Computing multiple roots of inexact polynomials

Zhonggang Zeng
Northeastern Illinois University

Contents

List of Figures

List of Tables

Computing multiple roots of inexact polynomials *

Zhonggang Zeng†

Abstract

We present a combination of two novel algorithms that accurately calculate multiple roots of general polynomials. For a given multiplicity structure and initial root estimates, Algorithm I transforms the singular root-finding into a regular nonlinear least squares problem on a pejorative manifold, and calculates multiple roots simultaneously. To fulfill the input requirement of Algorithm I, we employ a numerical GCD-finder, containing a partial singular value decomposition and an iterative GCD refinement, as the main engine of Algorithm II that calculates the multiplicity structure and the initial root approximation. The combined method calculates multiple roots with high forward accuracy 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 true sensitivity of the multiple roots, a pejorative condition number is proposed and error bounds are given. Extensive computational experiments and the error analysis confirm that a polynomial being ill-conditioned in the conventional sense can be well conditioned pejoratively, and its multiple roots can be computed with remarkable accuracy.

1 Introduction

In this paper, we present a combination of two novel numerical algorithms that accurately calculate multiple roots of polynomials with coefficients possibly being inexact without using multiprecision arithmetic.

Polynomial root-finding is one of the classical problems with longest and richest history. One of the most difficult issues in root-finding is computing multiple roots. In addition to requiring *exact* coefficients, it is widely believed that it is necessary to use multiprecision arithmetic when multiple roots are present [20].

It is also believed that there is a barrier of "attainable accuracy" in computing multiple roots [14, 20, 28]: 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. Multiprecision softwares [1, 21] are available. However, when polynomial coefficients are truncated, multiple roots would turn into clusters. In such cases, extending machine precision on the inexact polynomial would not 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.

*Mathematics Subject Classification 65F35, 65H05

†Dept of Math, Northeastern Illinois University, Chicago, IL 60625, email: zzeng@neiu.edu.

While many numerical analysts consider multiple roots being hypersensitive in numerical computation, W. Kahan [16] proved that if the multiplicities are preserved, the multiple roots can actually be well behaved. More precisely, polynomials with a 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 theory, 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 polynomial onto the manifold, the computation remains structure-preserving. As a result, all the roots, regardless of the multiplicities, are calculated simultaneously and accurately.

In applying our Algorithm I, one needs to have a priori knowledge of the multiplicity structure of the polynomial and its initial root estimates. To fulfill this input requirement, we propose Algorithm II in §4 that employs a numerical GCD-finder, containing a partial singular value decomposition of the Sylvester discriminant matrices and an iterative refinement strategy for the recursive GCD computation. The resulting algorithm calculates the multiplicity structure and its initial root approximation for a given polynomial.

In §3.4, we propose a pejorative condition number that measures the sensitivity of multiple roots. A polynomial that is ill-conditioned in conventional sense can be well conditioned pejoratively, and its roots can be calculated beyond the barrier of "attainable accuracy". This pejorative condition number can easily be calculated. Error bounds on the computed roots are given for inexact polynomials.

In §3.6 and §4.3, we present separate numerical results for Algorithm I and Algorithm II. The numerical results for the combined algorithm are shown in §5. Both algorithms and their combination are implemented as a Matlab package MultRoot that is electronically available[1].

The combined algorithm is accurate, stable and reasonably efficient. Taking the coefficient vector as the *only* input, it not only outputs the roots and multiplicities, but also verifies the solution automatically via the backward error, the estimated forward error, and the pejorative condition number as by-products. The total complexity is clearly no more than $O(n^3)$. The most significant features are its remarkable accuracy and robustness in handling inexact data. As shown in numerical examples, the hybrid code accurately identifies the multiplicity structure and multiple roots for polynomials with a coefficient accuracy being as low as 7 digits. With given multiplicities, Algorithm I converges even with lower data accuracy such as 3 decimal digits. This appears to be the first blackbox-type root-finder with such capability.

While numerical experiments reported in the literature rarely 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 we have seen in the literature can be repeated only if multiprecision is used

[1] `http://www.neiu.edu/~zzeng/multroot.htm`

on exact polynomials, such as the works of Farmer-Loizou [12] and Iliev[15]. A zero-finder for general analytic functions with multiple zeros has been developed by Kravanja and Van Barel [18]. The method uses an accuracy refinement with modified Newton's iteration that also requires multiprecision for multiple roots unless the polynomial is already factored [29].

The idea of exploiting the pejorative manifold and the problem structure has been used extensively for ill-conditioned problems. Besides Kahan's pioneer work 30 years ago, theories and computational strategies for the matrix canonical forms have been studied, such as [7, 9, 10, 19], 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 Notations

In this paper, $\mathbf{R}^n$ and $\mathbf{C}^n$ denote the n dimensional real and complex vector spaces respectively. All vectors are columns and denoted by boldface lower case letters. Matrices are denoted by upper case letters. The notation $(\cdot)^\top$ represents the transpose of $(\cdot)$, and $(\cdot)^H$ the Hermitian adjoint (i.e. conjugate transpose) of $(\cdot)$. When we use a (lower case) letter, say p, to denote a degree n polynomial, then $p_0, p_1, \cdots, p_n$ 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

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

unless defined otherwise.

2.2 Basic definitions and lemmas

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

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

is called the k*-th order* **Cauchy matrix** *associated with* p.

Lemma 2.1 *Let*

$$f(x) = f_0\, x^n + f_1\, x^{n-1} + \cdots + f_n, \quad g(x) = g_0\, x^m + g_1\, x^{m-1} + \cdots + g_m$$

and $h(x) = f(x)g(x)$. *Then* $\mathbf{h}$ *is the convolution of* $\mathbf{f}$ *and* $\mathbf{g}$ *defined by*

$$\mathbf{h} = conv(\mathbf{f}, \mathbf{g}) = C_{m+1}(f)\mathbf{g} = C_{n+1}(g)\mathbf{f}.$$

Proof. A straightforward verification. Q.E.D.

Definition 2.2 *Let* $p(x)$ *be a polynomial of degree* n *and* $p'(x)$ *be its derivative. For* $k = 1, 2, \cdots n$, *the matrix of size* $(n+k) \times (2k+1)$

$$S_k(p) = \left[\begin{array}{c|c} C_{k+1}(p') & C_k(p) \end{array} \right]$$

is called the k*-th* **Sylvester discriminant matrix**.

Lemma 2.2 *Let* $u(x) \equiv GCD(p, p')$ *be the greatest common divisor of* $p(x)$ *and* $p'(x)$. *Let* ς_j *be the smallest singular value of* $S_j(p)$, $j = 1, 2, \cdots, n$. *Then the following are equivalent.*

(a) The degree of $u(x)$, $deg(u)$, *is* m.

(b) The polynomial $p(x)$ *has* $k = n - m$ *distinct roots.*

(c) $\varsigma_1 \geq \varsigma_2 \geq \cdots \geq \varsigma_{k-1} > 0, \qquad \varsigma_k = \varsigma_{k+1} = \cdots = \varsigma_n = 0.$

Proof. If $p(x)$ has distinct roots $z_1, \cdots, z_k$, then there are positive integers $\ell_1, \cdots, \ell_k$ such that

$$\begin{aligned} p(x) &= p_0(x - z_1)^{\ell_1} \cdots (x - z_k)^{\ell_k}, \\ p'(x) &= p_0 \left[\prod_{j=1}^{k} (x - z_j)^{\ell_j - 1} \right] \left[\sum_{i=1}^{k} \left(\ell_i \prod_{l \neq i} (x - z_l) \right) \right]. \end{aligned}$$

Therefore $u(x) = GCD(p, p') = \prod_{j=1}^{k} (x - z_j)^{\ell_j - 1}$, making (a) and (b) equivalent. By Proposition 3.1 in [22], (a) holds if and only if $\varsigma_{k-1} > 0$ and $\varsigma_k = \cdots = \varsigma_n = 0$. Because the smallest singular value ς_{min} of a matrix A equals

$$\varsigma_{min} = \min_{\|\mathbf{x}\|_2 = 1} \left\| A\mathbf{x} \right\|_2,$$

it is easy to see that augmenting columns to A does not increase this minimum, and hence the inequalities $\varsigma_1 \geq \cdots \geq \varsigma_{k-1}$ follow. Q.E.D.

Lemma 2.3 *Let* m *be the degree of* $u(x) = GCD(p, p')$, *and*

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

with $deg(v) = k = n - m$. *Then*

(a) The normalized vector $\begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix}$ *is the right singular vector of* $S_k(p)$ *associated with the smallest singular value* ς_k*. Equivalently*

$$\varsigma_k = \left\| S_k(p) \begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix} \right\|_2 \cdot \left\| \begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix} \right\|_2^{-1} = \min_{\|\mathbf{y}\|_2=1} \left\| S_k(p)\mathbf{y} \right\|_2 = 0.$$

(b) When $\mathbf{v}$ *is known, the coefficient vector of* $u(x)$ *is the solution* $\mathbf{u}$ *of the linear system*

$$C_{m+1}(v)\mathbf{u} = \mathbf{p}$$

Proof. The vector

$$S_k(p) \begin{bmatrix} \mathbf{v} \\ -\mathbf{w} \end{bmatrix} = C_{k+1}\left(p'\right)\mathbf{v} - C_k(p)\mathbf{w}$$

is the coefficient vector of

$$p'(x)v(x) - p(x)w(x) \;=\; u(x)w(x)v(x) - u(x)v(x)w(x) \;\equiv\; 0,$$

yielding (a). The assertion (b) is a direct consequence of Lemma 2.1. Q.E.D.

Lemma 2.4 *Let* A *be a matrix, whose smallest two distinct singular values are* $\hat{\sigma} > \tilde{\sigma}$*. Write* $A = QR$*, where* Q *is unitary and* R *is upper triangular (i.e., its QR decomposition). From any vector* $\mathbf{x}_0$ *that is not orthogonal to the right singular subspace of* A *associated with* $\tilde{\sigma}$*, we generate the sequences* σ_i*,* $\mathbf{x}_i$*,* $i = 1, 2, \cdots$ *by the inverse iteration*

$$\begin{cases} \textit{Solve} & R^H\mathbf{y}_i = \mathbf{x}_{i-1} & \textit{for } \mathbf{y}_i \\ \textit{Solve} & R\,\mathbf{z}_i = \mathbf{y}_i & \textit{for } \mathbf{z}_i \\ \\ \textit{Calculate} & \mathbf{x}_i = \dfrac{\mathbf{z}_i}{\|\mathbf{z}_i\|_2}, \quad \sigma_i = \dfrac{\mathbf{y}_i}{\|\mathbf{z}_i\|_2} = \left\| R\mathbf{x}_i \right\|_2 & \\ & & i = 1, 2, \cdots. \end{cases} \tag{1}$$

Then there is a constant c *such that*

$$\left| \sigma_i - \tilde{\sigma} \right| \le \tau^i c, \quad \left\| A\mathbf{x}_i \right\|_2 = \tilde{\sigma} + O\left(\tau^i\right), \quad \textit{where } \tau = \left(\frac{\tilde{\sigma}}{\hat{\sigma}}\right)^2. \tag{2}$$

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

Proof. It is easy to verify that the process (1) is equivalent to the inverse iteration that calculates an eigenvector associated with the smallest eigenvalue of A^HA. Therefore, the theories of standard inverse iteration apply. For more general discussion on the inverse iteration in calculating singular subspaces, see [25]. Q.E.D.

