

The Amended DSeSC Power Method for Polynomial Root-Finding

V. Y. PAN

Department of Mathematics and Computer Science
Lehman College of CUNY, Bronx, NY 10468, U.S.A.
vpan@lehman.cuny.edu

(Received August 2004; accepted September 2004)

Abstract—Cardinal’s matrix version of the Sebastiao e Silva polynomial root-finder rapidly approximates the roots as the eigenvalues of the associated Frobenius matrix. We preserve rapid convergence to the roots but amend the algorithm to allow input polynomials with multiple roots and root clusters. As in Cardinal’s algorithm, we repeatedly square the Frobenius matrix in nearly linear arithmetic time per squaring, which yields dramatic speedup versus the recent effective polynomial root-finder based on the application of the inverse power method to the Frobenius matrix. © 2005 Elsevier Science Ltd. All rights reserved.

Keywords—Polynomial roots, Frobenius matrices, Repeated squaring, Root clusters.

1. INTRODUCTION

1.1. The Background and Related Works

Solving a polynomial equation is the oldest computational problem. It has important applications to computer algebra, algebraic geometric computations, and signal processing (see [1–7], and the bibliography therein). We wish to cite also the recent progress on the extension to the solution of the algebraic eigenproblem in [8].

The polynomial root-finder in [9–11] optimizes both arithmetic and Boolean time up to polylogarithmic factors, that is, up to these factors the solution involves as many arithmetic and Boolean operations as are required just to process the input and the output. The algorithm, however, is not simple to code, has not been implemented yet, and requires computations with extended precision. The users prefer alternative algorithms, and the design of more efficient practical polynomial root-finders remains an important research challenge.

Increasingly popular is the reduction to approximating the eigenvalues of the associated Frobenius companion matrix, whose spectrum is made up of the roots of the input polynomial. Such a root-finder is available from MATLAB; it relies on the application of the QR algorithm to the Frobenius matrix and has been dramatically accelerated in [7,12].

Another matrix approach to polynomial root-finding is a nontrivial extension of the so-called Graeffe’s iteration. The iteration was proposed by Dandelin, rediscovered by Lobachevsky, and

Supported by NSF Grant CCR 9732206 and PSC CUNY Awards 65393-0034 and 66437-0035.

popularized by Graeffe [13]. It generates the polynomials with recursively squared roots. Sebastiao e Silva in [14] (cf. [15]) complements this process with a nontrivial technique for the recovery of the roots of the original polynomial, although numerical stability of this recovery is generally a problem. In Cardinal's matrix version in [16], the roots are repeatedly squared by means of squaring the Frobenius matrix; its structure is preserved and exploited to run the computations fast, in nearly linear time per squaring. Cardinal also proposes a special initialization technique which simplifies and numerically stabilizes the recovery of the roots.

The resulting *DSeSC iteration* due to Dandelin, Sebastiao e Silva and Cardinal, however, shares with the algorithm in [14] (and possibly even aggravates) the severe convergence problems in the case where the input polynomial has multiple or clustered roots. In this important case Cardinal suggests shifting from repeated squaring to the matrix sign iteration. This is a distinct approach; its numerical stability at the recovery stage becomes a problem again.

1.2. The DSeSC Method with Amendments

We complement the DSeSC iteration with two simple but decisive amendments. Our novel initialization technique removes the problems with multiple and clustered roots. This technique, however, is not compatible with Cardinal's simplified and stabilized recovery of the approximations to the roots, and we propose a distinct recovery technique. Our resulting algorithm is equivalent to the application of the classical power iteration [17, Section 7.3; 18, Section 2.1] to the Frobenius matrix except that we repeatedly square this matrix, thus advancing to a root much faster. Our final recovery of the root/eigenvalue is closely related to its approximation by the Rayleigh quotients. If we apply our algorithm to the reverse polynomial and shift the variable, we arrive at the shifted inverse power iteration (with repeated squaring).

We call our algorithm the *amended DSeSC power iteration*. It is clearly superior to the inverse power algorithm in [19] (which performs with no repeated squaring). The latter algorithm has already nearly reached the efficiency level of the best practical root-finders, according to the results of extensive tests reported in [19]. Thus, the amended DSeSC power iteration has good chances to become the polynomial root-finder of choice.

1.3. Organization of the Paper

We first recall the relevant parts of the DSeSC algorithm and then specify our amendments. We introduce some definitions in Section 2, recall some basic facts in Section 3, and recall the relevant parts of the DSeSC algorithm in Sections 4–6. In Section 7, we cover the initialization techniques from [16] and their amendment. In Section 8 we recall Cardinal's results on computations in the algebra generated by a Frobenius (companion) matrix. In Section 9, we combine the algorithms in Sections 7 and 8 with the power method. In Section 10, we specify the recipes for multiple roots and root clusters.

2. SOME DEFINITIONS

$t(x)$ is a polynomial of degree n with n distinct roots z_1, \dots, z_n ,

$$t(x) = \sum_{i=0}^n t_i x^i = t_n \prod_{j=1}^n (x - z_j), \quad t_n \neq 0. \quad (2.1)$$

A_t is the algebra of polynomials reduced modulo $t(x)$.

$$l_j(x) = \frac{t(x)}{t'(z_j)(x - z_j)}, \quad j = 1, \dots, n, \quad (2.2)$$

are the Lagrange polynomials for the node set $\{z_1, \dots, z_n\}$.

$\|\cdot\|$ is a polynomial norm, such that $\|t(x)\| \rightarrow 0$ as $\sum_{i=0}^n |t_i| \rightarrow 0$, e.g., $\|t(x)\|_1 = \sum_{i=0}^n |t_i|$, $\|t(x)\|_2 = (\sum_{i=1}^n |t_i|^2)^{1/2}$, or $\|t(x)\|_{2,w} = (\sum_{i=1}^n w_i |t_i|^2)^{1/2}$ for fixed positive weights w_1, \dots, w_n .

