MATHEMATICS OF COMPUTATION
Volume 74, Number 250, Pages 869–903
S 0025-5718(04)01692-8
Article electronically published on July 22, 2004

COMPUTING MULTIPLE ROOTS OF INEXACT POLYNOMIALS

ZHONGGANG ZENG

Abstract. We present a combination of two algorithms that accurately calculate multiple roots of general polynomials.

Algorithm I transforms the singular root-finding into a regular nonlinear least squares problem on a pejorative manifold, and it calculates multiple roots simultaneously from a given multiplicity structure and initial root approximations. To fulfill the input requirement of Algorithm I, we develop a numerical GCD-finder containing a successive singular value updating and an iterative GCD refinement as the main engine of Algorithm II that calculates the multiplicity structure and the initial root approximation. While limitations exist in certain situations, the combined method calculates multiple roots with high accuracy and consistency in practice without using multiprecision arithmetic, even if the coefficients are inexact. This is perhaps the first blackbox-type root-finder with such capabilities.

To measure the sensitivity of the multiple roots, a structure-preserving condition number is proposed and error bounds are established. According to our computational experiments and error analysis, a polynomial being ill-conditioned in the conventional sense can be well conditioned with the multiplicity structure being preserved, and its multiple roots can be computed with high accuracy.

1. Introduction

In this paper, we present a combination of two numerical algorithms for computing multiple roots and the multiplicity structures of polynomials. According to our extensive computational experiments and error estimates, the method accurately calculates polynomial roots of nontrivial multiplicities without using multiprecision arithmetic, even if the coefficients are inexact.

Polynomial root-finding is among the classical problems with the longest and richest history. One of the most difficult issues in root-finding is computing multiple roots. In addition to requiring *exact* coefficients, multiprecision arithmetic may be needed when multiple roots are present [27, p. 196]. In fact, using multiprecision arithmetic has been a common practice in designing root-finding algorithms and software, such as those in [2, 14, 16]. Moreover, there is a so-called "attainable accuracy" in computing multiple roots [19, 27, 35]: to calculate an m-fold root to the precision of k correct digits, the accuracy of the polynomial coefficients *and* the machine precision must be at least mk digits. This "attainable accuracy" barrier also suggests the need of using multiprecision arithmetic. Multiprecision software such

Received by the editor January 13, 2003 and, in revised form, September 17, 2003.
2000 *Mathematics Subject Classification.* Primary 12Y05, 65H05; Secondary 65F20, 65F22, 65F35.
Key words and phrases. Polynomial, root, multiplicity.

©2004 American Mathematical Society

as [1] are available. However, when polynomial coefficients are truncated, multiple roots turn into clusters, and extending machine precision will never reverse clusters back to multiple roots. In the absence of accurate methods that are independent of multiprecision technology, multiple roots of perturbed polynomials would indeed be intractable.

While multiple roots are considered hypersensitive in numerical computation, W. Kahan [21] proved that if the multiplicities are preserved, those roots may actually be well behaved. More precisely, polynomials with a fixed multiplicity structure form a pejorative manifold. A polynomial is ill-conditioned if it is near such a manifold. On the other hand, for the polynomial on the pejorative manifold, its multiple roots are insensitive to multiplicity-preserving perturbations, unless the polynomial is also near a submanifold of higher multiplicities. Therefore, to calculate multiple roots accurately, it is important to maintain the computation on a proper pejorative manifold.

In light of Kahan's theoretical insight, we propose Algorithm I in §3 that transforms the singular root-finding into a regular nonlinear least squares problem on a pejorative manifold. By projecting the given polynomial onto the manifold, the computation remains structure preserving. As a result, the roots can be calculated simultaneously and accurately.

In applying Algorithm I, one needs a priori knowledge on the initial root approximation as well as multiplicity identification, which is often attempted by estimation (e.g., [30, 35]) or clustering (e.g., [4, 15, 25]) with unknown certainty. To fulfill the input requirement of Algorithm I, it is preferable to have an algorithm that systematically calculates the multiplicity structure. One of the main difficulties in identifying the multiplicity structure is the lack of a robust method for computing the polynomial greatest common divisor (GCD).

Recently, many different approaches and strategies have been proposed for the numerical computation of approximate GCD of univariate polynomials [5, 6, 13, 18, 22, 26, 28]. We have been inspired by those endeavors, especially the pioneering work of Corless, Gianni, Trager and Watt [6], which identifies the GCD degree by the total singular value decomposition (SVD) of the full Sylvester matrix followed by the suggestion of four possible alternative ways to compute the GCD using the degree. We propose a numerical GCD-finder that employs a successive updating on a sequence of Sylvester submatrices for the smallest singular values only, followed by extracting the degree *and* the coefficients of the GCD decomposition from the singular vector as the initial iterate, and finally applies the Gauss-Newton iteration to refine the approximate GCD decomposition. As a result, the GCD-finder is a blackbox-type algorithm in its own right and constitutes the main engine of our proposed Algorithm II in §4 which, with some limitations specified in §4.5, calculates the multiplicity structure and its initial root approximation for a given polynomial.

In §3.4, we propose a structure-preserving condition number that measures the sensitivity of multiple roots. A polynomial that is ill-conditioned in the conventional sense can be well conditioned with the multiplicity structure being preserved, and its roots can be calculated far beyond the barrier of "attainable accuracy". This condition number can be calculated easily. Error bounds on the roots are established for inexact polynomials.

In §3.6 and §4.6, we present separate numerical results for Algorithms I and II. The numerical results for the combined algorithm are shown in §5. Both algorithms and their combination are implemented as a Matlab package MultRoot, which is available electronically from the author. This paper also elaborates on certain preliminary results in [37] by providing detailed proofs and discussions.

Although our emphasis at this stage is on accuracy rather than fast computation, we prove local convergence of our Algorithm I when the multiplicity structure and close initial approximations are available. We also prove the local convergence of the GCD iteration in the midst of Algorithm II, which calculates the multiplicity structure with high accuracy and consistency in practice along with initial root approximations . The combined algorithm takes the coefficient vector as the *only* input and output results that include the roots and their multiplicities as well as the backward error, the estimated forward error, and the structure-preserving condition number. The most significant features of the algorithm are its high accuracy and its robustness in handling inexact data. As shown in numerical examples, the code accurately identifies the multiplicity structure and multiple roots for polynomials with a coefficient accuracy being as low as seven digits. With given multiplicities, Algorithm I converges even with data accuracy as low as three decimal digits. While limitations exist when the polynomial is ill-conditioned in the sense of the structure-preserving sensitivity that we define in §3.4, the code appears to be the first blackbox-type root-finder with the capability to calculate roots and multiplicities beyond the barrier of "attainable accuracy".

While numerical experiments reported in the literature seldom reach multiplicity 10, we successfully tested our algorithms on polynomials with root multiplicities as high as 400 without using multiprecision arithmetic. We are aware of no other reliable methods that calculate multiple roots accurately by using standard machine precision. Accurate results for multiple root computation that we have seen in the literature, such as the methods in [14], can be repeated only if multiprecision arithmetic is used on exact polynomials. A zero-finder for general analytic functions with multiple zeros has been developed by Kravanja and Van Barel [23]. The method uses an accuracy refinement with a modified Newton's iteration that may also require multiprecision arithmetic for multiple roots unless the polynomial is already factored [39].

There exist general-purpose root-finders using $O(n^2)$ flops or less, such as those surveyed in [27]. However, the barrier of "attainable accuracy" may prevent those root-finders from calculating multiple roots accurately when the polynomials are inexact (e.g., see Figure 10 in §4.6) even if multiprecision arithmetic is used. Our algorithms provide the option of reaching high accuracy on multiple roots at the higher computing cost of $O(n^3)$, which may not be a lofty price to pay.

The idea of exploiting the pejorative manifold and problem structure has been applied extensively for ill-conditioned problems. Besides Kahan's pioneering work 30 years ago, theories and computational strategies for the matrix canonical forms have been studied (see [9, 11, 12, 24]) to take advantage of the pejorative manifolds or varieties. At present, it is not clear if those methods can be applied to polynomials with multiple roots.

2. Preliminaries

2.1. **Notation.** In this paper, $\mathbb{R}^n$ and $\mathbb{C}^n$ denote the n-dimensional real and complex vector spaces, respectively. Vectors, always considered columns, are denoted by boldface lower-case letters, and matrices are denoted by upper case letters. Blank entries in a matrix are filled with zeros. The notation $(\cdot)^\top$ represents the transpose of $(\cdot)$, and $(\cdot)^H$ is the Hermitian adjoint (i.e., conjugate transpose) of $(\cdot)$. When we use a (lower case) letter, say p, to denote a polynomial of degree n, then $p_0, p_1, \ldots, p_n$ are its coefficients as in

$$p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_n.$$

The same letter in boldface (e.g., $\mathbf{p}$) denotes the coefficient (column) vector

$$\mathbf{p} = (\, p_0, p_1, \ldots, p_n \,)^\top,$$

unless it is defined otherwise. The degree of p is $\deg(p)$. For a pair of polynomials p and q, their greatest common divisor (GCD) is denoted by $\mathrm{GCD}(p, q)$.

2.2. **Basic definitions and lemmas.**

Definition 2.1. Let $p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_n$ be a polynomial of degree n. For any integer $k \geq 0$, the matrix

$$C_k(p) = \overbrace{\begin{bmatrix} p_0 & & \\ p_1 & \ddots & \\ \vdots & \ddots & p_0 \\ p_n & & p_1 \\ & \ddots & \vdots \\ & & p_n \end{bmatrix}}^{k+1}$$

is called the kth order **convolution matrix** associated with p.

Lemma 2.2. *Let f and g be polynomials of degrees n and m, respectively, with $h(x) = f(x)g(x)$. Then* $\mathbf{h}$ *is the* **convolution** *of* $\mathbf{f}$ *and* $\mathbf{g}$ *defined by*

$$\mathbf{h} = \mathrm{conv}(\mathbf{f}, \mathbf{g}) = C_m(f)\mathbf{g} = C_n(g)\mathbf{f}.$$

Proof. A straightforward verification. □

Definition 2.3. Let p be a polynomial of degree n, and let p' be its derivative. For $k = 1, 2, \ldots n-1$, the matrix of size $(n+k) \times (2k+1)$

$$S_k(p) = \left[\; C_k(p') \;\middle|\; C_{k-1}(p) \;\right]$$

is called the kth **Sylvester discriminant matrix**.

Lemma 2.4. *Let p be a polynomial of degree n, and let p' be its derivative with $u = \mathrm{GCD}(p, p')$. For $j = 1, \ldots, n$, let ς_j be the smallest singular value of $S_j(p)$. Then the following are equivalent:*

(a) $\deg(u) = m$,
(b) *p has $k = n - m$ distinct roots,*
(c) $\varsigma_1,\ \varsigma_2,\ \ldots, \varsigma_{k-1} > 0, \qquad \varsigma_k = \varsigma_{k+1} = \cdots = \varsigma_n = 0.$

Proof. The equivalence between (a) and (b) is trivial to verify, and the assertion that (a) is equivalent to (c) is part of Proposition 3.1 in [28]. ☐

Lemma 2.5. *Let p be a polynomial of degree n, and let p' be its derivative with $u = \mathrm{GCD}(p, p')$ and $\deg(u) = m = n - k$. Let v and w be polynomials that satisfy*

$$u(x)v(x) = p(x), \quad u(x)w(x) = p'(x).$$

Then

(a) *v and w are coprime, namely they have no common factors;*
(b) *the (column) rank of $S_k(p)$ is deficient by one;*
(c) *the normalized vector $\begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix}$ is the right singular vector of $S_k(p)$ associated with the smallest (zero) singular value ς_k;*
(d) *if $\mathbf{v}$ is known, the coefficient vector $\mathbf{u}$ of $u = \mathrm{GCD}(p, p')$ is the solution to the linear system $C_m(v)\mathbf{u} = \mathbf{p}$.*

Proof. Assertion (a) is trivial. $S_k(p) \begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix} = C_k(p')\,\mathbf{v} - C_{k-1}(p)\mathbf{w} = 0$ because it is the coefficient vector of $p'v - pw \equiv (uw)v - (uv)w \equiv 0$. Let $\hat{\mathbf{v}} \in \mathbb{C}^{k+1}$ and $\hat{\mathbf{w}} \in \mathbb{C}^k$ be coefficient vectors of polynomials $\hat{v}$ and $\hat{w}$, respectively, which also satisfy $C_k(p')\,\hat{\mathbf{v}} - C_{k-1}(p)\hat{\mathbf{w}} = 0$. Then we also have $(uw)\hat{v} - (uv)\hat{w} = 0$, namely $w\hat{v} = v\hat{w}$. Since v and w are coprime, there is a polynomial c such that $\hat{v} = cv$ and $\hat{w} = cw$, and c is obviously a constant. Therefore, the single vector $\begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix}$ forms the basis for the null space of $S_k(p)$. Consequently, both assertions (b) and (c) follow. The assertion (d) is a direct consequence of Lemma 2.2. ☐

Lemma 2.6. *Let $A \in \mathbb{C}^{n\times m}$ with $n \geq m$ be a matrix whose smallest two distinct singular values are $\hat{\sigma} > \tilde{\sigma}$. Let $Q \begin{pmatrix} R \\ 0 \end{pmatrix} = A$ be the QR decomposition of A, where $Q \in \mathbb{C}^{n\times n}$ is unitary and $R \in \mathbb{C}^{m\times m}$ is upper triangular. From any vector $\mathbf{x}_0 \in \mathbb{C}^m$ that is not orthogonal to the right singular subspace of A associated with $\tilde{\sigma}$, we generate the sequences $\{\sigma_j\}$ and $\{\mathbf{x}_j\}$ by the inverse iteration*

$$\left\{ \begin{array}{llcll} \textit{Solve} & R^H\mathbf{y}_j & = & \mathbf{x}_{j-1} & \textit{for } \mathbf{y}_j \in \mathbb{C}^m, \\ \textit{Solve} & R\,\mathbf{z}_j & = & \mathbf{y}_j & \textit{for } \mathbf{z}_j \in \mathbb{C}^m, \\ \textit{Calculate} & \mathbf{x}_j & = & \dfrac{\mathbf{z}_j}{\|\mathbf{z}_j\|_2}, & \sigma_j = \left\| R\mathbf{x}_j \right\|_2, \end{array} \right. \qquad j = 1, 2, \ldots. \tag{2.1}$$

Then $\lim_{j\to\infty} \sigma_j = \lim_{j\to\infty} \|A\mathbf{x}_j\|_2 = \tilde{\sigma}$ and

$$\sigma_j = \left\| A\mathbf{x}_j \right\|_2 = \tilde{\sigma} + O\left(\tau^j\right), \quad \textit{where} \quad \tau = \left(\frac{\tilde{\sigma}}{\hat{\sigma}}\right)^2.$$

If $\tilde{\sigma}$ is simple, then $\mathbf{x}_j$ converges to the right singular vector $\tilde{\mathbf{x}}$ of A associated with $\tilde{\sigma}$.

Proof. See [32] for straightforward verifications. ☐

2.3. **The Gauss-Newton iteration.** The Gauss-Newton iteration is an effective method for solving nonlinear least squares problems. Let $G : \mathbb{C}^m \longrightarrow \mathbb{C}^n$ with $n > m$, and $\mathbf{a} \in \mathbb{C}^n$. The nonlinear system $G(\mathbf{z}) = \mathbf{a}$ for $\mathbf{z} \in \mathbb{C}^m$ is overdetermined with no conventional solutions in general. We thereby seek a *weighted least squares solution.* Let $W = \mathrm{diag}(\omega_1, \ldots, \omega_n)$ be a diagonal weight matrix with positive weights ω_j's. Let $\|\cdot\|_W$ denote the weighted 2-norm

$$\left\| \mathbf{v} \right\|_W \equiv \left\| W\,\mathbf{v} \right\|_2 \equiv \sqrt{\sum_{j=1}^{n} \omega_j^2 v_j^2}, \quad \text{for all } \mathbf{v} = (v_1, \ldots, v_n)^\top \in \mathbb{C}^n. \tag{2.2}$$

Our objective is to solve the minimization problem $\min_{\mathbf{z}\in\mathbb{C}^m} \left\| G(\mathbf{z}) - \mathbf{a} \right\|_W^2$.

