A Method Computing Multiple Roots of Inexact Polynomials *

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

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 nonsingular least squares problem on a pejorative manifold, and calculates multiple roots simultaneously. To fulfill the input requirement of Algorithm I, we design a numerical GCD-finder, including a partial singular value decomposition and an iterative refinement, as the main engine for 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 are presented. The error analysis and numerical results confirm that a polynomial being ill-conditioned in conventional sense can be well conditioned pejoratively. In those cases, the multiple roots can be computed with remarkable accuracy.

Categories and Subject Descriptors

G.1.5 [**Mathematics of Computing**]: Roots of Nonlinear Equations – Iterative methods; methods for polynomials; G.4 [**Mathematics of Computing**]: Mathematical Software – Algorithm design and analysis

General Terms

Polynomial root-finding

*This is an extended abstract of a preliminary report. Detailed results and updates are available at `http://www.neiu.edu/~zzeng`

ISSAC'03, August 3–6, 2003, Philadelphia, Pennsylvania, USA

1. INTRODUCTION

In this paper, we present a combination of two numerical algorithms that accurately calculate multiple roots of polynomials with coefficients possibly being inexact, without using multiprecision arithmetic. As a result, a blackbox-type root-finder MultRoot is implemented as a Matlab package.

Polynomial root-finding is one of the classical problems with longest and richest history. It remains a challenge today. 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 [12]. If polynomial coefficients are truncated, multiple roots would turn into clusters. In that case, extending machine precision beyond IEEE standard would have little effect. Without accurate methods that are independent of multiprecision technology, multiple roots of perturbed polynomials would indeed be intractable.

Although many numerical analysts consider multiple roots highly ill-conditioned in numerical computation, W. Kahan [10] pointed that this is a misconception, and proved that if the multiplicities are preserved on a proper pejorative manifold, the multiple roots can actually be well behaved.

In light of Kahan's theory, we propose our 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, structure altering perturbations are filtered out, and the computation remains on the manifold. As a result, all the roots, regardless of the multiplicities, are calculated simultaneously and accurately.

Algorithm I requires prior knowledge of the multiplicity structure and initial root estimates. To fulfill this input requirement, we employ a numerical GCD-finder, including a partial singular value decomposition of the Sylvester discriminant matrices, and an iterative refinement strategy for the recursive GCD computation. The GCD-finder here can easily be adapted for general GCD calculation of univariate polynomials. The resulting Algorithm II calculates the multiplicity structure and the initial root approximation for a given polynomial.

In §3.3, 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" [9, 12].

In §3.4 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 are implemented in Matlab and are available from the author upon request.

The combined algorithm is efficient, accurate and stable. It requires the polynomial coefficients as the *only* input, and outputs the roots, multiplicities, the backward error, the estimated forward error, and the pejorative condition number. 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 combined code accurately identifies the multiplicity structure and multiple roots for polynomials with coefficients accuracy as low as 7 digits. With given multiplicities, Algorithm I needs even lower data accuracy such as 3 decimal digits. To the best of our knowledge, this is the first blackbox-type root-finders with such capability.

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 $(\cdot)$. If 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 vector $\mathbf{p} = (\, p_0, p_1, \cdots, p_n \,)^\top$ unless defined otherwise.

2.2 Basic definitions and lemmas

DEFINITION 2.1. *Let* $p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_n$ *with* $p_0 \neq 0$. *For any integer* $k > 0$, *the* $(n+k) \times k$ *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.

DEFINITION 2.2. *Let* $p(x)$ *be a polynomial of degree* n *and* $p'(x)$ *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 2.1. *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 $u(x)$ *is* $deg(u) = m$.

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

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

LEMMA 2.2. *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.

