Available online at www.sciencedirect.com

SCIENCE @ DIRECT•

ELSEVIER Computers and Mathematics with Applications 51 (2006) 605–620

An International Journal
computers & mathematics with applications

www.elsevier.com/locate/camwa

Method for Finding Multiple Roots of Polynomials

CHANG-DAU YAN AND WEI-HUA CHIENG*
Department of Mechanical Engineering
National Chiao Tung University
Hsinchu City, Taiwan, R.O.C.

(Received April 2002; revised and accepted July 2005)

Abstract—Conventional numerical methods for finding multiple roots of polynomials are inaccurate. The accuracy is unsatisfactory because the derivatives of the polynomial in the intermediate steps of the associated root-finding procedures are eliminated. Engineering applications require that this problem be solved. This work presents an easy-to-implement method that theoretically completely resolves the multiple-root issue. The proposed method adopts the Euclidean algorithm to obtain the greatest common divisor (GCD) of a polynomial and its first derivative. The GCD may be approximate because of computational inaccuracy. The multiple roots are then deflated into simple ones and then determined by conventional root-finding methods. The multiplicities of the roots are accordingly calculated. A detailed derivation and test examples are provided to demonstrate the efficiency of this method. © 2006 Elsevier Ltd. All rights reserved.

Keywords—Multiple root, Root finding, Zero finding, Polynomial GCD, Approximate divisibility, Approximate GCD.

1. INTRODUCTION

Finding the roots or zeros of polynomials is a fundamental problem in mathematics. Conventional methods for numerically solving polynomial or algebraic equations include bisection method, linear interpolation method, fixed-point iteration methods, Muller's method, Newton's method. Horner's method, and Bairstow's method [1–5]. Bisection method suffers from slow convergence. Linear interpolation, fixed-point iteration, Muller's, and Newton's methods can suffer from divergence. Horner's and Bairstow's methods are strong in terms of convergence and computational efficiency. However, they lose their accuracy and rate of convergence for polynomials with multiple roots. So do Muller's and Newton's methods.

Methods of finding multiple roots are elucidated in [1,3,6]. However, their formulations are valid only for double roots and not, in general, for multiple roots. Today, well-known mathematics software packages such as MATLAB and MATHEMATICA are also weak in finding multiple roots of polynomials, as discussed in Section 7. Many engineering applications may have suffered from calculation results by conventional methods. An effective resolution that avoids the inaccuracy of multiple-root finding is in great demand.

*Author to whom all correspondence should be addressed.

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

doi:10.1016/j.camwa.2005.07.018

Pan [7] has outlined a quadtree algorithm that defines initial suspect squares and performs proximity tests to isolate the roots of a polynomial. Hull and Mathon [8] have presented an algorithm that simultaneously approximates the roots of a polynomial with quadratic convergence by utilizing Weierstrass-Durand-Kerner formulas. Fortune [9] has developed an algorithm that approximates all roots of a polynomial by means of computing iteratively eigenvalues of the generalized companion matrix of the polynomial. Malek and Vaillancourt [10,11] have proposed a three-stage composite algorithm that reduces the multiple roots of a polynomial into simple ones, subsequently obtains roots by finding the eigenvalues of the first block of Schmeisser's companion matrix, and finally calculates the multiplicity of each root found by means of Lagouanelle's modified limiting formula. All aforementioned methods can obtain more accurate results than those obtained by conventional methods when multiple roots are encountered in polynomials.

This paper proposes a method similar to that derived in [11] for finding multiple roots. However, this paper will focus more on the derivation of an accurate polynomial GCD when the round-off error is concerned. The performance analysis addressed in Section 7 shows that the proposed method can yield better precision of polynomial multiple roots than other known methods do.

2. GENERAL SCHEME FOR ROOTS

A solution of an equation $f(x) = 0$ is a root, λ_0, of the function $f(x)$, that is $f(\lambda_0) = 0$. A root is characterized by its degree or multiplicity, which may prevent conventional root-finding methods to find exact solutions. The following defines a multiple root, as a basis for further discussions.

DEFINITION. MULTIPLICITY OF A ROOT (MR). *(See [5].) Assume that $f(x) \in C^m$, that is $f(x)$ and its derivatives $f'(x), \dots, f^{(m)}(x)$ are defined and continuous in a neighborhood of λ_0. Then, $f(x) = 0$ has a root of multiplicity m at $x = \lambda_0$, if and only if*

$$f(\lambda_0) = 0,\ f'(\lambda_0) = 0,\ \dots,\ f^{(m-1)}(\lambda_0) = 0, \qquad \text{and} \qquad f^{(m)}(\lambda_0) \neq 0.$$

∎

Specifically, a root is a simple root if $m = 1$ and is a multiple root if $m \geqslant 2$.

If $f(x)$is continuous in a neighborhood of its simple root, λ_0, then it can be factored in the form,

$$f(x) = (x - \lambda_0)\varphi_0(x),$$

where $\varphi_0(\lambda)$ is continuous in a neighborhood of λ_0 and $\varphi_0(\lambda_0) \neq 0$ [6]. This factorization is a deflation, such that the simple root, λ_0, is removed from $f(x)$, which is thus deflated into $\varphi_0(x)$. Extending deflation to a multiple root yields the following corollary.

COROLLARY. DEFLATION OF A MULTIPLE ROOT (DMR). *If $f(x)$ is in C^m and has a root of multiplicity m at λ_0, then there exists a function $\varphi_0(x)$, such that $f(x)$ can be expressed in the form*

$$f(x) = (x - \lambda_0)^m \varphi_0(x),$$

where $\varphi_0(x)$ is continuous in a neighborhood of λ_0 and $\varphi_0(\lambda_0) \neq 0$. ∎

PROOF. Assume that $n > m$ and $f(x) \in C^{n+1}$. A finite Taylor expansion of $f(x)$ at λ_0 leads to

$$f(x) = \sum_{k=0}^{n} \frac{(x-\lambda_0)^k}{k!} f^{(k)}(\lambda_0) + \frac{(x-\lambda_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi),$$

for some $\xi \in [\lambda_0, x]$. If $f(x)$ has a root of multiplicity m at λ_0, then by Definition MR,

$$\begin{aligned} f(x) &= \sum_{k=m}^{n} \frac{(x-\lambda_0)^k}{k!} f^{(k)}(\lambda_0) + \frac{(x-\lambda_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi) \\ &= (x-\lambda_0)^m \varphi_0(x), \end{aligned}$$

where $\varphi_0(x) \in C^{n-m}$,

$$\varphi_0(x) = \sum_{k=m}^{n} \frac{(x-\lambda_0)^{k-m}}{k!} f^{(k)}(\lambda_0) + \frac{(x-\lambda_0)^{n-m+1}}{(n+1)!} f^{(n+1)}(\xi),$$

and $\varphi_0(\lambda_0) \neq 0$. The corollary thus follows. ∎

Applying Corollary DMR to all the roots of $f(x)$ yields a general expression for $f(x)$, as shown in the following corollary.

