Real Polynomial Root-finding by Means of Matrix and Polynomial Iterations*

Victor Y. Pan [1,2],[a] and Liang Zhao[2],[b]

[1] Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
[2] Ph.D. Programs in Mathematics and Computer Science
The Graduate Center of the City University of New York
New York, NY 10036 USA
[a] victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/
[b] lzhao1@gc.cuny.edu

Abstract
Univariate polynomial root-finding is a classical subject, still important for modern computing. Frequently one seeks just the real roots of a polynomial with real coefficients. They can be approximated at a low computational cost if the polynomial has no nonreal roots, but for high degree polynomials, nonreal roots are typically much more numerous than the real ones. The challenge is known for long time, and the subject has been intensively studied. Nevertheless, we produce some novel ideas and techniques and obtain dramatic acceleration of the known algorithms. To achieve our progress we exploit the correlation between the computations with matrices and polynomials, randomized matrix computations, and complex plane geometry, extend the techniques of the matrix sign iteration, and use the structure of the companion matrix of the input polynomial. The results of our extensive tests with benchmark polynomials and random matrices are quite encouraging. In particular in our tests the number of iterations required for convergence of our algorithms grew very slowly (if at all) as we increased the degree of the univariate input polynomials and the dimension of the input matrices from 64 to 1024.

Keywords: Polynomials, Real roots, Matrices, Matrix sign iteration, Companion matrix, Frobenius algebra, Square root iteration, Root squaring

1 Introduction

Assume a univariate polynomial of degree $n$ with real coefficients,

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - x_j), \quad p_n \neq 0,$$

which has $r$ real roots $x_1, \ldots, x_r$ and $s = (n - r)/2$ pairs of nonreal complex conjugate roots. In some applications, e.g., to algebraic and geometric optimization, one seeks only the $r$ real roots, which make up just a small fraction of all roots. This is a well studied subject (see [10] Section

*This work has been supported by NSF Grant CCF 1116736 and PSC CUNY Award 67699-00 45. Some of its results have been presented at CASC 2014
10.3.5], [39], [44], and the bibliography therein), but the most popular packages of subroutines for root-finding such as MPSolve 2000 [5], Eigensolve 2001 [17], and MPSolve 2012 [9] approximate the \ r real roots about as fast and as slow as all the \ n complex roots. It can be surprising, but we present some novel methods that accelerate the solution by a factor of \ n/r, which means dramatic speed up in the cited important applications.

The springboard for our real root-finders is the matrix sign iterations, which we apply to the companion matrix of an input polynomial. It is a well known technique [20], but never used in this direction. We show that it is particularly efficient for our purpose of real root-finders, and we combine it with various old and new techniques that support fast convergence of the iterations and their numerical validity.

Our tests for benchmark polynomials confirm the efficiency of this approach. In particular, the number of iterations was typically quite small and grew very slowly (if it grew at all) as the polynomial degree increased.

Some of our techniques are of potential independent interest, e.g., our numerical stabilization in Section 3.3, our exploitation of matrix functions and randomized matrix computations in Algorithm 3.4, and the combination of our maps of the complex plane and the rational transformations of the variable and the roots throughout the paper. Some of our algorithms (e.g., the ones of Section 3.4) take advantage of combining the power of operating with matrices and polynomials. This demonstrates once again the value of synergistic combinations of this kind, which we have been advocating since [26] and [6].

We organize our paper as follows. In the next section we cover the background for our algorithms based on the modified matrix sign iteration and its extensions. We present a variety of these algorithms in Section 3. In Section 4, which is the contribution of the second author, we cover the results of our numerical tests. In the Appendix we cover some auxiliary results.

2 Basic Definitions and Results for Matrix Computations and for Root-finding by Means of Eigen-solving

Hereafter “flop” stands for “arithmetic operation”.

2.1 Some Basic Definitions for Matrix Computations

\( M^T = (m_{ji})_{i,j=1}^{m,n} \) is the transpose of a matrix \( M = (m_{ij})_{i,j=1}^{m,n} \). \( M^H \) is its Hermitian transpose.

\( ||M|| \) is its spectral norm \( ||M||_2 \).

\( I = I_n \) is the \( n \times n \) identity matrix.

\( \text{diag}(b_1, \ldots, b_s) \) is the \( s \times s \) diagonal matrix with the diagonal entries \( b_1, \ldots, b_s \).

\( \mathcal{R}(M) \) is the range of a matrix \( M \), that is, the linear space generated by its columns.

A matrix of full column rank is a matrix basis of its range.

A matrix \( Q \) is unitary if \( Q^HQ = I \) or \( QQ^H = I \).

\((Q, R) = (Q(M), R(M))\) for an \( m \times n \) matrix \( M \) of rank \( n \) denotes a unique pair of unitary \( m \times n \) matrix \( Q \) and upper triangular \( n \times n \) matrix \( R \) such that \( M = QR \) and all diagonal entries of the matrix \( R \) are positive [18, Theorem 5.2.3].

\( M^+ \) is the unique Moore–Penrose pseudo inverse of \( M \) [18, Section 5.5.2], equal to \( M^H \) if and only if the matrix \( M \) is unitary.