Lemma 2.7. *Let $F : \mathbb{C}^m \longrightarrow \mathbb{C}^n$ be analytic with its Jacobian being $\mathcal{J}(\mathbf{z})$. If there is a neighborhood Ω of $\tilde{\mathbf{z}}$ in $\mathbb{C}^m$ such that $\|F(\tilde{\mathbf{z}})\|_2 \le \|F(\mathbf{z})\|_2$ for all $\mathbf{z} \in \Omega$, then $\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = 0$.*

Proof. The real case $F : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ of the lemma is proved in [10]. The proof for the complex case is nearly identical, except for using the Cauchy-Riemann equation. □

By Lemma 2.7, let $J(\mathbf{z})$ be the Jacobian of $G(\mathbf{z})$. To find a local minimum of $\|F(\mathbf{z})\|_2 \equiv \|W[G(\mathbf{z}) - \mathbf{a}]\|_2$ with $\mathcal{J}(\mathbf{z}) = WJ(\mathbf{z})$, we look for $\tilde{\mathbf{z}} \in \mathbb{C}^m$ satisfying

$$\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = \Big[WJ(\tilde{\mathbf{z}})\Big]^H W\Big[G(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = J(\tilde{\mathbf{z}})^H\, W^2\Big[G(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = 0.$$

In other words, $G(\tilde{\mathbf{z}}) - \mathbf{a}$ is orthogonal, with respect to $\langle \mathbf{v}, \mathbf{w} \rangle \equiv \mathbf{v}^H W^2 \mathbf{w}$, to the tangent plane of the manifold $\Pi = \Big\{ \mathbf{u} = G(\mathbf{z}) \;\Big|\; \mathbf{z} \in \mathbb{C}^m \Big\}$ at $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$.

The Gauss-Newton iteration can be derived as follows (see Figure 1). To find a least squares solution $\mathbf{z} = \tilde{\mathbf{z}}$ to the equation $G(\mathbf{z}) = \mathbf{a}$, we look for the point $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$ that is the orthogonal projection of $\mathbf{a}$ onto Π. Let $\mathbf{u}_0 = G(\mathbf{z}_0)$ in Π be near $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$. We can approximate the manifold Π with the tangent plane $P_0 = \Big\{ G(\mathbf{z}_0) + J(\mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0) \;\Big|\; \mathbf{z} \in \mathbb{C}^m \Big\}$. Then the point $\mathbf{a}$ is orthogonally

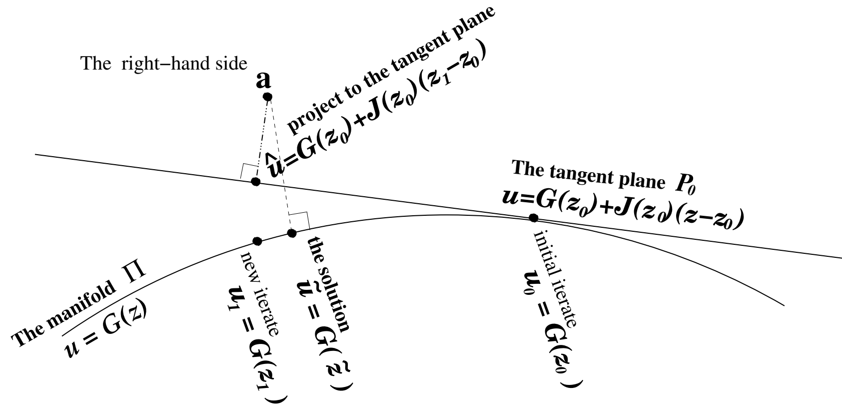

FIGURE 1. Illustration of the Gauss-Newton iteration.

projected onto the tangent plane P_0 at $\hat{\mathbf{u}} = G(\mathbf{z}_0) + J(\mathbf{z}_0)(\mathbf{z}_1 - \mathbf{z}_0)$ by solving the overdetermined linear system

$$G(\mathbf{z}_0) + J(\mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0) = \mathbf{a} \quad \text{or} \quad J(\mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0) = -[G(\mathbf{z}_0) - \mathbf{a}] \tag{2.3}$$

for its weighted least squares solution

$$\mathbf{z}_1 = \mathbf{z}_0 - \left[J(\mathbf{z}_0)_W^+\right][G(\mathbf{z}_0) - \mathbf{a}] \quad \text{with} \quad J(\mathbf{z}_0)_W^+ = \left[J(\mathbf{z}_0)^H W^2 J(\mathbf{z}_0)\right]^{-1} J(\mathbf{z}_0)^H W^2. \tag{2.4}$$

As long as $J(\mathbf{z}_0)$ is of full (column) rank, the pseudo inverse $J(\mathbf{z}_0)_W^+$ exists. Therefore $\mathbf{u}_1 = G(\mathbf{z}_1)$ is well defined and is expected to be a better approximation to $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$ than $\mathbf{u}_0 = G(\mathbf{z}_0)$. The Gauss-Newton iteration is then a recursive application of (2.4) (also see [7, 10]).

The convergence theory of the Gauss-Newton iteration has been well established for overdetermined systems in real spaces [10]. The following lemma is a straightforward generalization of Theorem 10.2.1 in [10] to complex spaces. Since the lemma itself as well as the proof are nearly identical to those in the real case in [10], we shall present the lemma without proof.

Lemma 2.8. *Let $\Omega \subset \mathbb{C}^m$ be a bounded open convex set and $F : D \subset \mathbb{C}^m \longrightarrow \mathbb{C}^n$ be analytic in an open set $D \supset \overline{\Omega}$. Let $\mathcal{J}(\mathbf{z})$ be the Jacobian of $F(\mathbf{z})$. Suppose that there exists $\tilde{\mathbf{z}} \in \Omega$ such that $\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = 0$ with $\mathcal{J}(\tilde{\mathbf{z}})$ full rank. Let σ be the smallest singular value of $\mathcal{J}(\tilde{\mathbf{z}})$. Let $\delta \geq 0$ be a constant such that*

$$\left\| \left[\mathcal{J}(\mathbf{z}) - \mathcal{J}(\tilde{\mathbf{z}})\right]^H F(\tilde{\mathbf{z}}) \right\|_2 \leq \delta \left\| \mathbf{z} - \tilde{\mathbf{z}} \right\|_2 \quad \text{for all} \quad \mathbf{z} \in \Omega. \tag{2.5}$$

If $\delta < \sigma^2$, then for any $c \in \left(\frac{1}{\sigma}, \frac{\sigma}{\delta}\right)$, there exists $\varepsilon > 0$ such that for all $z_0 \in \Omega$ with $\|\mathbf{z}_0 - \tilde{\mathbf{z}}\|_2 < \varepsilon$, the sequence generated by the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \mathcal{J}(\mathbf{z}_k)^+ F(\mathbf{z}_k), \quad \text{where} \quad \mathcal{J}(\mathbf{z}_k)^+ = [\mathcal{J}(\mathbf{z}_k)^H \mathcal{J}(\mathbf{z}_k)]^{-1} \mathcal{J}(\mathbf{z}_k)^H$$

for $k = 0, 1, \ldots$, is well defined inside Ω, converges to $\tilde{\mathbf{z}}$, and satisfies

$$\left\| \mathbf{z}_{k+1} - \tilde{\mathbf{z}} \right\|_2 \leq \frac{c\delta}{\sigma} \left\| \mathbf{z}_k - \tilde{\mathbf{z}} \right\|_2 + \frac{c\alpha\gamma}{2\sigma} \left\| \mathbf{z}_k - \tilde{\mathbf{z}} \right\|_2^2, \tag{2.6}$$

where $\alpha > 0$ is the upper bound of $\|\mathcal{J}(\mathbf{z})\|_2$ on $\overline{\Omega}$, and $\gamma > 0$ is the Lipschitz constant of $\mathcal{J}(\mathbf{z})$ in Ω, namely, $\| \mathcal{J}(\mathbf{z}+\mathbf{h}) - \mathcal{J}(\mathbf{z}) \|_2 \leq \gamma \| \mathbf{h} \|$ for all $\mathbf{z}$, $\mathbf{z}+\mathbf{h} \in \Omega$.

3. Algorithm I: Root-finding with given multiplicities

In this section, we assume that the multiplicity structure of a given polynomial is known. We shall deal with the problem of determining this multiplicity structure in §4. A condition number will be introduced to measure the sensitivity of multiple roots. When the condition number is moderate, the multiple roots can be calculated accurately by our algorithm.

3.1. Remarks on previous work.

In Part II of [21], Kahan discussed the sensitivity of polynomial roots with enlightening insight, and pointed out that it may be a misconception to consider multiple roots to be infinitely ill-conditioned. Kahan's work on roots of a polynomial p in [21] can be briefly summarized as follows. First, Kahan describes the "pejorative manifold" of polynomials with multiple roots. Secondly, the differentiability is proved for an m-fold isolated root with respect to coefficients that are constrained to preserve the multiplicity m of that root. This

differentiability then naturally leads to the existence of a finite local condition number of an *isolated* m-fold root under the perturbation that is constrained to preserve the multiplicity m of that root. Kahan also proves the existence of a vanishing point for $p^{(m-1)}(x)$ in a region containing a cluster of m roots of p. Finally, a possible approach is proposed, based on the Lagrange multipliers, for finding the polynomial nearest to p while possessing an m-fold root.

Kahan's work [21] emphasizes theoretical analysis rather than computational methodology. The sensitivity analysis is rigorous while the description of the pejorative manifold is heuristic. The condition number defined in [21] exists in theory with unknown practical attainability. The implementability of the proposed Lagrange multiplier method in numerical computation is still unknown.

In this section, we shall attempt to formulate the pejorative manifold rigorously. More importantly, we emphasize the practical computation of multiple roots. Our main contribution in this section also includes the following. First, we convert the singular root-finding problem to a least squares problem and prove its regularity. Second, we establish the local convergence of the Gauss-Newton iteration for solving the least squares problem. Our third contribution is the formulation of a global structure-preserving condition number measuring the combined sensitivity of all roots that are constrained in a multiplicity structure instead of an isolated multiple root considered by Kahan. Not only is this condition number easily computable, but it also enables us to estimate the computing error quite accurately according to our error analysis and numerical experiments. Finally, we establish practical procedures that carry out the necessary computation on the pejorative manifold. By assembling these elements, we construct our Algorithm I.

3.2. **The pejorative manifold.**

A polynomial of degree n corresponds to a vector (or point) in $\mathbb{C}^n$

$$p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_n \quad \sim \quad \mathbf{a} = (a_1, \ldots, a_n)^\top \equiv \left(\frac{p_1}{p_0}, \ldots, \frac{p_n}{p_0}\right)^\top,$$

where "$\sim$" denotes this correspondence. For a partition of n, namely a fixed array of positive integers $\ell_1, \ldots, \ell_m$ with $\ell_1 + \cdots + \ell_m = n$, a polynomial p that has roots $z_1, \ldots, z_m$ with multiplicities $\ell_1, \ldots, \ell_m$, respectively, can be written as

$$\frac{1}{p_0} p(x) = \prod_{j=1}^{m} (x - z_j)^{\ell_j} \quad = \quad x^n + \sum_{j=1}^{n} g_j(z_1, \ldots, z_m)\, x^{n-j}, \tag{3.1}$$

where each g_j is a polynomial in $z_1, \ldots, z_m$. We have the correspondence

$$p \sim G_\ell(\mathbf{z}) \equiv \begin{pmatrix} g_1(z_1, \ldots, z_m) \\ \vdots \\ g_n(z_1, \ldots, z_m) \end{pmatrix} \in \mathbb{C}^n, \qquad \text{where} \quad \mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} \in \mathbb{C}^m. \tag{3.2}$$

We now define the pejorative manifold rigorously based on Kahan's heuristic description.

Definition 3.1. An ordered array of positive integers $\ell = [\ell_1, \ldots, \ell_m]$ is called a ***multiplicity structure*** of degree n if $\ell_1 + \cdots + \ell_m = n$. For any such given multiplicity structure ℓ, the collection of vectors $\Pi_\ell \equiv \left\{ G_\ell(\mathbf{z}) \;\middle|\; \mathbf{z} \in \mathbb{C}^m \right\} \subset \mathbb{C}^n$ is called the ***pejorative manifold*** of multiplicity structure ℓ, where

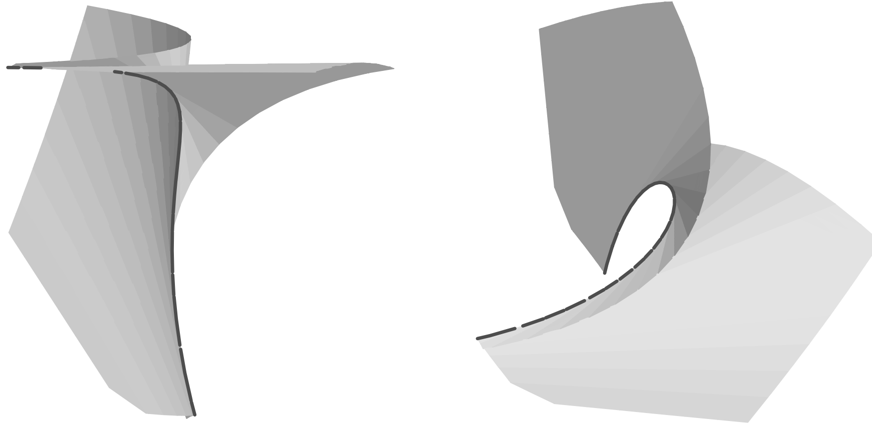

FIGURE 2. Pejorative manifolds of polynomials with degree 3 (viewed from two angles).

$G_\ell : \mathbb{C}^m \longrightarrow \mathbb{C}^n$ defined in (3.1)–(3.2) is called the ***coefficient operator*** associated with the multiplicity structure ℓ.

For example, we consider polynomials of degree 3. First, for multiplicity structure $\ell = [1, 2]$,

$$(x - z_1)(x - z_2)^2 = x^3 + (\boxed{-z_1 - 2z_2})\, x^2 + (\boxed{2z_1 z_2 + z_2^2})\, x + (\boxed{-z_1 z_2^2}).$$

A polynomial with one simple root z_1 and one double root z_2 corresponds to the vector

$$G_{[1,2]}(\mathbf{z}) \equiv \begin{pmatrix} \boxed{-z_1 - 2z_2} \\ \boxed{2z_1 z_2 + z_2^2} \\ \boxed{-z_1 z_2^2} \end{pmatrix} \in \mathbb{C}^3, \quad \text{with} \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \in \mathbb{C}^2. \tag{3.3}$$

The vectors $G_{[1,2]}(\mathbf{z})$ in (3.3) for all $\mathbf{z} \in \mathbb{C}^2$ form the pejorative manifold $\Pi_{[1,2]}$. Similarly,

$$\Pi_{[3]} = \left\{ (-3\, z,\, 3\, z^2,\, -z^3)^\top \;\middle|\; z \in \mathbb{C} \right\}$$

when $\ell = [3]$. $\Pi_{[3]}$ is a submanifold of $\Pi_{[1,2]}$ that contains all polynomials with a single triple root. Figure 2 shows the manifolds $\Pi_{[1,2]}$ (the wings) and $\Pi_{[3]}$ (the sharp edge) in $\mathbb{R}^3$.