2.3 The Gauss-Newton iteration

The Gauss-Newton iteration is an effective method for solving nonlinear least squares problems. Let $G : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ with $n > m$, and $\mathbf{a} \in \mathbf{C}^n$. The nonlinear system

$$G(\mathbf{z}) = \mathbf{a}, \quad \text{for} \quad \mathbf{z} \in \mathbf{C}^m$$

is overdetermined, and there is no conventional solution. We thereby seek a *weighted least squares solution*. Let $W = diag(\omega_1, \cdots, \omega_n)$ be a diagonal weight matrix with $\omega_j > 0$, $j = 1, \cdots n$. Let $\left\| \cdot \right\|_W$ denote the weighted 2-norm:

$$\left\| \mathbf{v} \right\|_W \equiv \left\| W\,\mathbf{v} \right\|_2 \equiv \sqrt{\sum_{j=1}^{n} \omega_j^2 v_j^2}, \quad \text{for all} \quad \mathbf{v} = (v_1, \cdots, v_n)^\top \in \mathbf{C}^n. \tag{3}$$

Our objective here is to solve the minimization problem

$$\min_{\mathbf{z}\in\mathbf{C}^m} \left\| G(\mathbf{z}) - \mathbf{a} \right\|_W^2 \equiv \min_{\mathbf{z}\in\mathbf{C}^m} \left\| W\Big(G(\mathbf{z}) - \mathbf{a}\Big) \right\|_2^2.$$

Lemma 2.5 *Let $F : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ whose components are analytic in every variable entry of $\mathbf{z}$. Let $\mathcal{J}(\mathbf{z})$ be the Jacobian of $F(\mathbf{z})$. If there is a neighborhood Ω of $\tilde{\mathbf{z}}$ in $\mathbf{C}^m$ such that*

$$\left\| F(\tilde{\mathbf{z}}) \right\|_2 \le \left\| F(\mathbf{z}) \right\|_2, \qquad \forall \mathbf{z} \in \Omega.$$

Then

$$\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = 0.$$

Proof. Let

$$F(\mathbf{z}) = \Big[\, f_1(\mathbf{z}), \cdots, f_n(\mathbf{z}) \Big]^\top, \quad \mathbf{z} = (\, z_1, \cdots z_m\,)^\top = \mathbf{x} + i\,\mathbf{y} \in \mathbf{C}^m, \quad i = \sqrt{-1};$$

$$\mathbf{x} = (x_1, \cdots, x_m)^\top \in \mathbf{R}^m, \;\; \mathbf{y} = (y_1, \cdots, y_m)^\top \in \mathbf{R}^m;$$

$$f_j(\mathbf{z}) = u_j(\mathbf{x}, \mathbf{y}) + i\, v_j(\mathbf{x}, \mathbf{y}), \quad j = 1, \cdots, n; \qquad g(\mathbf{x}, \mathbf{y}) = \frac{1}{2} F(\mathbf{z})^H F(\mathbf{z}) = \frac{1}{2} \sum_{j=1}^{n} (u_j^2 + v_j^2).$$

Since $\tilde{\mathbf{z}} = \tilde{\mathbf{x}} + i\tilde{\mathbf{y}}$ is a local minimum of $\left\| F(\mathbf{z}) \right\|_2^2 = F(\mathbf{z})^H F(\mathbf{z})$, we have

$$\frac{\partial g}{\partial x_k}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \frac{\partial g}{\partial y_k}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = 0, \quad k = 1, \cdots, m.$$

Namely, at $\tilde{\mathbf{z}} = \tilde{\mathbf{x}} + i\,\tilde{\mathbf{y}}$, using the Cauchy-Riemann equation, we have

$$\begin{aligned} 0 &= \sum_{j=1}^{n} \left[\left(u_j \frac{\partial u_j}{\partial x_k} + v_j \frac{\partial v_j}{\partial x_k} \right) + i \left(u_j \frac{\partial u_j}{\partial y_k} + v_j \frac{\partial v_j}{\partial y_k} \right) \right] = \sum_{j=1}^{n} \left[\left(\frac{\partial u_j}{\partial x_k} - i \frac{\partial v_j}{\partial x_k} \right) (u_j + i v_j) \right] \\ &= \sum_{j=1}^{n} \overline{\frac{\partial f_j}{\partial z_k}}\, f_j, \qquad k = 1, \cdots, m, \end{aligned}$$

which leads to $\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = 0$. Q.E.D.

By Lemma 2.5, let $J(\mathbf{z})$ be the Jacobian of $G(\mathbf{z})$. To find a local minimum of $\Big\| F(\mathbf{z}) \Big\|_2 \equiv \Big\| W\Big[G(\mathbf{z}) - \mathbf{a}\Big] \Big\|_2$ with $\mathcal{J}(\mathbf{z}) = WJ(\mathbf{z})$, we look for $\tilde{\mathbf{z}} \in \mathbf{C}^m$ such that

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

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

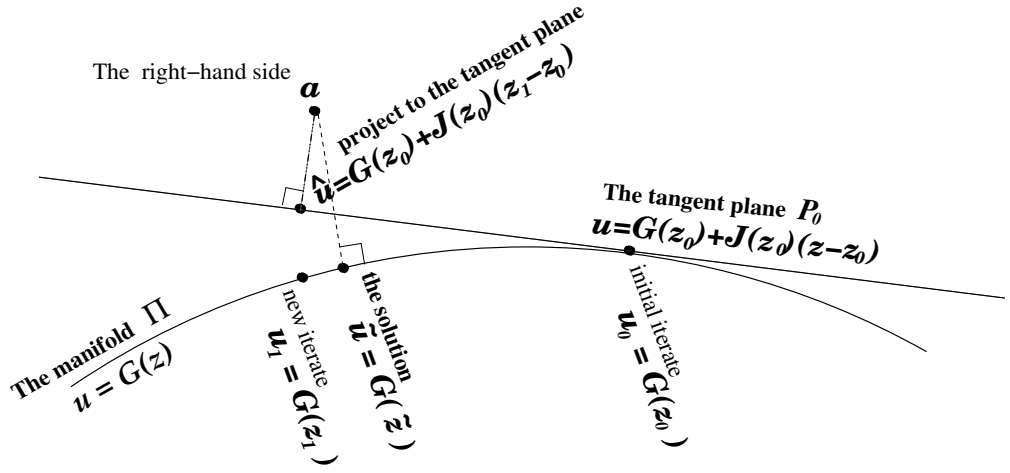


Figure 1: Illustration of the Gauss-Newton iteration

The Gauss-Newton iteration can be derived as follows. For the overdetermined nonlinear system $G(\mathbf{z}) = \mathbf{a}$, we look for $\tilde{\mathbf{z}}$ where $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$ is the orthogonal projection of the point $\mathbf{a}$ to Π. At $\mathbf{u}_0 = G(\mathbf{z}_0)$ in Π near $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$, we can approximate the manifold Π with the tangent plane $P_0 = \Big\{ G(\mathbf{z}_0) + J(\mathbf{z}_0)\,(\mathbf{z} - \mathbf{z}_0) \;\Big|\; \mathbf{z} \in \mathbf{C}^m \Big\}$. We then orthogonally project the point $\mathbf{a}$ to 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

$$G(\mathbf{z}_0) + J(\mathbf{z}_0)\,(\mathbf{z} - \mathbf{z}_0) = \mathbf{a} \quad \text{or} \quad J(\mathbf{z}_0)\,(\mathbf{z} - \mathbf{z}_0) = -[G(\mathbf{z}_0) - \mathbf{a}] \tag{4}$$

for its weighted least squares solution

$$\mathbf{z}_1 = \mathbf{z}_0 - \Big[J(\mathbf{z}_0)^+_W\Big]\,[G(\mathbf{z}_0) - \mathbf{a}]. \tag{5}$$

Here, $J(\mathbf{z}_0)^+_W$ is the weighted pseudo-inverse

$$J(\mathbf{z}_0)^+_W = \Big[J(\mathbf{z}_0)^H W^2 J(\mathbf{z}_0)\Big]^{-1} J(\mathbf{z}_0)^H W^2$$

of $J(\mathbf{z}_0)$, as long as $J(\mathbf{z}_0)^H W^2 J(\mathbf{z}_0)$ is invertible. Then $\mathbf{u}_1 = G(\mathbf{z}_1)$ is expected to be a better approximation of $\tilde{\mathbf{u}} = G(\tilde{\mathbf{z}})$ than $\mathbf{u}_0 = G(\mathbf{z}_0)$ (see Figure 1).

It is neither necessary nor desirable to calculate the pseudo-inverse $J(\mathbf{z}_0)^+_W$ explicitly. The weighted least squares solution to (4) is the least squares solution to the scaled linear system

$$W J(\mathbf{z}_0)\,(\mathbf{z}-\mathbf{z}_0) = -W\Big[\,G(\mathbf{z}_0)-\mathbf{a}\Big]$$

that can be solved by the standard linear least squares solvers. The Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \Big[J(\mathbf{z}_k)^+_W\Big]\,\Big[\,G(\mathbf{z}_k)-\mathbf{a}\Big], \quad k=0,1,\cdots$$

is then a recursive application of (5) (also see [5, 8]).

The convergence theory of the Gauss-Newton iteration has been well established for overdetermined systems in real spaces [8]. The following lemma is a straightforward generalization of Theorem 10.2.1 in [8] to complex spaces. Since the lemma itself as well as its proof are nearly identical to those in the real case as in [8], we shall present the lemma without proof.