For a polynomial $p(x) = \sum_{i=0}^k p_i x^i$, define the *reverse polynomial*

$$\bar{p}(x) = x^k \sum_{i=0}^k p_i x^{-i} = \sum_{i=0}^k p_i x^{k-i}. \quad (2.3)$$

0 and I denote the null and identity matrices of appropriate sizes, respectively. \mathbf{e}_{i-1} is the i^{th} column of I . \mathbf{v}^\top and M^\top are the transposes of a vector \mathbf{v} and a matrix M , respectively. $T = (t_{i,j})_{i,j=0}^{n-1}$ is a Toeplitz matrix if $t_{i+1,j+1} = t_{i,j}$ wherever i and j are in the range from 0 to $n-2$.

The matrix entries represented by the blank space are zeros.

$$Z = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & \ddots & \ddots & & \\ & & & 1 & 0 \end{bmatrix} \quad (2.4)$$

is the $n \times n$ down-shift matrix, such that $Z^n = 0$, $Z\mathbf{v} = (v_{i-1})_{i=0}^{n-1}$ for a vector $\mathbf{v} = (v_i)_{i=0}^{n-1}$ and $v_{-1} = 0$.

$L(\mathbf{x}) = \sum_{i=0}^{n-1} x_i Z^i$ for a vector $\mathbf{x} = (x_i)_{i=0}^{n-1}$ is the lower triangular Toeplitz matrix defined by its first column given by the vector \mathbf{x} .

3. BASIC FACTS

FACT 3.1. LAGRANGE INTERPOLATION FORMULA. For any pair of polynomials $t(x)$ in (2.1) and $f(x) \in A_t$ and for the Lagrange polynomials $l_j(x)$ in (2.2), we have $f(x) = \sum_{i=1}^n f(z_i) l_i(x)$.

FACT 3.2. LAGRANGE POLYNOMIALS AS IDEMPOTENTS OR PROJECTORS. For all j and $k \neq j$ we have

- (a) $l_j(x) l_k(x) = 0 \pmod{t(x)}$,
- (b) $l_j^2(x) = l_j(x) \pmod{t(x)}$,
- (c) $f(x) l_j(x) = f(z_j) l_j(x) \pmod{t(x)}$.

PROOF. Part (a) is immediate by inspection. To prove Part (b), recall that $t(z_j) = 0$, apply the Lagrange expansion formula to obtain that $t(x) = t(x) - t(z_j) = t'(z_j)(x - z_j) \pmod{(x - z_j)^2}$, and deduce that $x - z_j$ divides $l_j(x) - 1$. Finally, Part (c) follows when we multiply the Lagrange interpolation formula by $l_j(z)$ and then substitute the equations of Parts (a) and (b). \blacksquare

By combining Facts 3.1 and 3.2, we extend Fact 3.1 as follows.

COROLLARY 3.1. For any polynomial $f(x) \in A_t$, we have

$$(f(x))^m = \sum_{j=1}^n (f(z_j))^m l_j(x) \pmod{t(x)}, \quad m = 1, 2, \dots$$

Corollary 3.1 implies the following result.

COROLLARY 3.2. For $f(x) \in A_t$ and an integer j , $1 \leq j \leq n$, let

$$\theta = \max_{k:k \neq j} \left| \frac{f(z_k)}{f(z_j)} \right| < 1. \quad (3.1)$$

Then $(f(x)/f(z_j))^m \pmod{t(x)} = l_j(x) + h(x)$, where $h_{j,m}(x)$ is a polynomial of degree at most $n-1$, $\|h_{j,m}(x)\|$ is in $O(\theta^m)$ as $m \rightarrow \infty$.

4. THE BASIC ALGORITHM

Corollary 3.2 motivates using the following algorithm.

ALGORITHM 4.1. REPEATED SQUARING MODULO A POLYNOMIAL. (See [14,15].)

INPUT. The coefficients of a polynomial $t(x)$ in (2.1) and a polynomial norm $\|\cdot\|$.

OUTPUT. An approximation to a root z_j of $t(x)$ or FAILURE.

INITIALIZATION. Fix a polynomial $r_0(x) = f(x)$, $0 < \deg f(x) < n$, a large integer k , and a small positive ε .

COMPUTATIONS. Recursively compute the polynomials

$$r_{h+1}(x) = \frac{1}{n_h} ((r_h(x))^2 \bmod t(x)) \quad (4.1)$$

for the positive scalars n_h , such that

$$\|r_{h+1}(x)\| = 1 \quad (4.2)$$

and for $h = 0, 1, \dots, k-1$. If $\|r_{h+1}(x) - r_h(x)\| < \varepsilon$, compute the quotient $ax - b \approx t(x)/r_{h+1}(x)$, such that $\|(ax - b)r_{h+1}(x) - t(x)\|$ is minimum (see [20; 21, Proposition 2.2]), output an approximation b/a to a root z_j of $t(x)$, and stop. If $h = k$, stop and output FAILURE. (Due to Corollary 3.2, the algorithm does not output FAILURE if (3.1) holds and k is large enough.)

Let us briefly analyze the algorithm. By virtue of Corollary 3.2, a multiple of the residue polynomial $(f(x)^m) \bmod t(x)$ approximates $l(x)$ within an error norm of the order of θ^m . The scalar factors $(f(z_j))^m$ remain unknown until we compute z_j , but this is immaterial in the process (4.1) of repeated squaring due to the scaling by n_h . In h steps of this process an approximation $r_h(x)$ to a scalar multiple of $l_j(x)$ is computed within an error norm of the order of θ^{2^h} . If (3.1) holds and h is large enough, the sequence $\{r_h(x)\}$ stabilizes, and we approximate $x - z_j$ as a scaled quotient in the division of $t(x)$ by $r_h(x)$ because $t(x)/l_j(x) = t(z_j)(x - z_j)$.