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

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

LEMMA 2.3. *Let* A *be a matrix with a QR decomposition* $A = QR$, *whose smallest two distinct singular values are* $\hat{\sigma} > \tilde{\sigma}$. *From any vector* $\mathbf{x}_0$ *that is not orthogonal to the right singular subspace of* A *associated with* $\tilde{\sigma}$, *let* σ_j, $\mathbf{x}_j$, $j = 1, 2, \cdots$ *be sequences generated by the inverse iteration*

$$\begin{cases} \textit{Solve} & R^H \mathbf{y}_j = \mathbf{x}_{j-1} & \textit{for } \mathbf{y}_j \\ \textit{Solve} & R\,\mathbf{z}_j = \mathbf{y}_j & \textit{for } \mathbf{z}_j \\ \textit{Calculate} & \mathbf{x}_j = \dfrac{\mathbf{z}_j}{\|\mathbf{z}_j\|_2}, & \sigma_j = \dfrac{\mathbf{y}_j}{\|\mathbf{z}_j\|_2} \\ & & j = 1, 2, \cdots \end{cases} \tag{1}$$

Then there is a constant c *such that*

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

2.3 The Gauss-Newton iteration

The Gauss-Newton iteration [3, 5] is an effective method for the nonlinear least squares problem. Let $G : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ with $n > m$. The nonlinear system

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

is overdetermined, and there is no conventional solution. We thereby seek a *least squares solution*. The objective is to solve the minimization problem

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

With $J(\mathbf{z})$ be the Jacobian of $G(\mathbf{z}) - \mathbf{a}$, this minimum occurs at $\tilde{\mathbf{z}}$ where

$$J(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = J(\tilde{\mathbf{z}})^H \left[G(\tilde{\mathbf{z}}) - \mathbf{a} \right] = 0.$$

If $J(\tilde{\mathbf{z}})$ is of full rank, and the residual $\left\| G(\tilde{\mathbf{z}}) - \mathbf{a} \right\|_2$ is sufficiently small, then the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left[J(\mathbf{z}_k)^+ \right] [G(\mathbf{z}_k) - \mathbf{a}]. \tag{3}$$

converges locally to $\tilde{\mathbf{z}}$. Here, $J(\mathbf{z}_0)^+$ is the pseudo-inverse of $J(\mathbf{z}_0)$, with

$$J(\mathbf{z}_0)^+ = \left[J(\mathbf{z}_0)^H J(\mathbf{z}_0) \right]^{-1} J(\mathbf{z}_0)^H$$

3. ALGORITHM I: ROOT-FINDING WITH KNOWN MULTIPLICITIES

3.1 The pejorative manifold

Introduced by Kahan, polynomials with roots in certain multiplicity structure form a pejorative manifold. Here we describe Kahan's concept in an explicit and precise way.

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

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

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

$$p(x) = \prod_{j=1}^{m} (x - z_j)^{\ell_j} = x^n + \sum_{j=1}^{n} g_j(z_1, \cdots, z_m)\, x^{n-j}. \quad (4)$$

Each $g_j(z_1, \cdots, z_m)$ is a polynomial in $z_1, \cdots, z_m$ and we have the correspondence

$$p(x) \sim G_\ell(\mathbf{z}) \equiv \begin{pmatrix} g_1(\mathbf{z}) \\ \vdots \\ g_n(\mathbf{z}) \end{pmatrix} \in \mathbf{C}^n, \text{ where } \mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} \in \mathbf{C}^m. \quad (5)$$

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 (4) – (5) is called the **coefficient operator** associated with the multiplicity structure ℓ.*

Figure 1 shows pejorative manifold $\Pi_{[1,2]}$ (the wings) and $\Pi_{[3]}$ (the sharp edge) in $\mathbf{R}^3$.

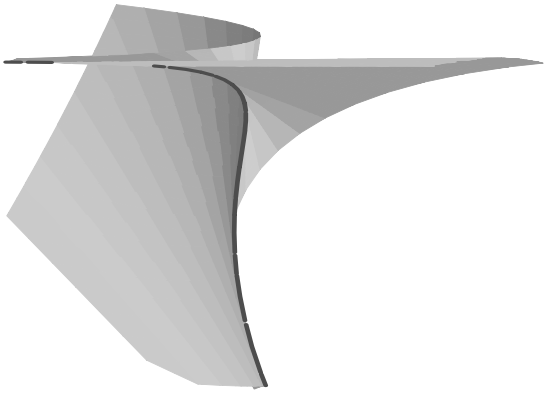

Figure 1: Pejorative manifolds of polynomials with degree 3

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 the root-finding for $p(x)$ is equivalent to solving the overdetermined polynomial system