An \( m \times n \) matrix \( M \) has an \( n \times m \) left inverse matrix \( X = M^{(l)} \) such that \( XM = I_n \) if and only if it has full column rank \( n \). In this case \( M^+ \) is a left inverse. The left inverse is unique if and only if \( M \) is a nonsingular matrix, and then \( M^{(l)} = M^{-1} \).

The \( \epsilon \)-rank of a matrix \( M \) is the minimal rank of the matrices in its \( \epsilon \)-neighborhood. Numerical rank \( \text{rank}(M) \) is the \( \epsilon \)-rank where \( \epsilon \) is small in context.

Definition 2.1. Eigenvalues, eigenvectors and eigenspaces.

- \( S \) is an invariant subspace of a square matrix \( M \) if \( MS = \{ Mv : v \in S \} \subseteq S \).
• A scalar $x$ is an eigenvalue of a matrix $M$ associated with an eigenvector $v$ if $Mv = xv$.

• An eigenvalue $x$ of a matrix $M$ is a root of the characteristic polynomial $\det(xI - M)$. The multiplicity of this root is the algebraic multiplicity of the eigenvalue $x$, denoted $am(x)$.

• The set $\mathcal{X}(M)$ of all eigenvalues of a matrix $M$ is called its spectrum.

• The eigenvectors associated with an eigenvalue $x$ or with any set of the eigenvalues $\mathcal{X} \in \mathcal{X}(M)$ form the invariant spaces $\mathcal{S}(M, x)$ and $\mathcal{S}(M, \mathcal{X})$, respectively, called the eigenspaces associated with the eigenvalue $x$ and the set $\mathcal{X}$ of eigenvalues, respectively.

• An eigenvalue $x$ of a matrix $M$ as well as a set of eigenvalues $\mathcal{X}$ are dominant if they are absolutely larger than all the other eigenvalues. The eigenspaces associated with dominant eigenvalues or sets of eigenvalues are also called dominant.

• The dimension $gm(x) = \dim(\mathcal{S}(M, x))$ is the geometric multiplicity of $x$, never exceeding $am(x)$. An eigenvalue $x$ is simple if $gm(x) = 1$.

### 2.2 The Companion Matrix and the Frobenius Algebra

Let $e_n^T = (0, 0, \ldots, 0, 1)$ denote the $n$th coordinate vector and write $p = (p_i/p_n)_{i=0}^{n-1}$.

\[
Z = C_0 = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & \ddots & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix} \quad \text{and} \quad C_p = \begin{pmatrix} 0 & -p_0/p_n & \cdots & 0 \\ 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 1 & -p_{n-1}/p_n \end{pmatrix} = Z - e_n^T p. \tag{2.1}
\]

$Z$ is the down-shift matrix. $C_p$ is the companion matrix of the polynomial $p(x)$ of (1.1), which is the characteristic polynomial of this matrix. Hence real root-finding for the polynomial $p(x)$ turns into real eigen-solving for this matrix.

$Z(v) = (v_{i-1})_{i=1}^n$ for a vector $v = (v_i)_{i=1}^n$ and for $v_0 = 0$.

**Theorem 2.1.** (The Cost of Computations in the Frobenius Matrix Algebra.) The companion matrix $C_p \in \mathbb{C}^{n \times n}$ of a polynomial $p(x)$ of (1.1) generates the Frobenius matrix algebra. (i) One needs $O(n)$ flops for addition, $O(n \log(n))$ for multiplication, and $O(n \log^2(n))$ for inversion in this algebra. One needs $O(n \log(n))$ flops to multiply a matrix in this algebra by a vector. (ii) Numerically stable inversion of a matrix in the Frobenius matrix algebra can be performed by using $O(n \log(n))$ flops.

**Proof.** The estimates of part (i) can be found in [12] and [33]. Part (ii) has been supported by the algorithms of [30] accelerated by a factor of $\log(n)$ in [30] Section 9.8. \qquad \square

### 2.3 Decreasing the Size of an Eigenproblem

The Power Method [18, Section 7.3.1] computes the vector $M^k v$ for a random vector $v$ and a sufficiently large integer $k$. If an eigenvalue $x$ is dominant and simple, then the 1-dimensional vector space $\{tM^k v\}$ for $t \in \mathbb{C}$ is expected to approximate the eigenspace associated with this eigenvalue. One can choose $k = 1$ if the domination of the eigenvalue $x$ in the spectrum of $M$ is strong. Let us extend the Power Method for $k = 1$ to the approximation of a strongly dominant eigenspace of dimension $r$.

**Algorithm 2.1.** Approximation of the dominant eigenspace.

**INPUT:** an $n \times n$ matrix $M$, the dimension $r$ of its dominant eigenspace $U$, $0 < r < n$, and two tolerance bounds $\tau$ and $\epsilon$. 

---

3
OUTPUT: FAILURE (with a low probability) or a unitary matrix $U$ whose range approximates the eigenspace $U$.

Computations:

1. Apply the randomized algorithm of [19], which at first generates a standard Gaussian random $n \times r_+$ matrix $G$ for a proper integer $r_+ > r$ and then computes the matrix $H = MG$ and the numerical rank $\text{nrank}(H)$.