5. ARITHMETIC OPERATIONS IN THE ALGEBRA A_t

Let us compute products, squares, and reciprocals in the algebra A_t .

The product $u(x)v(x)$ in A_t is the polynomial

$$r(x) = w(x) - q(x)t(x), \quad (5.1)$$

where

$$w(x) = u(x)v(x), \quad (5.2)$$

$$\deg w(x) = 2n - h, \quad h > 1, \quad \deg r(x) = k < n, \quad \deg q(x) = n - h. \quad (5.3)$$

Let us compute $r(x)$ in (5.1). W. l. o. g. let $h \leq n$. Substitute $1/x$ for x in (5.1), multiply the resulting equation by x^{2n-h} , and obtain that $\bar{w}(x) - \bar{t}(x)\bar{q}(x) = x^{2n-h-k}\bar{r}(x) = 0 \bmod x^{2n-h-k}$, where $\bar{w}(x)$, $\bar{t}(x)$, $\bar{q}(x)$, and $\bar{r}(x)$ denote the reverse polynomials of $w(x)$, $t(x)$, $q(x)$, and $r(x)$, respectively (cf. (2.3)). Since $t_n \neq 0$, the polynomial $\bar{t}(x)$ has the reciprocal modulo x^{n-h+1} , and we write

$$\bar{s}(x) = (\bar{t}(x))^{-1} \bmod x^{n-h+1}. \quad (5.4)$$

By multiplying the equation $\bar{w}(x) = \bar{q}(x)\bar{t}(x) \bmod x^{n-h+1}$ by $\bar{s}(x)$ and recalling from (5.4) that $\deg \bar{q}(x) = n - h$, we obtain

$$\bar{q}(x) = \bar{s}(x)\bar{w}(x) \bmod x^{n-h+1}. \quad (5.5)$$

These observations lead us to the following algorithm, where we write $d = \lceil \log_2(2n - h + 1) \rceil$, $N = 2^d$.

ALGORITHM 5.1. MULTIPLICATION OF TWO POLYNOMIALS MODULO A POLYNOMIAL.

INPUT. Three polynomials, $t(x)$, $u(x)$, and $v(x)$ satisfying (2.1), (5.2), and (5.3).

OUTPUT. The polynomial $r(x)$ in (5.1).

PREPROCESSING. Compute the coefficients of the polynomial $\bar{s}(x)$ in (5.4) and the values of the polynomials $t(x)$ and $\bar{s}(x)$ at the $2^{d^{\text{th}}}$ roots of 1.

COMPUTATIONS. Successively compute the coefficients of the polynomials $w(x)$ in (5.2), $\bar{q}(x)$ in (5.5), and $r(x)$ in (5.1), with the auxiliary transitions to the reverse polynomials defined by (2.3).

To perform Algorithm 5.1 economically, apply the fundamental evaluation-interpolation techniques introduced by Toom in [22]. Choose the N^{th} roots of 1, $N = 2^d$ as the nodes of Toom's evaluation and interpolation and write $\text{FFT}(d)$ to denote the arithmetic complexity of performing the FFT on these nodes. Then the arithmetic cost of the preprocessing is given by the sum of $O(n \log n)$ arithmetic operations for computing the coefficients of $\bar{s}(x)$ (see [23, Section 2.5]) and $2\text{FFT}(d)$ for computing the values of $t(x)$ and $\bar{s}(x)$ at the $2^{d^{\text{th}}}$ roots of 1. In addition, Algorithm 5.1 requires $7\text{FFT}(d) + O(n)$ arithmetic operations. $4\text{FFT}(d)$ of them come from performing FFT on the $2^{d^{\text{th}}}$ roots of 1 for the polynomials $u(x)$, $v(x)$, $\bar{w}(x) \bmod x^{n-h+1}$, and $q(x)$, whereas the other $3\text{FFT}(d)$ come from performing the inverse FFT on the $2^{d^{\text{th}}}$ roots of 1 for the polynomials $w(x)$, $(\bar{w}(x) \bmod x^{n-h+1})\bar{s}(x)$, and $q(x)t(x)$. Besides the FFTs and inverse FFTs, the algorithm involves $3N + n$ arithmetic operations for the pairwise multiplication of the values of the polynomials $u(x)$ and $v(x)$, $\bar{s}(x)$ and $\bar{w}(x) \bmod x^{n-h+1}$, and $q(x)$ and $t(x)$ at the N^{th} roots of 1 and for subtracting modulo x^n the polynomial $q(x)t(x)$ from $w(x)$.

In the special case of squaring in A_t , we save an $\text{FFT}(d)$ because $u(x) = v(x)$.

Finally, computing a reciprocal in the algebra A_t is reduced to application of the Euclidean algorithm or to solving a Sylvester (resultant) linear system of equations [23, Sections 2.7, 2.10]. This can be done by using $O(n \log^2 n)$ arithmetic operations but with possible numerical stability problems.