COROLLARY. GENERALIZED DEFLATION OF ROOTS (GDR). *Assume that $f(x)$ has n roots. among them there are k distinct roots, each denoted by λ_i and with multiplicity m_i for $i = 1 \sim k$. respectively. Then, $f(x)$ can be expressed in factored form as*

$$f(x) = \varphi(x) \prod_{i=1}^{k} (x-\lambda_i)^{m_i}, \tag{1}$$

where $\varphi(x) \neq 0$ is continuous in each neighborhood of λ_i for $i = 1 \sim k$, respectively, and $n = \sum_{i=1}^{k} m_i$. ∎

PROOF. According to Corollary DMR,

$$f(x) = (x-\lambda_i)^{m_i} \varphi_i(x),$$

can be derived with respect to each distinct root λ_i for $i = 1 \sim k$. The corollary thus follows if we set

$$\varphi_i(x) = \varphi(x) \prod_{j=1, j\neq i}^{k} (x-\lambda_j)^{m_j}.$$

∎

3. GENERALIZED SIMPLIFICATION OF MULTIPLE ROOTS

Taking the first derivative of $f(x)$ in (1) and factoring yields

$$f'(x) = \left\{ \varphi(x) \sum_{i=1}^{k} \left(m_i \prod_{j=1, j\neq i}^{k} (x-\lambda_j) \right) + \varphi'(x) \prod_{i=1}^{k} (x-\lambda_i) \right\} \prod_{i=1}^{k} (x-\lambda_i)^{m_i-1}. \tag{2}$$

Considering (1) and (2), we see that $f(x)$ and $f'(x)$ have common factors. Let $f_c(x)$ represents the common factor of $f(x)$ and $f'(x)$,

$$f_c(x) = \prod_{i=1}^{k} (x-\lambda_i)^{m_i-1}. \tag{3}$$

Equation (3) shows that all simple roots are removed from $f(x)$,

$$f_c(\lambda_i) \neq 0, \qquad \text{for all } \lambda_i \text{ with } m_i = 1. \tag{4}$$

According to Definition MR,

$$f_c(\lambda_i) = 0,\ f_c'(\lambda_i) = 0,\ \dots,\ f_c^{(m_i-2)}(\lambda_i) = 0 \qquad \text{and} \qquad f_c^{(m_i-1)}(\lambda_i) \neq 0, \tag{5}$$

for all λ_i with $m_i \geqslant 2$.

Factoring (1),(2) by means of $f_c(x)$ yields

$$f(x) = f_c(x) f_0(x) \qquad \text{and} \qquad f'(x) = f_c(x) f_1(x), \tag{6}$$

respectively, and the two factored functions are as follows.

$$f_0(x) = \varphi(x) \prod_{i=1}^{k} (x - \lambda_i), \tag{7}$$

$$f_1(x) = \varphi(x) \sum_{i=1}^{k} \left(m_i \prod_{j=1,\ j \neq i}^{k} (x - \lambda_j) \right) + \varphi'(x) \prod_{i=1}^{k} (x - \lambda_i). \tag{8}$$

Utilizing (6) to relate $f(x)$ to $f'(x)$ leads to

$$f_c(x) f_1(x) = f'(x) = f_c'(x) f_0(x) + f_c(x) f_0'(x).$$

Rearranging the above equation yields

$$f_1(x) = \frac{f_c'(x) f_0(x)}{f_c(x)} + f_0'(x). \tag{9}$$

The first term on the right side of (9) is eliminated since $f_0(\lambda_i) = 0$ and $f_c(\lambda_i) \neq 0$ for simple roots, λ_i, according to (4),(7). Thus,

$$f_1(\lambda_i) = \frac{f_c'(\lambda_i) f_0(\lambda_i)}{f_c(\lambda_i)} + f_0'(\lambda_i) = f_0'(\lambda_i), \qquad \text{for } m_i = 1.$$

However, a zero-divided-by-zero situation arises because $f_c(\lambda_i) = 0$ when a multiple root, λ_i, is encountered, according to (5). l'Hôpital's Rule [12], is used to resolve this indeterminate situation. For example, (5) shows that $f_c(\lambda_i) = 0$ and $f_c'(\lambda_i) \neq 0$ for any root λ_i with $m_i = 2$. Applying l'Hôpital's rule to the first term on the right side of (9) yields

$$f_1(\lambda_i) = \frac{f_c''(\lambda_i) f_0(\lambda_i) + f_c'(\lambda_i) f_0'(\lambda_i)}{f_c'(\lambda_i)} + f_0'(\lambda_i) = 2 f_0'(\lambda_i), \qquad \text{for } m_i = 2.$$

The above derivations imply that a general equation may exist. Therefore, considering a multiple root, λ_i, with $m_i = m + 1$, and applying l'Hôpital's rule m times to the first term on the right side of (9) gives

$$f_1(\lambda_i) = \frac{[f_c'(\lambda_i) f_0(\lambda_i)]^{(m)}}{f_c^{(m)}(\lambda_i)} + f_0'(\lambda_i), \qquad \text{for } m_i = m + 1. \tag{10}$$

Expanding the numerator using the product rule of combining derivatives [12] and the binomial theorem [13] yields

$$[f_c'(\lambda_i) f_0(\lambda_i)]^{(m)} = \sum_{j=0}^{m} \binom{m}{j} f_c^{(m-j+1)}(\lambda_i) f_0^{(j)}(\lambda_i), \qquad \text{for } m_i = m + 1. \tag{11}$$

According to the following relations derived from (5),

$$f_c(\lambda_i) = 0,\ f_c'(\lambda_i) = 0,\ \ldots,\ f_c^{(m-1)}(\lambda_i) = 0 \qquad \text{and} \qquad f_c^{(m)}(\lambda_i) \neq 0,$$

only the terms containing $f_c^{(m)}(\lambda_i)$ and $f_c^{(m+1)}(\lambda_i)$ remain in (11) excluding those that contain $f_0^{(0)}(\lambda_i)$, that is $f_0(\lambda_i)$. Therefore,

$$[f_c'(\lambda_i) f_0(\lambda_i)]^{(m)} = \binom{m}{1} f_c^{(m)}(\lambda_i) f_0^{(1)}(\lambda_i) = m f_c^{(m)}(\lambda_i) f_0'(\lambda_i).$$

Consequently, (10) becomes

$$f_1(\lambda_i) = \frac{m f_c^{(m)}(\lambda_i) f_0'(\lambda_i)}{f_c^{(m)}(\lambda_i)} + f_0'(\lambda_i) = (m+1) f_0'(\lambda_i), \qquad \text{for } m_i = m+1.$$

Since $f_0'(\lambda_i) \neq 0$, considering (7) for any root, λ_i, of $f(x)$, the above derivations yield the following relation,

$$f_1(\lambda_i) = m_i f_0'(\lambda_i) \neq 0,$$

for λ_i, $i = 1 \sim k$, such that $f_0(\lambda_i) = 0$. Therefore, $f_0(x)$ and $f_1(x)$ share no common roots and no common factors. Thus, the following corollary (the readers may also refer to [11]) can be inferred.