As a special case, $\Pi_{[1,1,\ldots,1]} = \mathbb{C}^n$ is the vector space of all monic polynomials with degree n.

3.3. **Solving the nonsingular least squares problem.** Let $\ell = [\ell_1, \ldots, \ell_m]$ be a multiplicity structure of degree n, and let Π_ℓ be the corresponding pejorative manifold. If the polynomial $p \sim \mathbf{a} \in \Pi_\ell$, then there is a vector $\mathbf{z} \in \mathbb{C}^m$ such that $G_\ell(\mathbf{z}) = \mathbf{a}$. In general, the polynomial system

$$\left\{ \begin{array}{rcl} g_1(z_1, \ldots, z_m) & = & a_1 \\ g_2(z_1, \ldots, z_m) & = & a_2 \\ \vdots & \vdots & \vdots \\ g_n(z_1, \ldots, z_m) & = & a_n \end{array} \right. \qquad \text{or} \qquad G_\ell(\mathbf{z}) = \mathbf{a} \tag{3.4}$$

is overdetermined except for the plain structure $\ell = [1, 1, \ldots, 1]$. Let the weight matrix be $W = \mathrm{diag}(\omega_1, \ldots, \omega_n)$. and $\|\cdot\|_W$ denote the weighted 2-norm defined in (2.2). We seek a *weighted least squares solution* to (3.4) by solving the minimization problem

(3.5)

$$\min_{\mathbf{z}\in\mathbb{C}^m} \left\|G_\ell(\mathbf{z}) - \mathbf{a}\right\|_W^2 \equiv \min_{\mathbf{z}\in\mathbb{C}^m} \left\|W\Big(G_\ell(\mathbf{z}) - \mathbf{a}\Big)\right\|_2^2 \equiv \min_{\mathbf{z}\in\mathbb{C}^m} \left\{ \sum_{j=1}^n \omega_j^2 \Big|g_j(\mathbf{z}) - a_j\Big|^2 \right\}.$$

Two common types of weights can be used. To minimize the overall backward error of the roots, we set $W = \mathrm{diag}(1, 1, \ldots, 1)$. On the other hand, the weights

$$\omega_j = \min\left\{ 1,\ |a_j|^{-1} \right\}, \quad j = 1, \ldots, n, \tag{3.6}$$

lead to minimization of the relative backward error at every coefficient larger than one. All our numerical experiments for Algorithm I are conducted using the weights (3.6).

From Lemma 2.7, let $J(\mathbf{z})$ be the Jacobian of $G_\ell(\mathbf{z})$. In order to find a local minimum point of $F(\mathbf{z}) \equiv W\Big[G_\ell(\mathbf{z}) - \mathbf{a}\Big]$ with $\mathcal{J}(\mathbf{z}) = WJ(\mathbf{z})$, we look for $\tilde{\mathbf{z}} \in \mathbb{C}^m$ such that

$$\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = \Big[WJ(\tilde{\mathbf{z}})\Big]^H W\Big[G_\ell(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = J(\tilde{\mathbf{z}})^H\, W^2 \Big[G_\ell(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = 0. \tag{3.7}$$

Definition 3.2. Let $p \sim \mathbf{a}$ be a polynomial of degree n. For any given multiplicity structure ℓ of the same degree, the vector $\tilde{\mathbf{z}}$ satisfying (3.7) is called a ***pejorative root vector*** or simply a ***pejorative root*** of p corresponding to the multiplicity structure ℓ and weight W.

Our algorithms emanate from the following fundamental theorem by which one may convert the singular problem of computing multiple roots with standard methods to a regular problem by seeking the least squares solution of (3.4).

Theorem 3.3. *Let $G_\ell : \mathbb{C}^m \longrightarrow \mathbb{C}^n$ be the coefficient operator associated with a multiplicity structure $\ell = [\ell_1, \ldots, \ell_m]$. Then the Jacobian $J(\mathbf{z})$ of $G_\ell(\mathbf{z})$ is of full (column) rank if and only if the components of $\mathbf{z} = (z_1, \ldots, z_m)^\top$ are distinct.*

Proof. Let $z_1, \ldots, z_m$ be distinct. To prove $J(\mathbf{z})$ is of full (column) rank, or the columns of $J(\mathbf{z})$ are linearly independent, write the jth column of $J(\mathbf{z})$ as $J_j = \left(\frac{\partial g_1(\mathbf{z})}{\partial z_j}, \ldots, \frac{\partial g_n(\mathbf{z})}{\partial z_j}\right)^\top$. For $j = 1, \ldots, m$, let $q_j(x)$, a polynomial in x, be defined as follows,

$$\begin{aligned} q_j(x) &= \left(\frac{\partial g_1(\mathbf{z})}{\partial z_j}\right) x^{n-1} + \cdots + \left(\frac{\partial g_{n-1}(\mathbf{z})}{\partial z_j}\right) x + \left(\frac{\partial g_n(\mathbf{z})}{\partial z_j}\right) \\ &= \frac{\partial}{\partial z_j}\Big[x^n + g_1(\mathbf{z})x^{n-1} + \cdots + g_n(\mathbf{z})\Big] \\ &= \frac{\partial}{\partial z_j}\Big[(x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}\Big] \\ &= -\ell_j\,(x - z_j)^{\ell_j - 1}\left[\prod_{k\neq j}(x - z_k)^{\ell_k}\right]. \end{aligned} \tag{3.8}$$

If $c_1 J_1 + \cdots + c_m J_m = 0$ for constants $c_1, \ldots, c_m$, then

$$\begin{aligned} q(x) &\equiv c_1\, q_1(x) + \cdots + c_m\, q_m(x) = -\sum_{j=1}^{m} \left\{ c_j \ell_j\, (x - z_j)^{\ell_j - 1} \left[\prod_{k \neq j} (x - z_k)^{\ell_k} \right] \right\} \\ &= -\left[\prod_{\sigma=1}^{m} (x - z_\sigma)^{\ell_\sigma - 1} \right] \sum_{j=1}^{m} \left[c_j \ell_j \prod_{k \neq j} (x - z_k) \right] \end{aligned}$$

is a zero polynomial, yielding $r(x) = \sum_{j=1}^{m} c_j \ell_j \left[\prod_{k \neq j} (x - z_k) \right] \equiv 0$. Therefore, for $l = 1, \ldots, m$, $r(z_l) = c_l \left[\ell_l \prod_{k \neq l} (z_l - z_k) \right] = 0$ implies $c_l = 0$ since ℓ_l's are positive and z_k's are distinct. Therefore, J_j's are linearly independent.

On the other hand, suppose $z_1, \ldots, z_m$ are not distinct, say, for instance, $z_1 = z_2$. Then the first two columns of $J(\mathbf{z})$ are coefficients of polynomials $h_1(x)$ and $h_2(x)$ defined as

$$-\ell_1\, (x - z_1)^{\ell_1 - 1} (x - z_2)^{\ell_2} \prod_{k=3}^{m} (x - z_k)^{\ell_k} \quad \text{and} \quad -\ell_2\, (x - z_1)^{\ell_1} (x - z_2)^{\ell_2 - 1} \prod_{k=3}^{m} (x - z_k)^{\ell_k},$$

respectively. Since $z_1 = z_2$, these two polynomials differ by constant multiples ℓ_1 and ℓ_2. Therefore $J(\mathbf{z})$ is (column) rank deficient. □

With the system (3.4) being nonsingular from Theorem 3.3, the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left[J(\mathbf{z}_k)_W^{+} \right] [G_\ell(\mathbf{z}_k) - \mathbf{a}], \quad k = 0, 1, \ldots, \tag{3.9}$$

on Π_ℓ is well defined. Moreover, we have the convergence theorem based on Lemma 2.8.

Theorem 3.4. *Let $\tilde{\mathbf{z}} = (\tilde{z}_1, \ldots, \tilde{z}_m)^\top \in \mathbb{C}^m$ be a pejorative root of $p \sim \mathbf{a}$ associated with multiplicity structure ℓ and weight W. Assume $\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_m$ are distinct. Then there are $\varepsilon, \epsilon > 0$ such that, if $\|\, \mathbf{a} - G_\ell(\tilde{\mathbf{z}})\,\|_W < \varepsilon$ and $\|\, \mathbf{z}_0 - \tilde{\mathbf{z}}\,\|_2 < \epsilon$, iteration* (3.9) *is well defined and converges to the pejorative root $\tilde{\mathbf{z}}$ with at least a linear rate. If we have $\mathbf{a} = G_\ell(\tilde{\mathbf{z}})$ in addition, then the convergence is quadratic.*

Proof. Let $F(\mathbf{z}) = W\left[G_\ell(\mathbf{z}) - \mathbf{a}\right]$ and $\mathcal{J}(\mathbf{z})$ be its Jacobian. $F(\mathbf{z})$ is obviously analytic. From Theorem 3.3, the smallest singular value σ of $\mathcal{J}(\tilde{\mathbf{z}})$ is strictly positive. If $\mathbf{a}$ is sufficiently close to $G_\ell(\tilde{\mathbf{z}})$, then

$$\left\| \, F(\tilde{\mathbf{z}}) \, \right\|_2 = \left\| \, G_\ell(\tilde{\mathbf{z}}) - \mathbf{a} \, \right\|_W$$

will be small enough, making (2.5) hold with $\delta < \sigma^2$. Therefore all conditions of Lemma 2.8 are satisfied and there is a neighborhood Ω of $\tilde{\mathbf{z}}$ such that if $\mathbf{z}_0 \in \Omega$, the iteration (3.9) converges and satisfies (2.6). If in addition $\mathbf{a} = G_\ell(\tilde{\mathbf{z}})$, then $F(\tilde{\mathbf{z}}) = 0$ and, therefore, $\delta = 0$ in (2.5) and (2.6). The convergence becomes quadratic. □

As a special case for the structure $\ell = [1, 1, \ldots, 1]$, equations in (3.4) form Viéte's system of n-variate polynomial system. Solving this system via Newton's iteration is equivalent to the Weierstrass (Durand-Kerner) algorithm [27]. When a polynomial has multiple roots, Viéte's system becomes singular at the nondistinct root vector. This singularity appears to be the very reason that causes the ill-condition of conventional root-finders: a wrong pejorative manifold is used.

3.4. **The structure-preserving condition number.** There are many insightful discussions on the numerical condition of polynomial roots in the literature, such as [8, 17, 21, 33, 29, 34]. In general, a condition number can be characterized as the smallest number satisfying

$$\text{(3.10)} \qquad \Big[\text{forward_error}\Big] \le \Big[\text{condition_number}\Big] \times \Big[\text{backward_error}\Big] + \text{h.o.t.},$$

where *h.o.t* represents higher order terms of the backward error. For a polynomial with multiple roots, under *unrestricted* perturbation, the only condition number satisfying (3.10) is infinity. For a simple example, let polynomial $p(x) = x^2$. A backward error ε makes the perturbed polynomial $\tilde{p}(x) = x^2 + \varepsilon$, which has roots $\pm\sqrt{\varepsilon} i$ with forward error $\sqrt{\varepsilon}$ in magnitude. The only "constant" c which accounts for $\sqrt{\varepsilon} \le c\,\varepsilon$ for *all* $\varepsilon > 0$ must be infinity.

By changing the computational objective from solving a polynomial equation $p(x) = 0$ to the nonlinear least squares problem in the form of (3.5), the structure-altering noise is filtered out, and the multiplicity structure is preserved. With this shift in computing strategy, the sensitivity of the roots can be analyzed differently.

Let us consider the root vector $\mathbf{z}$ of $p \sim \mathbf{a} = G_\ell(\mathbf{z})$. The polynomial p is perturbed, with multiplicity structure ℓ being preserved, to be $\hat{p} \sim \hat{\mathbf{a}} = G_\ell(\hat{\mathbf{z}})$. In other words, both p and $\hat{p}$ are on the same pejorative manifold Π_ℓ. Then

$$\hat{\mathbf{a}} - \mathbf{a} = G_\ell(\hat{\mathbf{z}}) - G_\ell(\mathbf{z}) = J(\mathbf{z})(\hat{\mathbf{z}} - \mathbf{z}) + O\left(\|\hat{\mathbf{z}} - \mathbf{z}\|^2\right),$$

where $J(\mathbf{z})$ is the Jacobian of $G_\ell(\mathbf{z})$. Assuming the entries of $\mathbf{z}$ are distinct, by Theorem 3.3, $J(\mathbf{z})$ is of full rank. Consequently,

$$\begin{aligned} \left\| W(\hat{\mathbf{a}} - \mathbf{a}) \right\|_2 &= \left\| [W J(\mathbf{z})](\hat{\mathbf{z}} - \mathbf{z}) \right\|_2 + h.o.t., \\ \text{namely,} \quad \left\| \hat{\mathbf{a}} - \mathbf{a} \right\|_W &\ge \sigma_{\min} \left\| \hat{\mathbf{z}} - \mathbf{z} \right\|_2 + h.o.t., \\ \text{(3.11)} \qquad \text{or} \quad \left\| \hat{\mathbf{z}} - \mathbf{z} \right\|_2 &\le \left(\frac{1}{\sigma_{\min}} \right) \left\| \hat{\mathbf{a}} - \mathbf{a} \right\|_W + h.o.t. \end{aligned}$$

where $\sigma_{\min}$, the smallest singular value of $WJ(\mathbf{z})$, is strictly positive since W and $J(\mathbf{z})$ are of full rank. The distance $\|\hat{\mathbf{z}} - \mathbf{z}\|_2$ is the forward error and the weighted distance $\|\hat{\mathbf{a}} - \mathbf{a}\|_W$ measures the backward error. Therefore, the sensitivity of the root vector is asymptotically bounded by $\frac{1}{\sigma_{\min}}$ times the size of the multiplicity-preserving perturbation. In this sense, the multiple roots are not infinitely sensitive.

Definition 3.5. Let p be a polynomial and $\mathbf{z}$ be its pejorative root corresponding to a given multiplicity structure ℓ and weight W. Let G_ℓ be the coefficient operator associated with ℓ, J be its Jacobian, and $\sigma_{\min}$ be the smallest singular value of $WJ(\mathbf{z})$. Then the ***condition number*** **of z with respect to the multiplicity structure** ℓ **and weight** W is defined as

$$\kappa_{\ell,w}(\mathbf{z}) = \frac{1}{\sigma_{\min}}.$$

Remark. The condition number $\kappa_{\ell,w}(\mathbf{z})$ is structure dependent. The array $\ell = [\ell_1, \ldots, \ell_m]$ may or may not be the *actual* multiplicity structure. A polynomial has different condition numbers corresponding to different pejorative roots on various pejorative manifolds. For example, see Table 4 in §3.6.2.

We now estimate the error on pejorative roots of polynomials with inexact coefficients. In this case, the given polynomial $\hat{p}$ is assumed to be arbitrarily perturbed

from p with both polynomials near a pejorative manifold Π_ℓ. In the exact sense, neither polynomial possesses the structure ℓ. The nearby pejorative manifold causes both polynomials to be ill-conditioned in the conventional sense. Consequently, the exact roots of $\hat{p}$ can be far from those of p even if two polynomials are close to each other. However, the following theorem ensures that their pejorative roots, not exact roots, may still be insensitive to perturbation.

