{d^2} \; < \; |S'_d(P_1, P_2)| \; = \; \prod_{i=1}^{d} \prod_{j=1}^{d} \left|\frac{(\xi_i - \zeta_j)}{\xi_i}\right| \; = \; \prod_{i=1}^{d} \prod_{j=1}^{d} \left|1 - \frac{\zeta_j}{\xi_i}\right| \; < \; 2^{d^2}.$$

∎

So

Corollary 3.3 *Since, in equation 3.2, computing τ and hence the S'_j is trivial, if, in algorithm* **FACTORPOL**, *step [3], we use at most $n^2(-\log\delta + 1)$ additional bits in our intermediate computation, the computational error introduced will be no larger than the inherent error estimated above.*

Finally, since we need perform at most $\log n$ PGCD computations in any computation path of algorithm **FACTORPOL**, the total number of bits of precision lost in step [3] – assuming we do each computation (in particular step [2], the constant isolation ratio factorization) with precision

$$\sigma > \bar{\phi} = -\log\delta + 2n^2m + 3n + \log n + 3 + n^2(-\log\delta + 1)$$

bits in the boolean case – is bounded above by

$$\bar{\phi} + \log\log n\,.$$

Thus, taking $\sigma = \pi + \log\log n + \bar{\phi}$, and combining all the results of this section we have proved theorem 3.1.

4 Splitting Sets and Centered Points

The previous section provides us with a method for efficiently factoring a polynomial into factors of smaller degree, once we have a disk with a "good" isolation ratio – that is one with δ reasonably large. However, in order to optimize the overall complexity of the root finding algorithm, it is necessary that the disk also divide the root set geometrically in a "balanced" manner.

In the case where the input polynomial, f, is *totally real* (i.e. has *only* real roots), the notion of a "splitting point" is used in order to find the required geometrically balanced division of the root set [BFKT 86, P 89, R 93b, N 94]. It is the purpose of this section to introduce the idea of a *splitting set* of points for general complex polynomials. In some sense it is a generalization of the splitting point idea, although not entirely. Its usefulness hinges on the fact that there is a fast method for approximating the moduli of the roots of f with small *relative* error. As a result of this, it will turn out that we will be able to find a balanced factorization of f as long as we can choose an origin

which is "close" to any "big" cluster of roots of f. A splitting set allows us to find such an origin. Let $D(z;R)$ denote a complex disk with center z and radius R.

Definition 4.1 *Fix $0 < \alpha < 1$ and $k > 0$. A finite set $S = \{s_1, \ldots, s_N\}$ of complex numbers is an* **(α, k)-splitting set for f,** *if for every disk $D(z_0\ ;\ R)$ in the complex plane containing more than $\alpha\, n$ roots of f, there is a point $s_j \in S$ that lies in $D(z_0\ ;\ kR)$.*

For the remainder of the paper we shall use the following shorthand whenever it is not ambiguous.

Definition 4.2 *We call a disk D* **α-full** *(for f) if it contains* **more** *than αn roots of f.*

Definition 4.3 *If $D = D(z;R)$ is a complex disk, and k is a non-negative real number, we use kD to represent the dilated disk $D(z;kR)$.*

Remark 4.1 *The preceding definition differs from the often intended meaning for kD, which is the point set obtained by multiplying every point of D by the scalar k.*

We can now restate the definition of a splitting set as follows: S is an (α, k)-splitting set for f if, and only if, for every disk D which is α-full for f, $kD \cap S \neq \emptyset$.

The usefulness of a splitting set is that it allows us to quickly find (assuming that k and N are reasonably small) a "good" origin for factoring f in a balanced way. This will be made more precise after the following series of lemmas.

Lemma 4.1 *If S is a (α, k)-splitting set for f, and $\alpha \geq 1/2$, then at least one point $s_0 \in S$ has the property that, if D is any α-full disk, $s_0 \in (k+2)D$.*

Proof : Let D_0 be an α-full disk of *minimal* radius, and let $s_0 \in S$ be a point of S which is contained in kD_0. *Any other* α-full disk D must have radius at least as large as the radius of D_0, and, since $\alpha \geq 1/2$, at least one root of f is in *both* D_0 and D. So $D \cap D_0 \neq \emptyset$. But then $(k+2)D$ must contain kD, and in particular, $s_0 \in (k+2)D$. ∎

Definition 4.4 *A point z with the properties of s_0 above will be called an* **(α, k)-centered point** *for f.*

Centered points will be, for us, the complex analogy of the splitting points used in the case of totally real root finding. We choose not to call them splitting points though, since their geometric properties are not absolutely identical.