2. Unless $\text{nrank}(H) = r$, reapply the algorithm of [19] up to $\tau$ times until the equation $\text{nrank}(H) = r$ is observed. If it is never observed, output FAILURE (this occurs with a probability near 0).

3. If $\text{nrank}(H) = r$, then compute the QR factorization $H = Q(H)R(H)$, output an $n \times r$ unitary matrix $U$ approximating the first $r$ columns of the matrix $Q(H)$, and stop. (The analysis in [19, Section 4], [37, Section 7.4] shows that, with a probability close to 1, the columns of the matrix $U$ closely approximate a unitary basis of the invariant space $U$ and that $||M - UU^HM|| \leq \epsilon ||M||$. The latter bound would certify correctness of the output.)

The arithmetic cost of performing the algorithm is $O(n^2 r_+)$, but decreases to $O(nr_+ (r_+ + \log(n))$ for $M = C_P$ by virtue of Theorem 2.1. It increases by a factor of $\log(r)$ if the dimension $r$ of the eigenbasis $U$ is not available, but is computed by using binary search that begins with recursive doubling of the candidate integer values 1, 2, 4, etc. The algorithm generates $nr_+$ random parameters, but its modification using the structured (so called SRFT) multipliers $H$ involves only $n$ such parameters (see [19] Section 11 and [37] Section 7.5).

Remark 2.1. Actually the algorithm of [19] works even where the input includes an upper bound $r_+$ on the dimension $r$ of the dominant eigenspace $U$, rather than the dimension itself, and then the algorithm can compute this dimension $r$ within the above computational cost as by-product. (The integer $r = \text{nrank}(H)$ can be obtained from rank revealing QR factorization of the matrix $H$.)

Now suppose that we have an eigenspace generated by $r$ eigenvalues of an $n \times n$ matrix. Then the following simple theorem (extending the recipe of the Rayleigh quotients) enables us to approximate these eigenvalues as the eigenvalues of an auxiliary $r \times r$ matrix.

Theorem 2.2. (Decreasing the Eigenproblem Size to the Dimension of an Invariant Space, cf. [48, Section 2.1].)

Suppose that $M \in \mathbb{C}^{n \times n}$, $U \in \mathbb{C}^{n \times r}$, and the matrix $U$ has full column rank $r \leq n$ and generates the space $U = \mathcal{R}(U)$. Then

(i) $U$ is an invariant space of $M$ if and only if there exists a matrix $L \in \mathbb{C}^{k \times k}$ such that $MU = UL$ or equivalently if and only if $L = U^{(l)}MU$,

(ii) $\chi(L) \subseteq \chi(M)$,

(iii) $MUv = xUv$ if $Lv = xv$,

(iv) the matrix $L$ is unique, that is, its choice is independent of the choice of a matrix $U$ and its left inverse $U^{(l)}$, and so $L = U^{H}MU$ for a unitary matrix $U$.

The algorithm and the theorem enable us to approximate the $r$ real eigenvalues of a matrix as the $r$ dominant eigenvalues of an auxiliary matrix. Theorems 2.2 and 2.3 (below) together suggest a direction to such a reduction, and we achieve it in Sections 3.1 and 5.2

2.4 Matrix Functions and Eigenspaces

Theorem 2.3. (The Eigenproblems for a Matrix and Its Function.)

Suppose that $M$ is a square matrix and a rational function $f(x)$ is defined on its spectrum.

(i) Then $f(M)v = f(x)v$ if $Mv = xv$.

(ii) Let $Y_f(x)$ denote the set $\{y: y = f(x)\}$ for a complex $x$ and let $U = U_{\mu, f}$ be the eigenspace of the matrix $f(M)$ associated with its eigenvalue $\mu$. Then this is an invariant space of the matrix $M$ generated by its eigenspace associated with all its eigenvalues $x$ such that $f(x) = \mu$.

(iii) The space $U$ has dimension 1 and is associated with a single eigenvalue of $M$ if $\mu$ is a simple eigenvalue of $f(M)$.
Proof. We readily verify part (i), which implies parts (ii) and (iii).

Remark 2.2. The matrix $Z^k$ for $1 \leq k \leq n$ has the single eigenvalue 0 satisfying $am(0) = n$ and $gm(0) = k$, and so $dim(U_{0,f}) = k$ for $M = Z$, $f(x) = x^k$, and $k = 1, \ldots, n$.

Suppose that we have computed a matrix basis $U \in C^{n \times r}$ for an invariant space $U$ of a matrix function $f(M)$ of an $n \times n$ matrix $M$. By virtue of Theorem 2.3, this is a matrix basis of an invariant space of the matrix $M$. We can first compute a left inverse $U^{(l)}$ or the orthogonalization $Q = Q(U)$ and then approximate the eigenvalues of $M$ associated with this eigenspace as the eigenvalues of the $r_+ \times r_+$ matrix $L = U^{(l)} MU = Q^H MQ$ (cf. Theorem 2.2).