COROLLARY. POLYNOMIAL ROOTS WITH MULTIPLICITIES (PRM). *Assume that $f(x)$ has n roots and among them there are k distinct roots, each denoted by λ_i with multiplicity m_i for $i = 1 \sim k$, respectively. Then, $f(x)$ and its first derivative, $f'(x)$, have only one greatest common factor based on the generalized deflation of roots, according to Corollary GDR,*

$$f_c(x) = \prod_{i=1}^{k} (x - \lambda_i)^{m_i - 1},$$

such that

$$f(x) = f_c(x) f_0(x) \qquad \text{and} \qquad f'(x) = f_c(x) f_1(x),$$

where $f_0(x)$ has exactly the same k distinct roots, λ_i, as those of $f(x)$, which are all simple roots. The multiplicity of any root, λ_i, can be determined by

$$m_i = \frac{f_1(\lambda_i)}{f_0'(\lambda_i)}, \qquad \text{for } i = 1 \sim k.$$

∎

Corollary PRM can be proven directly by evaluating (7),(8) at all distinct roots. However, the derivations are not trivial.

Corollary PRM implies that the specific factorization of a function may be decisive in effectively determining its roots. The extracted function, $f_0(x)$, eliminates the concerns of inaccuracy in the possible multiple roots of the original function, $f(x)$. Consequently, conventional methods efficiently and accurately find roots of the new target since the roots to be determined are all simple ones. Simultaneously, the multiplicities of the roots are concisely obtained.

However, Corollary PRM is applicable to a function only when the greatest common factor of the function and its first derivative are available based on the generalized deflation of roots. Hence, it is suitable for algebraic equations but not transcendental equations. The following sections establish a method for resolving the multiple-root issue of polynomial equations.

4. EUCLIDEAN ALGORITHM FOR POLYNOMIALS

An algebraic equation is defined as an equation $p(x) = 0$, in which $p(x)$ is an algebraic function obtained from algebraic operations on polynomials [4,12]. A polynomial of degree n has the general form

$$p(x) = \sum_{i=0}^{n} a_i x^i,$$

where $a_n \neq 0$ and $n \geqslant 1$. A polynomial, $p(x)$, is monic if its leading coefficient $\mathrm{lc}(p(x)) = a_n = 1$.

An algorithm for polynomial division is introduced to factorize an algebraic function or a polynomial.

DEFINITION. POLYNOMIAL DIVISION (PD). *(See [14,15].) Let $p(x)$ and $d(x)$ be polynomials with $d(x) \neq 0$. Then there exist unique polynomials, the quotient $q(x)$ and the remainder $r(x)$. such that*

$$p(x) = d(x)\, q(x) + r(x),$$

where either $r(x) = 0$ or the degree of $r(x)$ is less than the one of $d(x)$. ∎

Polynomial division, according to the above definition, involves no arithmetic division if $d(x)$ is monic. The number of arithmetic operations is essentially proportional to $n(m-n+1)$, where m and n are the degrees of $p(x)$ and $d(x)$, respectively, and $m \geqslant n$. Thus, the computational cost or complexity of the algorithm is of order $O(n(m-n+1))$. However, some fast algorithms may have a lower computational complexity of $O((m-n+1)\log(m-n+1))$ [14].

A polynomial, $p(x)$, is divisible by $d(x)$ if $r(x) = 0$. If so, $d(x)$ is called a divisor of $p(x)$, and so is $q(x)$. Moreover, a polynomial, $q(x)$, is said to be a common divisor of two or more polynomials if $q(x)$ is a divisor of each of those polynomials. The greatest common divisor (GCD) of polynomials is the divisor divisible by all the common divisors, so that the GCD is the largest of all the common divisors. Finding the GCD of two polynomials is surprisingly easier than finding any other common divisor and is performed by extending to polynomials the Euclidean algorithm for obtaining the GCD of two positive integers. The Euclidean theorem is described for polynomials and proven as follows.

THEOREM. EUCLIDEAN THEOREM FOR POLYNOMIALS (ETP). *Let $f(x)$ and $g(x)$ be nonzero polynomials and $r(x)$ be the remainder obtained by dividing $f(x)$ by $g(x)$. If $\mathrm{GCD}(f(x), g(x))$ denotes the GCD of $f(x)$ and $g(x)$, then $\mathrm{GCD}(f(x), g(x)) = \mathrm{GCD}(g(x), r(x))$.* ∎

PROOF. Let $d_{gc}(x)$ be the GCD of $f(x)$ and $g(x)$, such that

$$f(x) = d_{gc}(x)\, q_f(x) \qquad \text{and} \qquad g(x) = d_{gc}(x)\, q_g(x),$$

where $q_f(x)$ and $q_g(x)$ are nonzero and have no common divisors. According to Definition PD, dividing $f(x)$ by $g(x)$ yields

$$d_{gc}(x)\, q_f(x) = d_{gc}(x)\, q_g(x)\, q(x) + r(x).$$

Rearranging this leads to

$$r(x) = d_{gc}(x)\left(q_f(x) - q_g(x)\, q(x)\right).$$

$q_f(x) - q_g(x)q(x)$ and $q_g(x)$ have no common divisor since there is no common divisor of $q_f(x)$ and $q_g(x)$. Therefore, $d_{gc}(x)$ is the GCD of $g(x)$ and $r(x)$. The theorem thus follows. ∎

The Euclidean algorithm for polynomials repetitively applies theorem ETP to successive results of polynomial divisions, starting by dividing $f(x)$ by $g(x)$. The divisor and the remainder of any division are arranged as the dividend and the divisor, respectively, in the subsequent division, yielding a sequence of remainders, called a Euclidean remainder sequence. The repetitive divisions are continued until the remainder is zero. Accordingly, the penultimate remainder in the sequence is exactly the GCD of $f(x)$ and $g(x)$.