6. HORNER'S BASIS AND THE ARITHMETIC OPERATIONS IN THE ALGEBRA A_t

Cardinal in [16] proposes to represent the polynomials in the algebra A_t by using the basis $\{h_{n-i}(x) = (t(x) - (t(x) \bmod x^{i-1}))/x^i = t_n x^{n-i} + t_{n-1} x^{n-i-1} + \dots + t_i, i = 1, \dots, n-1\}$, which he calls *Horner's*. For a polynomial $f(x) = \sum_{i=0}^{n-1} f_i x^i = \sum_{i=0}^{n-1} y_i h_{n-i}(x)$ in A_t , its coefficient vectors in the monomial and Horner's bases are related via a lower triangular Toeplitz linear system of equations,

$$\begin{bmatrix} t_n & & & \\ \cdot & \ddots & & \\ & & \ddots & \\ t_2 & \cdot & \cdot & t_n \\ t_1 & t_2 & & t_n \end{bmatrix} \begin{bmatrix} y_0 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} f_{n-1} \\ \vdots \\ f_1 \\ f_0 \end{bmatrix}, \quad (6.1)$$

that is, $L(\mathbf{t}')\mathbf{y} = \mathbf{f}'$ where $\mathbf{y} = (y_i)_{i=0}^{n-1}$, $\mathbf{f}' = (f_{n-i})_{i=1}^{n-1}$, $\mathbf{t}' = (t_{n-i})_{i=0}^{n-1}$. For the transition from the monomial basis representation with the coefficient vector \mathbf{f} to the coefficient vector \mathbf{y} in Horner's basis, we first compute the first column $(L(\mathbf{t}'))^{-1}\mathbf{e}_0$ of the matrix $(L(\mathbf{t}'))^{-1}$ and then multiply this matrix by \mathbf{f}' . (The inverse $(L(\mathbf{t}'))^{-1}$ is a lower triangular Toeplitz matrix, completely defined by its first column [23, Section 2.5].) The computation of the vector $(L(\mathbf{t}'))^{-1}\mathbf{e}_0$ is the preprocessing stage; it does not involve the vector \mathbf{f}' .

Cardinal in [16] proposes two algorithms for squaring and multiplication in A_t in Horner's basis. Here is his squaring algorithm.

ALGORITHM 6.1. SQUARING IN A_t WITH RESPECT TO THE HORNER BASIS.

INPUT. The coefficient vector \mathbf{t} of the polynomial $t(x)$ and the vector \mathbf{y} defining a polynomial $f(x) \in A_t$ in the Horner basis.

OUTPUT. The vector \mathbf{w} defining the polynomial $(f(x))^2 \in A_t$ in the Horner basis.

COMPUTATIONS.

STAGE 1. Compute the vector \mathbf{c} being the convolution of \mathbf{y} and \mathbf{t} .

STAGE 2. Change the sign of the first n components of \mathbf{c} ; denote the resulting vector \mathbf{c}^* .

STAGE 3. Compute and output the vector \mathbf{w} formed by the n components from the n^{th} to the $(2n-1)^{\text{st}}$ in the $(3n-1)$ -dimensional vector \mathbf{z} which denotes the convolution of \mathbf{c}^* and \mathbf{y} .

Up to n additional sign changes at Stage 2 and selecting n components in the middle of the $(3n-1)$ -dimensional vector at Stage 3, the algorithm amounts to computing two convolutions; the cost of its performance is dominated by the cost of performing three forward and two inverse FFTs associated with the vectors \mathbf{t} , \mathbf{c} , \mathbf{c}^* , \mathbf{y} , and \mathbf{z} . The single inverse FFT at Stage 1 and the three FFTs at Stage 3 (one of them is the inverse FFT) are performed on the $2^{d^{\text{th}}}$ and $2^{D^{\text{th}}}$ roots of 1, respectively, for $d = \lceil \log_2(2n-1) \rceil$, $D = \lceil \log_2(3n-1) \rceil$; the FFT for the vector \mathbf{y} at Stage 1 is covered by the FFT for the same vector at Stage 3; the FFT for \mathbf{t} does not depend on \mathbf{y} and can be precomputed. The overall number of arithmetic operations involved in Algorithm 6.1 (not counting the cost of precomputing the FFT for \mathbf{t}) is slightly below $6\text{FFT}(d)$, that is, a little less than in Algorithm 5.1 for the same squaring problem in A_t . Thus, the repeated squaring in A_t can be rapidly performed in both monomial and Horner's bases. The results of numerical experiments and a theoretical argument in [16] indicate that this computation is more stable numerically in the Horner's than the monomial basis.

For the multiplication of two polynomials $u = u(x)$ and $v = v(x)$ in A_t with respect to Horner's basis, Cardinal specifies an algorithm which in addition to $O(n)$ arithmetic operations involves six FFTs on the $2^{D^{\text{th}}}$ roots of 1 and two FFTs on the $2^{d^{\text{th}}}$ roots of 1; then he points out that the same complexity bound (up to the term in $O(n)$) can be achieved by combining Algorithm 6.1 with the simple equation $4uv = (u+v)^2 - (u-v)^2$.

For completeness we next recall Cardinal's multiplication algorithm. It turns into Algorithm 6.1 wherever both input polynomials coincide with each other.

ALGORITHM 6.2. MULTIPLICATION IN A_t WITH RESPECT TO THE HORNER BASIS.

INPUT. The coefficient vector \mathbf{t} of the polynomial $t(x)$ and the vectors \mathbf{x} and \mathbf{y} defining two polynomials $f(x), g(x) \in A_t$ in the Horner basis.

OUTPUT. The vector \mathbf{w} defining the polynomial $f(x)g(x) \in A_t$ in the Horner basis.

COMPUTATIONS.

STAGE 1. Compute the vectors \mathbf{c} and \mathbf{d} being the convolutions of \mathbf{x} and \mathbf{t} and of \mathbf{y} and \mathbf{t} , respectively.

STAGE 2. Replace the last n components of \mathbf{c} and the first n components of \mathbf{d} with zeros; denote the resulting vectors \mathbf{c}^* and \mathbf{d}^* .

STAGE 3. Compute and output the vector \mathbf{w} formed by the n components from the n^{th} to the $(2n-1)^{\text{st}}$ in the $(3n-1)$ -dimensional vector $\mathbf{z} = \mathbf{u} - \mathbf{v}$ where \mathbf{u} and \mathbf{v} denote the convolutions of \mathbf{c}^* and \mathbf{y} and of \mathbf{d}^* and \mathbf{x} , respectively.