Remark 2.3. Root Refinement and the Selection of Real Roots.

In the following we seek $r$ real eigenvalues of a matrix $M$, but in the presence of rounding errors our numerical algorithms approximate both $r$ real and $r_+ - r$ nearly real eigenvalues of the matrix $M$ for some $r_+ \geq r$. These eigenvalues, however, are the roots of $p(x)$ and, under some mild assumptions about the isolation of every such root from the $n - 1$ other roots, we can refine their approximations very fast (cf. Theorems 1.1 and 4.4). Then we can readily select the $r$ real eigenvalues among the $r_+$ real and nearly real ones.

If $r = 1$, then the matrix $U$ turns into an eigenvector $u$, shared by the matrices $f(M)$ and $M$, while the matrix $L$ turns into the the Rayleigh Quotient $u^T MU$ or the simple quotient $(MU)_{1}/u_1$ for any $i$ such that $u_i \neq 0$.

2.5 Some Maps in the Frobenius Matrix Algebra

Part (i) of Theorem 2.3 implies that for a polynomial $p(x)$ of $1$ and a rational function $f(x)$ defined on the set $\{x_i\}_{i=1}^n$ of its roots, the rational matrix function $f(C_p)$ has the spectrum $\lambda(f(C_p)) = \{f(x_i)\}_{i=1}^n$. In particular, the maps

$$C_p \to C_p^{-1}, \quad C_p \to aC_p + bI, \quad C_p \to C_p^2, \quad C_p \to \frac{C_p + C_p^{-1}}{2}, \quad \text{and} \quad C_p \to \frac{C_p - C_p^{-1}}{2}$$

induce the maps of the eigenvalues of the matrix $C_p$, and thus induce the maps of the roots of its characteristic polynomial $p(x)$ given by the equations

$$y = 1/x, \quad y = ax + b, \quad y = x^2, \quad y = 0.5(x + 1/x), \quad \text{and} \quad y = 0.5(x - 1/x),$$

respectively. By using the reduction modulo $p(x)$, define the five dual maps

$$y = (1/x) \mod p(x), \quad y = ax + b \mod p(x), \quad y = x^2 \mod p(x),$$

$$y = 0.5(x + 1/x) \mod p(x), \quad \text{and} \quad y = 0.5(x - 1/x) \mod p(x),$$

where $y = y(x)$ denotes polynomials. Apply the two latter maps recursively, to define two iterations with polynomials modulo $p(x)$ as follows, $y_0 = x$, $y_{h+1} = 0.5(y_h + 1/y_h) \mod p(x)$ and $y_0 = x$, $y_{h+1} = 0.5(y_h - 1/y_h) \mod p(x)$, $h = 0, 1, \ldots$. More generally, define the iteration $y_0 = x$, $y_{h+1} = ay_h + b/y_h \mod p(x)$, $h = 0, 1, \ldots$, for any pair of scalars $a$ and $b$.

3 Real Root-finders with Modified Matrix Sign Iteration.

Variations and Extensions

In this section we present some efficient numerical real root-finders based on modification of the matrix sign iteration applied to the companion matrix of the input polynomial.
3.1 A Modified Matrix Sign Iteration

Our first algorithm approximates the real roots of a polynomial \( p(x) \) as the real eigenvalues of the companion matrix \( C_p \). It applies the matrix iterations

\[
M_0 = C_p, \quad M_{h+1} = 0.5(M_h - M_h^{-1}) \quad \text{for } h = 0, 1, \ldots
\]

which modify the matrix sign iterations \( \hat{M}_{h+1} = 0.5(\hat{M}_h + \hat{M}_h^{-1}) \) (cf. [20]).

For every eigenvalue \( x_j \) of the matrix \( M_0 = C_p \), define its trajectory made up of the eigenvalues of the matrices \( M_h \), being its images in the maps \( M_h \rightarrow M_h \), for \( h = 1, 2, 3, \ldots \). More generally the modified M"obius iterations below define a trajectory initiated at any complex point \( x(0) \).

\textbf{Theorem 3.1.} (Convergence of the modified M"obius Iterations.) Fix a complex \( x = x(0) \) and define the modified M"obius iterations

\[
x^{(h+1)} = \frac{1}{2}(x^{(h)} - 1/x^{(h)}) \quad \text{for } h = 0, 1, \ldots
\]

(i) The values \( x^{(h)} \) are real for all \( h \) if \( x(0) \) is real.
(ii) \(|x^{(h)} - \text{sign}(x)\sqrt{-1}| \leq \frac{2r_h}{1-r_h^2} \) for \( r = |x - \text{sign}(x)| \) and \( h = 0, 1, \ldots \).

\textbf{Proof.} Part (i) is immediately verified. Part (ii) readily extends the similar estimate on [7], page 500].

Theorem 3.1 implies the following result.

\textbf{Corollary 3.1.} As \( h \rightarrow \infty \), the trajectories of the \( 2s \) nonreal eigenvalues of \( M_0 = C_p \) converge to \( \pm \sqrt{-1} \) with the quadratic rate of convergence right from the start, whereas the trajectories of the \( r \) real eigenvalues are real for all \( h \).