ALGORITHM POLYNOMIALS GCD (PG). Let the input polynomials $f(x)$ and $g(x)$ of degree m and n, respectively, be

$$f(x) = \sum_{i=0}^{m} a_i x^i \qquad \text{and} \qquad g(x) = \sum_{i=0}^{n} b_i x^i,$$

where the leading coefficients $\mathrm{lc}(f(x)) = a_m \neq 0$ and $\mathrm{lc}(g(x)) = b_n \neq 0$. The algorithm is as follows.

```
Step 1. Set $p_1(x) = f(x)$, $d_1(x) = g(x)/\mathrm{lc}(g(x))$, and $i = 1$.
Step 2. Compute $r_i(x)$, such that $p_i(x) = d_i(x)q_i(x) + r_i(x)$.
Step 3. Go to next step while $r_i(x) = 0$. Otherwise, set $p_{i+1}(x) = d_i(x)$, $d_{i+1}(x) =$
        $r_i(x)/\mathrm{lc}(r_i(x))$, and $i = i + 1$. And then go to Step 2.
Step 4. Output $r_{i-1}(x)$. ∎
```

The obtained GCD may be a constant (polynomial). If so, the polynomials are said to be coprime. In this worst case, the algorithm has a computational complexity of $O(n^2)$ although a fast version can perform at $O(n \log^2 n)$ [14].

5. ALGORITHM FOR POLYNOMIAL ROOTS WITH MULTIPLICITIES

The above sections developed decisive tools for finding the roots of polynomials, whether simple or multiple. The critical idea is to deflate all multiple roots into simple ones. Based on the deflated polynomial, the distinct roots are searched for, and determined through standard or conventional root-finding methods. Thereafter, the multiplicities of the roots are determined.

As mentioned earlier, Corollary PRM states that the deflated function can be obtained by using the greatest common factor of the target function and its first derivative. For polynomials, the aforementioned greatest common factor is exactly the GCD of the target polynomial and its first derivative. Most importantly, the deflated polynomial is always determinable. This polynomial is determined by Algorithm PG and a successive polynomial division. Effective, practical procedures are accordingly implemented to resolve the multiple-root issue and determine all the distinct roots and their multiplicities, as follows.

Algorithm Polynomial Roots with Multiplicities (PRM). Let the input polynomial $f(x)$ be of degree n and have k distinct roots, each denoted by λ_i with multiplicity m_i for $i = 1 \sim k$, respectively.

```
Step 1. Compute $f'(x)$ of degree $n - 1$.
Step 2. Find the GCD $d_{gc}(x)$ of $f(x)$ and $f'(x)$ by Algorithm PG.
Step 3. Compute $q_f(x) = f(x)/d_{gc}(x)$ and $q_g(x) = f'(x)/d_{gc}(x)$.
Step 4. Employ a conventional method to determine all the $k$ roots $\lambda_i$, distinct and simple,
        of $q_f(x)$.
Step 5. The multiplicities $m_i = 1$ for $i = 1 \sim k$ if the GCD $d_{gc}(x)$ is a constant (polynomial).
        Otherwise, calculate the multiplicities $m_i = q_g(\lambda_i)q'_f(\lambda_i)$ rounded to the nearest
        integer.
Step 6. Output the $k$ roots $\lambda_i$ with their multiplicities $m_i$. ∎
```

Two computational aspects of this algorithm are considered. Step 2 involves algebraic operations to search for the GCD of two polynomials. The step has computational complexity $O(n^2)$ with Algorithm PG, or $O(n \log^2 n)$ with a fast version of the algorithm. Nevertheless, Step 4 uses numerical, iterative methods to find simple roots of polynomials. The efficiency of these methods that reveals their computational complexity is obtained by measuring the order of convergence. The order of convergence of Newton's method is 2 while that of the Secant method is 1.618 and that of Muller's method 1.839 [5,6].

The following polynomial is used as a test example to demonstrate the effectiveness of Algorithm PRM,

$$f_t(x) = (x+1)^3 \left(x^2 + x + 1\right)^2. \tag{12}$$

Expanding the above equation yields a polynomial of degree seven. It can be expressed by arranging its coefficients in descending order of the degrees of polynomial terms. The resultant list of coefficients is

$$f_t(x) = \{1, 5, 12, 18, 18, 12, 5, 1\} \text{ of degree } 7. \tag{13}$$

Taking its first derivative and making it monic leads to

$$f_t'(x) = \left\{1, \frac{30}{7}, \frac{60}{7}, \frac{72}{7}, \frac{54}{7}, \frac{24}{7}, \frac{5}{7}\right\} \text{ of degree 6.} \tag{14}$$

Applying Algorithm PG to the above two polynomials yields the following Euclidean remainder sequence.

$$r_{t_1}(x) = \left\{\frac{18}{49}, \frac{78}{49}, \frac{144}{49}, \frac{150}{49}, \frac{90}{49}, \frac{24}{49}\right\} \text{ of degree 5,} \tag{15}$$

$$r_{t_2}(x) = \left\{\frac{7}{9}, \frac{7}{3}, \frac{28}{9}, \frac{7}{3}, \frac{7}{9}\right\} \text{ of degree 4,} \tag{16}$$

$$r_{t_3}(x) = \{0\} \text{ of degree } -\infty, \text{ i.e., zero polynomial.} \tag{17}$$

Hence, $r_{t_2}(x)$ is found as the GCD of $f_t(x)$ and $f_t'(x)$.

According to Corollary PRM,

$$f_{t_0}(x) = \frac{f_t(x)}{r_{t_2}(x)} = x^3 + 2x^2 + 2x + 1, \tag{18}$$

where $r_{t_2}(x)$ has been made monic. Therefore, the usual root-finding methods from the textbooks or off-the-shelf software can be used to solve $f_{t_0}(x)$. The results are very accurate at the distinct roots, λ_i, of (12). In fact, (18) can be factored exactly as the minimal factorization of (12),

$$f_{t_0}(x) = (x+1)\left(x^2+x+1\right),$$

that has the roots $\lambda_1 = -1$, $\lambda_2 \cong -0.5 - 0.866i$, and $\lambda_3 \cong -0.5 + 0.866i$. Subsequently, the following are computed

$$f_{t_0}'(x) = 3x^2 + 4x + 2,$$

$$f_{t_1}(x) = \frac{f_t'(x)}{r_{t_2}(x)} = 7x^2 + 9x + 5,$$

and then the root multiplicities, m_i, are

$$m_i = \frac{f_{t_1}(\lambda_i)}{f_{t_0}'(\lambda_i)} = \left.\frac{7x^2+9x+5}{3x^2+4x+2}\right|_{x=\lambda_i} = 3,\ 2, \text{ and } 2,$$

for λ_1, λ_2, and λ_3, respectively.