We call the root-finding approach in this and the previous sections *the DSeSC iteration* (cf. [14,16]), where D reflects Dandelin's initial contribution of the "Graeffe's" iteration.

7. INITIALIZATION POLICIES

The efficiency of Algorithm 4.1 can be enhanced with an appropriate initial choice of the polynomial $f(x)$ in A_t . Let us specify the choices which

- (a) simplify the root-finding by relaxing the negative affect from the multiple and clustered roots,

- (b) enable implicit deflation, and
- (c) simplify the final recovery of the root.

(a) By choosing $f(x) = t'(x)$ or, more generally, $f(x) = t'(x)g(x) \bmod t(x)$ for any polynomial $g(x)$, we ensure that $f(z_j) = 0$ as soon as z_j is a multiple root of $t(x)$ and that $f(z_j) \approx 0$ if z_j is in a cluster of the roots of $t(x)$. Then the term $(f(z_j))^m l_j(x)$ is dominated in the sum in Corollary 3.1, so that the influence of the multiple (and clustered) roots of $t(x)$ on the convergence of process (4.1) to a simple isolated root of $t(x)$ is suppressed. (For a random polynomial $g(x)$, property (3.1) is likely to hold provided a polynomial $t(x)$ has a simple isolated root.)

(b) If we have computed a root z_j of $t(x)$ and seek the next root, we may repeat the same process starting with $f(x) = \bar{g}(x)(1 - l_j(x)) \bmod t(x)$ for any fixed $\bar{g}(x) \in A_t$ because $l_j(z_j) = 1$ for all j . As soon as the second root z_k of $t(x)$ has been approximated, we may obtain the next root if we apply the same process starting with $f(x) = \bar{g}(x)(1 - l_j(x) - l_k(x)) \bmod t(x)$ for any fixed $\bar{g}(x) \in A_t$; indeed $l_j(x) + l_k(x) = 1$ for $x = z_j$ and $x = z_k$ for any j and any $k \neq j$. The process can be continued recursively. The latter choices of $f(x)$ are compatible with the previous one in Part (a) because we may choose $\bar{g}(x) = t'(x)g(x)$ for any fixed or random polynomial $g(x)$.

The explicit deflation of a root z via the division of $t(x)$ by $x - z$ only requires $2n - 2$ arithmetic operations, but the implicit deflation may have better numerical stability.

(c) As we mentioned, Cardinal in [16] contends that using Horner's basis representation improves numerical stability of the computation of $(f(x))^m \bmod t(x)$ with repeated squaring of $f(x)$ in A_t . He shows that the element $t'(z_j)l_j(x)$ of A_t has the representation $(1, z_j, z_j^2, \dots, z_j^{n-1})$ in Horner's basis. Then z_j is immediately obtained as the ratio of the first two basis coordinates of $l_j(x)$, and we may spare the division of $t(x)$ by $l_j(x)$. Furthermore, we may force convergence to the simple root z_j of $t(x)$ that most closely approximates a fixed complex value c provided all other roots lie farther from c . To achieve this, one may start with an element $f(x)$ of A_t represented by the vector $(1, c, c^2, \dots, c^{n-1})$ in Horner's basis because in this case $f(z_j) = t(c)/c - z_j$ [16].

The latter initialization technique simplifies and numerically stabilizes the stage of the final recovery of the root. Can we simultaneously relax the negative effect of the multiple and clustered roots? The above techniques are not compatible, not allowing us to yield the two improvements simultaneously, but in the next sections we achieve this goal by shifting to appropriate matrix methods for polynomial root-finding.

8. MATRIX REPRESENTATION OF THE ALGEBRA A_t

The elements $f(x)$ of A_t can be viewed as linear operators f on A_t ,

$$f : g(x) \rightarrow g(x)f(x), \quad \text{for all } g(x) \in A_t.$$

In [16], the matrix $F_t(f)$ of this linear operator in the monomial basis is called the *Frobenius matrix associated with the operator f* . In Horner's basis this operator has the matrix $F_t^\top(f)$ [16]. In particular, for $f(x) = x$, we have

$$F_t(f) = C = \begin{bmatrix} 0 & \cdots & 0 & -t_0^* \\ 1 & \ddots & & -t_1^* \\ & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -t_{n-1}^* \end{bmatrix}, \quad (8.1)$$

$t_i^* = t_i/t_n$, $i = 0, 1, \dots, n - 1$, that is, the Frobenius matrix of the operator of multiplication by x in A_t is the companion matrix of $t(x)$ in (2.1). The eigenvalues of C coincide with the roots z_1, \dots, z_n of $t(x)$. The algebra A_t has an isomorphic representation A_C where a polynomial $f(x) = \sum_{i=0}^{n-1} f_i x^i \in A_t$ is mapped into the matrix $F_t(f) = \sum_{i=0}^{n-1} f_i C^i$ with the first column filled with the coefficients f_0, \dots, f_{n-1} .

The following result of [16] is an immediate consequence of the Barnett factorization (see Proposition 2.9.2 of [24]).

THEOREM 8.1. For two polynomials, $t(x)$ in (2.1) and $f(x) \in A_t$, let \mathbf{f} and $\mathbf{y} = [y_0, y_1, \dots, y_{n-1}]^\top$ denote the coefficient vectors of $f(x)$ in the monomial and Horner's bases, respectively. Then we have

$$F_t(f) = L(\mathbf{f}) - L(\mathbf{t})L^\top(Z\mathbf{y}), \quad (8.2)$$

where $\mathbf{t} = [t_0, \dots, t_{n-1}]^\top/t_n$, the matrix $L(\mathbf{v})$ denotes the lower triangular Toeplitz matrix with the first column given by a vector \mathbf{v} , and Z is the down-shift matrix Z of (2.4).