\textbf{Algorithm 3.1.} Matrix sign iteration modified for real eigen-solving.

\textbf{INPUT:} two integers \( n \) and \( r \), \( 0 < r < n \), and the coefficients of a polynomial \( p(x) \) of equation (1.1). 

\textbf{OUTPUT:} approximations to the real roots \( x_1, \ldots, x_r \) of the polynomial \( p(x) \) or FAILURE with a probability close to 0.

\textbf{COMPUTATIONS:}

1. Write \( M_0 = C_p \) and recursively compute the matrices \( M_{h+1} \) of (3.1) for \( h = 0, 1, \ldots \) (cf. Corollary 3.1).
2. Fix a sufficiently large integer \( k \) and compute the matrix \( M = M_k^2 + I_n \).
   (By extending Corollary 3.1 observe that the map \( M_0 = C_p \rightarrow M \) sends all nonreal eigenvalues of \( C_p \) into a small neighborhood of the origin 0 and sends all real eigenvalues of \( C_p \) into the ray \( \{x : x \geq 1 \} \).)
3. Apply our randomized Algorithm 2.1 in order to approximate a unitary matrix \( U \) whose columns form a basis for the eigenspace associated with the \( r \) dominant eigenvalues of the matrix \( M \). (By virtue of Theorem 2.3 this is expected to be the eigenspace associated with the real eigenvalues of the matrix \( C_p \), although with a probability close to 0 the algorithm can output FAILURE and stop the computations.)
4. Compute and output approximations to the \( r \) eigenvalues of the \( r \times r \) matrix \( L = U^H C_p U \).
   (They approximate the \( r \) real eigenvalues of the matrix \( C_p \) by virtue of Theorem 2.3 and consequently approximate the \( r \) real roots of the polynomial \( p(x) \).)

Stages 1 and 2 involve \( O(kn \log^2(n)) \) flops by virtue of Theorem 2.1. Stage 3 adds \( O(n^2r) \) flops and the cost \( a_{rn} \) of generating \( n \times r \) standard Gaussian random matrix. Add \( O(r^3) \) flops performed at Stage 4 and arrive at the overall arithmetic cost bound \( O((kn \log^2(n) + m^2) + a_{rn}) \).
**Remark 3.1.** (Counting Real Eigenvalues.) The binary search can produce the number of real eigenvalues as the numerical rank of the matrices $M_k^2 + I$ when this rank stabilizes as $k$ increases.

**Remark 3.2.** (Acceleration by Means of Scaling.) One can dramatically accelerate the initial convergence of Algorithm 3.1 by applying determinantal scaling (cf. [20]), that is, by replacing the matrix $M_0 = C_p$ by the matrix $M_0 = 0.5(\nu C_p - (\nu C_p)^{-1})$ for $\nu = 1/|\det(C_p)|^{1/n} = |p_n/p_0|.$

## 3.2 Inversion-free Variations of the Modified Matrix Sign Iteration and Hybrid Algorithms

The overall arithmetic cost of the Modified Matrix Sign Iteration is dominated by the cost of $k$ matrix inversions, that is, $O(kn \log^2(n))$ flops (cf. Theorem 2.1). If all nonreal eigenvalues of the matrix $M_0$ lie in the two discs $D(\pm \sqrt{-1}, 1/2) = \{x : |x \pm \sqrt{-1}| \leq 1/2\},$ then we can avoid this deficiency by replacing iterations (3.1) with any of the two iteration processes

$$M_{h+1} = 0.5(M_h^2 + 3M_h) \quad (3.3)$$

and

$$M_{h+1} = -0.125(3M_h^2 + 10M_h^3 + 15M_h) \quad (3.4)$$

for $h = 0, 1, \ldots$. In this case, right from the start both iterations send the nonreal roots toward the two points $\pm \sqrt{-1}$ with quadratic and cubic convergence rates, respectively. (To prove this, extend the proof of [7] Proposition 4.1.) Both iteration processes keep the real roots real and use $O(n \log(n))$ flops per iteration loop.

What if the nonreal roots do not lie in these discs? We can apply the following combination of iterations (3.1), (3.3), and (3.4) and Corollary D.1 of Section D.

**Algorithm 3.2.** A Hybrid Algorithm.

**INPUT, OUTPUT** as in Algorithm 3.1.

**COMPUTATIONS:** Perform the iterations of Algorithm 3.1 using matrix inversions and involving $O(n \log^2(n))$ flops per iteration, until a test shows that the $2s$ nonreal eigenvalues of the input companion matrix are mapped into the discs $D(\pm \sqrt{-1}, 1/2).$ Then shift the computations to the inversion-free iterations (3.3) or (3.4) converging faster and using $O(n \log(n))$ flops per iteration. (For testing the latter condition, apply the algorithm supporting Corollary D.1. To keep the computational cost down, apply this test periodically, according to a fixed policy, based on heuristic rules or previous statistics.)