Theorem 3.6. *For a fixed $\ell = [\ell_1, \ldots, \ell_m]$, let the polynomial $\hat{p} \sim \hat{\mathbf{b}}$ be an approximation to $p \sim \mathbf{b}$ with pejorative roots $\hat{\mathbf{z}}$ and $\mathbf{z}$, respectively, that are corresponding to the multiplicity structure ℓ and a weight W. Assume the components of $\mathbf{z}$ are distinct while $\| G_\ell(\hat{\mathbf{z}}) - \hat{\mathbf{b}} \|_W$ reaches a local minimum at $\hat{\mathbf{z}}$. If $\| \mathbf{b} - \hat{\mathbf{b}} \|_W$ and $\| G_\ell(\mathbf{z}) - \mathbf{b} \|_W$ are sufficiently small, then*

$$\left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 \leq 2 \cdot \kappa_{\ell,w}(\mathbf{z}) \cdot \left(\left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W + \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W \right) + h.o.t. \tag{3.12}$$

Proof. From (3.11),

$$\begin{aligned} \left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 &\leq \kappa_{\ell,w}(\mathbf{z}) \left\| G_\ell(\mathbf{z}) - G_\ell(\hat{\mathbf{z}}) \right\|_W + h.o.t. \\ &\leq \kappa_{\ell,w}(\mathbf{z}) \left(\left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W + \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W + \left\| G_\ell(\hat{\mathbf{z}}) - \hat{\mathbf{b}} \right\|_W \right) + h.o.t. \end{aligned}$$

Since $\| G_\ell(\hat{\mathbf{z}}) - \hat{\mathbf{b}} \|_W$ is a local minimum, we have

$$\left\| G_\ell(\hat{\mathbf{z}}) - \hat{\mathbf{b}} \right\|_W \leq \left\| G_\ell(\mathbf{z}) - \hat{\mathbf{b}} \right\|_W \leq \left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W + \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W,$$

and the assertion of the theorem follows. □

By the above theorem, when a polynomial is perturbed, the error on the pejorative roots depends on the magnitude of the perturbation (i.e., $\| \mathbf{b} - \hat{\mathbf{b}} \|_W$), the distance to the pejorative manifold (namely $\| G_\ell(\mathbf{z}) - \mathbf{b} \|_W$), as well as the condition number $\kappa_{\ell,w}(\mathbf{z})$. Although the (exact) roots may be hypersensitive, their pejorative roots are stable if $\kappa_{\ell,w}(\mathbf{z})$ is moderate.

For a polynomial p having multiplicity structure ℓ, we can now estimate the error of its multiple roots computed from its (inexact) approximation $\hat{p}$. The perturbation from p to $\hat{p}$ can be arbitrary, such as rounding up digits in coefficients. The (exact) roots of $\hat{p}$ are all simple in general and far from the multiple roots of p. The following corollary ensures that the *pejorative* root $\hat{\mathbf{z}}$ of $\hat{p}$ with respect to the multiplicity structure ℓ can be an accurate approximation to the multiple roots $\mathbf{z}$ of p.

Corollary 3.7. *Under the condition of Theorem 3.6, if $\mathbf{z}$ is the exact root vector of p with multiplicity structure ℓ, then*

$$\left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 \leq 2 \cdot \kappa_{\ell,w}(\mathbf{z}) \cdot \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W + h.o.t. \tag{3.13}$$

Proof. Since $\mathbf{z}$ is exact, $\| G_\ell(\mathbf{z}) - \mathbf{b} \|_W = 0$ in (3.12). □

The "attainable accuracy" barrier suggests that when multiplicity increases, the root sensitivity intensifies. However, there is no apparent correlation between the magnitude of the multiplicities and the structure-constraint sensitivity. For example, consider the polynomials

$$p_\ell(x) = (x+1)^{\ell_1}(x-1)^{\ell_2}(x-2)^{\ell_3}$$

multiplicities			condition
ℓ_1	ℓ_2	ℓ_3	number
1	1	1	3.1499
1	2	3	2.0323
10	20	30	0.0733
100	200	300	0.0146

FIGURE 3. The sensitivity and multiplicities.

with different multiplicities $\ell = [\ell_1, \ell_2, \ell_3]$. For the weight W defined in (3.6), Figure 3 lists the condition numbers for different multiplicities. As seen in this example, the magnitude of root error can actually be *less* than that of the data error when the condition number is less than one. The condition theory described above indicates that multiprecision arithmetic may *not* be a necessity, and the "attainable accuracy" barrier appears to be highly questionable.

In §3.6 and §5, more examples will show that our iterative algorithm indeed reaches the accuracy permissible by the condition number $\kappa_{\ell,w}(\mathbf{z})$, which can be calculated with negligible cost. The QR decomposition of the Jacobian $J(\mathbf{z})$ is required by the iteration (3.9), and can be recycled to calculate $\kappa_{\ell,w}(\mathbf{z})$. The inverse iteration in Lemma 2.6 is suitable for finding the smallest singular value.

3.5. **The numerical procedures.** Iteration (3.9) requires calculation of the vector value of $G_\ell(\mathbf{z}_k)$ and matrix value of $J(\mathbf{z}_k)$, where the components of $G_\ell(\mathbf{z})$ are defined in (3.1) and (3.2) as coefficients of the polynomial $p(x) = (x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}$. While the explicit formulas for each $g_j(z_1, \ldots, z_m)$ and $\frac{\partial g_j}{\partial z_i}$ can be symbolically (inefficiently in general) computed using software like Maple, we propose more efficient numerical procedures for computing $G_\ell(\mathbf{z})$ and $J(\mathbf{z})$ in Figure 4.

Polynomial multiplication is equivalent to vector convolution (Lemma 2.2). The polynomial $p(x) = (x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}$ can thereby be constructed from recursive convolution with vectors $(1, -z_j)^\top$, $j = 1, 2, \ldots, m$. As a result, $G_\ell(\mathbf{z})$ is

```
Algorithm EvalG:
input:  m, n, z = (z_1,...,z_m)^T,
             ℓ = [ℓ_1,...,ℓ_m]
output:  vector G_ℓ(z) ∈ C^n

    s = (1)
    for i = 1,2,...m do
        for l = 1,2,...,ℓ_i do
            s = conv(s, (1,-z_i)^T)
        end do
    end do
    g_j(z) = (j+1)th component
        of s for j = 1,...,n
```

```
Algorithm EvalJ:
input:  m, n, z = (z_1,...,z_m)^T,
             ℓ = [ℓ_1,...,ℓ_m]
output:  Jacobian J(z) ∈ ℂ^{n×m}

    u ~ ∏(x - z_j)^{ℓ_j - 1} by EvalG
    for j = 1,2,...,m do
        s = -ℓ_j u
        for l = 1,...,m, l ≠ j do
            s = conv(s, (1,-z_l)^T)
        end do
        jth column of J(z) = s
    end do
```

FIGURE 4. Pseudo-codes for evaluating $G_\ell(\mathbf{z})$ and $J(\mathbf{z})$.

```
Pseudo-code PEJROOT (Algorithm I):
 input:  m, n, a ∈ ℂ^n, weight matrix W, initial iterate z_0,
             multiplicity structure ℓ, error tolerance τ
 output:  Roots z = (z_1, ..., z_m), or message of failure

    for k = 0,1,... do
        Calculate G_ℓ(z_k) and J(z_k) with EVALG and EVALJ
        Compute the least squares solution Δz_k to the linear
                system [W J(z_k)](Δz_k) = W[G_ℓ(z_k) − a]
        Set z_{k+1} = z_k − Δz_k and δ_k = ||Δz_k||_2
        if k ≥ 1 then
            if δ_k ≥ δ_{k−1} then, stop, output failure message
            else if δ_k^2/(δ_{k−1} − δ_k) < τ then,  stop, output z = z_{k+1}
            end if
        end if
    end do
```

FIGURE 5. Pseudo-code of Algorithm I.

computed through the nested loops shown in Figure 4 as Algorithm EVALG. It takes $n^2 + O(n)$ flops (additions and multiplications) to calculate $G_\ell(\mathbf{z})$.

The jth column of the Jacobian $J(\mathbf{z})$, as shown in the proof of Theorem 3.3, can be considered to be the coefficients of the polynomial $q_j(x)$ defined in (3.8). The cost of computing $J(\mathbf{z})$ is no more than $mn^2 + O(n)$ flops. Each step of the Gauss-Newton iteration takes $O(nm^2)$ flops. Therefore, for a polynomial of degree n with m distinct roots, the complexity of Algorithm I is $O(m^2 n + mn^2)$. The worst case occurs when $m = n$ and the complexity becomes $O(n^3)$. The complete pseudo-code of the Algorithm I is shown in Figure 5.

3.6. **Numerical results for Algorithm I.** Algorithm I is implemented as a Matlab code PEJROOT. All the tests of PEJROOT are conducted with IEEE double precision (16 decimal digits) without extension. In comparison, other algorithms and software may use unlimited machine precision in some cases.

3.6.1. *The effect of "attainable accuracy".* Conventional methods, such as Farmer-Loizou methods [14], are subject to the "attainable accuracy" barrier. We made a straightforward implementation of the Farmer-Loizou third-order iteration suggested in [14] and applied it to the same example they used,

$$p_1(x) = (x-1)^4(x-2)^3(x-3)^2(x-4).$$

Both iterations start from $\mathbf{z}_0 = (1.1, 1.9, 3.1, 3.9)$ using the standard IEEE double precision. The "attainable accuracy" of the roots are 4, 5, 8, 16 digits, respectively. For 100 iteration steps, the Farmer-Loizou method produces iterates that bounce around the roots. In contrast, our iteration smoothly converges to the roots and reaches an accuracy of 14 digits. The "attainable accuracy" barrier has no effect on our algorithm. The iterations are shown in Table 1 for three roots $x = 1, 2, 3$ with highest multiplicities.

In the same problem, we increase the multiplicities 10 times as large, generating

$$p_2(x) = (x-1)^{40}(x-2)^{30}(x-3)^{20}(x-4)^{10}$$

Table 1. Comparison with the Farmer-Loizou third-order iteration in the low multiplicity case. Three roots are shown with unimportant digits truncated.

Farmer-Loizou third order iteration				PejRoot result			
step	iterateas			step	iterates		
1	1.0009	1.998	3.001	1	1.03	1.8	3.4
2	0.99997	1.9999992	3.000000008	2	0.997	1.98	2.6
3	0.01	3.4	2.9988	3	1.00009	2.05	2.8
4	0.8	2.3	3.000007	4	0.99994	1.994	2.98
5	0.998	2.007	3.0000001	5	1.000003	2.0001	2.9990
6	1.0000007	2.00000007	2.99996	6	0.999999997	2.000000005	2.9999990
...	...			7	1.00000000000000	2.0000000000002	2.999999999998
100	1.00000008	3.3	2.99999997	8	1.00000000000000	2.00000000000000	2.99999999999999

Table 2. Comparison with the Farmer-Loizou third-order iteration in the high multiplicity case. Three roots are shown with unimportant digits truncated.

Farmer-Loizou third order iteration				PejRoot result			
step	iterateas			steps	iterates		
1	0.47	-.33	3.02	1	1.004	1.98	3.05
2	32.92	-4.65	2.69	2	1.0001	1.998	3.003
3	4.75	-1.80	1.75	3	0.9999998	2.000006	2.99997
4	205.96	.40	1.54	4	0.999999999994	2.00000000001	2.9999999990
...	...			5	1.00000000000000	2.00000000000001	2.99999999999997
100	5.99	1.10	0.30	6.	1.00000000000000	2.00000000000001	2.99999999999998

with 16-digit accuracy in coefficients. In this test, our method still uses the standard 16-digit arithmetic and attains 14 correct digits on the roots, while the Farmer-Loizou method uses 1000-digit operations in Maple and fails (three root iterations are shown in Table 2).

The true accuracy barrier of Algorithm I is the condition number $\kappa_{\ell,w}(\mathbf{z})$. Matlab constructed the test polynomial with a relative coefficient error of 4.56×10^{-16}, the condition number is 29.3. The root error is approximately 1×10^{-14}, which is within the error bound $2 \times (29.3) \times (4.56 \times 10^{-16}) = 2.67 \times 10^{-14}$ established in Corollary 3.7.

There are state-of-art root-finding packages available using multiprecision arithmetic, such as MPSolve implemented by Bini et al. [2] and Eigensolve by Fortune [16]. If the given polynomial is exact (e.g., a polynomial with rational coefficients), those packages in general are capable of calculating all roots to the desired accuracy via extending the machine precision according to "attainable accuracy". For inexact polynomials, the accuracy of those packages on multiple roots is limited no matter how many digits the machine precision is extended to. For example, consider the polynomial

$$p(x) = \left(x - \sqrt{2}\right)^{20} \left(x - \sqrt{3}\right)^{10}.$$

The coefficients are calculated to 100-digit accuracy. The "attainable accuracy" for the roots $\sqrt{2}$ and $\sqrt{3}$ are 5 and 10 digits, respectively. MPSolve and Eigensolve output nearly identical results in accordance with this "attainable accuracy". In contrast, our software using only 16-digit precision in coefficients without extending the machine precision, still outputs roots of 15-digit accuracy along with accurate multiplicities (see Table 3).

Table 3. Comparison with multiprecision root-finders MPSolve and Eigensolve.

```
MPSolve and Eigensolve results         | MultRoot results with 16-digit input/machine precision
  with 100-digit input accuracy        |
(unimportant digits are truncated)     | THE CONDITION NUMBER:                       0.90775
  1.41412    - 0.000013i               | THE BACKWARD ERROR:                    6.66E-016
  1.41412    + 0.000013i               | THE ESTIMATED FORWARD ROOT ERROR:   1.21e-15
   ... ...                             |
  1.73205077 - 0.0000000094i           | computed roots              multiplicities
  1.73205077 + 0.0000000094i           |      1.732050807568876                  10
   ... ...                             |      1.414213562373096                  20
```

3.6.2. *Clustered multiple roots.* Let

$$f(x) = (x - 0.9)^{18}(x - 1)^{10}(x - 1.1)^{16}.$$

The roots are highly multiple and clustered. The Matlab function ROOTS produces 44 ill-conditioned roots scattered in a box of 2.0×2.0 (see Figure 6). In contrast, the Algorithm I code PejRoot obtains all three multiple roots for at least 14 digits in accuracy by taking two additional iteration steps on the information of multiplicity structure and the initial iterate provided by Algorithm II in §4.

step	z_1	z_2	z_3
0	0.89999999993	0.9999999993	1.0999999998
1	0.999999999999991	1.00000000000001	1.10000000000001
2	0.999999999999991	1.000000000000001	1.10000000000001

The backward accuracy can easily be verified to be less than 1.36×10^{-15}. The condition number is 60.4. Therefore, with perturbation at the 16th digit of the coefficients, 14 correct digits constitute the best possible accuracy that can be expected from any method.

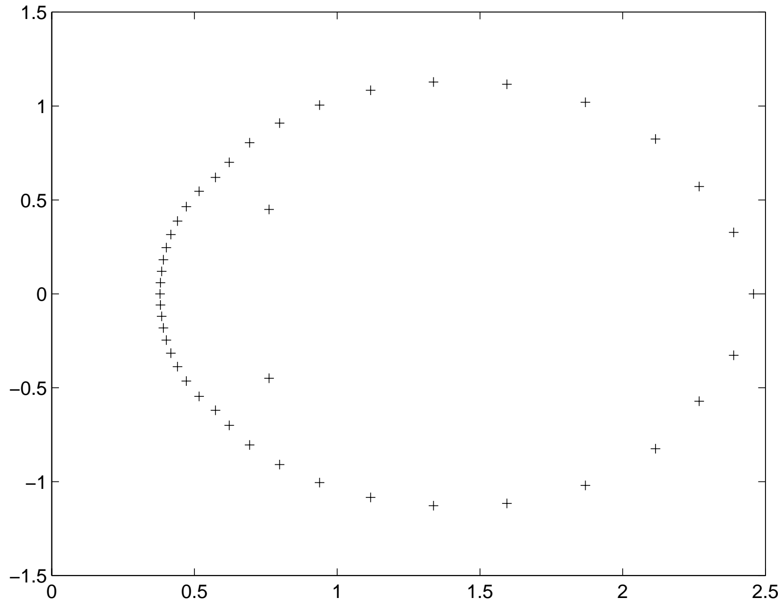

Figure 6. The root cluster from three multiple roots calculated by Matlab roots.