$$G_\ell(\mathbf{z}) = \mathbf{a} \quad (6)$$

for its least squares solution. Let $J(\mathbf{z})$ be the Jacobian of $G_\ell(\mathbf{z})$. In order to find a local minimum point of $\left\| F(\mathbf{z}) \right\|_2 \equiv \left\| G_\ell(\mathbf{z}) - \mathbf{a} \right\|_2$, with $J(\mathbf{z})$ being the Jacobian of $F(\mathbf{z})$, we look for the point $\tilde{\mathbf{z}} \in \mathbf{C}^m$ such that

$$J(\tilde{\mathbf{z}})^H F(\tilde{\mathbf{z}}) = J(\tilde{\mathbf{z}})^H \left[G_\ell(\tilde{\mathbf{z}}) - \mathbf{a} \right] = 0. \quad (7)$$

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 (7) is called a **pejorative root vector** or simply **pejorative root** of $p(x)$ corresponding to the multiplicity structure ℓ.*

Computing multiple roots with standard methods is a singular problem. The following theorem shows that by seeking the least squares solution of (6), the problem becomes nonsingular.

THEOREM 3.1. *Let $G_\ell : \mathbf{C}^m \longrightarrow \mathbf{C}^n$ be the coefficient operator associated with a multiplicity structure ℓ. 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.*

We apply the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left[J(\mathbf{z}_k)^+ \right] [G_\ell(\mathbf{z}_k) - \mathbf{a}], \quad k = 0, 1, \cdots \quad (8)$$

on Π_ℓ. Theorem 3.1 also ensures that the Gauss-Newton iteration is well defined and locally convergent.

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. Assume that $\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_m$ are distinct. Then there are $\varepsilon, \epsilon > 0$ such that, if $\left\| \mathbf{a} - G_\ell(\tilde{\mathbf{z}}) \right\|_2 < \varepsilon$ and $\left\| \mathbf{z}_0 - \tilde{\mathbf{z}} \right\|_2 < \epsilon$, the Gauss-Newton iteration (8) is well defined and converges to the pejorative root $\tilde{\mathbf{z}}$ with at least a linear rate. If, in addition to the above assumptions, we have $\mathbf{a} = G_\ell(\tilde{\mathbf{z}})$, then the convergence is quadratic.*

The main cost of the iteration (8) is the evaluation of $G_\ell(\mathbf{z})$ and $J(\mathbf{z})$ at every iterative step, where the components of $G_\ell(\mathbf{z})$ are defined in (4) as coefficients of the polynomial

$$p(x) = (x - z_1)^{\ell_1} \cdots (x - z_m)^{\ell_m}. \quad (9)$$

It is important to employ efficient numerical procedures to compute $G_\ell(\mathbf{z})$ and $J(\mathbf{z})$.

Polynomial multiplication is equivalent to vector convolution. Therefore, the components of $G_\ell(\mathbf{z})$, namely coefficients of $p(x)$ in (9)), can be calculated by recursive convolution of $(1, z_j)^\top$, $j = 1, \cdots, m$.

It is easy to verify that the j-th column of $J(\mathbf{z})$ is the

coefficient vector of

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

that can also be calculated using recursive convolution.

It takes $n^2+O(n)$ flops (additions and multiplications) to calculate $G_\ell(\mathbf{z})$, and no more than $mn^2+O(n)$ for $J(\mathbf{z}_k)$.

3.3 The pejorative condition number and inexact polynomials

There are many insightful discussions on the numerical condition of polynomial roots, such as [4, 8, 14]. A condition number, by its concept, can be characterized as the smallest number κ that satisfies

$$\Big[forward\_error\Big] \le \kappa\,\Big[backward\_error\Big], \qquad (10)$$

ignoring the higher order terms. For a polynomial with multiple roots, under *unrestricted* perturbation, the only condition number satisfying (10) is infinity.