Next we state the important root modulus approximation theorem which has been used many times in previous literature. This method due to Turin in 1968 [T 68, T 75, T 84] can be used to determine approximations to the magnitudes of all the roots of a polynomial. Schönhage [S 82] gave an efficient implementation of Turin's method. (See [P 87, R 93b] for other uses of Turin's method.)

Lemma 4.2 *There is an algorithm, which, given a polynomial, f, of degree n, and fixed positive constants c_1 and c_2, computes, with sequential complexity $O(n \log^2 n)$ or parallel time $O(\log^2 n)$ with n processors, for each root r_i of $f(z)$, an interval $I_i = [L_i, U_i]$ containing r_i, such that*

$$U_i \leq L_i(1+\gamma)\,,$$

and $\gamma = \frac{1}{c_1 n^{c_2}}$.

We now need to introduce the somewhat technical definition of *relative length*.

Definition 4.5 *Suppose $I = [L, U]$, is an interval. If $0 < L \leq U$, then the* **relative length** *of I is the quantity*

$$| I |_{rel} \;=\; \frac{U}{L} - 1$$

If $L \leq 0$ and $U > 0$ we define the relative length of I to be ∞, and if $0 < U < L$, we take the relative length of I to be 0. In any other case, we must have $U \leq 0$, and we define the relative length of I to be the relative length of $[-U, -L]$.

Similarly, if $D = D(c; R)$ is a disk in the complex plane, we define the relative diameter, $d(D)_{rel}$, of D to be the relative length of the interval $[\,|c|-R\,,\;|c|+R\,]$.

Finally, for an annulus, A, centered at the origin, we let its relative width, $|A|_{rel}$, be the relative length of its intersection with the positive real axis.

By combining the results of the previous section with lemma 4.2, we are led to the following

Theorem 4.1 *Let f be a polynomial of degree $d \leq n$, and with $|f| < 2^{2n+nm}$. Then there is an algorithm requiring $O(d \log^2 d \log b \log d^2) \leq O(d \log^3 d \log b)$ arithmetic operations, and at most ϕ bits of precision, which computes a sequence of $q \leq 5$ full precision factors $f_0, \ldots, f_q$, and a sequence of corresponding geometric regions $V_0, \ldots, V_q$ with these properties:*

1. *The region V_0 is the only unbounded region.*
2. *The regions $V_1, \ldots, V_q$ are all contained in a disk, $\mathcal{D}$, of radius R.*
3. *If $0 \leq i \neq j \leq q$, then the distance between the two regions V_i and V_j, $\delta(V_i, V_j)$, satisfies*

$$\delta(V_i, V_j) \;>\; \frac{1}{168 d^2} R^2. \tag{33}$$

4. *If ζ is a root of f, then $\zeta \in V_i$ for some i.*

5. *If V_i contains more than $d/2$ roots of f, then V_i is contained in a disk, $\mathcal{D}'$, with relative diameter smaller than $dR/21$.*

Remark 4.2 *In the statement of theorem 4.1, we have made what appears to be an odd choice of constants (168, 21 ...). The reason for the choices will become clear later. We could have chosen any other pair of constants with the same ratio, and the statement would still be correct, but the complexity of the algorithm is dependent on the choice, so we choose to fix them explicitly, otherwise this fact would be obscured in the theorem's statement.*

Proof of theorem 4.1: Apply lemma 4.2 with $c_1 = 168$ and $c_2 = 2$. For $\lfloor d/4 \rfloor \leq i \leq \lceil 3d/4 \rceil$, we consider the intervals $J_i = [\, U_i, L_{i+1} \,]$. If any one of these intervals, J_k, has relative length greater than or equal to $\frac{1}{168d^2}$, then by applying the factorization algorithm of the previous section, with $\delta = \frac{1}{400d^2}$, we can take $R = U_k$, $q = 1$, and let $V_1 = D(0;R)$, and let V_0 be the complement of $D(0;L_{k+1})$, and take f_1 and f_0 to be the factors with roots in the corresponding regions.