Consequently, one may represent a matrix $F_t(f)$ in the algebra A_C by means of any of the vectors \mathbf{f} and \mathbf{y} by using the memory space n (cf. (6.1)) not counting the memory space for the storage of the $n + 1$ coefficients of the polynomial $t(x)$, which defines the algebra A_t . Due to the isomorphism between the algebras A_t and A_C , we immediately extend the algorithms for squaring and multiplication in A_t in Sections 5 and 6 to the same operations in A_C . Computing the reciprocal of a polynomial $f(x)$ in A_t is equivalent to the inversion of the matrix $F_t(f)$ in A_C . We observe that $F_t(f) - \delta I = F_t(f(x) - \delta)$ for a scalar δ and $f(x)$ in A_t ; furthermore, the representations of $f(x) - \delta$ and $f(x)$ in Horner's basis coincide with one another except that the coefficient y_{n-1} for $f(x)$ exceeds the one for $f(x) - \delta$ by δ/t_n (see (6.1)).

Based on representation (8.2) in the algebra A_C , we immediately simplify the final recovery of the root z_j from the scaled Lagrange polynomial $cl_j(x)$ for a scalar $c \neq 0$. Indeed (2.2) implies that $cl_j(F)(F - z_jI) = 0$. Therefore,

$$z_j = \frac{(l_j(F))_{0,1}}{(l_j(F))_{0,0}}, \quad (8.3)$$

where $(M)_{i,j}$ denotes the $(i, j)^{\text{th}}$ entry of a matrix M , $i = 0, 1, \dots, n-1$. We may perform repeated squaring in Horner's basis and define z_j in (8.3) only at the recovery stage, by combining (6.1) and (8.2). Thus, we solve both problems of the final recovery and handling multiple roots and root clusters if we apply the initialization rule (a) from Section 7 and apply equation (8.3) for the recovery.

9. THE POWER METHOD FOR THE MATRIX EIGENVALUES AND ITS APPLICATION TO POLYNOMIAL ROOTS

The classical power method [17, Section 7.3; 18, Section 2.1] enables us to approximate the eigenvalues of a matrix M . If $M = C = F_t(f)$ for $f(x) = x$, we approximate the roots of a polynomial $t(x)$. Let us recall this method.

Suppose M is a diagonalizable $n \times n$ matrix and $X = (\mathbf{x}_i)_{i=1}^n$ is the matrix of its right eigenvectors, so that $X^{-1}MX = \lambda = \text{diag}(\lambda_i)_{i=1}^n$, $M\mathbf{x}_i = \lambda_i\mathbf{x}_i$, $i = 1, \dots, n$. Also suppose that $\theta = \max_{i>1} |\lambda_i/\lambda_1| < 1$, $\mathbf{v} = \sum_{i=1}^n b_i\mathbf{x}_i$, $b_1 \neq 0$. Then for large k the vectors $b_1\lambda_1^k\mathbf{x}_1$ dominate in the sums $\sum_{i=1}^n b_i\lambda_i^k\mathbf{x}_i$ representing the vectors $\mathbf{v}_k = M^k\mathbf{v}$. One may estimate that the Rayleigh quotients

$$r_k = \frac{\mathbf{v}_k^\top M \mathbf{v}_k}{\mathbf{v}_k^\top \mathbf{v}_k} \quad (9.1)$$

approximate the dominant eigenvalue λ_1 with the error $|r_k - \lambda_1|$ in $O(\theta^k)$, whereas the scaled vectors $\mathbf{v}_k/\|\mathbf{v}_k\|$ approximate the associated dominant eigenvector $\mathbf{x}_1/\|\mathbf{x}_1\|$. Application of this method to the matrices $M - \delta I$ and $(M - \delta I)^{-1}$ instead of M (where δ is a fixed scalar) yields the approximations to the dominant eigenvalues $\lambda_j - \delta$ and $(\lambda_k - \delta)^{-1}$ of these matrices as well as the associated eigenvectors \mathbf{x}_j and \mathbf{x}_k (provided the eigenvalues are unique). The pairs $(\lambda_j, \mathbf{x}_j)$ and $(\lambda_k, \mathbf{x}_k)$ are the eigenpairs of M , such that λ_j is the farthest from δ and λ_k is the closest to δ among all eigenvalues in the spectrum of M .

Now let $M = C$ for C in (8.1). Then the power iteration should converge to a root of $t(x)$. By applying explicit or (better) implicit deflation (see the recipe for the latter in Section 7), we may compute the other roots of $t(x)$ recursively. We accelerate the convergence by applying repeated

squaring of the initial matrix $F_t(f)$. The Cardinal's techniques in Section 8 (cf. (8.2)) combined with Algorithms 4.1, 5.1, and 6.1 enable us to perform each squaring fast, by using $O(n \log n)$ arithmetic operations per squaring. We combine the first two recipes of the initialization in Section 7 to relax the effect of the multiple and clustered roots on the convergence, to yield bound (3.1), and to allow implicit deflation. The third initialization recipe in Section 7 is not compatible with the first two but is not needed anymore because of our alternative recovery recipe based on (8.3) or (9.1). The proponents of the power method should be pleased to observe that recipe (8.3) coincides with using the Rayleigh quotient (9.1) for $\mathbf{v}_k = F^k \mathbf{e}_0$.

A choice of the normalization scalars $n_h = \text{trace}((F_t(f))^H)$ in (4.1) has been pointed out in [16]; another effective choice is $n_h = \mathbf{v}^\top F_t(f)^H \mathbf{v} / \mathbf{v}^\top \mathbf{v}$ for $H = 2^h$ and a vector \mathbf{v} which can be fixed or random and may vary or not vary with h . The computation of n_h takes $O(n \log n)$ arithmetic operations for each h .