Lemma 2.6 *Let $\Omega \subset \mathbf{C}^m$ be a bounded open convex set and $F : D \subset \mathbf{C}^m \longrightarrow \mathbf{C}^n$, 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*

$$\Big\|\,\Big[\,\mathcal{J}(\mathbf{z})-\mathcal{J}(\tilde{\mathbf{z}})\,\Big]^H F(\tilde{\mathbf{z}})\,\Big\|_2 \leq \delta\,\Big\|\,\mathbf{z}-\tilde{\mathbf{z}}\,\Big\|_2, \quad \forall \mathbf{z}\in\Omega. \tag{6}$$

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

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \mathcal{J}(\mathbf{z}_k)^+ F(\mathbf{z}_k), \quad k=0,1,\cdots, \quad \textit{where} \quad \mathcal{J}(\mathbf{z}_k)^+ = [\mathcal{J}(\mathbf{z}_k)^H\,\mathcal{J}(\mathbf{z}_k)]^{-1}\mathcal{J}(\mathbf{z}_k)^H,$$

is well defined inside Ω, converges to $\tilde{\mathbf{z}}$, and satisfies

$$\Big\|\mathbf{z}_{k+1}-\tilde{\mathbf{z}}\Big\|_2 \leq \frac{c\delta}{\sigma}\Big\|\mathbf{z}_k-\tilde{\mathbf{z}}\Big\|_2 + \frac{c\alpha\gamma}{2\sigma}\Big\|\mathbf{z}_k-\tilde{\mathbf{z}}\Big\|_2^2, \tag{7}$$

$$\Big\|\mathbf{z}_{k+1}-\tilde{\mathbf{z}}\Big\|_2 \leq \frac{c\delta+\sigma}{2\sigma}\Big\|\mathbf{z}_k-\tilde{\mathbf{z}}\Big\|_2 < \Big\|\mathbf{z}_k-\tilde{\mathbf{z}}\Big\|_2, \tag{8}$$

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

$$\Big\|\,\mathcal{J}(\mathbf{z}+\mathbf{h})-\mathcal{J}(\mathbf{z})\,\Big\|_2 \leq \gamma\,\Big\|\,\mathbf{h}\,\Big\|_2, \quad \forall \mathbf{z},\ \mathbf{z}+\mathbf{h}\ \in\ \Omega.$$

3 Algorithm I: root-finding with given multiplicities

In this section, we assume the multiplicity structure of a given polynomial is known. We shall deal with the problem of determining this multiplicity structure in the next section. The pejorative condition number will be introduced here to measure the sensitivity of multiple roots. When the pejorative condition number is moderate, the multiple roots can be calculated accurately by our Algorithm I.

3.1 The pejorative manifold

Introduced by Kahan [16], polynomials with roots in a given multiplicity structure form a *pejorative manifold*. More explicitely, a monic polynomial of degree n corresponds to a vector (or point) in $\mathbf{C}^n$

$$p(x) = x^n + a_1\, x^{n-1} + \cdots + a_{n-1}\, x + a_n \quad \sim \quad \mathbf{a} \;=\; (a_1, \cdots, a_n)^\top \in \mathbf{C}^n,$$

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

$$p(x) = \prod_{j=1}^{m} (x - z_j)^{\ell_j} \quad = \quad x^n + \sum_{j=1}^{n} g_j(z_1, \cdots, z_m)\; x^{n-j}. \tag{9}$$

where each $g_j(z_1, \cdots, z_m)$ is a polynomial in $z_1, \cdots, z_m$. This leads to the correspondence

$$p(x) \sim G_\ell(\mathbf{z}) \;\equiv\; \begin{pmatrix} g_1(z_1, \cdots, z_m) \\ \vdots \\ g_n(z_1, \cdots, z_m) \end{pmatrix} \in \mathbf{C}^n, \qquad \text{where} \quad \mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} \in \mathbf{C}^m. \tag{10}$$

Definition 3.1 *An ordered array of positive integers $\ell = [\ell_1, \cdots, \ell_m]$ 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 n-vectors*

$$\Pi_\ell \equiv \left\{ G_\ell(\mathbf{z}) \;\middle|\; \mathbf{z} \in \mathbf{C}^m \right\} \subset \mathbf{C}^n$$

is called the ***pejorative manifold*** *of multiplicity structure ℓ, where $G_\ell : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ defined in (9) - (10) is called the* ***coefficient operator*** *associated with the multiplicity structure ℓ.*

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

$$(x - z_1)(x - z_2)^2 = x^3 + (-z_1 - 2z_2)\, x^2 + (2z_1 z_2 + z_2^2)\, x + (-z_1 z_2^2),$$

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

$$G_{[1,2]}(\mathbf{z}) \equiv \begin{pmatrix} -z_1 - 2z_2 \\ 2z_1 z_2 + z_2^2 \\ -z_1 z_2^2 \end{pmatrix} \in \mathbf{C}^3, \quad \text{with} \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \in \mathbf{C}^2. \tag{11}$$

The vectors $G_{[1,2]}(\mathbf{z})$ in (11) for all $\mathbf{z} \in \mathbf{C}^2$ form the pejorative manifold $\Pi_{[1,2]}$. When $\ell = [3]$,

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

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

As a special case, the pejorative manifold $\Pi_{[1,1,\cdots,1]}$ is $\mathbf{C}^n$, representing the vector space of all polynomials with degree n.

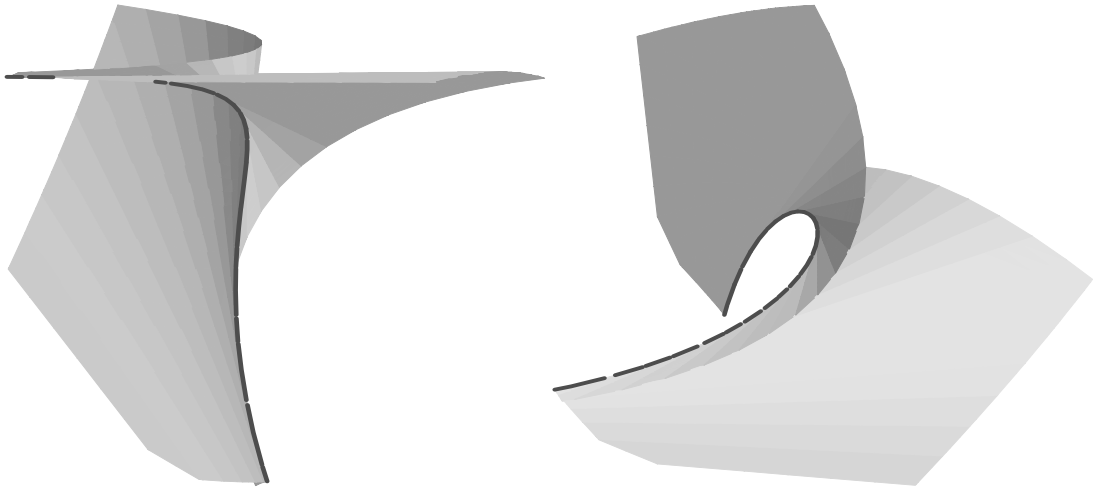


Figure 2: Pejorative manifolds of polynomials with degree 3 (view from two angles)

3.2 The nonlinear least squares problem

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

$$\left\{ \begin{array}{rcl} g_1(z_1, \cdots, z_m) & = & a_1 \\ g_2(z_1, \cdots, z_m) & = & a_2 \\ \vdots & \vdots & \vdots \\ g_n(z_1, \cdots, z_m) & = & a_n \end{array} \right. \qquad \text{or} \qquad G_\ell(\mathbf{z}) = \mathbf{a} \tag{12}$$

is overdetermined except for the plain structure $\ell = [1, 1, \cdots, 1]$. Usually, there are no conventional solutions. We thereby seek a *weighted least squares solution* to (12). Let $W = diag(\omega_1, \cdots, \omega_n)$ be a weight matrix with weights $\omega_j > 0$, $j = 1, \cdots n$. Let $\left\| \cdot \right\|_W$ denote the weighted 2-norm defined in (3). The objective is to solve the minimization problem

$$\min_{\mathbf{z} \in \mathbf{C}^m} \left\| G_\ell(\mathbf{z}) - \mathbf{a} \right\|_W^2 \equiv \min_{\mathbf{z} \in \mathbf{C}^m} \left\| W\Big(G_\ell(\mathbf{z}) - \mathbf{a} \Big) \right\|_2^2 \equiv \min_{\mathbf{z} \in \mathbf{C}^m} \left\{ \sum_{j=1}^{n} \omega_j^2 \Big| g_j(\mathbf{z}) - a_j \Big|^2 \right\}. \tag{13}$$

The residual $G_\ell(\mathbf{z}) - \mathbf{a}$ is a by-product of the computation. Its magnitude is also a measure of the backward error and is handily verifiable.

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

$$\omega_j = \min \left\{ 1, \ \frac{1}{|a_j|} \right\}, \quad j = 1, \cdots, n \tag{14}$$

lead to minimization of the relative backward error at every coefficient larger than one. All the numerical experiments for Algorithm I are conducted using the weights (14).

By Lemma 2.5, 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 $\tilde{\mathbf{z}} \in \mathbf{C}^m$ such that

$$\mathcal{J}(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = \Big[WJ(\tilde{\mathbf{z}})\Big]^H W\Big[G_\ell(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = J(\tilde{\mathbf{z}})^H W^2\Big[G_\ell(\tilde{\mathbf{z}}) - \mathbf{a}\Big] = 0. \tag{15}$$

Namely, $G_\ell(\tilde{\mathbf{z}}) - \mathbf{a}$ is orthogonal with respect to the inner product $\langle \mathbf{v}, \mathbf{w} \rangle \equiv \mathbf{v}^H W^2 \mathbf{w}$ to the tangent plane

$$\Big\{ \mathbf{u} = G_\ell(\tilde{\mathbf{z}}) + J(\tilde{\mathbf{z}})\,(\mathbf{z} - \tilde{\mathbf{z}}) \in \mathbf{C}^n \;\Big|\; \mathbf{z} \in \mathbf{C}^m \Big\}$$

of the pejorative manifold $\Pi_\ell = \Big\{ \mathbf{u} = G_\ell(\mathbf{z}) \;\Big|\; \mathbf{z} \in \mathbf{C}^m \Big\}$ at $\tilde{\mathbf{u}} = G_\ell(\tilde{\mathbf{z}}) \in \Pi_\ell$.

Definition 3.2 *Let $p(x) \sim \mathbf{a}$ be a monic polynomial of degree n. For any given multiplicity structure ℓ of the same degree, the vector $\tilde{\mathbf{z}}$ satisfying (15) is called a* ***pejorative root vector*** *or simply* ***pejorative root*** *of $p(x)$ corresponding to the multiplicity structure ℓ and weight W.*

The following theorem is the foundation of our algorithms. This theorem indicates that one may convert the singular problem of computing multiple roots with standard methods to a regular problem by seeking the least squares solution of (12).

Theorem 3.1 *Let $G_\ell : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ be the coefficient operator associated with a multiplicity structure $\ell = [\ell_1, \cdots, \ell_m]$. Then the Jacobian $J(\mathbf{z})$ of $G_\ell(\mathbf{z})$ is of full (column) rank if and only if the entries of $\mathbf{z} = (z_1, \cdots, z_m)^\top$ are distinct.*

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

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

If $c_1 J_1 + \cdots + c_m J_m = 0$ for constants $c_1, \cdots, c_m$, we let

$$q(x) = c_1\, q_1(x) + \cdots + c_m\, q_m(x).$$

Then,

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

is a zero polynomial implies $r(x) = \sum_{j=1}^{m} c_j \ell_j \left[\prod_{k \neq j} (x - z_k) \right] \equiv 0$. Therefore,

$$0 = r(z_j) = c_j \left[\ell_j \prod_{k \neq j} (z_j - z_k) \right], \quad j = 1, \cdots, m.$$

So, $c_j = 0$ for $j = 1, \cdots, m$ since ℓ_j's are positive and z_k's are distinct.