Thus we shall suppose that

$$| J_i |_{rel} < \frac{1}{168d^2}$$

for all $\lfloor d/4 \rfloor \leq i \leq \lceil 3d/4 \rceil$. Let

$$i_0 = \max_{i < \lfloor d/4 \rfloor} \{ i : | I_i |_{rel} > \frac{1}{168d^2} \}$$

and

$$i_1 = \min_{i > \lceil 3d/4 \rceil} \{ i : | I_i |_{rel} > \frac{1}{168d^2} \} .$$

Finally let

$$I = [U_{i_0}, L_{i_1}] .$$

One easily checks then that

$$| I |_{rel} < \frac{1}{84d} . \tag{34}$$

Set

$$V_0 = (D(0; U_{i_1}))^c$$

and, using the factorization algorithm of the previous section, again with $\delta = \frac{1}{400d^2}$, compute f_0, the full precision factor of f corresponding to the roots of f in V_0. Let g_0 be the other full precision factor of f, so all roots of g_0 lie in $D(0; L_{i_1})$, and set

$$V_1 = D(0; L_{i_0})$$

and, as with V_0, compute f_1, the full precision factor of g_1 corresponding to the roots of g_1 in V_1. Let g_2 be the other full precision factor of g_1. Then all roots of g_2 must lie in the annulus

$$A = D(0; L) - D(0; U) ,$$

(where $L = L_{i_1}$ and $U = U_{i_0}$) which, by equation 34, satisfies

$$| A |_{rel} < \frac{1}{84d} . \tag{35}$$

Now let $c = 2L_{i_1}$ and translate coordinates to the point $X_1 = c + 0i$. Repeat the construction once again using the polynomial g_2 in place of f. If this successfully factors g_2 into factors of degree at most $d/2$ we are done. If not, we will have constructed another annulus

$$A_1 = D(X_1; L_1) - D(X_1; U_1) ,$$

(and corresponding polynomial factor g_4) containing more than $d/2$ roots of g_2. (Recall that d is the degree of f, even though the degree of g_2 may be smaller.) And, incidentally, we will also have computed V_2 and f_2, and V_3 and f_3, which correspond to the factors f_0 and g_0 that we computed when we began with f instead of with g_2, but, by design, these are guaranteed to be of degree less than $d/2$.

Since all the roots of g_2 were in A, we can conclude two facts.

1. All the roots of g_4 lie in $A \cap A_1$.

2. $$(1 - \frac{1}{84d})L < L_1 < 3(1 + \frac{1}{84d})L \, . \tag{36}$$

We now distinguish two cases:

1. **Case 1:** $(4/3)L \leq L_1 \leq (8/3)L$.

2. **Case 2:** $L_1 < (4/3)L$ *or* $(8/3)L < L_1$.

In case 1, it is easy to see that $A \cap A_1$ consists of two convex regions of bounded aspect ratio, V_4 and V_5, each one contained in a disk of radius at most R/d, and satisfying $\delta(V_4, V_5) > (1/4)R$. So, here we are almost done. We can apply one more factorization step (in fact, this time – and only this time – we can use the simpler factorization algorithm of lemma 3.1 rather than the more complicated factorization method developed in the previous section) in order to produce the corresponding factors f_4 and f_5 of g_4. The properties of the regions V_i can now be easily verified using equation 36.

In case 2, we abandon our attempt to use X_1 as an origin, and translate coordinates instead to the new origin $X_2 = 0 + ci$ (relative to the very first coordinate system). Since case 1 did not occur with respect to the first change of coordinates, it now *must* occur in this coordinate system – one simply looks at the elementary geometry of the three intersecting annuli. Hence, by the reasoning of the previous paragraph, we have completed the proof. ∎

We can now state the following crucial corollary, whose proof follows immediately from the definition of a centered point, and from the stated properties of the regions V_i.

Corollary 4.1 *If the origin is a* $(1/2, 21d)$*-centered point for* f

of degree d, then none of the regions V_i, $0 \leq i \leq q \leq 5$, contain more than $d/2$ roots of f.

5 The Root Finding Algorithm and its Complexity

We now have the tools to give a recursive formulation of the entire root finding algorithm. However, the proof of its correctness depends on the following crucial theorem, which is a simple consequence of the main result in [CN 94].

Theorem 5.1 *If $0 < \beta < 1$, then the set of roots of the $(\lceil \beta n \rceil - 1)^{th}$ derivative of f, $f^{(\lceil \beta n \rceil - 1)}(z)$, is a $(\beta, 21n)$-splitting set for f.*

In fact, the results of [CN 94] show that the roots of $f^{(\lceil \beta n \rceil - 1)}(z)$ are even better than a $(\beta, 21n^{1/3})$-splitting set for f (recall we are assuming $\beta \geq 1/2$), but in this paper we will not take advantage of this stronger statement.