To apply the shifted or shifted inverse iteration, we may begin with $F_t(f) - \delta I$ or $(F_t(f) - \delta I)^{-1}$, or we may do this implicitly by replacing the polynomial $t(x)$ with $s(x) = t(x - \delta)$ or the reverse polynomial $\bar{s}(x)$ and applying the same original algorithm to A_s or $A_{\bar{s}}$ (cf. (2.3)).

If the coefficients of $t(x)$ are real, we automatically yield the dual complex conjugate root for any nonreal root.

We call the resulting root-finder *the amended DSeSC power iteration*.

10. TO THE MULTIPLE ROOTS AND ROOT CLUSTERS VIA THE (HIGHER-ORDER) DERIVATIVES

The amended DSeSC power algorithm recursively approximates simple roots of $t(x)$, beginning with the better isolated roots at which the derivative takes on the absolutely larger values. For the remaining multiple roots, one may apply the same algorithm to the derivative $t'(x)$ and then, recursively, higher-order derivatives $t^{(h)}(x)$, $h = 1, 2, \dots, n-1$ (cf. [25, Section 9.2]), incorporating also implicit or explicit deflation. If one has computed an approximation \tilde{x} to a root of the polynomials $t^{(h)}(x)$ but has chosen not to incorporate deflation, then one should test if \tilde{x} also approximates a root of $t(x)$. Instead of the customary criteria in the test, one may apply a few steps of a known root-finder for $t(x)$, e.g., one of such root-finders in [25] or [19] initialized at \tilde{x} or Newton's iteration $x_0 = \tilde{x}$, $x_{i+1} = x_i - m(t(x_i)/t'(x_i))$ where $i = 0, 1, \dots$, and where m is the guessed multiplicity of the root of $t(x)$.

The latter approach can be applied to approximate the roots of multiplicity exactly m : apply the algorithm to approximate the roots of $t^{(m-1)}(x)$ with the initial choice of $f(x) = t^{(m)}(x)g(x)$; then test if the computed approximation is also a root of $t^{(i)}(x)$ for $i = 0, 1, \dots, m-2$ (probabilistically we may just test if it is a root of the polynomial $\sum_{i=0}^{m-2} h_i t^{(i)}(x)$ for random h_i). Similar tricks enable us to begin with approximating the roots in the clusters made up of exactly, at least, or at most m roots.

REFERENCES

1. V.Y. Pan, Solving a polynomial equation: Some history and recent progress, *SIAM Review* **39** (2), 187–220, (1997).
2. B. Mourrain and V.Y. Pan, Multivariate polynomials, duality and structured matrices, *J. of Complexity* **16** (1), 110–180, (2000).
3. J.M. McNamee, Bibliography on roots of polynomials, *J. Comp. Appl. Math.* **47**, 391–394, (1993).
4. J.M. McNamee, A supplementary bibliography on roots of polynomials, *J. Computational Applied Mathematics* **78** (1), (1997); <http://www.elsevier.nl/homepage/sac/cam/mcnamee/index.html>.
5. J.M. McNamee, An updated supplementary bibliography on roots of polynomials, *J. Computational Applied Mathematics* **110**, 305–306, (1999).
6. J.M. McNamee, A 2000 updated supplementary bibliography on roots of polynomials, *J. Computational Applied Mathematics* **142**, 433–434, (2000).
7. D.A. Bini, L. Gemignani and V.Y. Pan, Improved initialization of the accelerated and robust QR-like polynomial root-finding, *Electronic Transaction on Numerical Analysis* (to appear).

8. V.Y. Pan, New reduction of the algebraic eigenproblem to polynomial root-finding via similarity transforms to generalized companion matrices of the characteristic polynomial, In *SIAM Conference on Linear Algebra*, Williamsburg, VA, (July 2003).
9. V.Y. Pan, Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros, In *Proc. 27th Ann. ACM Symp. on Theory of Computing*, pp. 741–750, ACM Press, New York, (May 1995).
10. V.Y. Pan, Optimal and nearly optimal algorithms for approximating polynomial zeros, *Computers Math. Applic.* **31** (12), 97–138, (1996).
11. V.Y. Pan, Univariate polynomials: Nearly optimal algorithms for factorization and rootfinding, *Journal of Symbolic Computations* **33** (5), 701–733, (2002).
12. D.A. Bini, L. Gemignani and V.Y. Pan, QR-like algorithms for generalized semiseparable matrices, Technical Report 1470, Department of Math., University of Pisa, Pisa, Italy, (July 2003).
13. A.S. Householder, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill, New York, (1970).
14. J. Sebastiao e Silva, Sur une méthode d'approximation semblable a celle de Graeffe, *Portugal Math.* **2**, 271–279, (1941).
15. A.S. Householder, Generalization of an algorithm by Sebastiao e Silva, *Numerische Math.* **16**, 375–382, (1971).
16. J.P. Cardinal, On two iterative methods for approximating the roots of a polynomial, In *Proceedings of AMS-SIAM Summer Seminar: Mathematics of Numerical Analysis: Real Number Algorithms*, (Edited by J. Renegar, M. Shub and S. Smale), Park City, UT, 1995; *Lectures in Applied Mathematics, Volume 32*, pp. 165–188, American Mathematical Society, Providence, RI, (1996).
17. G.H. Golub and C.F. Van Loan, *Matrix Computations*, Third Edition, Johns Hopkins University Press, Baltimore, MD, (1996).
18. G.W. Stewart, *Matrix Algorithms, Vol. II, Eigensystems*, SIAM, Philadelphia, PA, (1998).
19. D.A. Bini, L. Gemignani and V.Y. Pan, Inverse power and Durand/Kerner iteration for univariate polynomial root-finding, *Computers Math. Applic.* **47** (2/3), 447–459, (2004).
20. R.M. Corless, P.M. Gianni, B.M. Trager and S.M. Watt, The singular value decomposition for polynomial systems, In *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pp. 195–207, ACM Press, New York, (1995).
21. V.Y. Pan, Computation of approximate polynomial GCDs and an extension, *Information and Computation* **167**, 71–85, (2001).
22. A.L. Toom, The complexity of a scheme of functional elements realizing the multiplication of integers, *Soviet Mathematics Doklady* **3**, 714–715, (1963).
23. V.Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston, MA, (2001).
24. D. Bini and V.Y. Pan, *Polynomial and Matrix Computations*, Vol. 1: Fundamental Algorithms, Birkhäuser, Boston, MA, (1994).
25. V.Y. Pan, B. Murphy, R.E. Rosholt, Y. Tang and X. Wang, Root-finding via eigen-solving, perturbations and realizations, Preprint, (2004).