6. APPROXIMATE POLYNOMIAL GCD

As stated in the previous section, the proposed method successfully solves the test polynomial (12) or (13) by deflating multiple roots into simple ones. The root multiplicities are subsequently obtained. Notably, the polynomial has only integer coefficients. Algorithm PRM is executed by arithmetic operations on rational numbers. Exact computation is thus possible. The Euclidean remainder sequence (15)–(17) reveals this fact. However, inexact computation usually occurs.

The number of digits in both the numerator and denominator of the fractional coefficients involved in the division of high-degree polynomials may drastically increase [1,16]. Even though such an occurrence is not obvious in the aforementioned example, because the Euclidean remainder sequence, $r_{t_i}(x)$, ends early at $i = 3$ due to the relatively high degree of the GCD. If the GCD is of low degree or even a constant, the list will be longer and show the growth of the digits of the coefficients. It may hinder the arithmetic computations in two ways. First, the computations require more memory to store the coefficients. Second, the intermediate computation steps might simplify the fractional coefficients and thus reduce their numbers of digits, seriously decreasing the computational performance. For polynomials with real coefficients that cannot conveniently or at all be transformed into fractions, the supposed GCD probably cannot be found.

For example, the Euclidean remainder sequence (15)–(17) expresses the coefficients in a floating-point representation as follows.

$$\begin{aligned}
r_{t_1}(x) &= \{0.367347, 1.59184, 2.93878, 3.06122, 1.83673, 0.489796\}, \\
r_{t_2}(x) &= \{0.777778, 2.33333, 3.11111, 2.33333, 0.777778\}, \\
r_{t_3}(x) &= \{-5.32907, -8.88178, -9.76996, -3.77476\} \times 10^{-15}.
\end{aligned}$$

Clearly, $r_{t_2}(x)$ does not exactly divide $r_{t_1}(x)$ because the remainder, $r_{t_3}(x)$, is not zero, although it is very small. The inexactness of the division follows from the inaccurate representation of polynomial coefficients in the computer system and the round-off errors in the intermediate steps of the computation. An approximation is therefore adopted to deal with such inaccuracy and yield correct results [13,17].

A polynomial norm that is the same as the Euclidean norm for vectors is introduced to measure the elimination of a polynomial [6,14]. The Euclidean norm is a generalization of vector length. As the vector length tends to zero, the vector vanishes. Considering the coefficient list of a polynomial as a vector, the polynomial norm tends to zero when the polynomial is vanishing.

DEFINITION POLYNOMIAL NORM (PN). *(See [1].) A polynomial $p(x)$ of degree n, for $a_n \neq 0$ and $n \geqslant 1$, can be identified with its coefficient list*

$$p(x) = \sum_{i=0}^{n} a_i x^i = \{a_n, a_{n-1}, \cdots, a_1, a_0\}.$$

Then, the Euclidean norm (or 2-norm) of the polynomial is

$$\|p(x)\| = \left(\sum_{i=0}^{n} a_i^2\right)^{1/2}.$$

∎

The Euclidean remainder sequence that results from using Algorithm PG to search for the GCD of polynomials (13) and (14) with real coefficients is therefore established. The following presents the remainder norm sequence, rather than the coefficient lists.

$$\{\|r_{t_i}(x)\|\} = \{4.92848, 4.66667, 1.47304 \times 10^{-14}, 0.171796, 5.82458, 0.818418, 0.0\}, \quad \text{for } i = 1 \sim 7.$$

The algorithm clearly finds no GCD or a constant (polynomial). However, the above remainder norm sequence implies that the third division yields a remainder of approximately zero, as determined by the very small norm of the third remainder. However, simply comparing the norms of remainders one may not be completely sure when to stop calculating the remainder sequence. The following presents another test polynomial of degree 30 to test the effectiveness of determining the approximately zero remainder.

$$(x+1)^{10}\left(x^2+x+1\right)^{10}. \tag{19}$$

Curve 1 in Figure 1 shows the scaled remainder norm sequence obtained by applying Algorithm PG to polynomial (19) and its derivative. Compared to the other remainders, the third remainder seems to be the one which is approximately zero. However, the decrease of Curve 1 questions the absoluteness of the norm of the third remainder being minimum in the remainder norm sequence. The concept of approximate divisibility of polynomials is introduced to solve this problem.

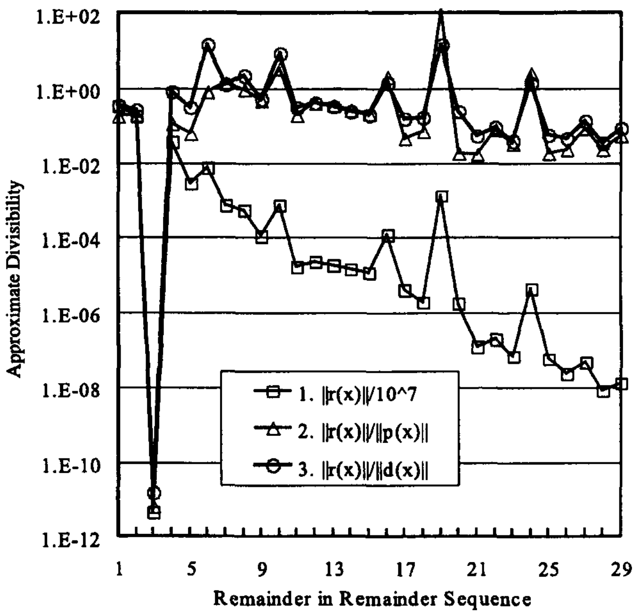

Figure 1. Sequences of approximate divisibility according to different definitions while searching for the GCD of polynomial (19) and its first derivative.

DEFINITION APPROXIMATE DIVISIBILITY (AD). *(See [13].) Consider the polynomials $p(x)$, $d(x)$, and $r(x)$, such that $r(x)$ is the remainder of $p(x)$ divided by $d(x)$. Thus, $d(x)$ divides $p(x) - r(x)$. The approximate divisibility is defined as either*

$$\Delta = \frac{\|r(x)\|}{\|p(x)\|} \quad \text{or} \quad \Delta = \frac{\|r(x)\|}{\|d(x)\|}.$$

Then, $d(x)$ is said to be an ε-divisor of $p(x)$ if $\Delta < \varepsilon$ for a given $\varepsilon > 0$. ∎

Curves 2 and 3 in Figure 1, illustrate two different sequences of approximate divisibility according to Definition AD. Clearly, both definitions of approximate divisibility can determine which remainder is approximately zero. Therefore, the remainder before the one which is approximately zero in the Euclidean remainder sequence is found as the approximate GCD. It is said to be an ε-GCD of polynomial (19) and its derivative based on Definition PN. Algorithm PG may thus be modified to accommodate the computationally inaccurate search of GCD with floating-point arithmetic.