We begin with a subroutine for computing a balanced factorization of f. The idea of this subroutine is to use the same construction as we used in the main algorithm of the previous section. In fact, the output of the balanced factorization subroutine will have exactly the same form as the output of that algorithm, except that we will do some extra work, and rely on theorem 5.1 to guarantee that none of the regions V_i contain more than $n/2$ roots – and hence that none of the full precision factors f_i have degree greater than $n/2$.

Balanced Factorization Algorithm

Let

$$\varepsilon = \frac{1}{(21)(336)n^3}\,.$$

1. Compute $g(z) = f^{(\lceil n/2 \rceil)}(z)$.

2. Recursively compute the balanced factorization $g = \prod_{i=0}^{q} g_i$ $(0 \leq q \leq 5)$ of g, along with the corresponding regions V_i. Let R be the bounding radius, as before, and let $\mathcal{D} = D(0; R)$.

3. Apply one iteration of the algorithm of theorem 4.1 to f. If none of the resulting factors has degree greater than $n/2$,

   then **RETURN**.

   Otherwise, let $\mathcal{D}' = D(c; r)$ be the "small disk" containing more than $n/2$ roots of f computed in the algorithm.

4. Translate coordinates so that c is the origin.

5. If the diameter of $\mathcal{D}'$, $d(\mathcal{D}')$, satisfies $d(\mathcal{D}') < \varepsilon R$, go on to step 6. Otherwise, **LOOP** to step 3. (In the following discussion of the algorithm, we will prove that we cannot loop more than 3 times.)

6. At this point, c must lie either in, or within a distance of $\frac{1}{336n^2}$ of *at least one* of the regions V_{i_0} computed in step 2 (we shall prove this claim shortly). However, it follows from the properties of these regions (theorem 4.1, property 3) that this region must be unique.

7. Set $g = g_{i_0}$, the polynomial factor corresponding to V_{i_0} (the degree is now reduced by a factor of at least 2) and **LOOP** to step 2.

We now need to show that this algorithm actually does terminate, and to analyze its complexity. We will do this with the following lemmas.

Definition 5.1 *There are two nested loops in the algorithm above. We call the loop from step* 5 *to step* 3*, the internal loop, and the loop from step* 7 *to step* 2*, the external loop.*

Lemma 5.1 *Each iteration of the inner loop decreases the diameter of the disk* $\mathcal{D}$ *by a factor smaller than* $\frac{1}{21n}$.

Proof : Let r be the radius of $\mathcal{D}'$ at the start of step 3. (Recall that at this point $\mathcal{D}'$ is centered at the origin.) We know that $|\mathcal{D}'|_{rel} < \frac{1}{21n}$.

Since we have not terminated the computation by returning, *the number of roots in* $\mathcal{D}'$ *is greater than* $n/2$, so we also know

$$\mathcal{D}' \cap \mathcal{D} \neq \emptyset \,.$$

The conclusion of the lemma thus follows immediately from the definition of relative diameter. ∎

The following corollary now follows immediately from the definition of ε.

Corollary 5.1 *The inner loop executes no more than* 3 *times.*

Lemma 5.2 *In step* 6*, the point* c *lies either within, or within a distance of* $\frac{R}{336n^2}$ *of, at least one of the regions* V_i*.*

Proof : First, assume inductively that at least one of the roots of $g(z)$ is a $(1/2, 21n)$-centered point for f. At the start of the computation, step 1, this is true by theorem 5.1. Now, in step 6, the disk $\mathcal{D}'$ has radius at most εR, and it contains more than $n/2$ roots of f, so by definition, *any* $(1/2, 21n)$-centered point for f must lie within a distance of $21\varepsilon R$ of c. Hence, by the inductive assumption, c lies within a distance of $\frac{R}{336n^2}$ of at least one root of g.