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

$$\hat{\mathbf{a}}-\mathbf{a}\ =G_\ell(\hat{\mathbf{z}})-G_\ell(\mathbf{z})=J(\mathbf{z})(\hat{\mathbf{z}}-\mathbf{z})+O\left(\left\|\,\hat{\mathbf{z}}-\mathbf{z}\,\right\|_2^2\right)$$

where $J(\mathbf{z})$ is the Jacobian of $G_\ell(\mathbf{z})$. Assuming that the entries of $\mathbf{z}$ are distinct, $J(\mathbf{z})$ is of full rank (Theorem 3.1). Ignoring higher order terms (*h.o.t.*), we have

$$\begin{aligned}\left\|\,\hat{\mathbf{a}}-\mathbf{a}\,\right\|_2 &= \left\|\,J(\mathbf{z})(\hat{\mathbf{z}}-\mathbf{z})+h.o.t.\,\right\|_2,\quad \text{namely,}\\ \left\|\,\hat{\mathbf{a}}-\mathbf{a}\,\right\|_2 &\ge \sigma_{min}\left\|\,\hat{\mathbf{z}}-\mathbf{z}\,\right\|_2+h.o.t.,\quad \text{or}\\ \left\|\,\hat{\mathbf{z}}-\mathbf{z}\,\right\|_2 &\le \left(\frac{1}{\sigma_{min}}\right)\left\|\,\hat{\mathbf{a}}-\mathbf{a}\,\right\|_2+h.o.t. \qquad (11)\end{aligned}$$

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

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

There is no obvious correlation between the pejorative condition number and the 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 $[\ell_1,\ell_2,\ell_3]$. Table 1 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 $\kappa_\ell<1$. The roots are well conditioned pejoratively.

multiplicities			pejorative 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

Table 1: The pejorative condition number and root multiplicities

3.4 Numerical results for Algorithm I

For Algorithm I described above, a Matlab code PEJROOT is implemented. If the multiplicity structure is known and an initial root approximation is given, the algorithm calculates the roots accurately, even if the polynomial is inexact with huge multiplicities.

Conventional methods are subject to the "attainable accuracy" barrier [9, 12], such as the modified Newton iteration and Farmer-Loizou methods [7]. For a simple polynomial

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

the "attainable accuracy" of the roots are 4, 5, 8, 16 digits respectively using standard IEEE double precision (16 digits). The Farmer-Loizou method produces iterates that bounce around the roots. PEJROOT smoothly converges to the roots and reaches accuracy of 14 digits. The table below compares the convergence pattern of the two algorithms.

```
    Farmer-Loizou third order iteration
steps  iterateas (unimportant digits are truncated)
  1      1.0009          1.998           3.001
  2      0.99997         1.9999992       3.000000008
  3      0.01            3.4             2.9988
  4      0.8             2.3             3.000007
  5      0.998           2.007           3.0000001
  6      1.0000007       2.00000007      2.99996
  7    -11.0             1.6             3.0000001
 ...   ...
100      1.00000008      3.3             2.99999997

         PejRoot result
 steps  iterates (unimportant digits are truncated)
  1    1.03              1.8             3.4
  2    0.997             1.98            2.6
  3    1.00009           2.05            2.8
  4    0.99994           1.994           2.98
  5    1.000003          2.0001          2.9990
  6    0.999999997       2.000000005     2.9999990
  7    1.00000000000000  2.00000000000002 2.999999999998
  8    1.00000000000000  2.00000000000000 2.99999999999999
```

Numerical experiments in root-finding literature rarely use double digit multiplicities. In contrast, PEJROOT can handle polynomials with multiplicities in hundreds, even if the coefficients are truncated. For example, let

$$p(x)=(x-z_1)^{100}(x-z_2)^{200}(x-z_3)^{300}(x-z_4)^{400}$$

with $z_1=0.3+0.6i$, $z_2=0.1+0.7i$, $z_3=0.7+0.5i$, $z_4=0.3+0.4i$. Every coefficient of the polynomial is rounded up to 6 significant digits. PEJROOT takes a few seconds under Matlab to calculate the all roots up to 7 digits accuracy,

which is optimal from the pejorative condition number 0.58. The computed roots are

$$\begin{aligned} z_1 &= 0.3000002 + 0.60000005i \\ z_2 &= 0.09999995 + 0.69999998i \\ z_3 &= 0.69999997 + 0.49999998i \\ z_4 &= 0.29999997 + 0.400000002i \end{aligned}$$