ALGORITHM APPROXIMATE POLYNOMIAL GCD (APG). Consider two input polynomials $f(x)$ and $g(x)$ with nonzero leading coefficients. Choose ε to be sufficiently small, say 10^{-8}, as the threshold of the approximate divisibility. The following steps describe the algorithm.

```
Step 1. Set $p_1(x) = f(x)$, $d_1(x) = g(x)/\mathrm{lc}(g(x))$, and $i = 1$.
Step 2. Compute $r_i(x)$, such that $p_i(x) = d_i(x)q_i(x) + r_i(x)$.
Step 3. Compute $\|r_i(x)\|$ and $\Delta = \|r_i(x)\|/\|d_i(x)\|$. Go to Step 5 while $\Delta < \varepsilon$.
Step 4. Compute $d_{i+1}(x) = r_i(x)/\mathrm{lc}(r_i(x))$ and $\|d_{i+1}(x)\| = \|r_i(x)\|/\mathrm{lc}(r_i(x))$. Set $p_{i+1}(x) =$
        $d_i(x)$ and $i = i + 1$ and then go to Step 2.
Step 5. Output $r_{i-1}(x)$.
```

Consequently, for polynomials with real coefficients, Algorithm PRM that simultaneously finds polynomial roots and their multiplicities must depend on Algorithm APG in Step 2 to search for the approximate GCD rather than Algorithm PG to search for the exact GCD.

7. PERFORMANCE ANALYSIS

This section presents examples to verify the performance of the proposed root-finding method. Two commercial software packages, MATLAB and MATHEMATICA, are used to solve polynomial equations as compared with the performance of the proposed method.

The MATLAB software [18] provides a function, *roots()*, to find polynomial roots. The function basically constructs a companion matrix by arranging the coefficients of the polynomial to be solved. It then determines the eigenvalues of the matrix through the QR algorithm. Some errors may occur during the computation of eigenvalues. Another software package, MATHEMATICA [19], takes a different approach. It essentially manipulates mathematical expressions and equations through symbolic operations. Hence, when solving a polynomial equation, MATHEMATICA tries to factor it and then decompose it if possible. For example, the function Solve$[f[x] == 0, x]$ may be used to find the roots of $f(x)$ in polynomial (12). However, the function Solve$[f[x] == 0, x]$ may induce fixed-point arithmetic for root finding when $f(x)$ is in a symbolically decomposable form. We chose the function NSolve$[f[x] == 0, x]$ to find roots in the floating-point arithmetic benchmark given in following sequels.

The polynomial to be tested is

$$(x+1)^m \left(x^2+x+1\right)^m \text{ of degree } 3m, \tag{20}$$

where m, to be specified, denotes the root multiplicity. Polynomial (20) clearly has three distinct, multiple roots at $x = -1$ and $x = (-1 \pm i\sqrt{3})/2$. Figure 2 illustrates the results of finding the roots of polynomial (20) using the built-in functions of MATLAB and MATHEMATICA, as well as the proposed method. The results are presented for calculated roots, as average percentages of the computational errors with respect to the root multiplicity specified for the test polynomial. The computational errors are meaningful only because the roots are located on the unit circle. Curve 1 in the figure shows the results obtained using MATLAB. Significant errors begin to be observed at root multiplicity $m = 4$, increasing quadratically until $m = 7$. The curve reveals that the root-finding inaccuracy dramatically increases when the root multiplicity $m > 7$.

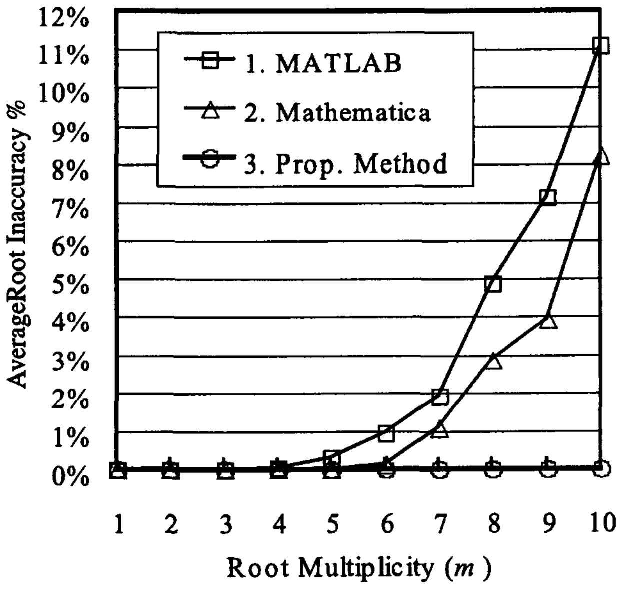

Figure 2. Average root-finding inaccuracy with respect to root multiplicity for polynomial (20) with floating-point arithmetic.

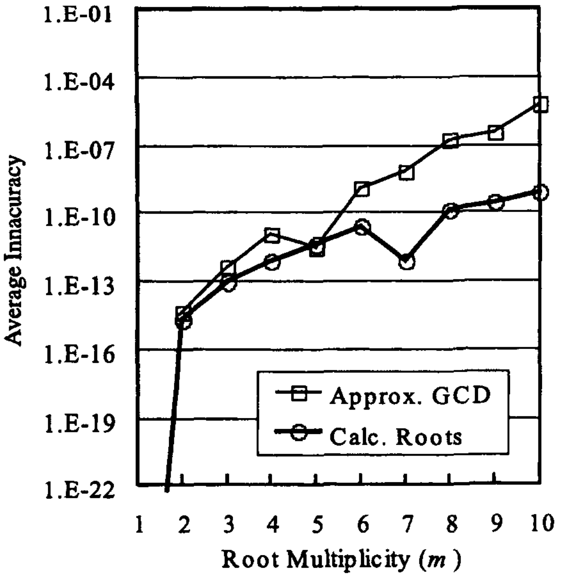

Figure 3. Average inaccuracy of found approximate GCDs and calculated roots with respect to root multiplicity for polynomial (20).