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

$$\begin{aligned} q_1(x) &= -\ell_1 \, (x - z_1)^{\ell_1 - 1} (x - z_2)^{\ell_2} \left[\prod_{k=3}^{m} (x - z_k)^{\ell_k} \right] \quad \text{and} \\ q_2(x) &= -\ell_2 \, (x - z_1)^{\ell_1} (x - z_2)^{\ell_2 - 1} \left[\prod_{k=3}^{m} (x - z_k)^{\ell_k} \right] \end{aligned}$$

as above. Since $z_1 = z_2$, these two polynomials differ by constant multiples ℓ_1 and ℓ_2. Therefore $J(\mathbf{z})$ is of (column) rank deficient. Q.E.D.

As a special case for the structure $\ell = [1, 1, \cdots, 1]$, equations in (12) form Viéte's system of n-variate polynomial system. Solving such a system via Newton's iteration is equivalent to the Weierstrass (Durand-Kerner) algorithm [20]. When a polynomial has multiple roots, Viéte's system becomes singular at the non-distinct root vector. Apparently, this singularity is the very reason that causes the ill-conditioning of the conventional root-finding: it is on a wrong pejorative manifold.

3.3 The Gauss-Newton iteration on the pejorative manifold

We apply the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left[J(\mathbf{z}_k)_W^+ \right] [G_\ell(\mathbf{z}_k) - \mathbf{a}], \quad k = 0, 1, \cdots \tag{16}$$

on Π_ℓ. When the pejorative root $\tilde{\mathbf{z}} = (\tilde{z}_1, \cdots, \tilde{z}_m)^\top \in \mathbf{C}^m$ has distinct components, the system (12) is nonsingular by Theorem 3.1, making the Gauss-Newton iteration well defined. Based on Lemma 2.6, we have the convergence theorem.

Theorem 3.2 *Let $\tilde{\mathbf{z}} = (\tilde{z}_1, \cdots, \tilde{z}_m)^\top \in \mathbf{C}^m$ be a pejorative root of $p(x) \sim \mathbf{a}$ associated with multiplicity structure ℓ and weight W. Assume that $\tilde{z}_1$, $\tilde{z}_2$, $\cdots$, $\tilde{z}_m$ are distinct. Then*

there are $\varepsilon, \epsilon > 0$ *such that, if* $\left\| \mathbf{a} - G_\ell(\tilde{\mathbf{z}}) \right\|_W < \varepsilon$ *and* $\left\| \mathbf{z}_0 - \tilde{\mathbf{z}} \right\|_2 < \epsilon$, *the iteration*

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \Big[J(\mathbf{z}_k)_w^+ \Big] \Big[G_\ell(\mathbf{z}_k) - \mathbf{a} \Big], \quad k = 0, 1, \cdots \tag{17}$$

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 $F(\mathbf{z}) = W\Big[G_\ell(\mathbf{z}) - \mathbf{a}\Big]$ and $\mathcal{J}(\mathbf{z})$ be its Jacobian. $F(\mathbf{z})$ is obviously analytic. From Theorem 3.1, 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 (6) holds with $\delta < \sigma^2$. Therefore all conditions of Lemma 2.6 are satisfied and there is a neighborhood of $\tilde{\mathbf{z}}$ such that, if $\mathbf{z}_0$ is in the neighborhood, the iteration (17) converges and satisfies (7). If in addition $\mathbf{a} = G_\ell(\tilde{\mathbf{z}})$, then $F(\tilde{\mathbf{z}}) = 0$ and thereby $\delta = 0$ in (6) and (7), and the convergence becomes quadratic. Q.E.D.

3.4 The pejorative condition number and inexact polynomials

There are many discussions on numerical condition of polynomial roots in the literature such as [6, 13, 16, 23, 26, 27]. In general, a condition number can be characterized as the smallest number satisfying

$$\Big[forward\_error\Big] \leq \Big[condition\_number\Big] \times \Big[backward\_error\Big] \tag{18}$$

For a polynomial with multiple roots, under *unrestricted* perturbation, the only condition number satisfying (18) is infinity. This fact can easily be seen from a simple perturbed polynomial

$$\tilde{p}(x) = p(x) - \varepsilon = (x-1)^2 - \varepsilon$$

where the double root $x = 1$ breaks into two simple roots $1 \pm \sqrt{\varepsilon}$. The forward error is $\sqrt{\varepsilon}$ and the backward error is ε. There is no finite number c that makes

$$\sqrt{\varepsilon} \leq c\,\varepsilon, \quad \forall \varepsilon > 0.$$

By changing the computational objective from solving a polynomial equation $p(x) = 0$ to the nonlinear least squares problem in the form of (13), the structure-altering noise is filtered out, and the perturbation is restricted to be *multiplicity preserving*. In such cases, the sensitivity of the roots can be analyzed differently.

Let's consider the pejorative root vector $\mathbf{z}$ of the polynomial $p(x) \sim \mathbf{a} = G_\ell(\mathbf{z})$. The polynomial $p(x)$ is perturbed to be $\hat{q}(x) \sim \hat{\mathbf{a}} = G_\ell(\hat{\mathbf{z}})$ with multiplicity structure ℓ preserved. Then

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

where $J(\mathbf{z})$ is the Jacobian of $G_\ell(\mathbf{z})$. Assuming the entries of $\mathbf{z}$ are distinct, by Theorem 3.1, $J(\mathbf{z})$ is of full rank. We have, using *h.o.t.* to stand for higher order terms,

$$\begin{aligned} \left\| W(\hat{\mathbf{a}} - \mathbf{a}) \right\|_2 &= \left\| [WJ(\mathbf{z})](\hat{\mathbf{z}} - \mathbf{z}) + h.o.t. \right\|_2, \\ \text{namely,} \qquad \left\| \hat{\mathbf{a}} - \mathbf{a} \right\|_W &\geq \sigma_{min} \left\| \hat{\mathbf{z}} - \mathbf{z} \right\|_2 + h.o.t., \\ \text{or} \qquad \left\| \hat{\mathbf{z}} - \mathbf{z} \right\|_2 &\leq \left(\frac{1}{\sigma_{min}} \right) \left\| \hat{\mathbf{a}} - \mathbf{a} \right\|_W + h.o.t. \end{aligned} \tag{19}$$

where $\sigma_{min} > 0$ is the smallest singular value of $WJ(\mathbf{z})$. The distance $\left\| \hat{\mathbf{z}} - \mathbf{z} \right\|_2$ is the forward error and the weighted distance $\left\| \hat{\mathbf{a}} - \mathbf{a} \right\|_W$ measures the backward error. The sensitivity of the root vector is thereby 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 3.3 *Let $p(x)$ be a polynomial and $\mathbf{z}$ be its pejorative root corresponding to multiplicity structure ℓ and weight W. Let G_ℓ be the coefficient operator associated with ℓ, J be its Jacobian, and σ_{min} be the smallest singular value of $WJ(\mathbf{z})$. Then the* ***pejorative condition number*** *of $\mathbf{z}$ is defined as*

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

We now estimate the error on pejorative roots of polynomials with inexact coefficients.

Theorem 3.3 *Let $\hat{\mathbf{b}} \sim \hat{p}(x)$ be an approximation to the polynomial $\mathbf{b} \sim p(x)$. Corresponding to the multiplicity structure ℓ and weight W, let $\mathbf{z}$ and $\hat{\mathbf{z}}$ be pejorative roots of $p(x)$ and $\hat{p}(x)$ respectively, with components of $\mathbf{z}$ being distinct. If $\left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W$ is sufficiently small, then*

$$\left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 \leq 2 \cdot \kappa_\ell(\mathbf{z}) \cdot \left(\left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W + \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W \right) + h.o.t. \tag{20}$$

where $\sigma_{min} > 0$ is the smallest singular value of $WJ(\mathbf{z})$.

Proof. From (19),

$$\begin{aligned} \left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 &\leq \frac{1}{\sigma_{min}} \left\| G_\ell(\mathbf{z}) - G_\ell(\hat{\mathbf{z}}) \right\|_W + h.o.t. \\ &\leq \frac{1}{\sigma_{min}} \left(\left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W + \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W + \left\| G_\ell(\hat{\mathbf{z}}) - \hat{\mathbf{b}} \right\|_W \right) + h.o.t. \end{aligned}$$

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

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

That proves the theorem. Q.E.D.

Corollary 3.1 *Under the condition of Theorem 3.3, if $\mathbf{z}$ is the exact root vector of $p(x)$ with multiplicity structure ℓ, then*

$$\left\| \mathbf{z} - \hat{\mathbf{z}} \right\|_2 \leq 2 \cdot \kappa_\ell(\mathbf{z}) \cdot \left\| \mathbf{b} - \hat{\mathbf{b}} \right\|_W + h.o.t. \tag{21}$$

Proof. Since $\mathbf{z}$ is exact, $\left\| G_\ell(\mathbf{z}) - \mathbf{b} \right\|_W = 0$ in (20). Q.E.D.

As a consequence, when an inexact polynomial approximates a polynomial that has exact multiple roots, then the pejorative root approximates the exact (multiple) roots. The (forward) root error is no more than being proportional to the (backward) coefficient error, while the proportionality constant is the doubled pejorative condition number.

The pejorative condition number has no obvious correlation with the magnitude of multiplicities. For example, consider polynomials

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

with different multiplicities $[\ell_1, \ell_2, \ell_3]$. For the weight W defined in (14), Figure 3 lists the pejorative 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 pejorative condition number is less than one. The pejorative condition theory here indicates that multiprecision arithmetic should *not* be a necessity. The "attainable accuracy" barrier appears to be highly questionable.

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

Figure 3: The pejorative condition number and root multiplicities

The pejorative condition number $\kappa_\ell(\mathbf{z})$ can be calculated with negligible cost. The Jacobian $J(\mathbf{z})$ and its QR decomposition are required by the Gauss-Newton iteration, and can be recycled to calculate $\kappa_\ell(\mathbf{z})$. The inverse iteration described in Lemma 2.4 is suitable for finding the smallest singular value. Since only the size, rather than the accurate digits, of the smallest singular value is required, a few inverse iteration steps are sufficient.

In §3.6 and §5, more examples will show that our iterative algorithm indeed reaches the accuracy permissible by the pejorative condition.

3.5 The numerical procedures

The iteration (16) requires evaluation of $G_\ell(\mathbf{z}_k)$ and $J(\mathbf{z}_k)$ at every iterative step, where the components of $G_\ell(\mathbf{z})$ are defined in (9) as coefficients of the polynomial

$$p(x) = (x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}. \tag{22}$$

While the explicit formulas for each $g_j(z_1, \cdots, z_m)$ and $\frac{\partial g_j}{\partial z_i}$ can be symbolically, inefficiently in general, computed using softwares like Maple, we list more efficient numerical procedures for computing $G_\ell(\mathbf{z})$ and $J(\mathbf{z})$ in Figure 4.

The polynomial multiplication is equivalent to the vector convolution (Lemma 2.1). The polynomial $p(x)$ in (22) can thereby be constructed from recursive convolution with vectors $(1, -z_j)^\top$. As a result, $G_\ell(\mathbf{z})$ is computed through the nested loops shown in Figure 4 as Algorithm EvalG. It takes $n^2 + O(n)$ floating point operations (additions and multiplications) to calculate $G_\ell(\mathbf{z})$.