But *all* roots of g lie *in* one of the V_i, so the distance from c to at least one V_i, say V_{i_0}, must be less than $\frac{R}{336n^2}$. By the choice of ε, and by theorem 4.1, property 3, V_{i_0} is the only region which can lie this close to c. Moreover, by the previous paragraph, the other regions can not contain any $(1/2, 21n)$-centered points, so V_{i_0} must contain all of the roots of g which *are* $(1/2, 21n)$-centered points. Since there is at least one of these, by the inductive assumption again, the chosen factor g_{i_0} must have at least one root

which is a $(1/2, 21n)$-centered point. This then, in turn, proves the inductive hypothesis. ∎

As a consequence of the proof, we see that the polynomial g always has a root which is $(1/2, 21n)$-centered for f. Since the degree of g is reduced by a factor of at least $1/2$ each time we pass through the outer loop, by corollary 4.1:

Theorem 5.2 *After at most* $\log n$ *iterations of the outer loop, the Balanced Factorization Algorithm returns from step* 3*, with a factorization of* f *having no factors of degree greater than* $n/2$*.*

Corollary 5.2 *The arithmetic complexity of the Balanced Factorization Algorithm is* $O(n \log^4 n \log b)$*.*

Proof : Let $T(n)$ be the number of arithmetic operations performed by the Balanced Factorization Algorithm on an input polynomial of degree n. We need to determine a constant $C > 0$, such that

$$T(n) \; < \; Cn \log^4 n \log b. \tag{37}$$

By theorem 4.1, if we neglect the number of operations in the recursive call to the Balanced Factorization Algorithm in step 2, the number of arithmetic operations performed in each pass through the outer loop is bounded above by $Kn \log^3 n \log b$ for a constant K. Since the number of operations performed in the recursive call is at most $T(n/2)$, we have

$$T(n) \; < \; \sum_{j=0}^{\log n - 1} \left[T(\frac{n}{2^{j+1}}) \; + \; K \frac{n}{2^j} \log^3 (\frac{n}{2^j}) \log b \right] \; . \tag{38}$$

Since the roots of quadratic polynomials have explicit formulas which can be easily approximated, we can assume that $n > 2$, in which case, for $i \geq 1$,

$$\log^4(\frac{n}{2^i}) \;=\; (\log n - i)^4 \;<\; \log^4 n - i\log^3 n\,. \tag{39}$$

We claim then that it suffices to take $C = 2K$. To see this, suppose this value for C has been picked, and assume, inductively, that equation 37 is satisfied for all $n' < n$. Substituting into equation 38, and using the estimate of equation 39 we have

$$T(n) \;<\; \sum_{j=0}^{\log n-1} C\frac{n}{2^{j+1}}\log^4 n\log b - \sum_{j=0}^{\log n-1} C\frac{n}{2^{j+1}}(j+1)\log^3 n\log b + \sum_{j=0}^{\log n-1} K\frac{n}{2^j}\log^3 n\log b. \tag{40}$$

Since

$$\sum_{j=0}^{\log n-1} \frac{n}{2^{j+1}}\log^4 n\log b \;<\; n\log^4 n\log b$$

and

$$\sum_{j=0}^{\log n-1} \frac{n}{2^{j+1}}(j+1)\log^3 n\log b \;>\; n\log^3 n\log b$$

and

$$\sum_{j=0}^{\log n-1} \frac{n}{2^j}\log^3 n\log b \;<\; 2n\log^3 n\log b$$

we have

$$T(n) \;<\; (Cn\log^4 n - Cn\log^3 n + 2Kn\log^3 n)\log b \;=\; Cn\log^4 n\log b. \tag{41}$$

∎

The full root finding algorithm is now immediate; we simply call the Balanced Factorization Algorithm recursively until all factors are linear. Let $Q(n)$ be the number of arithmetic operations for the full root finding algorithm. Then

$$Q(n) \ < \ T(n) + 2T(\frac{n}{2}) + \cdots = \sum_{j=0}^{\log n - 1} 2^j T(\frac{n}{2^j}),$$

so

$$Q(n) \ < \ \sum_{j=0}^{\log n - 1} 2^j C \frac{n}{2^j} \log^4(\frac{n}{2^j}) \log b = \sum_{j=0}^{\log n - 1} Cn \log^4(\frac{n}{2^j}) \log b \ < \ Cn \log^5 n \log b.$$

6 Parallization of our Root Finding Algorithm

We now briefly discuss the parallel execution of our root finding algorithm, which can be immediately parallelized. We assume an arithmetic CREW PRAM model, where each processor can execute an arithmetic scalar operation in a single step, and with concurrent reads and exclusive writes on the stored memory. Our root algorithm relies on a number of basic operations on polynomials of degree n, which can be executed using n processors in the indicated times (see [BM 75, J 92, R 93a])

- polynomial translation, sum, and product in $O(\log n)$ time,
- polynomial evaluation and interpolation at n points in $O(\log^2 n)$ time.