4. ALGORITHM II: THE MULTIPLICITIES AND THE INITIAL ROOT ESTIMATES

Algorithm I calculates multiple roots accurately, provided that the multiplicity structure is known and a good set of initial root estimates is given. We now present a GCD-based algorithm to fulfill this input requirement.

As in Lemma 2.1, for $p(x) = (x-z_1)^{\ell_1} \cdots (x-z_m)^{\ell_m}$ with z_j's distinct, the key to calculating z_j's is to factor $p(x)$ and $p'(x)$ with a GCD triple (u, v, w):

$$\left\{ \begin{array}{rcl} u(x)\, v(x) &=& p(x) \\ u(x)\, w(x) &=& p'(x) \end{array} \right. , \quad v(x),\, w(x) \text{ are co-prime.} \quad (12)$$

If this can be accomplished, then we have

$$u(x) = u_0 \prod_{j=1}^{m} (x - z_j)^{\ell_j - 1}, \quad v(x) = v_0\,(x - z_1) \cdots (x - z_m).$$

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

Numerical computation of GCD's has been studied extensively, such as in [1, 2, 6, 11, 13]. Our algorithm employs a partial singular value decomposition by an implicit inverse iteration (Lemma 2.3) and iterative refinement by the Gauss-Newton iteration. As a result, the GCD's can be calculated accurately and efficiently.

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. Set up the system (12).
STEP 3. Find an initial approximation to u, v and w for the GCD system (12).
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 2.1, the degree of $u(x) = GCD(p, p')$ is $m = n - k$ if and only if the first rank-deficient Sylvester discriminant matrix is the k-th. 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 the ς_k that is zero *numerically*. Since only one singular pair is needed, the inverse iteration described in Lemma 2.3 is suitable for this purpose.

The right singular vector $\mathbf{y}_k$ of $S_k(p)$ associated with ς_k contains the initial approximation to $\mathbf{v}$ and $\mathbf{w}$ (see Lemma 2.2), and will be used to refine the GCD triple (u, v, w).

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 $(\mathbf{u}, \mathbf{v}, \mathbf{w})^\top \in \mathbf{C}^{m+1} \times \mathbf{C}^{n-m+1} \times \mathbf{C}^{n-m}$:

$$F(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \left[\begin{array}{c} \mathbf{p} \\ \mathbf{p}' \end{array} \right] \quad (13)$$

where

$$F(\mathbf{u}, \mathbf{v}, \mathbf{w}) \equiv \left[\begin{array}{c} C_{m+1}(v)\ \mathbf{u} \\ C_{m+1}(w)\ \mathbf{u} \end{array} \right].$$

The following lemma ensures that solving this quadratic system is a nonsingular problem at the GCD triple (u, v, w).

LEMMA 4.1. *The Jacobian of $F(\mathbf{u}, \mathbf{v}, \mathbf{w})$ in (13) is*

$$J\,(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \left[\begin{array}{ccc} C_{m+1}(v) & C_{k+1}(u) & 0 \\ C_{m+1}(w) & 0 & C_k(u) \end{array} \right]. \quad (14)$$

If $u(x) = GCD(p, p')$ with (u, v, w) satisfying (13), then $J(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is of full (column) rank.