The j-th column of the Jacobian $J(\mathbf{z})$, as shown in the proof of Theorem 3.1, can be considered

Algorithm EVALG:
input: m, n, $\mathbf{z} = (z_1, \cdots, z_m)^\top$,
$\ell = [\ell_1, \cdots, \ell_m]$
output: vector $G_\ell(\mathbf{z}) \in C^n$

$\mathbf{s} = (1)$
for $i = 1, 2, \cdots m$ do
for $l = 1, 2, \cdots, \ell_i$ do
$\mathbf{s} = conv\left(\mathbf{s}, (1, -z_i)^\top\right)$
end do
end do
$g_j(\mathbf{z})$ =$(j+1)$-th component of $\mathbf{s}$
$j = 1, \cdots, n$

Algorithm EVALJ:
input: m, n, $\mathbf{z} = (z_1, \cdots, z_m)^\top$,
$\ell = [\ell_1, \cdots, \ell_m]$
output: Jacobian matrix $J(\mathbf{z}) \in \mathbf{C}^{n \times m}$

$\mathbf{u} \sim \prod (x - z_j)^{\ell_j - 1}$ by EVALG
for $j = 1, 2, \cdots, m$ do
$\mathbf{s} = -\ell_j \, \mathbf{u}$
for $l = 1, \cdots, m$, $l \neq j$ do
$\mathbf{s} = conv\left(\mathbf{s}, (1, -z_l)^\top\right)$
end do
j-th column of $J(\mathbf{z}) = \mathbf{s}$
end do

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

as the coefficients of the following polynomial in x

$$-\ell_j \, (x - z_j)^{\ell_j - 1} \left[\prod_{i \neq j} (x - z_i)^{\ell_i} \right] = \left[\prod_{k=1}^{m} (x - z_k)^{\ell_k - 1} \right] \left[-\ell_j \prod_{i \neq j} (x - z_i) \right].$$

The cost of computing $J(\mathbf{z})$ is no more than $mn^2 + O(n)$ flops. The complete pseudo-code of the Algorithm I is shown in Figure 5.

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 recursive polynomial multiplication appears to be robust and stable from our experience on all the polynomials we have tested.

3.6 Numerical results for Algorithm I

A Matlab code PEJROOT is implemented for Algorithm I. All the testing are conducted with IEEE double precision (16 decimal digits). The testing polynomials are in general form.

3.6.1 The effect of "attainable accuracy"

Conventional methods, such as Farmer-Loizou methods [12], are subject to the "attainable accuracy" barrier. We made straightforward Matlab and Maple implementations of the Farmer-Loizou third order iteration

$$z_i^{(k+1)} = z_i^{(k)} - \left[1 - \left(\sum_{j \neq i} \frac{\ell_j}{z_i^{(k)} - z_j^{(k)}} \right) \left(\frac{p(z_i^{(k)})}{p'(z_i^{(k)})} \right) \right]^{-1} \left[\ell_i \frac{p(z_i^{(k)})}{p'(z_i^{(k)})} \right], \quad \begin{array}{l} i = 1, \cdots, m; \\ k = 1, 2, \cdots, \end{array}$$

and use the same example in that paper

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

Pseudo-code PEJROOT (Algorithm I):

input: m, n, $\mathbf{a} \in \mathbf{C}^n$, weight matrix W, initial iterate $\mathbf{z}_0$, multiplicity structure ℓ, error tolerance τ

output: Roots $\mathbf{z} = (z_1, \cdots, z_m)$, 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

The test uses standard IEEE double precision (16 digits). Both iterations start from $\mathbf{z}_0 = (1.1, 1.9, 3.1, 3.9)$. 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. On the other hand, our iteration smoothly converges to the roots and reaches accuracy of 14 digits. The "attainable accuracy" barrier has no effect on our algorithm. The iterations are shown below for three roots $x = 1, 2, 3$ with highest multiplicities

(three roots are shown, unimportant digits are truncated)

Farmer-Loizou third order iteration				PejRoot result			
steps	iterateas			steps	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	-11.0	1.6	3.0000001	7	1.00000000000000	2.0000000000002	2.999999999998
...	...			8	1.00000000000000	2.00000000000000	2.99999999999999
100	1.00000008	3.3	2.99999997				

The multiplicities of the Farmer-Loizou test problem are quite low. In the same problem, we increase the multiplicities 10 times as large:

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

The polynomial is constructed with 16-digit accuracy in coefficients. To make the comparison more clear, our method still uses the standard IEEE 16 digits arithmetic, while Farmer-Loizou method uses 1000-digit operations in Maple. Since the polynomial coefficients are inexact, conventional method fails even with multiprecision arithmetic, while PEJROOT still attains 14 correct digits of the four roots (Three roots iterations are shown below).

Farmer-Loizou third order iteration				PejRoot result			
steps	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	3.41	-.07	.88	5	1.00000000000000	2.00000000000001	2.99999999999997
6	11.55	1.16	.14	6.	1.00000000000000	2.00000000000001	2.99999999999998
7	3.51	-.96	.72	7.	1.00000000000000	2.00000000000000	2.99999999999999
...	...						
100	5.99	1.10	0.30				

The true barrier of accuracy for Algorithm I is the pejorative condition. Matlab constructed the test polynomial with a relative coefficient error of 4.56×10^{-16}, the pejorative 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 (21).

3.6.2 Clustered multiple roots

Consider $f(x) = (x - 0.9)^{18}(x - 1)^{10}(x - 1.1)^{16}$ in expanded form. 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, Algorithm I code PejRoot obtains all three roots for at least 14 correct digits each by taking two additional steps on the information of multiplicity structure and initial root approximation provided by our Algorithm II in §4.

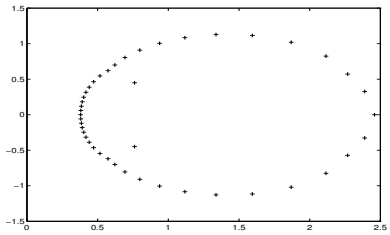


Figure 6: The root cluster from three multiple roots calculated by Matlab function `roots`

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 pejorative condition number is 60.4. Therefore, with perturbation at the sixteenth digit of the coefficients, 14 digits accuracy is the best possible accuracy that can be expected from any method.

An important feature of Algorithm I is that it does *not* require the correct multiplicity structure. Computation with different structures is often needed and is permissible with Algorithm I. If the computation is on a "wrong" pejorative manifold, then either the pejorative condition number or the backward error becomes large. Table 1 is a partial list of pejorative roots under different multiplicity structures. As shown in Table 1, if the computing objective is *unconstraint minimization* of the backward error, like standard methods, then we would get simple, clustered, and incorrect roots as shown in Figure 6. On the other hand, if the objective is to minimize backward error, subject to the condition requirement $\kappa < 100$, we would obtain accurate approximation of the desired roots.

3.6.3 Roots with huge multiplicities

The accuracy as well as stability of Algorithm I seem independent of the multiplicities of the roots. For instance, let's consider the polynomial of degree 1000

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

multiplicity structure	pejorative roots	backward error (relative)	pejorative condition number
[1,1,...,1]	(see Figure 9)	.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

Table 1: Partial list of multiple roots on different pejorative manifolds

The multiplicities of the roots are 100, 200, 300 and 400. These multiplicities are "huge"in the sense that numerical examples with multiplicities of 10 or more are rarely used in the literature.

In addition to such high multiplicities, we perturb the sixth digits of all coefficients of $g(x)$ by multiplying (1 ± 10^{-6}) on each one of them. Using any conventional approach, this perturbation will result in the total loss of forward accuracy even if multiprecision is used. The code PejRoot of Algorithm I takes a few seconds under Matlab to calculate all roots up to 7 digits accuracy. Taking the pejorative 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, see Figure 7).

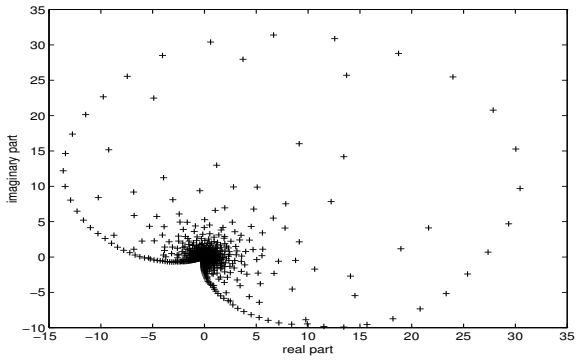


Figure 7: Result for the degree 1000 polynomial by Matlab function `roots`

z_1		z_2		z_3		z_4	
.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	+.70008i	.69996	+.50003i	.29996	+.40007i
.300005	+.600006	.099998	+.6999992i	.69999992	+.4999993i	.2999992	+.3999992i
.3000002	+.60000005i	.09999995	+.69999998i	.69999997	+.49999998i	.29999997	+.400000002i

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 much more preferred if obtainable. In this section, we present Algorithm II that calculates the multiplicity structure of a given polynomial as well as the initial root approximation for Algorithm I. As in Lemma 2.2, for $p(x) = p_0(x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}$ with z_j's being distinct, the key to calculating z_j's is to factor $p(x)$ and $p'(x)$ with a GCD

(Greatest Common Divisor) triple (u, v, w):

$$\begin{cases} u(x)\,v(x) &= p(x) \\ u(x)\,w(x) &= p'(x) \end{cases}, \quad v(x) \text{ and } w(x) \text{ are co-prime.} \tag{23}$$

If this can be accomplished, then we have

$$u(x) = u_0 \prod_{j=1}^{m} (x - z_j)^{\ell_j - 1}, \quad v(x) = \frac{p_0}{u_0}\,(x - z_1)\cdots(x - z_m).$$

The polynomial factor $v(x)$ has all the distinct roots of $p(x)$ as its simple roots that can be solved with standard root-finders. By applying the same strategy on $u(x) = GCD(p, p')$ recursively, the roots and the multiplicity structure can be calculated.

Numerical computation of GCD's has been studied extensively, such as in [3, 4, 11, 17, 22]. Our algorithm employs a partial singular value decomposition by an implicit inverse iteration (Lemma 2.4) and an iterative refinement by the Gauss-Newton iteration. As a remarkable consequence, the GCD's can be calculated accurately and efficiently without using symbolic computation or multiprecision.

4.1 Calculating the greatest common divisor

Algorithm II is based on the following GCD-finder:

STEP 1. Find the degree m of $GCD(p, p')$.
STEP 2. According to the degree m, set up the system (23) for the GCD triple (u, v, w).
STEP 3. Find an initial approximation to u, v and w for the GCD system (23).
STEP 4. Use the Gauss-Newton iteration to refine the GCD triple (u, v, w).