Let us specify some alternative matrix iterations for real root-finding without inversions. Recall that $\text{sign}(M) = M(M^2)^{-0.5}$ and apply the Newton-Schultz iteration for the approximation of the matrix square root [20], equation (6.3)],

$$Y_{k+1} = 0.5 Y_k(3I - Z_k Y_k), \quad Y_0 = M^{-2},$$

and

$$Z_{k+1} = 0.5 (3I - Z_k Y_k) Z_k, \quad Z_0 = I$$

for $k = 0, 1, \ldots$. The iteration keeps real eigenvalues real and converges if $||I - M^{-2}||_p < 1$ for $p = 1, 2,$ or $\infty$. This assumption is easy to satisfy by means of scaling $M \rightarrow aM,$ which keeps real eigenvalues real for real $a$.

The similar coupling technique of [8] is even simpler, for it is applied directly to the modified matrix sign iteration (3.1), preserving its quadratic convergence to $\pm \sqrt{-1}$ right from the start. In our tests for real root-finding, however, we could perform safely only a small number of these inversion-free iterations at the initial stage, and then the images of the real eigenvalues of the matrix $C_p$ grew very large and the condition numbers of the computed matrices blew up.
3.3 Numerical Stabilization of the Modified Matrix Sign Iteration

The images of nonreal eigenvalues of the matrix $C_p$ converge to $\pm \sqrt{-1}$ in the iteration of Stage 1 of Algorithm 3.1, but if the images of some real eigenvalues of $C_p$ come close to 0, then at the next step we would have to invert an ill-conditioned matrix $M_h$ unless we are applying an inversion-free variant of the iteration of the previous subsection.

We can try to avoid this problem by shifting the matrix (and its eigenvalues), that is, by adding to the current matrix $M_h$ the matrix $sI$ for a reasonably small positive scalar $s$ or $-s$. We can select this scalar by applying heuristic methods and randomization. In our tests this policy has preserved convergence quite well, but towards a more radical recipe, we applied the following modification of Algorithm 3.1.

**Algorithm 3.3.** Numerical stabilization of the modified matrix sign iteration.

**Input, Output and Stages 3 and 4 of Computations are as in Algorithm 3.1 except that the input includes a small positive scalar $\alpha$ such that no eigenvalues of the matrix $C_p$ have imaginary parts close to $\pm \alpha \sqrt{-1}$ (see Remark 3.3 below), the set of $r$ real roots $x_1, \ldots, x_r$ of the polynomial $p(x)$ is replaced by the set of its $r_+$ roots having the imaginary parts in the range $[-\alpha, \alpha]$, and the integer $r$ is replaced by the integer $r_+$ throughout.

**Computations:**

1. Apply Stage 1 of Algorithm 3.1 to the two matrices $M_{0,\pm} = \alpha \sqrt{-1} I \pm C_p$, thus producing two sequences of the matrices $M_{h,+}$ and $M_{h,-}$ for $h = 0, 1, \ldots$.

2. Fix a sufficiently large integer $k$ and compute the matrix $M = M_{k,+} + M_{k,-}$.

Because of the assumed choice of $\alpha$, the matrices $\alpha \sqrt{-1} I \pm C_p$ have no real eigenvalues, and so the images of all their eigenvalues, that is, the eigenvalues of the matrices $M_{k,+}$ and $M_{k,-}$, converge to $\pm \sqrt{-1}$ as $\alpha \to \infty$. Moreover, one can verify that the eigenvalues of the matrix $M_{k,+} + M_{k,-}$ converge to 0 unless they are the images of the $r_+$ eigenvalues of the matrix $C_p$ having the imaginary parts in the range $[-\alpha, \alpha]$. The latter eigenvalues of the matrix $M_{k,+} + M_{k,-}$ converge to $2 \sqrt{-1}$.

This shows correctness and numerical stability of Algorithm 3.3.

The algorithm approximates the $r_+$ roots of $p(x)$ by using $O(kn \log^2(n) + nr_+^2) + a_{r+n}$ flops, versus $O(kn \log^2(n) + nr^2) + a_{rnn}$ involved in Algorithm 3.1.

**Remark 3.3.** We can test the proximity of the roots to a line in two stages: by at first moving the line into the unit circle $\{ x : |x| = 1 \}$ (cf. Theorem A.3) and then applying the algorithm that supports Corollary 7.7.

3.4 Square Root Iteration (a Modified Modular Version)

Next we describe a dual polynomial version of Algorithm 3.1. It extends the square root iteration $y_{h+1} = \frac{1}{2}(y_h + 1/y_h)$, $h = 0, 1, \ldots$, and at Stage 2 involves the computation of the polynomial $\text{gcd}(p, t_k)$, which denotes an approximate greatest common divisor of the input polynomial $p = p(x)$ and an auxiliary polynomial $t_k = t_k(x)$. We refer the reader to [31], [22], [4], [49], [10] for the definitions of this concept and the algorithms for its computation.

Compared to Algorithm 3.1 we replace all rational functions in the matrix $C_p$ by the same rational functions in the variable $x$ and reduce them modulo the input polynomial $p(x)$. The reduction does not affect the values of the functions at the roots of $p(x)$, and it follows that these values are precisely the eigenvalues of the rational matrix functions computed in Algorithm 3.1.