TABLE 4. Partial list of multiple roots on different pejorative manifolds.

multiplicity structure	pejorative roots	backward error (relative)	condition number
[1,1,...,1]	(see Figure 6)	.0000000000000006	1390704851032436
[18,10,16]	(.9000, 1.0000, 1.1000)	.000000000000002	60.4
[17,11,16]	(.8980, .9934, 1.1006)	.0000004	53.8
[14,16,14]	(.8890, .9892, 1.1090)	.000003	29.0
[10,24,10]	(.8711, .9906, 1.1315)	.000008	26.7
[2, 40, 2]	(.7390, .9917, 1.3277)	.00009	23.6
[1, 43]	(.5447, 1.0054)	.004	1.3
[44]	(.9925)	.04	.0058

An important feature of Algorithm I is that it does *not* require the correct multiplicity structure. Computation with different structures is permissible with Algorithm I and is often needed when the structure is unclear. If the computation is on a "wrong" pejorative manifold, then either the condition number or the backward error becomes large. Table 4 is a partial list of pejorative roots under different multiplicity structures. The apparent deviations on the pejorative roots in comparison with $(0.9, 1.0, 1.1)$ are the effect of the difference in structure, not the failure of the the error bounds in §3.4 where structure preservation is assumed.

For any polynomial of degree n, the nearest pejorative manifold(s) *always* include $\Pi_\ell \equiv \mathbb{C}^n$ with structure $\ell = [1, 1, \ldots, 1]$ because every other manifold is its subset. For that reason, an *unconstrained minimization* of the backward error (i.e., the distance to a pejorative manifold) will naturally lead to the simple, clustered, and incorrect roots as shown in Figure 6. Notice that pejorative roots with different multiplicity structures correspond to different backward errors and condition numbers $\kappa_{\ell,w}$. Generally, reduction in sensitivity may exclude root sets of higher backward accuracy. In this example, minimizing the backward error among all pejorative roots with a sensitivity constraint, say $\kappa_{\ell,w}(\mathbf{z}) < 100$, leads to the accurate roots with correct multiplicity structure. In short, conventional methods seek *unconstrained minimization* of the backward error among all pejorative roots, while computing multiple roots of inexact polynomials may be accomplished as a *constrained optimization* problem that minimizes the distance to a manifold subject to the condition that the roots are insensitive to perturbation with respect to the structure.

3.6.3. *Roots with huge multiplicities.* The accuracy as well as stability of Algorithm I seems independent of the multiplicities of the roots. For instance, let us consider the polynomial of degree 1000

$$g(x) = [x-(0.3+0.6i)]^{100}\,[x-(0.1+0.7i)]^{200}\,[x-(0.7+0.5i)]^{300}\,[x-(0.3+0.4i)]^{400}.$$

The multiplicities of the roots are 100, 200, 300 and 400. These multiplicities are "huge" compared to other numerical examples, usually with multiplicities less than

TABLE 5. Iterates on the degree 1000 polynomial.

.289	+.601i	.100	+.702i	.702	+.498i	.301	+.399i
.309	+.602i	.097	+.698i	.698	+.499i	.299	+.400i
.293	+.596i	.101	+.7003i	.7002	+.5005i	.3007	+.4003i
.3003	+.5994i	.09994	+.70008 i	.69996	+.50003i	.29996	+.40007i
.300005	+.600006	.099998	+.6999992i	.69999992	+.4999993i	.2999992	+.3999992i
.3000002	+.60000005i	.09999995	+.69999998i	.69999997	+.49999998i	.29999997	+.400000002i

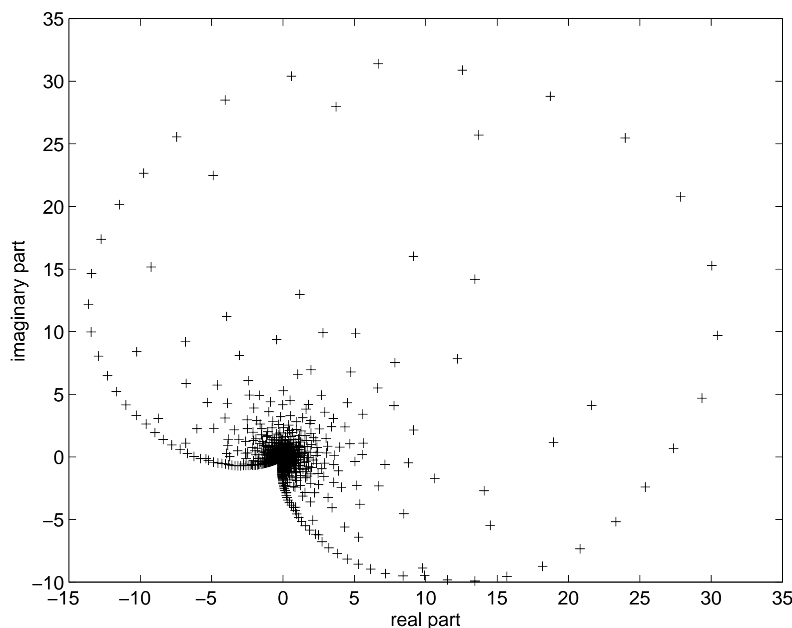

FIGURE 7. Result for the degree 1000 polynomial by Matlab function `roots`.

ten, used in the root-finding literature. In addition to such high multiplicities, we perturb the sixth digits of all coefficients of g by multiplying (1 ± 10^{-6}) on each one of them. Using any conventional approach, this perturbation will result in a total loss of forward accuracy, even if multiprecision arithmetic is used. The code PEJROOT of Algorithm I takes a few seconds under Matlab to calculate all roots up to seven digits accuracy as iterates, shown in Table 5. Taking the condition number 0.58 into account, this accuracy is optimal. On the same machine, Matlab function `roots` takes about 15 minutes to produce 1000 incorrect roots, as shown in Figure 7.

4. ALGORITHM II: THE MULTIPLICITY STRUCTURE AND INITIAL ROOT ESTIMATES

While Algorithm I can be used on any particular pejorative manifold, of course, the "correct" multiplicity structure is preferred if it is attainable. We present Algorithm II which calculates the multiplicity structure of a given polynomial as well as the initial root approximation for Algorithm I.

4.1. **Remarks on the univariate GCD computation.** For a given polynomial p with $u = \text{GCD}\,(p, p')$, Lagrange pointed out in 1769 that $v = p/u$ has the same distinct roots as p, and all roots of v are simple. If v is obtainable, its simple roots can be calculated using standard root-finders. Based on this observation, the following process, described by Gauss in 1863, is a natural approach to attain total

factorization of the polynomial p.

$$(4.1)\qquad \begin{cases} u_0 = p \\ \text{for } j = 1, 2, \ldots, \;\; \text{while } \deg(u_{j-1}) > 0 \text{ do} \\ \qquad \text{calculate } u_j = \text{GCD}\left(u_{j-1}, u'_{j-1}\right), \qquad v_j = \dfrac{u_{j-1}}{u_j} \\ \qquad \text{calculate the (simple) roots of } v_j(x) \\ \text{end do} \end{cases}$$

In this process, a k-fold root of p will be calculated k times as simple roots of v_j's. Other square-free factorization processes, such as Yun's algorithm [36], have also been proposed in the context of computer algebra.

The difficulty in carrying out the process (4.1) is the GCD computation. The classical Euclidean GCD algorithm requires recursive polynomial division which may not be numerically stable (see §4.2.3). Therefore, implementations of (4.1) based on the Euclidean GCD-finder [3, 31] may fail to reach desirable reliability or accuracy (see the numerical comparison in §4.6).

Numerical GCD computation has been studied extensively [5, 6, 13, 18, 22, 26, 28]. However, a reliable blackbox-type software is still not available. In [6], Corless, Gianni, Trager, and Watt proposed a novel approach using the singular-value decomposition in finding the degree of the GCD, and suggested the possibility of solving a GCD system similar to (4.3) below as a least squares problem, along with several other possibilities, including using the Euclidean algorithm.

There are several unresolved issues in the approach of Corless et al., especially in the stage of calculating the GCD after determining its degree. Among the possible avenues suggested, they seem to prefer using iterative methods to solve the least squares problem similar to (4.3) below. However, their least squares system is underdetermined by one equation. Moreover, with no clearly decided initial iterate being given, one can only leave this crucial ingredient to guessing or some sort of expensive global search [5]. From [6] and its follow-up work, such as [5, 22], it is also not clear which standard optimization algorithm should be selected. We shall demonstrate that the Gauss-Newton iteration, absent from the above works, is apparently the simplest, most efficient, and most suitable method for solving the GCD system (4.3), and it is at least locally convergent.

The key to carrying out the procedure (4.1) is the capability of factoring an arbitrary polynomial f and its derivative f' with a GCD triplet (u, v, w):

$$(4.2)\qquad \begin{cases} u(x)\,v(x) &= f(x), \\ u(x)\,w(x) &= f'(x), \end{cases} \qquad u \text{ is monic, } v \text{ and } w \text{ are coprime.}$$

In light of the Corless-Gianni-Trager-Watt approach, which calculates all singular values of a single Sylvester matrix $S_{n-1}(f)$, we employ a successive updating process that calculates only the smallest singular values of the Sylvester matrices $S_j(f)$, $j = 1, 2, \ldots$, and stop at the first rank-deficient matrix $S_k(f)$. With this $S_k(f)$, not only the degrees of the GCD triplet u, v, w are available, we also obtain coefficients of v and w automatically from the resulting right singular vector. In combination with a least squares division in §4.2.3 of the unstable long division, we can generate an approximation to the GCD triplet, and obtain an initial iterate that is not clearly indicated in the approach of Corless et al. Consequently, a blackbox-type software computing $\text{GCD}(f, f')$ is developed for the process (4.1).

The discussion on GCD computation is limited to $\mathrm{GCD}(f, f')$ because our objective is mainly root-finding in this paper. With minor modifications, our GCD-finder can easily be adapted to the general GCD problem of arbitrary polynomial pairs.

4.2. **Calculating the greatest common divisor.** Algorithm II is based on the following GCD-finder for an arbitrary polynomial f:

STEP 1. Find the degree m of $\mathrm{GCD}(f, f')$.
STEP 2. Set up the system (4.2) in accordance with the degree m.
STEP 3. Find an initial approximation to u, v and w for the GCD system (4.2).
STEP 4. Use the Gauss-Newton iteration to refine the GCD triplet (u, v, w).

We shall describe each step in detail.

4.2.1. *Finding the degrees of the GCD triplet.* Let f be a polynomial of degree n. By Lemma 2.4, the degree of $u = \mathrm{GCD}(f, f')$ is $m = n - k$ if and only if the kth Sylvester discriminant matrix is the first one being rank deficient. Therefore, $m = \deg(u)$ can be identified by calculating the sequence of the smallest singular values ς_j of $S_j(f)$, $j = 1, 2, \ldots$, until reaching ς_k that is *approximately* zero. Since only one singular pair (i.e., the singular value and the right singular vector) is needed, the inverse iteration described in Lemma 2.6 is suitable for this purpose. Moreover, we can reduce the computing cost even further by recycling and updating the QR decomposition of $S_j(f)$'s along the way. More specifically, let

$$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n, \quad f'(x) = b_0 x^{n-1} + b_1 x^{n-2} + \cdots + b_{n-1}.$$

We rotate the columns of $S_j(f)$ to form $\hat{S}_j(f)$ in such a way,

$$\left(\begin{array}{ccc|ccc} \multicolumn{3}{c}{\overbrace{\hphantom{b_0 \ \ddots \ b_0}}^{j+1}} & \multicolumn{3}{c}{\overbrace{\hphantom{a_0 \ \ddots \ a_0}}^{j}} \\ b_0 & & & a_0 & & \\ b_1 & \ddots & & a_1 & \ddots & \\ \vdots & \ddots & b_0 & \vdots & \ddots & a_0 \\ b_{n-1} & & b_1 & a_n & & a_1 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & b_{n-1} & & & a_n \end{array}\right)$$

$$\longrightarrow \overbrace{\begin{pmatrix} b_0 & a_0 & & & & & & \\ b_1 & a_1 & b_0 & a_0 & & & & \\ \vdots & \vdots & b_1 & a_1 & \ddots & & & \\ b_{n-1} & a_{n-1} & \vdots & \vdots & & b_0 & a_0 & \\ & a_n & b_{n-1} & a_{n-1} & & b_1 & a_1 & b_0 \\ & & & a_n & \ddots & \vdots & \vdots & b_1 \\ & & & & & b_{n-1} & a_{n-1} & \vdots \\ & & & & & & a_n & b_{n-1} \end{pmatrix}}^{2j+1},$$

that the odd and even columns of $\hat{S}_j(f)$ consist of the coefficients of f' and f, respectively. Consequently, the matrix $\hat{S}_{j+1}(f)$ is simply formed by adding a zero

row at the bottom and two columns in the right on $\hat{S}_j(f)$. Updating the QR decomposition of each $\hat{S}_j(f)$ requires only $O(n)$ additional flops. The inverse iteration (2.1) requires $O(j^2)$ flops at each $S_j(f)$.

Let θ be a given *zero singular-value threshold*, which shall be discussed more in §4.4. With successive QR updating and the inverse iteration, the process of finding the degrees of the GCD triplet (u, v, w) can be summarized as follows.

> Calculate the QR decomposition of the $(n+1) \times 3$ matrix $\hat{S}_1(f) = Q_1 R_1$
> `For` $j = 1, 2, \ldots$ `do`
>   use the inverse iteration (2.1) to find the smallest singular value ς_j
>   of $\hat{S}_j(f)$ and the corresponding right singular vector $\mathbf{y}_j$
>   `if` $\varsigma_j \le \theta \| \mathbf{f} \|_2$, `then` $k{=}j$, $m{=}n{-}k$, extract $\mathbf{v}$ and $\mathbf{w}$ from $\mathbf{y}_j$, `exit`
>   `else`   update $\hat{S}_j(f)$ to $\hat{S}_{j+1}(f) = Q_{j+1} R_{j+1}$
>   `end if`
> `end do`

4.2.2. *The quadratic GCD system.* Let $m = n - k$ be the degree of $\mathrm{GCD}(f, f')$ calculated in `STEP 1`. We now formulate the GCD system (4.2) of `STEP 2` in vector form with unknown vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$:

$$\begin{bmatrix} u_0 \\ \mathrm{conv}(\mathbf{u},\mathbf{v}) \\ \mathrm{conv}(\mathbf{u},\mathbf{w}) \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{f} \\ \mathbf{f}' \end{bmatrix}, \qquad \text{for} \quad \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix} \in \mathbb{C}^{m+1} \times \mathbb{C}^{k+1} \times \mathbb{C}^{k}. \tag{4.3}$$

Here, the convolution $\mathrm{conv}(\cdot, \cdot)$ is defined in Lemma 2.2. The following lemma ensures that this quadratic system is nonsingular.