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

THEOREM 4.1. *Let $\tilde{u}(x) = GCD(p, p')$ with $\tilde{v}(x)$ and $\tilde{w}(x)$ satisfying (13). 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

$$\begin{aligned} (\mathbf{u}_{j+1}, \mathbf{v}_{j+1}, \mathbf{w}_{j+1})^\top &= (\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j)^\top \\ &- J\,(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j)^+ \left[F(\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j) - \left(\begin{array}{c} \mathbf{p} \\ \mathbf{p}' \end{array} \right) \right] \quad (15) \\ j &= 0, 1, \cdots \end{aligned}$$

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

4.1.3 Setting up the initial iterate and refining the GCD with the Gauss-Newton iteration

In `STEP 1`, while calculating the degree of $GCD(p, p')$, the singular pair $(\varsigma_k, \mathbf{y}_k)$ of $S_k(p)$ is a by-product. Theoretically, the singular vector $\mathbf{y}_k$ consists of $\mathbf{v}$ and $\mathbf{w}$ that satisfy (13) (see Lemma 2.2). Numerically, $\mathbf{v}$ and $\mathbf{w}$ need refinement and are used as initial approximation $\mathbf{v}_0$ and $\mathbf{w}_0$.

In theory, a polynomial long division yields

$$p(x) \div v(x) = u(x).$$

However, it is not numerically stable to use straightforward long division, especially with inexact data. By Lemma 2.2, long division is equivalent to solving the linear system

$$C_{m+1}(v)\mathbf{u}_0 = \mathbf{p} \quad (16)$$

for a least squares solution $\mathbf{u}_0$. This "least squares division" is consistently more accurate in our numerical experiment.

By extracting $\mathbf{v}_0$ and $\mathbf{w}_0$ from the singular vector, and by solving (16) for $\mathbf{u}_0$, we obtain initial iterates for the Gauss-Newton iteration (15) that refines the GCD triple $(\mathbf{u}, \mathbf{v}, \mathbf{w})$.

4.2 Computing the multiplicity structure

The procedure in §4.1 calculates the GCD of $p(x)$ and $p'(x)$. From Lemma 2.2, if $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ satisfy the GCD system (13), then $v(x)$ has all simple roots that are identical to all the distinct roots of $p(x)$. This GCD calculation can be applied recursively.

Let $u_0(x) = p(x)$. For $j = 1, , 2\cdots$, until either $deg(u_j) = 0$ or $deg(v_j) = 1$, we use the GCD method in §4.1 to solve

$$\begin{cases} u_j(x)v_j(x) &= u_{j-1}(x) \\ u_j(x)w_j(x) &= u'_{j-1}(x) \end{cases}, \quad u_j = GCD(u_{j-1}, u'_{j-1})$$

From calculating the simple roots of $v_1(x), v_2(x), \cdots$ and counting the repetition of each root, we obtain the multiplicity structure and initial root estimates.

4.3 Numerical results for Algorithm II

Algorithm II is implemented as a Matlab code GcdRoot. Its effectiveness can be illustrated by the example

$$p(x) = (x-1)^{20}(x-2)^{15}(x-3)^{10}(x-4)^5 \qquad (17)$$

that is generated by Matlab function `poly`. GcdRoot correctly identifies the multiplicity structure. The roots are approximated to an accuracy of 10 digits or more. Starting from this result as input using Algorithm I program PejRoot, we obtained all multiple roots with at least 14 digits correct.

```
GcdRoot result: BACKWARD ERROR = 6.05e-010

          COMPUTED ROOTS          MULTIPLICITIES
        4.000000000109542               5
        3.000000000176196              10
        2.000000000030904              15
        1.000000000000353              20

PejRoot result: BACKWARD ERROR = 6.16e-016

          COMPUTED ROOTS          MULTIPLICITIES
        3.999999999999985               5
        3.000000000000011              10
        1.999999999999997              15
        1.000000000000000              20
```

4.4 Some remarks on the GCD calculation

In this paper, we are interested in root-finding. Our discussion on GCD-finding is limited to $GCD(p, p')$. With minor modifications, the algorithm can be applied to numerical computation of the general GCD's of two or more polynomials.

The idea of using singular values to calculate GCD is not new. Extensive discussions and analysis have been reported [2, 6, 13]. In [1, 11], GCD refinement has also been proposed using complicated nonlinear programming methods. In contrast, the Gauss-Newton iteration is simple and efficient, thereby suitable for numerical GCD computation.