**Algorithm 3.4.** Square root modular iteration modified for real root-finding.

**Input:** two integers $n$ and $r$, $0 < r < n$, and the coefficients of a polynomial $p(x)$ of equation (1.1).

**Output:** approximations to the real roots $x_1, \ldots, x_r$ of the polynomial $p(x)$.

**Computations:**
1. Write \( y_0 = x \) and (cf. [37]) compute the polynomials
\[
y_{h+1} = \frac{1}{2}(y_h - 1/y_h) \mod p(x), \quad h = 0, 1, \ldots
\] (3.5)

2. Periodically, for selected integers \( k \), compute the polynomials \( t_k = y_k^2 + 1 \mod p(x) \).

3. Write \( g_k(x) = \text{agcd}(p, t_k) \) and compute \( d_k = \text{deg}(g_k(x)) \). If \( d_k = n - r = 2s \), compute the polynomial \( v_k \approx p(x)/g_k(x) \) of degree \( r \). Otherwise continue the iteration of Stage 1.

4. Apply one of the algorithms of [1], [3], and [4] (cf. Theorem C.1) to approximate the \( r \) roots \( y_1, \ldots, y_r \) of the polynomial \( v_k \). Output these approximations.

Our comments preceding this algorithm show that the values of the polynomials \( t_k(x) \) at the roots of \( p(x) \) are equal to the images of the eigenvalues of the matrix \( C_p \) in Algorithm 3.1. Hence the values of the polynomials \( t_k(x) \) at the nonreal roots of \( p(x) \) converge to 0 as \( k \to \infty \), whereas their values at the real roots of \( p(x) \) stay far from 0. Therefore, for sufficiently large integers \( k \), \( \text{agcd}(p, t_k) \) turns into the polynomial \( \prod_{j=r+1}^n (x-x_j) \). This implies correctness of the algorithm. Its asymptotic computational cost is \( O(kn \log^3(n)) \) plus the cost of computing \( \text{agcd}(p, t_k) \) and choosing the integer \( k \) (see our next remark).

**Remark 3.4.** Compared to Algorithm 3.1, the latter algorithm reduces real root-finding essentially to the computation of \( \text{agcd}(p, t_k) \). One can apply quite efficient heuristic algorithms for this computation (cf. [22], [23], [5], [4], [49], [10]), but no good formal estimates are available for their complexity. One can, however, note that \( p(x)u_k(x) \approx t_k(x)v_k(x) \), and so, assuming that \( v_k(x) \) is a monic polynomial (otherwise we can scale it), obtain its other coefficients (as well as the coefficients of the polynomial \( u_k(x) \)) from the least-squares solution to the associated Sylvester linear system of equations. Its well known superfast divide and conquer solution involves order of \( n \log^2(n) \) flops (cf. [24, Chapter 5]), but the recent numerically stable algorithm of [51] accelerated by a factor of \( \log(n) \) in [30, Section 9.8] involves only \( O(n \log(n)) \) flops.

4 **Numerical Tests**

Extensive numerical tests of the algorithms of this paper, performed in the Graduate Center of the City University of New York, are the contribution of the second author (at some points he was assisted by Ivan Retamoso). The tests recorded the number of iterations and the error of the approximation of the real roots of benchmark polynomials to which we applied these algorithms. For Algorithms 3.1 and 3.3 similar data have been recorded also for the approximation of real eigenvalues of some random matrices \( M \). In the latter case the convergence of these algorithms and the number of their iterations depend mostly on the characteristic polynomials of \( M \), even though the estimates for the arithmetic cost of performing each iteration generally grow compared to the special case where \( M = C_p \).

In some cases we stop the iterations already when they produce crude approximation to the roots. This is because, instead of continuing the iterations, we can apply the algorithms of [40] followed by Newton’s or Ehrlich–Aberth’s iterations (cf. Section [13]), which refine very fast these crude approximations.

Finally we note that the test results in the present section, are quite encouraging, e.g., the numbers of iterations required for convergence of our algorithms have grown very slowly (if at all) when we increased the degree of the input polynomials and dimension of the input matrices from 64 to 1024.

4.1 **Tests for the Modified Matrix Sign Iteration (Algorithm 3.1)**

In the first series of the tests, Algorithm 3.1 has been applied to one of the Mignotte benchmark polynomials, namely \( p(x) = x^n + (100x - 1)^3 \). It is known that this polynomial has three ill
conditioned roots clustered about 0.01 and has \( n - 3 \) well conditioned roots. In the tests, Algorithm 3.1 has output the roots within the error less than \( 10^{-6} \) by using 9 iterations for \( n = 32 \) and \( n = 64 \) and by using 11 iterations for \( n = 128 \) and \( n = 256 \).