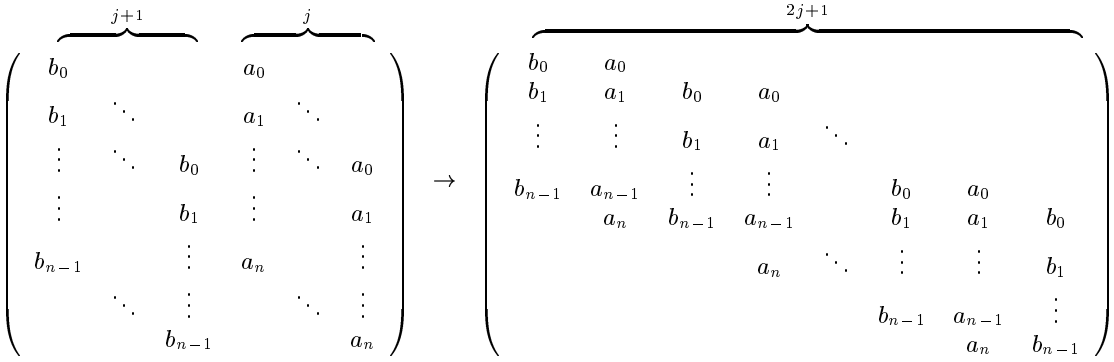
We shall describe each step in detail.

4.1.1 Finding the degrees of the GCD triple

Let $p(x)$ be a polynomial of degree n. By Lemma 2.2, the degree of $u(x) = GCD(p, p')$ is $m = n - k$ if and only if the k-th Sylvester discriminant matrix is the first rank-deficient matrix. Therefore, $m = deg(u)$ can be identified by calculating the sequence of the smallest singular values ς_j of $S_j(p)$, $j = 1, 2, \cdots$, until reaching ς_k that is zero *numerically*. Since only one singular pair is needed, the inverse iteration described in Lemma 2.4 is suitable for this purpose. Moreover, we can reduce the computing cost even further by recycling and updating the QR decomposition of $S_j(p)$'s along the way. More specifically, let

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

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



that the odd columns of $\hat{S}_j(p)$ consist of the coefficients of p', and even columns are the coefficients of p. Consequently, the matrix $\hat{S}_{j+1}(p)$ is formed simply by adding a zero row at bottom and augmenting two columns at right on $\hat{S}_j(p)$. It is easy to see that updating the QR decomposition of each $\hat{S}_j(p)$ requires only $O(n)$ additional flops. The inverse iteration (1) consists of a forward and a backward substitutions of triangular systems that requires $O(j^2)$ flops at each $S_j(p)$.

Let θ be a given *zero singular value threshold*. We shall discuss more about this number in §4.2. The process of finding the degree of the GCD triple (u, v, w) can be summarized as follows.

Calculate the QR decomposition of the $(n+1) \times 3$ matrix $\hat{S}_1(p) = Q_1 R_1$.
For $j = 1, 2, \cdots$ do
  Use the implicit inverse iteration (Lemma 2.4) to find the smallest
    singular value ς_j of $\hat{S}_j(p)$ and the corresponding right singular vector $\mathbf{y}_j$.
  if $\varsigma_j \leq \theta \left\| \mathbf{p} \right\|_2$, then
    $k = j$, $m = n - k$, extract $\mathbf{v}$ and $\mathbf{w}$ from $\mathbf{y}_j$, exit.
  else
    update $\hat{S}_j(p)$ to $\hat{S}_{j+1}(p) = Q_{j+1} R_{j+1}$
  end if
end do

4.1.2 The quadratic GCD system

Let $m = n - k$ be the degree of $GCD(p, p')$ calculated in `STEP 1`. We can now formulate the GCD system of `STEP 2`. This system can be written in vector form, with unknown vector $(\mathbf{u}, \mathbf{v}, \mathbf{w})^\top$:

$$F(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \begin{bmatrix} \mathbf{p} \\ \mathbf{p}' \end{bmatrix}, \quad \mathbf{u} \in \mathbf{C}^{m+1},\ \mathbf{v} \in \mathbf{C}^{n-m+1},\ \mathbf{w} \in \mathbf{C}^{n-m}, \tag{24}$$

$$\text{where} \quad F(\mathbf{u}, \mathbf{v}, \mathbf{w}) \equiv \begin{bmatrix} conv(\mathbf{u}, \mathbf{v}) \\ conv(\mathbf{u}, \mathbf{w}) \end{bmatrix}.$$

Here, the convolution $conv(\cdot, \cdot)$ is defined in Lemma 2.1. The following lemma ensures that solving this quadratic system is a nonsingular problem in finding the GCD triple (u, v, w).

Lemma 4.1 *The Jacobian of* $F(\mathbf{u}, \mathbf{v}, \mathbf{w})$ *in (24) is*

$$J(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \begin{bmatrix} C_{m+1}(v) & C_{k+1}(u) & 0 \\ C_{m+1}(w) & 0 & C_k(u) \end{bmatrix}. \tag{25}$$

If $u(x) = GCD(p, p')$ *with* $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ *satisfying (24), then* $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ *is of full (column) rank.*

Proof. It is straightforward to verify (25) using Lemma 2.1. This Jacobian is the generalized Sylvester resultant matrix for u, v, w (see [22]), that are co-prime since $u = GCD(p, p')$. By Proposition 3.1 in [22], it is of full rank. Q.E.D.

This nonsingularity also ensures that the Gauss-Newton iteration is locally convergent with quadratic rate.

Theorem 4.1 *Let* $\tilde{u}(x) = GCD(p, p')$ *with* $\tilde{v}(x)$ *and* $\tilde{w}(x)$ *satisfying (24), and* W *be a diagonal weight matrix. Then there exists* $\varepsilon > 0$ *such that for all* $\mathbf{u}_0$, $\mathbf{v}_0$, $\mathbf{w}_0$ *satisfying*

$$\left\| \mathbf{u}_0 - \tilde{\mathbf{u}} \right\|_2 < \varepsilon, \quad \left\| \mathbf{v}_0 - \tilde{\mathbf{v}} \right\|_2 < \varepsilon, \quad \left\| \mathbf{w}_0 - \tilde{\mathbf{w}} \right\|_2 < \varepsilon,$$

the Gauss-Newton iteration

$$\begin{bmatrix} \mathbf{u}_{j+1} \\ \mathbf{v}_{j+1} \\ \mathbf{w}_{j+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_j \\ \mathbf{v}_j \\ \mathbf{w}_j \end{bmatrix} - J(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j)_w^+ \left[F(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j) - \begin{pmatrix} \mathbf{p} \\ \mathbf{p}' \end{pmatrix} \right], \quad j = 0, 1, \cdots \tag{26}$$

converges to $[\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}]^\top$ *quadratically.*

Proof. $WF(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is obviously analytic. Since $(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ is a solution to (24), we have $WF(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}) = W(\mathbf{p}, \mathbf{p}')^\top$ and therefore $\delta = 0$ in (6) of Lemma 2.6. By Lemma 4.1, $WJ(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ is of full rank and its smallest singular value $\sigma > 0$. Let Ω be a neighborhood of $(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ in which

$$\left\| WJ(\mathbf{u}, \mathbf{v}, \mathbf{w}) \right\|_2 \leq \alpha, \quad \forall (\mathbf{u}, \mathbf{v}, \mathbf{w}) \in \overline{\Omega}.$$

Let γ be the Lipschitz constant of $WJ(\mathbf{u}, \mathbf{v}, \mathbf{w})$ over $\overline{\Omega}$. Then, by Lemma 2.6, we have

$$\left\| \begin{bmatrix} \mathbf{u}_{j+1} \\ \mathbf{v}_{j+1} \\ \mathbf{w}_{j+1} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{u}} \\ \tilde{\mathbf{v}} \\ \tilde{\mathbf{w}} \end{bmatrix} \right\|_2 \leq \frac{c\alpha\gamma}{2\sigma} \left\| \begin{bmatrix} \mathbf{u}_j \\ \mathbf{v}_j \\ \mathbf{w}_j \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{u}} \\ \tilde{\mathbf{v}} \\ \tilde{\mathbf{w}} \end{bmatrix} \right\|_2^2.$$

Q.E.D.

After finding the degree of $GCD(p, p')$, to employ the Gauss-Newton iteration to obtain the solution $[\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}]^\top$, we need an initial approximation $[\mathbf{u}_0, \mathbf{v}_0, \mathbf{w}_0]^\top$.

4.1.3 Setting up the initial iterate

In `STEP 1`, when the degree of $GCD(p, p')$ is calculated, the singular vector $\mathbf{y}_k$ of $\hat{S}_k(p)$ associated with the singular value ς_k consists of $\mathbf{v}$ and $\mathbf{w}$ that satisfy (24) (see Lemma 2.3). For refinement, we will use them as initial approximations $\mathbf{v}_0$ and $\mathbf{w}_0$. Because of the column rotation in §4.1.1, the odd and even entries of $\mathbf{y}_k$ form $\mathbf{v}_0$ and $\mathbf{w}_0$ respectively.

While polynomial long division yields $p(x) \div v(x) = u(x)$ in theory. The process may not be numerically stable especially with inexact data. By Lemma 2.3, long division is equivalent to solving the linear system

$$C_{m+1}(v)\mathbf{u}_0 = \mathbf{p} \tag{27}$$

for $\mathbf{u}_0$. When v and p are inexact, finding least squares solution to (27) is to minimize $\left\| conv(\mathbf{u}, \mathbf{v}) - \mathbf{p} \right\|_2$. This "least squares division" is consistently more accurate in our numerical experiment. The example shown in Table 2 is quite common, in which $\mathbf{p} = conv(\mathbf{u}, \mathbf{v})$ is rounded up at the eighth digit after decimal point. Substantial difference exists between long division (Matlab `deconv`) and least squares division.

approx. coef. of $p(x)$	coefficients of $v(x)$	coef.'s of $p(x) \div v(x)$	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				

Table 2: A numerical comparison between long division and least squares division

Extracting $\mathbf{v}_0$ and $\mathbf{w}_0$ from the singular vector, and solving (27) for $\mathbf{u}_0$, they will be used as the initial iterates for the Gauss-Newton iteration (26) that refines the GCD triple. Moreover, the linear system (27) is banded with the bandwidth being the number of distinct roots plus one. Therefore, the cost of solving (27) is insignificant (no more than $O(n^3)$) in the overall complexity.

4.1.4 Refining the GCD with the Gauss-Newton iteration

Each step of the Gauss-Newton iteration reduces the weighted residual

$$\left\| F(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j) - \begin{bmatrix} \mathbf{p} \\ \mathbf{p}' \end{bmatrix} \right\|_W = \left\| WF(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j) - W \begin{bmatrix} \mathbf{p} \\ \mathbf{p}' \end{bmatrix} \right\|_2 \tag{28}$$

until it is numerically unreducible. We stop the iteration when the residual no longer decreases.