5. NUMERICAL EXPERIMENTS FOR THE COMBINED METHOD

5.1 The effect of inexact coefficients

It is expected that application problems use inexact data. The following experiment is intended to test the effect of data error on the accuracy and stability of both GcdRoot and PejRoot.

Example: 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 for k-digit accuracy, where $k = 10, 9, 8, \cdots$. For this problem sequence, GcdRoot identifies the multiplicity structure if the coefficients has at least 7 correct digits. PejRoot continues to converge with data accuracy down to 3 digits, if the multiplicities are manually given instead of computed by GcdRoot.

k	code	$x_1 = 0.\dot{9}\dot{0}$	$x_2 = 1.\dot{8}\dot{1}$	$x_3 = 2.\dot{7}\dot{2}$
10	GcdRoot	0.90909090	1.8181818	2.7272727
	PejRoot	0.909090909	1.81818181	2.7272727
9	GcdRoot	0.909090	1.81818	2.72727
	PejRoot	0.9090909	1.8181818	2.727272
8	GcdRoot	0.90909	1.8182	2.727
	PejRoot	0.9090909	1.818181	2.72727
7	GcdRoot	0.9090	1.82	2.7
	PejRoot	0.90909	1.81818	2.7272

Table 2: Effect of coefficient error on computed roots

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 $x_1 = 0.9,\ 0.99,\ 0.999, \cdots$. GcdRoot is used to find initial input for PejRoot. Computing results for the two nearby roots are shown for both programs in Table 3. The decreasing root gap has no effect on the accuracy of the root $x_3 = -0.5$ in this experiment.

ε	code	$x_1 = 1 - \varepsilon$	$x_2 = 1$
0.1	GcdRoot	0.89999999999	0.99999999999
	PejRoot	0.9000000000000	0.9999999999999
0.01	GcdRoot	0.98999999	0.99999999
	PejRoot	0.989999999999	1.000000000000
0.001	GcdRoot	0.99900	1.00000
	PejRoot	0.99899999999	1.00000000000
0.0001	GcdRoot	0.9997	0.99996
	PejRoot	0.999900000	0.999999999

Table 3: Effect of decreasing root gap on computed roots

The combined algorithm MultRoot handles the root gap as small as 10^{-4}. When this gap is smaller than 10^{-4}, the constructed (inexact) polynomial is closer to a polynomial of two distinct roots of multiplicities 40 and 5.

5.3 A large inexact problem

By combining two methods, we implemented a Matlab code MULTROOT. We conclude this paper by testing this code on our final test problem. We randomly generated 20 complex numbers and use them as roots:

$$.5 \pm i,\ -1 \pm .2i,\ -.1 \pm i,\ -.8 \pm .6i,\ -.7 \pm .7i,$$

$$1.4,\ -.4 \pm .9i,\ .9,\ -.8 \pm .3i,\ .3 \pm .8i,\ .6 \pm .4i$$

From these roots we generate $f(x)$ of degree 20, and then we round all coefficients to 10 decimal digits. 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. MULTROOT faces no difficulty calculating all the roots and finding out multiplicities. The worst accuracy of the roots is 11-digit. Here is the final result.

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

COMPUTED ROOTS                                  MULTIPLICITIES

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

6. REFERENCES

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

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

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

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

[5] 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.

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

[7] M. R. Farmer and G. Loizou. An algorithm for the total, or partial, factorization of a polynomial. *Math. Proc. Camb. Phil. Soc.*, 82:427–437, 1977.

[8] W. Gautschi. Questions of numerical condition related to polynomials. In G. H. Golub, editor, *MAA Studies in Mathematics, Vol. 24, Studies in Numerical Analysis*, pages 140–177, USA, 1984. The Mathematical Association of America.

[9] M. Igarashi and T. Ypma. Relationships between order and efficiency of a class of methods for multiple zeros of polynomials. *J. Comput. Appl. Math.*, 60:101–113, 1995.

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

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

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

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

[14] H. J. Stetter. Condition analysis of overdetermined algebraic problems. In e. a. V.G. Ganzha, editor, *Computer Algebra in Scientific Computing–CASC 2000*, pages 345–365, Springer, 2000.