The following REFERENCES should be changed:

4. J.M. McNamee, ..., {\bf 78}, 1, (1997); http...,
that is, replace "(1)" with ",1"

6. J.M. McNamee, A 2002 updated..., 433-434, (2002).
that is, twice replace 2000 with 2002

7. D.A. Bini,... Analysis, {\bf 17}, 195-205, (2004).

8. V.Y. Pan, B. Murphy, R.E. Rosholt, Y. Tang, X. Wang and X. Yan,
Root-finding via eigen-solving, preprint, (2005).
That is, replace the former reference 8 with the updated reference 25.
This means that the new reference 8 should be cited wherever any of the
former references 8 and 25 were cited.

12. D.A. Bini, L. Gemignani and V.Y. Pan, Fast and stable QR eigenvalue
algorithms for generalized companion matrices and secular equations,
{\em Numerische Math.}, in print. (Also see Technical Report 1470,
Department of Math., University of Pisa, Pisa, Italy, (July 2003).)

Other Corrections

Page 4, the line between equations (4.1) and (4.2) should be read as
follows:

"for the positive scalar n_h such that"
that is, delete the comma after n_h

Page 4, the next line after equation (4.2) should be read as follows:
"and for $h=0, \dots, k$. If ..."
that is, replace $k-1$ with k

Page 4, line 4 from bottom:
replace "(5.4)" with "(5.3)"

Page 5, line 2 should be read as follows:
"INPUT. Three polynomials, $u(x)$ and $v(x)$, both of degrees of at most
 $n-1$, and $t(x)$ satisfying (2.1)."

Page 5, line 3 should be read as follows:
"OUTPUT. The polynomial $r(x)$ in (5.1)-(5.3)."
that is, insert -(5.3) after (5.1)

Page 5, line 7: insert "and from" after "transition to"

Page 5, line 2 in section 6:

replace " $i=1, \dots, n-1$," with " $i=1, \dots, n$,"

Page 5, line 3 in section 6:

replace " $\sum_{i=0}^{n-1} y_i$ " with " $\sum_{i=1}^n y_i$ "

Page 5, line 2 from bottom should be read as follows:

"basis. They actually amount to pre-multiplication of equation (8.2) in our Theorem 8.1 by the respective coefficient vector in Horner's basis. Here is ..."

Page 6, line 1: replace "the vector $\{\mathbf{y}\}$ defining a polynomial" with "the coefficient vector $\{\mathbf{y}\}$ of a polynomial"

Page 6, line 3: replace "The vector $\{\mathbf{w}\}$ defining the polynomial" with "The coefficient vector $\{\mathbf{w}\}$ of the polynomial"

Page 6, the INPUT line of Algorithm 6.2 : replace "the vectors $\{\mathbf{x}\}$ and $\{\mathbf{y}\}$ defining two polynomials" with "the coefficient vectors $\{\mathbf{x}\}$ and $\{\mathbf{y}\}$ of two polynomials"

Page 6, the OUTPUT line of Algorithm 6.2 : replace "the vector $\{\mathbf{w}\}$ defining the polynomial" with "the coefficient vector $\{\mathbf{w}\}$ of the polynomial"

Page 7, line 8: add z_j after "isolated root"

Page 7, in the beginning of line 5 of Part (c):

replace "of $l_j(x)$, and we may" with "of $(f(x))^m$, and we can"

Page 7, two lines below: replace "one may" with "we can"

Page 8, line 15: replace "coefficient y_{n-1} " with "coefficient y_n "

Then at the end of the line, after "(see (6.1))." add:

"Actually, Algorithms 6.1 and 6.2 just pre-multiply equation (8.2) by the vector $\{\mathbf{y}\}$."

Page 8, lines 21-22 should be read as follows:

"squaring in Horner's basis and (as soon as the ratio in equation (8.3) stabilizes) approximate a root with z_j in (8.3). (Here we also apply equations (6.1) and (8.2) for the transition between the monomial and Horner's bases). Thus, we solve..."

Page 8, the last line in Section 9 should be read as follows:

"the recovery. The ratio (8.3) can never stabilize if for $k > 1$ there are

k distinct absolutely largest roots of the polynomial $t(x)$ (e.g., a pair of complex conjugate roots where $t(x)$ has real coefficients). The invariant k -dimensional eigenspace of the matrix $F_t(x)$ associated with these roots, however, stabilizes; for smaller k we can readily approximate an orthogonal basis for this eigenspace and then the eigenvalues/roots themselves (cf. [18, Section 4.4]). For larger k , this computation becomes expensive, but for any k we have good chances to yield a unique absolutely largest root if we randomly shift the variable x . We can improve numerical stability by replacing shifts with transforms of the unit circle centered in the origin into itself. These transforms cannot help only if the k largest roots lie on this circle."

Besides,

I wish to replace "may" by "can"

on Page 7, lines 9, 11, 15, 17, and 22 (twice)

on Page 8, lines 7, 20, 33 (12 from bottom), and 47 (second from bottom)

on Page 9, lines 13 and 14