The diagonal weight matrix W is used to scale the GCD system (24) so that the entries of $W \begin{bmatrix} \mathbf{p} \\ \mathbf{p}' \end{bmatrix}$ are of similar magnitude. Each step of the Gauss-Newton iteration requires solving

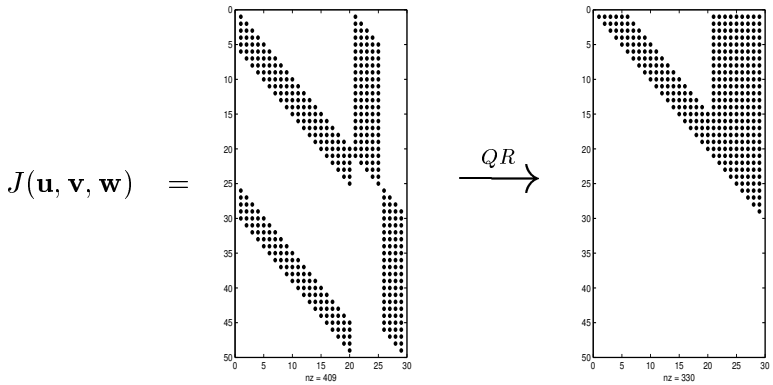


Figure 8: Sparsity of $J(\mathbf{u},\mathbf{v},\mathbf{w})$ and its triangularization

an overdetermined linear system

$$\left[\, WJ(\mathbf{u}_j,\mathbf{v}_j,\mathbf{w}_j)\,\right]\left(\begin{bmatrix}\mathbf{u}_{j+1}\\ \mathbf{v}_{j+1}\\ \mathbf{w}_{j+1}\end{bmatrix}-\begin{bmatrix}\mathbf{u}_j\\ \mathbf{v}_j\\ \mathbf{w}_j\end{bmatrix}\right)=W\left[F(\mathbf{u}_j,\mathbf{v}_j,\mathbf{w}_j)-\begin{pmatrix}\mathbf{p}\\ \mathbf{p}'\end{pmatrix}\right]$$

for its least squares solution, which requires a QR decomposition of the Jacobian $J(\mathbf{u}_j,\mathbf{v}_j,\mathbf{w}_j)$ and backward substitution for an upper triangular linear system. This Jacobian is a sparse matrix with a special structure. This sparsity can be largely preserved during the process. Figure 8 shows the typical sparsity of $J(\mathbf{u},\mathbf{v},\mathbf{w})$ and its triangularization. When $p(x)$ is a polynomial of degree n. A straightforward QR decomposition of $J(\mathbf{u},\mathbf{v},\mathbf{w})$ costs $O(n^3)$ flops. Taking the sparsity structure of $J(\mathbf{u},\mathbf{v},\mathbf{w})$ into account, it can be verified that the sparse QR decomposition costs $O(mk^2+m^2k+k^3)$ where k be the number of distinct roots and $m=n-k$. This complexity is significantly reduced to between $O(n^2)$ and $O(n^3)$.

4.2 Computing the multiplicity structure

From Lemma 2.3, if $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ satisfy the GCD system (24), then $v(x)$ has all simple roots that are identical to all the distinct roots of $p(x)$. On the other hand, this GCD calculation can be applied recursively to determine the multiplicity structure of $p(x)$:

$u_0(x)=p(x)$
for $m=1,2,\cdots s$, until $deg(u_s)=0$ do
  Find the GCD triple (u_m,v_m,w_m) that satisfies

$$\begin{cases} u_m(x)\,v_m(x) &= u_{m-1}(x)\\ u_m(x)\,w_m(x) &= u'_{m-1}(x)\end{cases},\quad v_m(x) \text{ and } w_m(x) \text{ are co-prime.}$$

  by using the the GCD finder given in §4.1.
end do

Let $d_m=deg(v_m)$, $m=1,2,\cdots,s$. To determine the multiplicity structure of $p(x)$, first of all, $p(x)$ has a total of $k=d_1$. It is straightforward to verify that the multiplicity structure is $\ell=[\ell_1,\cdots,\ell_k]$ with

$$\ell_j=\max\left\{\, t \,\middle|\, d_t\geq k-j+1\,\right\},\quad j=1,2,\cdots,k. \tag{29}$$

Moreover, an l-fold root of $p(x)$ appears l times as a simple root of $v_m(x)$, $m = 1, 2, \cdots, l$. Therefore, multiple root-finding on $p(x)$ is reduced to a sequence of simple root-finding on $v_j(x)$, $j = 1, 2, \cdots, s$. Since the the computation is carried out with inexact floating point operations, the computed roots of $v_j(x)$'s do not reach the optimal accuracy permitted by the pejorative condition of $p(x)$ in general. Using the multiplicity structure determined by (29), we may group the numerically "identical" roots of $\left\{ v_j(x) \right\}_1^s$ as the initial root approximation of $p(x)$.

In practice, the m-loop above stops either at $deg(u_m) = 0$ or $deg(v_m) = 1$. In latter case, there is only one root remaining in u_{m-1}. The structure is thereby determined with no need for further GCD calculation.

Each step in the m-loop above involves solving a GCD system

$$F_m(\mathbf{u}_m, \mathbf{v}_m, \mathbf{w}_m) \equiv \begin{bmatrix} conv(\mathbf{u}_m, \mathbf{v}_m) \\ conv(\mathbf{u}_m, \mathbf{w}_m) \end{bmatrix} = \begin{bmatrix} \mathbf{u}_{m-1} \\ \mathbf{u}'_{m-1} \end{bmatrix} \tag{30}$$

by the Gauss-Newton iteration

$$\begin{bmatrix} \mathbf{u}_m^{(j+1)} \\ \mathbf{v}_m^{(j+1)} \\ \mathbf{w}_m^{(j+1)} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_m^{(j)} \\ \mathbf{v}_m^{(j)} \\ \mathbf{w}_m^{(j)} \end{bmatrix} - J_m \left(\mathbf{u}_m^{(j)}, \mathbf{v}_m^{(j)}, \mathbf{w}_m^{(j)} \right)_w^+ \left[F_m \left(\mathbf{u}_m^{(j)}, \mathbf{v}_m^{(j)}, \mathbf{w}_m^{(j)} \right) - \begin{pmatrix} \mathbf{u}_{m-1} \\ \mathbf{u}'_{m-1} \end{pmatrix} \right], \tag{31}$$

$$j = 0, 1, \cdots$$

where $J_m \left(\mathbf{u}_m^{(j)}, \mathbf{v}_m^{(j)}, \mathbf{w}_m^{(j)} \right)$ is the Jacobian of $F_m \left(\mathbf{u}_m^{(j)}, \mathbf{v}_m^{(j)}, \mathbf{w}_m^{(j)} \right)$ and $J_m \left(\mathbf{u}_m^{(j)}, \mathbf{v}_m^{(j)}, \mathbf{w}_m^{(j)} \right)^+$ is its pseudo-inverse.

We use three control parameters this process. The first one is the *zero singular value threshold* θ that is used to identify the zero singular value. The default choice is $\theta = 10^{-8}$. In calculating u_m, the smallest singular value ς_l of $\hat{S}_l(u_{m-1})$ that is less than $\theta \left\| \mathbf{u}_{m-1} \right\|_2$, will be *tentatively* considered as a zero, pending confirmation from the residual information produced by the Gauss-Newton iteration.

When the smallest singular value is below the threshold, the Gauss-Newton iteration is initiated to further reduce the residual (28) to its numerical limit. We use the second control parameter, the *initial residual tolerance* ϱ, to decide if this refined residual is tiny enough. Our default choice is $\varrho = 10^{-10}$. When the (weighted) residual

$$\rho_m = \left\| F_m(\mathbf{u}_m, \mathbf{v}_m, \mathbf{w}_m) - \begin{bmatrix} \mathbf{u}_{m-1} \\ \mathbf{u}'_{m-1} \end{bmatrix} \right\|_W = \left\| W F_m(\mathbf{u}_m, \mathbf{v}_m, \mathbf{w}_m) - W \begin{bmatrix} \mathbf{u}_{m-1} \\ \mathbf{u}'_{m-1} \end{bmatrix} \right\|_2 \tag{32}$$

satisfies

$$\rho_m \leq \varrho \left\| \mathbf{u}_{m-1} \right\|_2,$$

we accept the GCD triple (u_m, v_m, w_m). Otherwise, the Sylvester's discriminant matrix is expanded to the next order and the partial singular value computation is continued.

The third parameter is the *residual tolerance growth factor* ϕ. Whenever a GCD triple (u_m, v_m, w_m) is calculated, there is a residual ρ_m defined in (32). The error in (u_m, v_m, w_m)

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 $\theta = 10^{-8}$, $\varrho = 10^{-10}$, $\phi = 100$)

output: the root estimates $(z_1, \cdots, z_k)^\top$ and multiplicity structure $[\ell_1, \cdots, \ell_k]$

```
Initialize u_0 = p
for m = 1, 2, ..., s, where 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 (1)
        if ς_l < θ ||u_{m-1}||_2 then
            set up the GCD system (30) according to l (see Section 4.1.2 )
            extract v_m^(0), w_m^(0) from y_l and calculate u_m^(0)  (see Section 4.1.3)
            apply the Gauss-Newton iteration (31) 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 (32)
        end if
    end do
    adjust the residual tolerance ϱ to be max{ϱ, φρ_j}
    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 simple roots z_j's of v_m(x), m = 1, 2, ..., s
    according to the multiplicities ℓ_j's.
```

Figure 9: Pseudo-code of Algorithm II

will cause the residual ρ_{m+1} of the next triple $(u_{m+1}, v_{m+1}, w_{m+1})$ to grow. Therefore, the residual tolerance ϱ may need to be adjusted after a GCD triple is found. Our default growth factor is 100. After obtaining ρ_m, the residual tolerance ϱ is adjusted to be

$$\max\left\{\ \varrho,\ \ \phi\,\rho_m\ \right\}.$$

Notice that the growth factor is applied to residual ρ_m rather than the residual tolerance ϱ to provide some breathing room. The residual tolerance ϱ itself may not grow at every step.

From our computing experience, the default control parameters works well for "normal" polynomials, such as those with unclustered roots of moderate multiplicities. For difficult problems, one may manual adjustment those parameters.

4.3 Numerical results for Algorithm II

The effectiveness of Algorithm II can be shown by the polynomial

$$p(x) = (x-1)^{20}(x-2)^{15}(x-3)^{10}(x-4)^5, \tag{33}$$

that is generated by Matlab function `poly`. Using the default control parameters, Algorithm II code GcdRoot correctly identifies the multiplicity structure. The roots are approximated

to an accuracy of 10 digits or more. Starting from this result as an input to Algorithm I code PejRoot, in the end we obtained all multiple roots with at least 14 digits correct (Table 4.3).

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

Table 3: Roots of $p(x)$ in (33) computed in two stages

This polynomial used to be extremely difficult by any standard for root-finding. The magnitude of its coefficients stretches from 1 to 10^{21}. Our algorithms have no difficulty finding all its multiple roots. To the best of our knowledge, there is no other method that can calculate multiple roots for such polynomials unless the machine precision is extended substantially according to the requirement of "attainable accuracy".

The Euclid method may also be used to find GCD in order to identify the multiplicities [2, 24]. F. Uhlig's `pzero` [24] is a Matlab implementation based on the Euclid method. The drawback of the Euclid method is its reliance on recursive long division that is numerically unstable (see Table 2). Moreover, it is not easy to identify a numerical zero remainder in the process. To identify multiplicities, `pzero` must rely on a root cluster matching that is inherently unreliable. Here we compare our code GcdRoot with `pzero` on the polynomial

$$p(x) = (x-1)^{4k}(x-2)^{3k}(x-3)^{2k}(x-4)^k.$$

for $k = 1, 2, \cdots, 8$. When the multiplicities increase, the root accuracy deteriorates with `pzero`. It successfully identifies the multiplicity structure for $k = 1$ and $k = 2$, but fails to do so after that. When $k > 5$, `pzero` no longer recognizes multiplicities, it only outputs simple clustered roots. In comparison, GcdRoot consistently attains a root accuracy of 11 or more digits with increasing multiplicities. The multiplicity structures are identified correctly for k up to 7. For the current implementation, the limitation of GcdRoot on this sequence is $k \leq 7$, although the root accuracy will stay the same for even larger k. When GcdRoot errs at $k = 8$, the multiplicities are up to 32, and the magnitude of coefficients stretches from 1 to 10^{35}.