Lemma 4.1. *The Jacobian of the quadratic system* (4.3) *is*

$$J(\mathbf{u},\mathbf{v},\mathbf{w}) = \begin{bmatrix} \mathbf{e}_1^\top & & \\ C_m(v) & C_k(u) & \\ C_m(w) & & C_{k-1}(u) \end{bmatrix}, \tag{4.4}$$
$$\text{where} \qquad \mathbf{e}_1 = (1, 0, \ldots, 0)^\top \in \mathbb{C}^{m+1}.$$

If $u = \mathrm{GCD}(f, f')$ *with* $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ *satisfying* (4.3), *then* $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ *is of full (column) rank.*

Proof. It is straightforward to verify (4.4) by using Lemma 2.2. To prove $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is of full rank, we assume the existence of polynomials $q(x) = \sum_{j=0}^m q_j x^{m-j}$, $r(x) = \sum_{j=0}^k r_j x^{k-j}$ and $s(x) = \sum_{j=0}^{k-1} s_j x^{k-j-1}$ such that

$$J(\mathbf{u},\mathbf{v},\mathbf{w}) \begin{pmatrix} \mathbf{q} \\ \mathbf{r} \\ \mathbf{s} \end{pmatrix} = 0, \quad \text{or} \quad \begin{cases} q_0 & = & 0, \\ vq + ur & = & 0, \\ wq + us & = & 0. \end{cases} \tag{4.5}$$

Here, as before, $\mathbf{q}$, $\mathbf{r}$ and $\mathbf{s}$ are coefficient vectors of q, r and s, respectively. From (4.5), we have $vq = -ur$ and $wq = -us$. So, $wvq - vwq = -uwr + uvs = 0$, namely $-wr + vs = 0$ or $wr = vs$. Since v and w are coprime, there is a polynomial t such that $r = tv$ and $s = tw$. Consequently, $vq = -ur = -utv$ leads to $q = -tu$. Because $\deg(q) = \deg(tu) \le m$, $\deg(u) = m \ge 0$ and $u_0 = 1$, the degree of t must be zero. So the polynomial t is a constant. Using the first equation in (4.5) and $u_0 = 1$, we have $q_0 = -tu_0 = -t = 0$. It follows that $q = -tu = 0$, $r = tv = 0$ and $s = tw = 0$. Consequently, $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is of full rank. ∎

The equation $u_0 = 1$, absent in the Corless-Gianni-Trager-Watt GCD method [6], needs to be included either explicitly with an equation or implicitly by eliminating u_0 from the variables to ensure the regularity of the system (4.3). In Remark 1 of [5, §4.3] with a heuristic explanation, Chin, Corless, and Corless realized that the restriction $u_0 = 1$ may make their divisor-quotient iteration converge, but abandoned it since their "test showed that the overall performance was worse when this constraint was in place" [5, §5.1.4]. Because we use a different refinement approach in our GCD-finder, preserving this constraint and, thereby the regularity of the GCD system (4.3), may be the very reason why our method obtains robust test results. Without this regularity, the local convergence of the Gauss-Newton iteration we use would not be guaranteed.

Theorem 4.2. *Let $\tilde{u} = \mathrm{GCD}(f, f')$ with $\tilde{v}$ and $\tilde{w}$ satisfying (4.3), and let W be a weight matrix. Then there exists $\varepsilon > 0$ such that for all $\mathbf{u}_0$, $\mathbf{v}_0$, $\mathbf{w}_0$ satisfying $\left\| \mathbf{u}_0 - \tilde{\mathbf{u}} \right\|_2 < \varepsilon$, $\left\| \mathbf{v}_0 - \tilde{\mathbf{v}} \right\|_2 < \varepsilon$ and $\left\| \mathbf{w}_0 - \tilde{\mathbf{w}} \right\|_2 < \varepsilon$, the Gauss-Newton iteration*

$$\begin{bmatrix} \mathbf{u}_{j+1} \\ \mathbf{v}_{j+1} \\ \mathbf{w}_{j+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_j \\ \mathbf{v}_j \\ \mathbf{w}_j \end{bmatrix} - J(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j)_W^+ \begin{bmatrix} \mathbf{e}_1^\top \mathbf{u}_j & - & 1 \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{v}_j) & - & \mathbf{f} \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{w}_j) & - & \mathbf{f}' \end{bmatrix}, \quad j = 0, 1, \ldots, \tag{4.6}$$

converges to $[\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}]^\top$ quadratically. Here $J(\cdot)_W^+ = [J(\cdot)^H W^2 J(\cdot)]^{-1} J(\cdot)^H W^2$ is the weighted pseudo-inverse of the Jacobian $J(\cdot)$ as defined in (4.4).

Proof. A straightforward verification by using Lemma 2.8 and Lemma 4.1. □

4.2.3. *Setting up the initial iterate.* We now need initial iterates $\mathbf{u}_0, \mathbf{v}_0, \mathbf{w}_0$ for the Gauss-Newton iteration (4.6). In `STEP 1`, when the singular value ς_k is calculated, the associated singular vector $\mathbf{y}_k$ consists of $\mathbf{v}_0$ and $\mathbf{w}_0$, which are approximations to $\mathbf{v}$ and $\mathbf{w}$ in (4.3), respectively (see Lemma 2.5). Because of the column rotation in §4.2.1, the odd and even entries of $\mathbf{y}_k$ form $\mathbf{v}_0$ and $\mathbf{w}_0$, respectively. For the initial approximation $\mathbf{u}_0$, notice that in theory the long division yields

$$f(x) = v_0(x) q(x) + r(x) \tag{4.7}$$

with $u_0(x) = q(x)$ and $r(x) = 0$. The process itself may not be numerically stable.

In the context of the Corless-Gianni-Trager-Watt method, Corless et al. [6, Lemma 3] propose the use of the least squares method to minimize $\|\Delta \mathbf{f}\|_2 = \| C_{n-m}(d)\mathbf{h} - \mathbf{f} \|_2$ whenever a candidate d of degree m approximating $\mathrm{GCD}(f, g)$ is available. In our approach, there is no candidate for $u = \mathrm{GCD}(f, f')$ at the end of `STEP 1`. Instead, we have v_0 and w_0 available which approximate v and w, respectively, such that $u = f/v = f'/w$; whereas we can adapt the least squares strategy (used by Corless et al. to calculate $\|\Delta \mathbf{f}\|_2$ from given u_0), to calculate the approximation u_0 of $\mathrm{GCD}(f, f')$ from given v_0 and w_0 in our method, and justify it via a condition theory of linear systems.

By Lemma 2.5, the long division (4.7) with $r(x) = 0$ is equivalent to solving the linear system

$$C_m(v_0)\, \mathbf{u}_0 = \mathbf{f} \tag{4.8}$$

for a least squares solution $\mathbf{u}_0$ that minimizes $\left\| \mathrm{conv}(\mathbf{u}_0, \mathbf{v}_0) - \mathbf{f} \right\|_2$. This "least squares division" is more accurate than the long division (4.7). In fact, the long

division (4.7) is equivalent to solving the $(n+1)\times(n+1)$ lower triangular linear system

$$(4.9)\quad L_m(v_0)\begin{pmatrix}\mathbf{q}\\ \mathbf{r}\end{pmatrix}=\mathbf{f},\quad\text{with}\quad L_m(v_0)=\left(\begin{array}{c|c} C_m(v_0) & \begin{array}{c}0_{(m+1)\times(n-m)}\\ I_{(n-m)\times(n-m)}\end{array}\end{array}\right).$$

The following theorem indicates that solving (4.8) for $\mathbf{u}_0$ may be preferable to using the long division (4.7).

Theorem 4.3. *Let $\kappa(A)$ denote the condition number of an arbitrary matrix A with respect to the matrix 2-norm. Then $\kappa\left(C_m(v)\right)\le\kappa(L_m(v))$ for any polynomial v and $m>0$.*

Proof. For any matrix A, $\kappa(A)=\frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$, where

$$\sigma_{\max}(A)=\max_{\|\mathbf{x}\|_2=1}\Big\|A\mathbf{x}\Big\|_2\quad\text{and}\quad\sigma_{\min}(A)=\min_{\|\mathbf{x}\|_2=1}\Big\|A\mathbf{x}\Big\|_2$$

are the largest and smallest singular values of A, respectively. Therefore

$$\sigma_{\max}(C_m(v))=\max_{\|\mathbf{u}\|_2=1}\Big\|C_m(v)\,\mathbf{u}\Big\|_2=\max_{\|\mathbf{q}\|_2=1,\mathbf{r}=0}\Big\|C_m(v)\,\mathbf{q}+\mathbf{r}\Big\|_2$$

$$=\max_{\|\mathbf{q}\|_2=1,\mathbf{r}=0}\left\|L_m(v)\begin{pmatrix}\mathbf{q}\\ \mathbf{r}\end{pmatrix}\right\|_2\le\max_{\|y\|_2=1}\Big\|L_m(v)\mathbf{y}\Big\|_2=\sigma_{\max}(L_m(v)).$$

Similarly, $\sigma_{\min}(C_m(v))\ge\sigma_{\min}(L_m(v))$, and consequently, $\kappa(C_m(v))\le\kappa(L_m(v))$. □

The magnitude gap between the condition numbers $\kappa(C_m(v))$ and $\kappa(L_m(v))$ can be tremendous for seemingly harmless v and moderate m. Actually, $L_m(v)$ can be pathetically ill-conditioned, making the long division (4.7) virtually a singular process, while $C_m(v)$ is still well conditioned. For example, consider a simple

Table 6. The comparison between the conditions of (4.7) and (4.8) for $v(x)=x+25$.

	$m=1$	$m=5$	$m=10$	$m=20$
$\kappa(C_m(v))$	1	1.0668	1.0791	1.0823
$\kappa(L_m(v))$	627	1.01×10^7	9.92×10^{13}	9.46×10^{27}

Table 7. A numerical comparison between long division and least squares division.

Data		Comparison		
approx. coef. of $f(x)$	coefficients of $v(x)$	known coef.'s of $f(x)\div v(x)$	least squares division	long division
1.00000000	1.00000000	1.00000000	0.9999999999	1.00000000
23.35360257	23.01829201	0.33531056	0.3353105599	0.33531056
29.89831582	22.05776405	0.12227539	0.1222753902	0.122275385
10.75803809		0.54726624	0.5472662398	0.5472663
15.57240922		0.27815340	0.2781534002	0.278151
18.76038493		0.28629915	0.2862991496	0.28634
13.73079603		1.00523653	1.0052365305	1.004
30.45600101		1.00205392	1.0020539195	1.02
46.21275197		0.97391204	0.9739120403	0.5
44.89871211		0.37785145	0.3778514500	11.
30.17981700				
8.33455813				

polynomial $v(x) = x + 25$. When m increases, $\kappa(L_m(v))$ grows exponentially but $\kappa(C_m(v))$ stays as nearly a constant (see Table 6). In fact, we have not encountered a truly ill-conditioned least squares division (4.8) in our extensive numerical experiments. On the other hand, the example shown in Table 7 is quite common. In which $\mathbf{f} = \mathrm{conv}(\mathbf{u}, \mathbf{v})$ is rounded up at the eighth digit after the decimal point. The difference between the long division (Matlab `deconv`) and the least squares division is quite substantial.

Extracting $\mathbf{v}_0$ and $\mathbf{w}_0$ from the singular vector and solving (4.8) for $\mathbf{u}_0$, we shall use them as the initial iterates for the Gauss-Newton iteration (4.6) that refines the GCD triplet. Moreover, the linear system (4.8) is banded, with bandwidth being one plus the number of distinct roots. Therefore, the cost of solving (4.8) is insignificant in the overall complexity.

4.2.4. *Refining the GCD with the Gauss-Newton iteration.* The Gauss-Newton iteration is expected to reduce the residual

$$\left\| \begin{pmatrix} \mathrm{conv}(\mathbf{u}_j, \mathbf{v}_j) \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{w}_j) \end{pmatrix} - \begin{pmatrix} \mathbf{f} \\ \mathbf{f}' \end{pmatrix} \right\|_W = \left\| W \begin{pmatrix} \mathrm{conv}(\mathbf{u}_j, \mathbf{v}_j) & - & \mathbf{f} \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{w}_j) & - & \mathbf{f}' \end{pmatrix} \right\|_2 \tag{4.10}$$

at each step until it is numerically unreducible. We stop the iteration when this residual no longer decreases. The diagonal weight matrix W is used to scale the GCD system (4.3) so that the entries of $W \begin{bmatrix} \mathbf{f} \\ \mathbf{f}' \end{bmatrix}$ are of similar magnitude. Each step of the Gauss-Newton iteration requires solving an overdetermined linear system

$$\begin{bmatrix} WJ(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j) \end{bmatrix} \mathbf{z} = W \begin{bmatrix} \mathbf{e}_1^\top \mathbf{u}_j & - & 1 \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{v}_j) & - & \mathbf{f} \\ \mathrm{conv}(\mathbf{u}_j, \mathbf{w}_j) & - & \mathbf{f}' \end{bmatrix}$$

for its least squares solution $\mathbf{z}$, and requires a QR decomposition of the Jacobian $WJ(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j)$ and a backward substitution for an upper triangular linear system. This Jacobian is a sparse matrix with a special sparsity structure that can largely be preserved during the process. Figure 8 shows the typical sparsity of $WJ(\mathbf{u}, \mathbf{v}, \mathbf{w})$

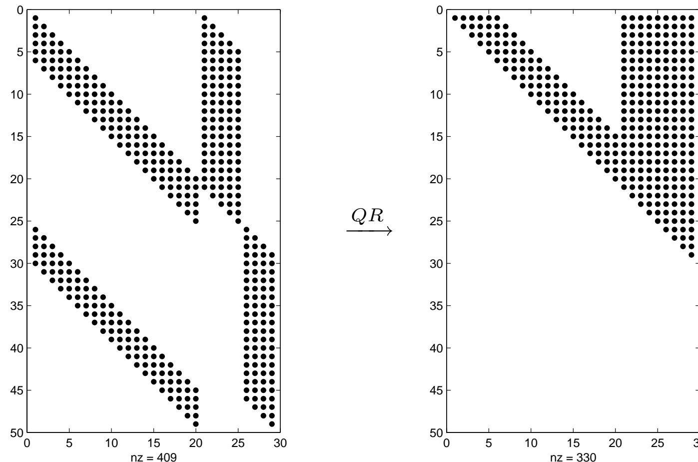

FIGURE 8. Sparsity of $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ and its triangularization.

along with its triangularization. When f is a polynomial of degree n, a straightforward QR decomposition of $WJ(\mathbf{u},\mathbf{v},\mathbf{w})$ costs $O(n^3)$ flops. Taking the sparsity of $WJ(\mathbf{u},\mathbf{v},\mathbf{w})$ into account, it can be verified that the sparse QR decomposition costs $O(mk^2 + m^2k + k^3)$, where, as before, k is the number of distinct roots and $m = n - k$. If k is $o(n)$, then the complexity is reduced to $O(kn^2)$.