Also, we can compute polynomial resultants in in parallel time $O(\log^2 n)$ using $n \log^\omega n$ processors, where $\omega = 2.376$, by use of the fast parallel Teoplitz and bounded displacement rank matrix algorithms given in [R 95]. Our root algorithm requires $O(\log^4 n) \log b$ stages of these polynomial operations as

well as resultant computations, and the sum of the degrees of all the polynomials operated on at each stage is at most $O(n)$. Thus the total parallel time is a factor $O(\log^2 n)$ more, namely $O(\log^6 n) \log b$ using $O(n \log^\omega n)$ processors. Using a constant factor slow down, we have:

Theorem 6.1 *All roots of a degree n complex polynomial can be computed in parallel time $O(\log^6 n) \log b$ using $n \log^\omega n$ processors.*

7 Conclusion and Open Problems

A previous draft of this paper [NR 94] gave a $O(n^{1+\epsilon} \log b)$ sequential bound for the complex root problem, which was improved to our current bounds of $O(n \log^5 n) \log b$ by a balancing routine developed by the first author. Pan has announced in [P95] a similar bound of $O(n \log^{O(1)} n) \log b$ for the complex root problem using the techniques developed in [NR 94] and also using this balancing routine communicated to Pan by the first author.

These bounds are really quite remarkable, when you consider that the number of steps required simply to *multiply* two polynomials of degree n by the elementary method taught in grade school is $O(n^2)$, and $O(n \log n)$ by convolution.

It remains an open problem to reduce the complexity of the complex root finding problem to cost of other basic polynomial operations. For example, polynomial evaluation and interpolation at n points costs $O(n \log^2 n)$ sequential time and $O(\log^2 n)$ parallel time using n processors. Can we reduce the sequential complexity of the root finding problem to the following bounds: $O(n \log n(b + \log n))$ sequential time or $O(\log n(b + \log n))$ parallel time using n processors?

References

[AHU 74] A.V.Aho, J.E.Hopcroft and J.D.Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

[BFKT 86] M.Ben-Or, E.Feig, D.Kozen and P.Tiwari, *A Fast parallel algorithm for determining all roots of a polynomial with Real Roots*, SIAM J. Comput. (1986).

[BT 90] M.Ben-Or and P.Tiwari, *Simple algorithms for approximating all roots of a polynomial with real roots*, Journal of Complexity **6** (1990), 417–442.

[B 92] D. Bini, *Divide and conquer techniques for the polynomial root-finding problem*, Proceedings of the 1st World (International) Congress of Nonlinear Analysts, Tampa, August (1992).

[BP 86] D. Bini and V. Pan, *Polynomial division and its computational complexity*, Journal of Complexity **2** (1986), 179-203.

[BA 80] R.R.Bitmead and D.O.Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra and its Applications **34**, (1980), 103–116.

[BM 75] A.Borodin and I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier (1975).

[BGY 80] R.P.Brent, F.G.Gustavson, and D.Y.Y.Yun, *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms **1** (1980), 259–295.

[BT 71] W.S.Brown and J.F.Traub, *On Euclid's algorithm and the theory of subresultants*, J. ACM **18** (1971), 505–514.

[BGH 82] A.Borodin, J.von zur Gathen and J.Hopcroft, *Fast parallel matrix and GCD computations*, Information and Control **52** (1982), 241–256.

[CR 93] J.Cheriyan and J.H.Reif, *Parallel and output sensitive algorithms for combinatoral and linear algebra problems*, Proc. of Symp. on Parallel Algorithms and Architectures '93 (1993).

[C 66] G.E.Collins, *Polynomial remainder sequences and determinants*, Amer. Math. Monthly **73** (1966), 708–712.

[CL 82] G.E.Collins and R.Loos, *Real zeros of polynomials*, Computing (1982), Springer-Verlag.

[CN 94] D.Coppersmith and C.A.Neff, *Roots of a polynomial and its derivatives,* Fifth Annual ACM - SIAM Symposium on Discrete Algorithms (SODA '94) Proceedings, (1994), 271–279.

[E 87] H.Edelsbrunner, *Algorithms in Computational Geometry*, Springer-Verlag (1987).