4.4 Some remarks on the GCD calculation

In this paper, our main interest is in root-finding. We therefore confine our discussion in GCD-finding only to calculating $GCD(p, p')$. Actually, with minor modifications, the algorithm can be applied to numerical computation of general GCD's of two or more polynomials.

The idea of using singular values to calculate GCD is not new. Extensive studies have been reported [4, 11, 22]. In [3, 17], GCD refinement by using complicated nonlinear programming methods has also been proposed. In contrast, the Gauss-Newton iteration is simple and efficient, it seems well suitable for numerical GCD computation.

k	code	$x_1 = 1$		$x_2 = 2$		$x_3 = 3$	
$k = 1$	`pzero`	1.00000000001	(4)	1.99999999998	(3)	3.000000000005	(2)
	GcdRoot	0.999999999999990	(4)	1.99999999999998	(3)	3.0000000000005	(2)
$k = 2$	`pzero`	1.0000000001	(8)	2.000000002	(6)	3.000000004	(4)
	GcdRoot	0.9999999999998	(8)	1.999999999983	(6)	2.99999999991	(4)
$k = 3$	`pzero`	0.9999999897	(**13**)	1.99999990	(**8**)	2.9999998	(**5**)
	GcdRoot	0.9999999999997	(12)	1.99999999997	(9)	2.9999999998	(6)
$k = 4$	`pzero`	0.9999995	(**21**)	1.999994	(**6**)	2.999990	(**7**)
	GcdRoot	1.0000000000003	(16)	2.00000000002	(12)	3.0000000001	(8)
$k = 5$	`pzero`	1.0000009	(**28**)	2.00001	(**8**)	3.00002	(**6**)
	GcdRoot	1.0000000000004	(20)	2.00000000003	(15)	3.0000000002	(10)
$k = 6$	`pzero`	$----$	(**1**)	$----$	(**1**)	$----$	(**1**)
	GcdRoot	1.0000000000002	(24)	2.00000000001	(18)	3.00000000004	(12)
$k = 7$	`pzero`	$----$	(**1**)	$----$	(**1**)	$----$	(**1**)
	GcdRoot	1.0000000000001	(28)	2.00000000001	(21)	3.00000000006	(14)
$k = 8$	`pzero`	$----$	(**1**)	$----$	(**1**)	$----$	(**1**)
	GcdRoot	1.0000000000002	(**47**)	2.00000000002	(**15**)	3.00000000001	(**11**)

Table 4: Partial results on $p(x) = (x-1)^{4k}(x-2)^{3k}(x-3)^{2k}(x-4)^k$ and comparison between `pzero` and GcdRoot. Numbers in parenthesis are computed multiplicities. Wrong multiplicities are in boldface.

5 Numerical results for the combined method

5.1 The effect of inexact coefficients

In applications, input data of problems are expected to be inexact. The following experiment tests the effect of data error on the accuracy as well as stability of both Algorithm I and II.

For the polynomial

$$p(x) = \Big(x - 10/11\Big)^5 \Big(x - 20/11\Big)^3 \Big(x - 30/11\Big)^2$$

in general form, every coefficient is rounded up to k-digit accuracy, where $k = 10, 9, 8, \cdots$. For this sequence of problems, Algorithm II code GcdRoot correctly identifies the multiplicity structure if the coefficients has at least 7 digits. If the multiplicities are manually given, Algorithm I code PejRoot continues to converge even when data accuracy is down to 3 digits. For lower data accuracy, the residual tolerance ϱ in GcdRoot needs to be adjusted accordingly. Table 5 shows the results of both programs.

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 decreasing root gap $\varepsilon = 0.1,\ 0.01, \cdots$. Namely, the first root $x_1 = 0.9,\ 0.99,\ 0.999, \cdots$.

When root gap decreases, the default control parameters may not work properly. In this test, we use the default parameters for all cases except $\varepsilon = 0.0001$, in which case, 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 6.

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
$k = 6$	$----$	GcdRoot	$----$	$----$	$----$	$----$
		PejRoot	0.9090	1.8181	2.727	3.7e-06
$k = 5$	$----$	GcdRoot	$----$	$----$	$----$	$----$
		PejRoot	0.909	1.818	2.72	2.4e-05
$k = 4$	$----$	GcdRoot	$----$	$----$	$----$	$----$
		PejRoot	0.90	1.81	2.7	1.9e-04
$k = 3$	$----$	GcdRoot	$----$	$----$	$----$	$----$
		PejRoot	0.9	1.8	2.8	1.8e-03

Table 5: Effect of coefficient error on computed roots

root gap ε	code	$x_1 = 1 - \varepsilon$	$x_2 = 1$	$x_3 = -0.5$	backward error	pejorative condition number
$\varepsilon = 0.1$	GcdRoot	0.89999999999	0.99999999999	-0.49999999999999	9.7e-10	
	PejRoot	0.9000000000000	0.9999999999999	-0.50000000000000	2.7e-13	.7
$\varepsilon = 0.01$	GcdRoot	0.98999999	0.99999999	-0.50000000000000	3.2e-07	
	PejRoot	0.989999999999	1.000000000000	-0.49999999999999	1.0e-12	6.7
$\varepsilon = 0.001$	GcdRoot	0.99900	1.00000	-0.49999999999999	1.9e-04	
	PejRoot	0.99899999999	1.00000000000	-0.500000000000000	4.1e-13	62.5
$\varepsilon = 0.0001$	GcdRoot	0.9997	0.99996	-0.4999999999999	1.1e-02	
	PejRoot	0.999900000	0.999999999	-0.50000000000000	4.0e-12	621.7
$\varepsilon = 0.00001$	PejRoot	0.999989990	1.0000000	-0.50000000000000	4.0e-10	5791.8

Table 6: Effect of decreasing root gap on computed roots

root gap ε	code	$x_1 = 1 - \varepsilon$	$x_2 = 1$	$x_3 = -0.5$	backward error	pejorative condition number
$\varepsilon = 0.0001$	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

Table 7: If the control parameter is not adjusted, tiny root gap makes computed roots identical. However, from the backward errors and pejorative condition, they are not necessarily wrong answers.

When the default growth factor stays the same as the default $\phi = 100$ and the gap $\varepsilon \leq 0.0001$, GcdRoot outputs a multiplicity structure $[40, 5]$. Namely, GcdRoot treats the two nearby 20-fold roots as a single 40-fold one. From the computed backward error and the pejorative condition, this may not necessarily be incorrect. See Table 7. When backward error becomes 10^{-12} and pejorative 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 6).

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

5.3 A large inexact problem

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 as roots:

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

These roots are used to generate polynomial $f(x)$ of degree 20. We then round all coefficients to 10 decimal digits. The coefficients are shown at the right side. We construct multiple roots by squaring $f(x)$ again and again. Namely,

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

At $k = 5$, $g_5(x)$ 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-digit. Here is the final result.

```
coefficients of f(x)
    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
```

```
 THE PEJORATIVE 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
```

Acknowledgment. The author is grateful for the e-correspondence with Frank Uhlig on the subject. Moreover, the author wishes to thank Frank Uhlig and Peter Kravanja for freely sharing their and codes.

References

[1] D. H. Bailey, *A Fortran-90 based multiprecision system*, ACM Trans. Math. Software, 21 (1995), pp. 379–387.

[2] L. Brugnanao and D. Trigiante, *Polynomial roots: the ultimate answer?*, Linear Alg. and Its Appl., 225 (1995), pp. 207–219.

[3] P. Chin, R. M. Corless, and G. F. Corless, *Optimization strategies for the approximate GCD problem*, in ISSAC 1998, New York, 1998, ACM Press, pp. 228–235.

[4] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt, *The singular value decomposition for polynomial systems*, in Proc. ISSAC 1995, New York, 1995, ACM Press, pp. 195–207.

[5] J.-P. Dedieu and M. Shub, *Newton's method for over-determined system of equations*, Math. Comp., 69 (1999), pp. 1099–1115.

[6] J. W. Demmel, *On condition numbers and the distance to the nearest ill-posed problem*, Numer. Math., 51 (1987), pp. 251–289.

[7] 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.

[8] 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.

[9] 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.

[10] ——, *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.

[11] I. Z. Emiris, A. Galligo, and H. Lombardi, *Certified approximate univariate GCDs*, J. Pure Appl. Algebra, 117/118 (1997), pp. 229–251.

[12] 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.

[13] 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.

[14] 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.

[15] A. I. Iliev, *A generalization of Ehrlich-Kjurkchiev method for multiple roots of algebraic equations*, Serdica Math. J., 24 (2000), pp. 215–224.

[16] W. Kahan, *Conserving confluence curbs ill-condition.* Technical Report 6, Computer Science, University of California, Berkeley, 1972.

[17] N. K. Karmarkar and Y. N. Lakshman, *On approximate polynomial greatest common divisors*, J. Symb. Comput., 26 (1998), pp. 653–666.

[18] P. Kravanja and M. Van Barel, *Computing Zeros of Analytic Functions, Lecture Notes in Mathematics, 1727*, Springer-Verlag, 2000.

[19] 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.

[20] V. Y. Pan, *Solving polynomial equations: some history and recent progress*, SIAM Review, 39 (1997), pp. 187–220.

[21] QNT Software Development Inc., *A variable- and mixed-precision linear algebra library for Fortran-77 and other languages.* QNT Software Development Inc, User Guide and Reference Manual, `http://www.quasinewtonian.com`, 2002.

[22] D. Rupprecht, *An algorithm for computing certified approximate GCD of n univariate polynomials*, J. Pure and Appl. Alg., 139 (1999), pp. 255–284.

[23] M. Shub and S. Smale, *Complexity of Bezout's Theorem III. Condition number and packing*, J. of Complexity, 9 (1993), pp. 4–14.

[24] 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.

[25] 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.

[26] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, N.J., 1963.

[27] J. R. Winkler, *Condition numbers of a nearly singular simple root of a polynomial*, Appl. Numer. Math., (2001), pp. 275–285.

[28] T. J. Ypma, *Finding a multiple zero by transformations and Newton-like methods*, SIAM Review, 25 (1983), pp. 365–378.

[29] Z. Zeng, *On ill-conditioned eigenvalues, multiple roots of polynomials, and their accurate computation.* MSRI Preprint No. 1998-048, (1998) [2].

Zhonggang Zeng
Department of Mathematics
Northeastern Illinois University
Chicago, IL 60625
email: `zzeng@neiu.edu`

[2] `http://www.neiu.edu/~zzeng/Papers/multiple.ps`