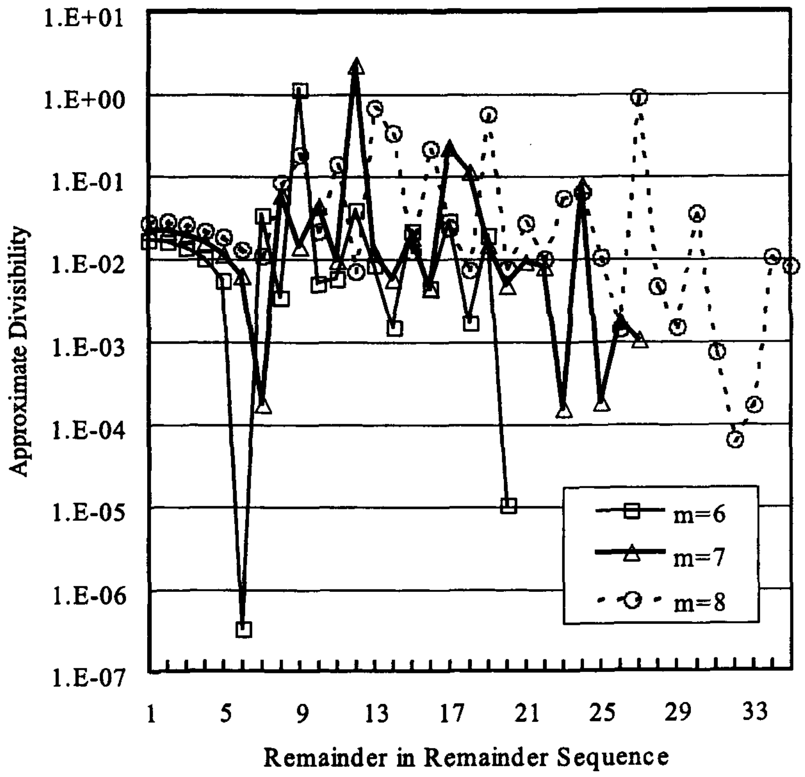

Figure 4. Sequences of approximate divisibility while searching for the GCDs of polynomial (21) and their first derivatives with respect to different maximum multiplicity, m, of roots.

Considering MATHEMATICA's root finding for polynomial (20), Curve 2 in Figure 2 plots the errors in the computed roots with respect to the multiplicity. The errors significantly increase

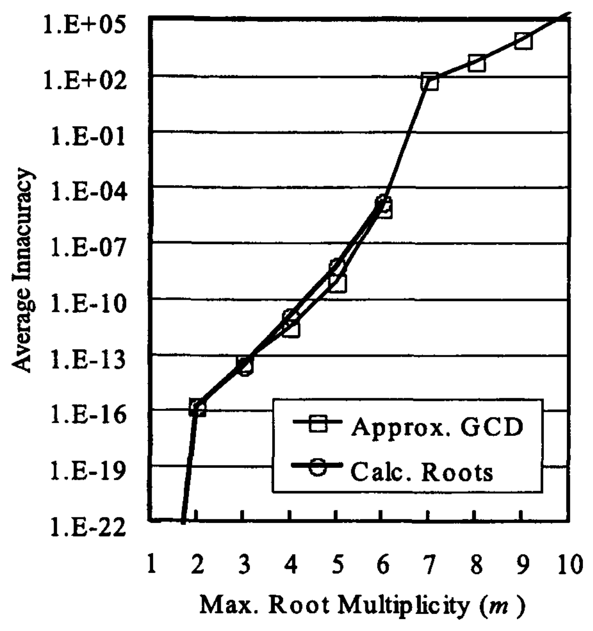

Figure 5. Average inaccuracy of found approximate GCDs and calculated roots with respect to root multiplicity for polynomial (21), in which those for $m > 6$ are not shown because no approximate GCDs are properly found.

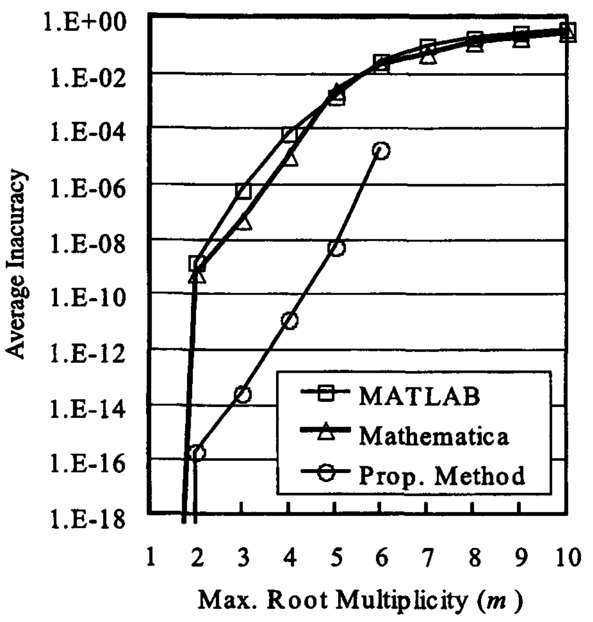

Figure 6. Average root-finding inaccuracy with respect to root multiplicity in logarithmic scaling for polynomial (21), in which those for $m > 6$ are not shown because no approximate GCDs are properly found.

from $m = 6$. The function apparently shows better results than MATLAB, perhaps because some symbolic manipulation or other techniques are used to find roots of polynomials with real coefficients. However, the errors that result from the root-finding functions of both MATLAB and MATHEMATICA are not acceptable for engineering applications.

Table 1. Results using the proposed method, MATHEMATICA, MATLAB, and corresponding programs from the DROOTS/IMSL library/NAG library [8].

Max. Error / Polynomial	Proposed Method	MATHEMATICA	MATLAB	DROOTS	IMSL	NAG
P4	7.28E − 15	1.08E − 07	7.02E − 06	1.71E − 12	2.46E − 07	2.83E − 05
P5	0	7.49E − 03	4.76E − 02	4.10E − 10	8.69E − 11	0
P6	5.97E − 13	3.44E − 05	1.29E − 04	5.01E − 05	2.96E − 06	1.61E − 04
P7	2.49E − 06	1.01E − 01	6.03E − 04	4.70E − 08	4.65E − 05	6.34E − 04
P9	3.87E − 13	8.06E − 07	8.48E − 05	1.31E − 06	7.23E − 07	2.81E − 04
P13	1.71E − 09	3.04E − 12	5.22E − 03	8.57E − 04	1.02E − 02	1.35E − 02
P19	5.14E − 12	1.12E − 05	3.71E − 08	7.14E − 10	6.47E − 07	7.34E − 08
P20	2.56E − 10	3.30E − 02	2.59E − 04	2.28E − 10	8.16E − 03	5.81E − 04

In contrast, Curve 3 drastically outperforms the other two curves in Figure 2. The results are obtained by implementing the proposed method in both MATLAB and MATHEMATICA. Both implementations produce virtually the same results. Figure 3 presents the average inaccuracy of the found approximate GCDs and calculated roots. The figures demonstrate the excellent performance of the proposed method in finding roots, as compared to that of MATLAB and MATHEMATICA. Furthermore, the proposed method yields the multiplicities of the roots.