[GM 77] S. Gal and W. Miranker, *Optimal sequential and parallel search for finding a root*, Journal of Combinatoral Theory **23** (1977).

[GH 72] I.Gargantini and P.Henrici, *Circular arithmetic and the determination of polynomial zeros*, Num. Math. **18** (1972), 305–320.

[H 74] P.Henrici, *Applied and Computational Complex Analysis, Vols. 1,2, and 3*, Wiley (1974).

[H 70] A.S.Householder, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill (1970).

[J 92] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley (1992).

[JT 70] M.A. Jenkins and J.F. Traub, *A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration*, Numer. Math., **14** (1970), 252-263.

[Mar 66] M.Marden, *Geometry of Polynomials*, American Mathematical Society (1966).

[M 92] M.Mignotte, *Mathematics for Computer Algebra*, Springer-Verlag (1992).

[N 90] C.A.Neff, *Specified precision polynomial root isolation is in NC*, Proc. 24th Annual IEEE Symp. on Foundations of Computer Science, 138–145. (1990).

[N 94] C.A.Neff, *Specified precision polynomial root isolation is in NC*, Journal of Computer and System Sciences **48** (1994), 429–463.

[NR 94] C.A.Neff and J.H.Reif, *An $O(n^{1+\epsilon} \log b)$ algorithm for the complex roots problem*, Proc. 35th Annual IEEE FOCS Santa Fe, NM.

[Pn 87] V.Y.Pan, *Algebraic complexity of computing polynomial zeros*, Comput. Math. Applic. **14** (1987), 285–304.

[P 87] V.Y.Pan, *Sequential and parallel complexity of approximate evaluation of polynomial zeros*, Comput. Math. Applic. **14** (1987), 591–622.

[P 89] V.Y.Pan, *Fast and efficient parallel evaluation of the zeros of a polynomial having only real zeros*, Computers Math. Applic. **17** (1989), 1475-1480.

[P 94] V.Y.Pan, *Deterministic improvement of complex polynomial factorization based on the properties of the associated resultant*, unpublished manuscript.

[P95] V. Y. Pan, *Improved algorithms for approximating complex polynomial zeros*, Proc. of the 27th ACM Symposium on Theory of Computing (STOC) Las Vegas, Nevada, (May, 1995).

[PS 85] F.P.Preparata and M.I.Shamos, *Computational Geometry: An Introduction*, Springer-Verlag (1985).

[R 93a] J.H.Reif, ed., *Synthesis of Parallel Algorithms*, Morgan Kaufmann (1993).

[R 93b] J.H. Reif, *An $O(nlog^3n)$ algorithm for the real root problem,* 34th Annual IEEE Conference on Foundations of Computer Science (FOCS '93) Proceedings, (November 1993), Palo Alto, CA.

[R 95] J.H.Reif, *Work efficient parallel solution of Toeplitz systems and polynomial GCD*, 27th Annual ACM Symposium of Theory of Computing (STOC95), Las Vegas, Nevada, May,1995.

[R 85] J.Renegar, *On the Cost of approximating all roots of a complex polynomial*, Math. Prog. **32** (1985), 319–336.

[R 87] J.Renegar, *On the worst-case arithmetic complexity of approximating zeros of polynomials*, Proc. 2nd Symp. on the Complexity of Approx. Solved Problems (1987).

[S 82] A.Schönhage, *The Fundamental Theorem of Algebra in terms of computational complexity*, Univ. of Tübingen, Tübingen, Germany, Unpublished Manuscript (1982).

[SS 85] M.Shub and S.Smale, *Computational Complexity: On the geometry of polynomials and a theory of cost, Part I*, Annls. Scient. Ecole Norm. Sup. **4** (1985), 107–142.

[SS 86] M.Shub and S.Smale, *Computational Complexity: On the geometry of polynomials and a theory of cost, Part II*, SIAM J. Computing **15** (1986), 145–161.

[S 81] S.Smale, *The Fundamental Theorem of Algebra and complexity theory*, Bull. A.M.S. **4** (1981), 1–36.

[T 68] P.Turan, *On the approximate solution of algebraic equations*, Comm. Math. Phys. Class Hung. Acad. **XVIII** (1968), 223–236.

[T 75] P.Turan, *Power sum method and approximative solution of algebraic equations*, Math. Computation **29** (1975), 311–318.

[T 84] P.Turan, *On a New Method of Analysis and Its Applications*, Wiley (1984).