4.3. **Computing the multiplicity structure and initial root estimates.** For a given polynomial p, the procedure (4.1) generates a sequence of square-free polynomials $v_1, v_2, \ldots, v_s$ of degrees $d_1 \geq d_2 \geq \cdots \geq d_s$, respectively, such that $p = v_1\, v_2\, \cdots\, v_s$ with

$$\{\text{roots of } v_1\} \supseteq \{\text{roots of } v_2\} \supseteq \cdots \supseteq \{\text{roots of } v_s\}.$$

Moreover, for each $j = 1, \ldots, s$, all roots of v_j are simple. Roots of v_1 consist of all distinct roots of p, while roots of v_2 consist of all distinct roots of p/v_1, etc. With these properties, the multiplicity structure is determined by the degrees $d_1, d_2, \ldots, d_s$. For example, consider

$$p(x) = (x-a)(x-b)^3(x-c)^4$$

for any a, b and c. We have the following.

	v_j's	degrees of v_j's	roots		
$v_1(x) =$	$(x-a)(x-b)(x-c)$	$d_1 = 3$	a	b	c
$v_2(x) =$	$(x-b)(x-c)$	$d_2 = 2$		b	c
$v_3(x) =$	$(x-b)(x-c)$	$d_3 = 2$		b	c
$v_4(x) =$	$(x-c)$	$d_4 = 1$			c
	multiplicity structure of p:		1	3	4

Without locating the roots a, b and c, the multiplicity structure $[\ell_1, \ell_2, \ell_3] = [1, 3, 4]$ is determined solely from the degrees $d_1, \ldots, d_4$.

$$\begin{array}{llllll} \ell_1 = 1 & \text{since} & d_1 & \geq 3 & = (d_1+1) - 1 \\ \ell_2 = 3 & \text{since} & d_1, d_2, d_3 & \geq 2 & = (d_1+1) - 2 \\ \ell_3 = 4 & \text{since} & d_1, d_2, d_3, d_4 & \geq 1 & = (d_1+1) - 3 \end{array}$$

Generally, we have the following theorem on identifying the multiplicity structure.

Theorem 4.4. *For a given polynomial p, let $v_1, \ldots, v_s$ be the square-free factors of p generated by the procedure* (4.1) *with degrees $d_1 \geq d_2 \geq \cdots \geq d_s$, respectively. Let $k = d_1 = \deg(v_1)$. Then the multiplicity structure ℓ of p consists of components*

$$\ell_j = \max\left\{ t \,\middle|\, d_t \geq (d_1 + 1) - j \right\}, \quad j = 1, 2, \ldots, k. \tag{4.11}$$

Proof. A straightforward verification. □

The location of the roots is not needed in deciding the structure.

The initial root approximation is determined based on the fact that an l-fold root of $p(x)$ appears l times as a simple root of each polynomial among $v_1, \ldots, v_l$. After calculating the roots of each v_j with a standard root-finder, numerically "identical" roots of v_j's are grouped in a straightforward manner, according to the multiplicity structure $[\ell_1, \ldots, \ell_k]$ determined by (4.11), to form the initial root approximation $(z_1, \ldots, z_k)$ that is needed by Algorithm I.

4.4. **Control parameters.** We use three control parameters for the recursive GCD computation. The default values of those parameters given below are selected under the assumption that the IEEE standard double precision of 16 decimal digits is used. The first control parameter is the *zero singular-value threshold* θ for identifying the zero singular value. The default choice is $\theta = 10^{-8}$. When the smallest singular value ς_l of $\hat{S}_l(u_{m-1})$ is less than $\theta \left\| \mathbf{u}_{m-1} \right\|_2$, it will be *tentatively* considered as a zero (pending confirmation from the residual information produced by the Gauss-Newton iteration). Then the Gauss-Newton iteration is initiated to further reduce the residual, as in (4.10), to its numerical limit. We use the second control parameter, the *initial residual tolerance* ϱ, to decide if the refined residual is acceptable. Our default choice is $\varrho = 10^{-10}$. We accept the GCD triplet (u_m, v_m, w_m) when the residual

$$\rho_m = \left\| \begin{pmatrix} \mathrm{conv}(\mathbf{u}_m, \mathbf{v}_m) - \mathbf{u}_{m-1} \\ \mathrm{conv}(\mathbf{u}_m, \mathbf{w}_m) - \mathbf{u}'_{m-1} \end{pmatrix} \right\|_W \leq \varrho \left\| \mathbf{u}_{m-1} \right\|_2. \tag{4.12}$$

Otherwise, we continue to update $S_l(u_{m-1})$ to $S_{l+1}(u_{m-1})$ and check $\varsigma_{l+1}, \ldots$.

Pseudo-code GcdRoot `(Algorithm II)`

```
 input:  The polynomial p of degree n, singular threshold θ,
             residual tolerance ϱ, residual growth factor φ.
             (If only p is provided, set θ = 10^-8, ϱ = 10^-10, φ = 100 )
 output:  the root estimates (z_1,...,z_k)^T and
          multiplicity structure [ℓ_1,...,ℓ_k]

     Initialize u_0 = p
     for m = 1,2,...,s, until deg(u_s) = 0 do
          for l = 1,2,... until residual ρ < ϱ‖u_{m-1}‖_2 do
               calculate the singular pair (ς_l, y_l) of Ŝ_l(u_{m-1})
                    by iteration (2.1)
               if ς_l < θ‖u_{m-1}‖_2 then
                    set up the GCD system (4.3) with f = u_{m-1}
                         (see Section 4.2.2 )
                    extract v_m^(0), w_m^(0) from y_l and calculate u_m^(0)
                         (see Section 4.2.3)
                    apply the Gauss-Newton iteration (4.6) from
                         u_m^(0), v_m^(0), w_m^(0) to obtain u_m, v_m, w_m
                    extract the residual ρ = ρ_m as in (4.12)
               end if
          end do
          adjust the residual tolerance ϱ to be max{ϱ, φρ_j}, and
                    set d_m = deg(v_m)
     end do
     set k = d_1, ℓ_j = max{ t | d_t ≥ k - j + 1 },   j = 1,2,...,k.
     match the roots of v_m(x), m = 1,2,...,s
               according to the multiplicities ℓ_j's.
```

FIGURE 9. Pseudo-code of Algorithm II.

The third parameter is the *residual tolerance growth factor* ϕ. Whenever a GCD triplet (u_m, v_m, w_m) and ρ_m are calculated, the error in (u_m, v_m, w_m) may cause the residual ρ_{m+1} of $(u_{m+1}, v_{m+1}, w_{m+1})$ to grow. Therefore, the tolerance ϱ may need adjustment. Our default growth factor is 100. After obtaining ρ_m, the residual tolerance ϱ is adjusted to be $\max\left\{ \varrho, \quad \phi\,\rho_m \right\}$. Notice that the growth factor is applied to the residual ρ_m rather than the residual tolerance ϱ. The residual tolerance ϱ itself may not grow at every step.

From our computing experience, the default control parameters work well for "normal" polynomials, such as those with unclustered roots of moderate multiplicities. For difficult problems, one may manually adjust the parameters. The overall Algorithm II shown in Figure 9 is implemented as Matlab code GcdRoot and included in the MultRoot package.

4.5. **Remarks on the convergence of Algorithm II.** There are two iterative components in Algorithm II. One of them is the inverse iteration (2.1). By Lemma 2.6, the iteration converges for all starting vectors $\mathbf{x}_0$, unless $\mathbf{x}_0$ is orthogonal to the intended singular vector $\mathbf{y}$. The probability of the occurrence of this orthogonality is zero. But even if it occurs, roundoff errors in the numerical computation will quickly destroy the orthogonality during iteration. Therefore, the inverse iteration (2.1) always converges. The other iterative component is the Gauss-Newton iteration (4.6) whose local convergence is ensured in Theorem 4.2. Therefore, as long as the rank decision on the Sylvester matrices is accurate and the error on the initial approximation of the GCD triplet is small, Algorithm II will produce correct multiplicity structure and a root approximation.

However, due to the nature of the problem, there is no guarantee that the original multiplicity structure can be identified from an inexact polynomial. When a polynomial is perturbed to a place that has equal distances to two or more different pejorative manifolds, it is somewhat unrealistic to expect any method to recover reliably from the perturbation. Therefore, we have conducted extensive numerical experiments in addition to the results exhibited in this paper. As reported in our software release note [38], we made a comprehensive test suite of 104 polynomials based on Jenkins-Traub Testing Principles [20]. These polynomials include all the test examples we have seen in the literature that have been used by experts to test the robustness, stability, accuracy, and efficiency of root-finders intended for multiple roots. On all the polynomials with multiple roots in the test suite, our package MultRoot consistently outputs accurate root/multiplicity results near machine precision. They are far beyond the "attainable accuracy" barrier that other algorithms are subject to. The test suite is available electronically from the author.

4.6. **Numerical results for Algorithm II.** The effectiveness of Algorithm II can be shown by the polynomial

$$p(x) = (x-1)^{20}(x-2)^{15}(x-3)^{10}(x-4)^5 \tag{4.13}$$

generated by the Matlab polynomial generator poly, with coefficients rounded up at 16 digits. Using the default control parameters, the Algorithm II code GcdRoot correctly identifies the multiplicity structure. The roots are approximated to an accuracy of 10 digits or more. With this result as input to Algorithm I code

TABLE 8. Roots of $p(x)$ in (4.13) computed in two stages.

```
Algorithm II (code GcdRoot) result:       | Algorithm I (code PejRoot) result
                                          |   THE BACKWARD ERROR:                   6.16e-016
The backward error is 6.057721e-010       |   THE ESTIMATED FORWARD ROOT ERROR:     9.46e-014
                                          |
   computed roots       multiplicities    |     computed roots        multiplicities
                                          |
   4.000000000109542          5           |     3.999999999999985            5
   3.000000000176196         10           |     3.000000000000011           10
   2.000000000030904         15           |     1.999999999999997           15
   1.000000000000353         20           |     1.000000000000000           20
```

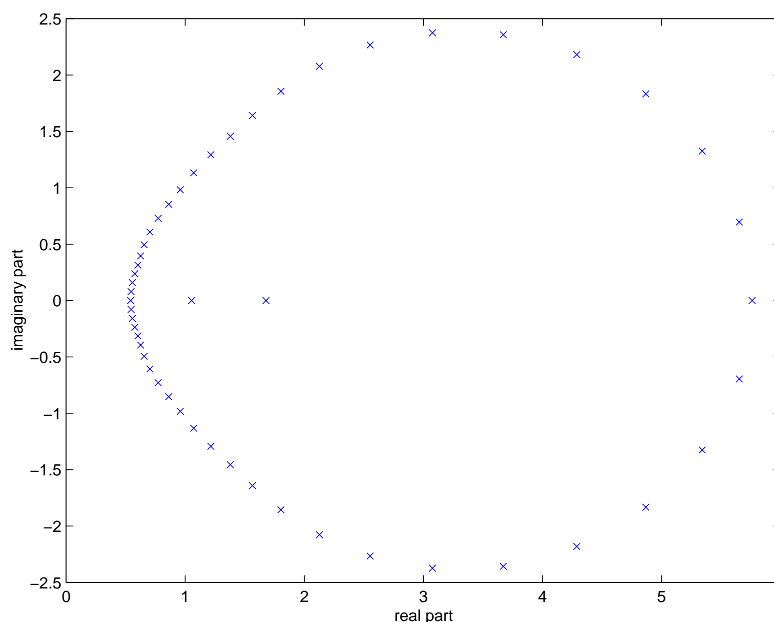

FIGURE 10. MPSOLVE results for the polynomial (4.13) using multiprecision.

PEJROOT, we obtained all multiple roots in the end with at least 14 correct digits (Table 8).

Polynomials with such high multiplicities are extremely difficult by any standard for root-finding. The magnitude of its coefficients stretches from 1 to 10^{21}. Remarkably, our algorithms have no difficulty finding all its multiple roots. To the best of our knowledge, there are no other methods that can calculate multiple roots for such polynomials. Since the coefficients are inexact, multiprecision root-finders also fail to calculate the roots with meaningful accuracy. Figure 10 shows the computed roots by MPSOLVE [2] using a virtually unlimited number of digits in machine precision. Those results are quite remote from the roots 1, 2, 3, 4.

The Euclidean method has also been used to find GCD in order to identify the multiplicities [3, 31]. Uhlig's PZERO [31] is a Matlab implementation based on the Euclidean method. The drawback of the Euclidean method is its reliance on recursive long division that is numerically unstable (see §4.2.3). Here we compare

TABLE 9. Partial results on $p_k(x)=(x-1)^{4k}(x-2)^{3k}(x-3)^{2k}(x-4)^k$ and comparison between PZERO and GCDROOT. Numbers in parentheses are computed multiplicities. Wrong multiplicities are in boldface.

k	code	$x_1 = 1$		$x_2 = 2$		$x_3 = 3$	
1	PZERO	1.00000000001	(4)	1.99999999998	(3)	3.000000000005	(2)
	GCDROOT	0.999999999999990	(4)	1.99999999999998	(3)	3.0000000000005	(2)
2	PZERO	1.0000000001	(8)	2.000000002	(6)	3.000000004	(4)
	GCDROOT	0.9999999999998	(8)	1.999999999983	(6)	2.99999999991	(4)
3	PZERO	0.9999999897	(**13**)	1.99999990	(**8**)	2.9999998	(**5**)
	GCDROOT	0.9999999999997	(12)	1.99999999997	(9)	2.9999999998	(6)
4	PZERO	0.9999995	(**21**)	1.999994	(**6**)	2.999990	(**7**)
	GCDROOT	1.0000000000003	(16)	2.00000000002	(12)	3.0000000001	(8)
5	PZERO	1.0000009	(**28**)	2.00001	(**8**)	3.00002	(**6**)
	GCDROOT	1.0000000000004	(20)	2.00000000003	(15)	3.0000000002	(10)
6	PZERO	− − −−	(**1**)	− − −−	(**1**)	− − −−	(**1**)
	GCDROOT	1.0000000000002	(24)	2.00000000001	(18)	3.00000000004	(12)
7	PZERO	− − −−	(**1**)	− − −−	(**1**)	− − −−	(**1**)
	GCDROOT	1.0000000000001	(28)	2.00000000001	(21)	3.00000000006	(14)

our code GCDROOT with PZERO on the polynomials

$$p_k(x) = (x-1)^{4k}(x-2)^{3k}(x-3)^{2k}(x-4)^k \quad \text{for } k = 1, 2, \ldots, 8.$$

When the multiplicities increase, the root accuracy deteriorates with PZERO, which successfully identifies the multiplicity structure for $k = 1$ and $k = 2$ but fails to do so afterwards. In comparison, GCDROOT consistently attains at least 11 digits in root accuracy with increasing multiplicities. The multiplicity structures are identified correctly for k up to 7 and multiplicities up to 28. For the current implementation, the limitation of GCDROOT on this sequence is for $k \leq 7$, whereas the root accuracy will stay the same for even larger k (see Table 9).

5. NUMERICAL RESULTS FOR THE COMBINED METHOD

5.1. The effect of inexact coefficients. In application, input data are expected to be inexact. The following experiment tests the effect of data error on the accuracy as well as robustness of both Algorithm I and II. For

$$p(x) = \left(x - \frac{10}{11}\right)^5 \left(x - \frac{20}{11}\right)^5 \left(x - \frac{30}{11}\right)^5$$

in general form, every coefficient is rounded up to k-digit accuracy, where $k = 10, 9, 8, \ldots$.

For this sequence of problems, Algorithm II code GCDROOT correctly identifies the multiplicity structure if the coefficients have at least seven accurate digits. If the multiplicities are manually given rather than computed by GCDROOT, Algorithm I code PEJROOT continues to converge even when data accuracy is down to three digits. For lower data accuracy, the residual tolerance ϱ in GCDROOT needs to be adjusted accordingly. Table 10 shows the results of both programs.

As shown in this test, both methods allow inexact coefficients to a certain extent. As usual, Algorithm I is more robust than Algorithm II, but Algorithm I depends on a structure identifier.