However, Figure 3 reveals that inaccuracy in finding roots grows as the root multiplicity increases, even with the proposed method. The following example incorporates the ill-conditioned property of polynomials to test the performance of the root-finding methods. As an extension of one of the test polynomials in [20], the following polynomial is used,

$$\prod_{n=1}^{m} (x - 0.1n)^{m-n+1} \text{ of degree } \frac{m(m+1)}{2}, \tag{21}$$

where m, to be specified, denotes the maximum multiplicity of the distinct roots. Polynomial (21) has constant term 1.2×10^{-5} for $m = 3$, -3.456×10^{-11} for $m = 5$, and 1.254×10^{-17} for $m = 7$. It is apparently ill-conditioned when m is large because some coefficients are too small to be handled by computer systems.

Figure 4 depicts the sequences of approximate divisibility while searching for the GCDs of polynomial (21) and their first derivatives for $m = 6, 7$, and 8. Algorithm APG easily and properly finds the approximate GCD for $m = 6$. The approximate GCD for $m = 7$ is correctly found if ε is chosen as high as 10^{-3}. If $m = 8$ or larger, an incorrectly approximate GCD or only a constant polynomial is obtained. The proposed method fails in such cases.

Figure 5 displays the average inaccuracy of the found approximate GCDs and calculated roots. The great inaccuracy of approximate GCDs at $m > 6$ reveals the failure of the search for GCDs. Consequently, no further calculation of polynomial roots is done. However, as compared with MATLAB and MATHEMATICA, the proposed method performs very well when tackling the ill-conditioned polynomials as shown in Fig. 6, if the approximate GCD is properly found.

A set of different polynomials with multiple roots, listed below, are tested to demonstrate the effectiveness of the proposed method in comparison with other known algorithms.

P4	$(x-1)^2(x-5i)^2(x+i)^3$
P5	$(x-1)^{10}$
P6	$(x-0.1)^4(x-0.2)^3(x-0.3)^2(x-0.4)$
P7	$(x-4-0.1i)(x-4+0.1i)(x-10)(x-5)(x-4)^2(x-3)^2(x-2)(x-1)$
P9	$(x-3)^3(x+1)^4(x+i)^2(x-1-2i)(x-1)$
P13	$x^6(x+10)^5(x-10)^5(x+i)^2(x-i)^2$
P19	$(x^{24}-x^{23}-x^{22}-\cdots-x-1)^2$
P20	$(x^{12}-x^{11}-x^{10}-\cdots-x-1)^4$

The above polynomials are labeled according to the ordinal numbers originally listed in Table II in [8]. Test results are shown in Table 1.

8. CONCLUSION

Conventional methods for finding the roots of algebraic functions or polynomials lose accuracy when multiple roots are involved. In particular, the inaccuracy is dramatically increased for polynomials that have roots with a large multiplicity. This work presents a method that resolves the multiple-root issue and provides the multiplicities of roots.

The proposed method is derived from the following findings. A GCD can be found from a polynomial and its first derivative. Dividing the polynomial by the GCD yields a resultant polynomial whose roots are proven to be simple and the same as the distinct roots of the original polynomial. Usually, the resultant polynomial with only simple roots can be accurately found using conventional methods. More advantageously, the multiplicity of each root can be determined easily. The results are confirmed by the test examples for which the functions of MATLAB and MATHEMATICA are inadequate.

However, the proposed method fails to find roots of a polynomial whose GCD is not correctly found. The failure follows from the round-off errors due to the inaccurate representation of floating-point coefficients and inexact polynomial division. Approximate divisibility is introduced to determine when to stop the computation of the remainder sequence, and then locate the approximate GCD. Experimental results have shown that the approximate GCD can be concisely and appropriately determined in the Euclidean remainder sequence. After the approximate GCD is correctly found, the proposed method yields highly accurate results for the roots and their multiplicities.

REFERENCES

1. J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, pp. 131–135, 147., Cambridge University Press, (1999).
2. C.F. Gerald and P.O. Wheatley, *Applied Numerical Analysis*, Sixth Edition, pp. 38–87, Addison-Wesley, (1999).
3. M.L. James, G.M. Smith and J.C. Wolford, *Applied Numerical Methods for Digital Computation*, Fourth Edition, pp. 66–119, HarperCollins, (1993).
4. M.J. Maron, *Numerical Analysis: A Practical Approach*, Second Edition pp. 41, 61-94, Macmillan, (1987).
5. J.H. Mathews and K.D. Fink, *Numerical Methods Using* MATLAB, Third Edition pp.40–100, Prentice-Hall, (1999).
6. C.W. Ueberhuber, *Numerical Computation: Methods, Software, and Analysis, Volume Two*, pp. 7–8, 296–314, Springer, (1997).
7. V.Y. Pan, Solving a polynomial equation: Some history and recent progress, *SIAM Review* **39** (2), 187–220, (1997).
8. T.E. Hull and R. Mathon, The mathematical basis and a prototype implementation of a new polynomial rootfinder with quadratic convergence, *ACM Transactions on Mathematical Software* **22** (3), 261–280, (1996).
9. S. Fortune, Polynomial root finding using iterated eigenvalue computation, In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation,* (ISSAC'2001) , pp. 121–128, (2001).
10. F. Malek and R. Vaillancourt, Polynomial zerofinding iterative matrix algorithms, *Computers Math. Applic.* **29** (1), 1-13., (1995).
11. F. Malek and R. Vaillancourt, A composite polynomial zerofinding matrix algorithm, *Computers Math. Applic.* **30** (2), 37–47, (1995).
12. G.R. Bradley and K.J. Smith, *Calculus*, Second Edition, pp. 31–32, 145, 301, Prentice-Hall, (1999).
13. V.Y. Pan, Computation of approximate polynomial GCDs and an extension, *Information and Computation* **167**, 71–85, (2001).
14. D. Bini and V. Pan, *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithm*, pp. 18–42, 90, Birkhäuser, (1994).
15. D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Second Edition, pp. 399–436, Addison-Wesley, (1981).
16. W.S. Brown, On Euclid's algorithm and the computation of polynomial greatest common divisors', *Journal of Association for Computing Machinery* **18** (4), 478–504, (1971).
17. I.Z. Emiris, A. Galligo and H. Lombardi, Certified approximate univariate GCDs, *Journal of Pure and Applied Algebra* **117** & **118**, 229–251, (1997).

18. MathWorks Inc., The Student Edition of MATLAB, Version 4, User's Guide, pp. 185–186, 559–560, Prentice-Hall, (1995).
19. S. Wolfram, *The* MATHEMATICA *Book*, Fourth Edition, pp. 85–88, 99–100, 786–790, Cambridge University Press, (1996).
20. M.A. Jenkins and J.F. Traub, Principles for testing polynomial zerofinding programs, *ACM Transactions on Mathematical Software* **1** (1), 26–34, (1975).