Table 10. Effect of coefficient error on computed roots.

number of correct digits	control parameters ϱ, θ	code	$x_1 = 0.\dot{9}\dot{0}$	$x_2 = 1.\dot{8}\dot{1}$	$x_3 = 2.\dot{7}\dot{2}$	backward error
$k = 10$	$\varrho = 1e-9$	GcdRoot	0.90909090	1.8181818	2.7272727	1.7e-08
	$\theta = 1e-7$	PejRoot	0.909090909	1.81818181	2.7272727	2.4e-10
$k = 9$	$\varrho = 1e-8$	GcdRoot	0.909090	1.81818	2.72727	7.0e-06
	$\theta = 1e-6$	PejRoot	0.9090909	1.8181818	2.727272	2.3e-09
$k = 8$	$\varrho = 1e-7$	GcdRoot	0.90909	1.8182	2.727	1.3e-04
	$\theta = 1e-5$	PejRoot	0.9090909	1.818181	2.72727	2.3e-08
$k = 7$	$\varrho = 1e-6$	GcdRoot	0.9090	1.82	2.7	1.3e-02
	$\theta = 1e-4$	PejRoot	0.90909	1.81818	2.7272	2.3e-07
$k = 6$	$----$	PejRoot	0.9090	1.8181	2.727	3.7e-06
$k = 5$	$----$	PejRoot	0.909	1.818	2.72	2.4e-05
$k = 4$	$----$	PejRoot	0.90	1.81	2.7	1.9e-04
$k = 3$	$----$	PejRoot	0.9	1.8	2.8	1.8e-03

5.2. **The effect of nearby multiple roots.** When two or more multiple roots are nearby, it can be difficult to identify the correct multiplicity structure. We test the example

$$p_\varepsilon(x) = (x - 1 + \varepsilon)^{20}(x-1)^{20}(x+0.5)^5$$

for the decreasing root gap $\varepsilon = 0.1,\ 0.01, \ldots,$ making the root $x_1 = 0.9$, 0.99, 0.999, ... along with fixed roots $x_2 = 1$ and $x_3 = -0.5$. When the root gap decreases, the control parameters may need adjustment. In this test, we use the default parameters for all cases except $\varepsilon = 0.0001$, in which the residual growth factor $\phi = 5$. GcdRoot is used to find the initial input for PejRoot. Computing results are shown for both programs in Table 11.

When the default growth factor stays the same as the default $\phi = 100$ and the gap $\varepsilon \le 0.0001$, GcdRoot outputs a multiplicity structure $[40, 5]$. Namely, GcdRoot treats the two nearby 20-fold roots 1 and $1 - \varepsilon$ as a single 40-fold one. From the computed backward error and the condition number, this may not necessarily be incorrect (see Table 12). When the backward error becomes 10^{-12} and the condition number is tiny (0.0066), they are numerically accurate! In contrast, using the "correct" multiplicity structure $[20, 20, 5]$, PejRoot outputs roots with backward error 10^{-10} and a large condition number 5791.8 (last line in Table 11).

By adjusting the control parameters, GcdRoot can find different pejorative manifolds that are close to the given polynomial. PejRoot then calculates corresponding pejorative roots. The selection of the most suitable solution should be application dependent.

Table 11. Effect of decreasing the root gap on computed roots.

gap ε	code	$x_1 = 1 - \varepsilon$	$x_2 = 1$	$x_3 = -0.5$	backward error	cond. num.
	GcdRoot	0.89999999999	0.99999999999	-0.49999999999999	9.7e-10	
10^{-1}	PejRoot	0.9000000000000	0.9999999999999	-0.50000000000000	2.7e-13	.7
	GcdRoot	0.98999999	0.99999999	-0.50000000000000	3.2e-07	
10^{-2}	PejRoot	0.989999999999	1.000000000000	-0.49999999999999	1.0e-12	6.7
	GcdRoot	0.99900	1.00000	-0.49999999999999	1.9e-04	
10^{-3}	PejRoot	0.99899999999	1.00000000000	-0.500000000000000	4.1e-13	62.5
	GcdRoot	0.9997	0.99996	-0.4999999999999	1.1e-02	
10^{-4}	PejRoot	0.999900000	0.999999999	-0.50000000000000	4.0e-12	621.7
10^{-5}	PejRoot	0.999989990	1.0000000	-0.50000000000000	4.0e-10	5791.8

TABLE 12. If the control parameter is not adjusted, the tiny root gap makes computed roots identical. However, from the backward errors and the condition number, they are not necessarily wrong answers.

root gap ε	code	$x_1 = 1 - \varepsilon$	$x_2 = 1$	$x_3 = -0.5$	backward error	cond. num.
$\varepsilon = 0.0001$	GCDROOT	0.99994999	0.99994999	-0.5000000000	5.7e-08	
	PEJROOT	0.999949999	0.999949999	-0.500000000	2.2e-08	0.0066
$\varepsilon = 0.00001$	GCDROOT	0.9999949999	0.9999949999	-0.500000000000	1.1e-10	
	PEJROOT	0.99999499999	0.99999499999	-0.50000000000	4.0e-12	0.0066

5.3. **A large inexact problem.** By implementing the combination of two methods, we have produced a Matlab code MULTROOT. We conclude this report by testing this code on our final test problem. First of all, 20 complex numbers are randomly generated and used as roots

$$.5 \pm i,\ -1 \pm .2i,\ -.1 \pm i,\ -.8 \pm .6i,\ -.7 \pm .7i,\ 1.4,\ -.4 \pm .9i,$$
$$.9,\ -.8 \pm .3i,\ .3 \pm .8i,\ .6 \pm .4i$$

to generate a polynomial f of degree 20. We then round all coefficients to 10 decimal digits. The coefficients are shown below.

coefficients of f
1
−0.7
−0.19
0.177
−0.7364
−0.43780
−0.952494
−0.2998258
−0.00322203
−0.328903811
−0.4959527435
−0.9616679762
0.4410459281
0.1090273141
0.6868094008
0.0391923826
0.0302248540
0.6603775863
−0.1425784968
−0.3437618593
0.4357949015

We construct multiple roots by squaring f repeatedly. Namely,

$$g_k(x) = [\, f(x)\,]^{2^k}, \quad k = 1, 2, 3, 4, 5.$$

At $k = 5$, g_5 has a degree 640 and 20 complex roots of multiplicity 32. Since the machine precision is 16 digits, the polynomials g_k are inexact. Using the default control parameters, our combined program encounters no difficulty in calculating all the roots as well as finding accurate multiplicities. The worst accuracy of the roots is 11-digits. Here is the final result.

```
 THE STRUCTURE PRESERVING CONDITION NUMBER:          0.0780464
 THE BACKWARD ERROR:                           6.38e-012
 THE ESTIMATED FORWARD ROOT ERROR:             9.96e-013

 computed roots                    multiplicities  |  computed roots                    multiplicities
                                                   |
 0.499999999999399 + 1.000000000006247 i    32     |  1.400000000000303 + 0.000000000000000 i    32
 0.499999999999399 - 1.000000000006247 i    32     | -0.399999999999482 + 0.899999999996264 i    32
-1.000000000003141 + 0.200000000004194 i    32     | -0.399999999999482 - 0.899999999996264 i    32
-1.000000000003140 - 0.200000000004193 i    32     |  0.899999999996995 - 0.000000000000000 i    32
-0.099999999996612 + 1.000000000001018 i    32     | -0.799999999987544 + 0.299999999995441 i    32
-0.099999999996612 - 1.000000000001018 i    32     | -0.799999999987544 - 0.299999999995441 i    32
 0.800000000001492 + 0.600000000001814 i    32     |  0.299999999995789 + 0.799999999976189 i    32
 0.800000000001492 - 0.600000000001815 i    32     |  0.299999999995789 - 0.799999999976189 i    32
-0.699999999997635 + 0.699999999997984 i    32     |  0.599999999989084 + 0.399999999997279 i    32
-0.699999999997635 - 0.699999999997984 i    32     |  0.599999999989084 - 0.399999999997279 i    32
```


Acknowledgments

The author wishes to thank the following scholars for their contributions which improved this paper. T. Y. Li, Ross Lippert, Hans Stetter, Joab Winkler, and the anonymous referees made valuable suggestions on the presentation. One of the referees pointed out some important previous works in [6, 36], Barry Dayton found an error in an early version of the manuscript. Frank Uhlig provided his insight on the subject in e-correspondence along with his software. Peter Kravanja also freely shared his code. D. A. Bini and G. Fiorentino, as well as S. Fortune made their root-finders freely available for electronic download. The author is grateful to the Program Committee of the ACM 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC) for their recognition of this work with its Distinguished Paper Award.

References

1. D. H. Bailey, *A Fortran-90 based multiprecision system*, ACM Trans. Math. Software, 21 (1995), pp. 379–387.
2. D. Bini and G. Fiorentino, *Numerical computation of polynomial roots using MPSolve – version 2.0*. manuscript, Software and paper available at `ftp://ftp.dm.unipi.it/pub/mpsolve/`, 1999.
3. L. Brugnanao and D. Trigiante, *Polynomial roots: the ultimate answer?*, Linear Alg. and Its Appl., 225 (1995), pp. 207–219. MR 96b:65050
4. L. Brugnano, *Numerical implementation of a new algorithm for polynomials with multiple roots*, J. Difference Eq. and Appl., 1 (1995), pp. 187–207. MR 96m:12001a
5. P. Chin, R. M. Corless, and G. F. Corless, *Optimization strategies for the approximate GCD problem*, Proceedings of 1998 International Symposium on Symbolic and Algebraic Computation (ISSAC '98), New York, 1998, ACM Press, pp. 228–235. MR 2001m:68004
6. R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt, *The singular value decomposition for polynomial systems*, Proceedings of 1995 nternational Symposium on Symbolic and Algebraic Computation (ISSAC '95), ACM Press, New York, 1995, pp. 195–207.
7. J.-P. Dedieu and M. Shub, *Newton's method for over-determined system of equations*, Math. Comp., 69 (1999), pp. 1099–1115. MR 2000j:65133
8. J. W. Demmel, *On condition numbers and the distance to the nearest ill-posed problem*, Numer. Math., 51 (1987), pp. 251–289. MR 88i:15014
9. J. W. Demmel and B. Kagström, *The generalized Schur decomposition of an arbitrary pencil $A - \lambda B$: robust software with error bounds and applications. Part I & Part II*, ACM Trans. Math. Software, 19 (1993), pp. 161–201. MR 96d:65060a; MR 96d:65060b
10. J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall Series in Computational Mathematics, Prentice-Hall, Englewood Cliffs, New Jersey, 1983. MR 85j:65001

11. A. Edelman, E. Elmroth, and B. Kagström, *A geometric approach to perturbation theory of matrices and and matrix pencils. Part I: Versal deformations*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 693–705. MR 99e:58021
12. A. Edelman, E. Elmroth, and B. Kagström, *A geometric approach to perturbation theory of matrices and and matrix pencils. Part II: a stratification-enhanced staircase algorithm*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 667–699. MR 2000c:65032
13. I. Z. Emiris, A. Galligo, and H. Lombardi, *Certified approximate univariate GCDs*, J. Pure Appl. Algebra, 117/118 (1997), pp. 229–251. MR 98f:65135
14. M. R. Farmer and G. Loizou, *An algorithm for the total, or partial, factorization of a polynomial*, Math. Proc. Camb. Phil. Soc., 82 (1977), pp. 427–437. MR 57:14402
15. M. R. Farmer and G. Loizou, *Locating multiple zeros interactively*, Comp. Math. Appl., 11 (1985), pp. 595–603. MR 86h:65067
16. S. Fortune, *An iterated eigenvalue algorithm for approximating roots of univariate polynomials*, J. Symbolic Comput., 33 (2002), pp. 627–646. MR 2003e:13039
17. W. Gautschi, *Questions of numerical condition related to polynomials*, in MAA Studies in Mathematics, Vol. 24, Studies in Numerical Analysis, G. H. Golub, ed., USA, 1984, The Mathematical Association of America, pp. 140–177. MR 88i:65007
18. V. Hribernig and H. J. Stetter, *Detection and validation of clusters of polynomial zeros*, J. Symb. Comput., 24 (1997), pp. 667–681. MR 2000b:68275
19. M. Igarashi and T. Ypma, *Relationships between order and efficiency of a class of methods for multiple zeros of polynomials*, J. Comput. Appl. Math., 60 (1995), pp. 101–113. MR 96f:65059
20. M. A. Jenkins and J. F. Traub, *Principles for testing polynomial zerofinding programs*, ACM Trans. Math. Software, 1 (1975), pp. 26–34. MR 53:2009
21. W. Kahan, *Conserving confluence curbs ill-condition*. Technical Report 6, Computer Science, University of California, Berkeley, 1972.
22. N. K. Karmarkar and Y. N. Lakshman, *On approximate polynomial greatest common divisors*, J. Symb. Comput., 26 (1998), pp. 653–666. MR 99j:68059
23. P. Kravanja and M. Van Barel, *Computing Zeros of Analytic Functions, Lecture Notes in Mathematics, 1727*, Springer-Verlag, 2000. MR 2001c:65004
24. R. A. Lippert and A. Edelman, *The computation and sensitivity of double eigenvalues*, in Advances in computational mathematics, Lecture Notes in Pure and Appl. Math. 202, New York, 1999, Dekker, pp. 353–393. MR 2000e:65043
25. T. Miyakoda, *Iterative methods for multiple zeros of a polynomial by clustering*, J. Comput. Appl. Math., 28 (1989), pp. 315–326. MR 91k:65086
26. V. Y. Pan, *Numerical computation of a polynomial gcd and extensions*. Research Report 2996, Institut National de Recherche en Informatique et en Automatique (INRIA), Sophia-Antipolis, France, 1996.
27. V. Y. Pan, *Solving polynomial equations: some history and recent progress*, SIAM Review, 39 (1997), pp. 187–220. MR 99b:65066
28. D. Rupprecht, *An algorithm for computing certified approximate GCD of n univariate polynomials*, J. Pure and Appl. Alg., 139 (1999), pp. 255–284. MR 2000d:65255
29. H. J. Stetter, *Condition analysis of overdetermined algebraic problems*, in Computer Algebra in Scientific Computing–CASC 2000, e. a. V.G. Ganzha, ed., Springer, 2000, pp. 345–365. MR 2002e:65088
30. J. A. Stolan, *An improved Šiljak's algorithm for solving polynomial equations converges quadratically to multiple zeros*, J. Comput. Appl. Math., 64 (1995), pp. 247–268. MR 96h:65074
31. F. Uhlig, *General polynomial roots and their multiplicities in $O(n)$ memory and $O(n^2)$ time*, Linear and Multilinear Algebra, 46 (1999), pp. 327–359. MR 2000i:12010
32. S. Van Huffel, *Iterative algorithms for computing the singular subspace of a matrix associated with its smallest singular values*, Linear Alg. Appl., 154-156 (1991), pp. 675–709. MR 92d:65065
33. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, N.J., 1963. MR 28:4661
34. J. R. Winkler, *Condition numbers of a nearly singular simple root of a polynomial*, Appl. Numer. Math., (2001), pp. 275–285.
35. T. J. Ypma, *Finding a multiple zero by transformations and Newton-like methods*, SIAM Review, 25 (1983), pp. 365–378. MR 85a:65078

36. D. Y. Y. Yun, *On square-free decomposition algorithms*, in Proceedings of 1976 ACM Symposium of Symbolic and Algebraic Computation (ISSAC'76), ACM Press, Yorktown Heights, New York, 1976, pp. 26–35.
37. Z. Zeng, *A method computing multiple roots of inexact polynomials*, Proceedings of 2003 International Symposium of Symbolic and Algebraic Computation (ISSAC '03), ACM Press, New York, 2003, pp. 266–272.
38. Z. Zeng, *Algorithm* 835 : *Multroot – a Matlab package computing polynomial roots and multiplicities*, ACM Trans. Math. Software, 30 (2004), pp. 218-235.
39. Z. Zeng, *On ill-conditioned eigenvalues, multiple roots of polynomials, and their accurate computation*. MSRI Preprint No. 1998-048, (1998).

Department of Mathematics, Northeastern Illinois University, Chicago, Illinois 60625
E-mail address: zzeng@neiu.edu