In the second series of the tests, polynomials \( p(x) \) of degree \( n = 50, 100, 150, 200, \) and 250 have been generated as the products \( p(x) = f_1(x)f_2(x) \), for the \( r \)th degree Chebyshev polynomial \( f_1(x) \) (having \( r \) real roots), \( r = 8, 12, 16 \), and \( f_2(x) = \sum_{i=0}^{n-r} a_i x^i \), \( a_i \) being i.i.d. standard Gaussian random variables, for \( j = 0, \ldots, n-r \). Algorithm 3.1 (performed with the IEEE standard double precision) was applied to 100 such polynomials \( p(x) \) for each pair of \( n \) and \( r \). Table 4.1 displays the output data, namely, the average values and the standard deviation of the numbers of iterations and of the maximum difference between the output values of the roots and their values produced by MATLAB root-finding function "roots()".

In the third series of the tests, Algorithm 3.1 approximated the real eigenvalues \( x_1, \ldots, x_r \) of a random complex symmetric matrix \( A = U^T \Sigma U \), for \( \Sigma = \text{diag}(x_1, \ldots, x_r, y_1, \ldots, y_{n-r}) \), \( r \) i.i.d. standard Gaussian real random variables, \( x_1, \ldots, x_r \), \( n-r \) i.i.d. standard Gaussian complex (non-real) random variables, \( y_1, \ldots, y_{n-r} \), and an orthogonal \( \Sigma \times \Sigma \) standard Gaussian random matrix \( U \). Table 4.2 displays the mean and standard deviation of the number of iterations and the error bounds in these tests for \( n = 50, 100, 150, 200, 250 \) and \( r = 8, 12, 16 \).

Table 4.1: Number of Iterations and Error Bounds for Algorithm 3.1 on Random Polynomials

| n  | r  | Iteration-mean | Iteration-std | Error-mean | Error-std |
|----|----|----------------|---------------|------------|-----------|
| 50 |   | 7.44           | 1.12          | 4.18 \times 10^{-6} | 1.11 \times 10^{-5} |
| 100|  8| 8.76           | 1.30          | 5.90 \times 10^{-6} | 1.47 \times 10^{-5} |
| 150|  8| 9.12           | 0.88          | 2.61 \times 10^{-5} | 1.03 \times 10^{-4} |
| 200|  8| 9.64           | 0.86          | 1.48 \times 10^{-6} | 5.93 \times 10^{-7} |
| 250|  8| 9.96           | 0.73          | 1.09 \times 10^{-7} | 5.23 \times 10^{-8} |
| 50 | 12 | 7.16           | 0.85          | 3.45 \times 10^{-4} | 9.20 \times 10^{-5} |
| 100| 12 | 8.64           | 1.15          | 1.34 \times 10^{-5} | 2.67 \times 10^{-6} |
| 150| 12 | 9.12           | 2.39          | 3.38 \times 10^{-4} | 1.08 \times 10^{-3} |
| 200| 12 | 9.76           | 2.52          | 6.89 \times 10^{-6} | 1.75 \times 10^{-6} |
| 250| 12 | 10.04          | 1.17          | 1.89 \times 10^{-5} | 4.04 \times 10^{-6} |
| 50 | 16 | 7.28           | 5.06          | 3.67 \times 10^{-3} | 7.62 \times 10^{-4} |
| 100| 16 | 10.20          | 5.82          | 1.44 \times 10^{-4} | 4.51 \times 10^{-5} |
| 150| 16 | 15.24          | 6.33          | 1.25 \times 10^{-3} | 4.90 \times 10^{-4} |
| 200| 16 | 13.36          | 5.38          | 1.07 \times 10^{-3} | 4.72 \times 10^{-4} |
| 250| 16 | 13.46          | 6.23          | 1.16 \times 10^{-4} | 2.45 \times 10^{-4} |
Table 4.2: Number of Iterations and Error Bounds for Algorithm 3.1 on Random Matrices

| n   | r  | Iteration-mean | Iteration-std | Error-mean   | Error-std   |
|-----|----|----------------|---------------|--------------|-------------|
| 50  | 8  | 10.02          | 1.83          | 5.51 \times 10^{-11} | 1.65 \times 10^{-10} |
| 100 | 8  | 10.81          | 2.04          | 1.71 \times 10^{-12} | 5.24 \times 10^{-12} |
| 150 | 8  | 14.02          | 2.45          | 1.31 \times 10^{-13} | 3.96 \times 10^{-13} |
| 200 | 8  | 12.07          | 0.94          | 2.12 \times 10^{-11} | 6.70 \times 10^{-11} |
| 250 | 8  | 13.59          | 1.27          | 2.75 \times 10^{-10}  | 8.14 \times 10^{-10}  |
| 50  | 12 | 10.46          | 1.26          | 1.02 \times 10^{-12}  | 2.61 \times 10^{-12}  |
| 100 | 12 | 10.60          | 1.51          | 1.79 \times 10^{-10}  | 3.66 \times 10^{-10}  |
| 150 | 12 | 11.25          | 1.32          | 5.69 \times 10^{-8}   | 1.80 \times 10^{-7}   |
| 200 | 12 | 12.36          | 1.89          | 7.91 \times 10^{-10}  | 2.50 \times 10^{-9}   |
| 250 | 12 | 11.72          | 1.49          | 2.53 \times 10^{-12}  | 3.84 \times 10^{-12}  |
| 50  | 16 | 10.10          | 1.45          | 1.86 \times 10^{-9}   | 5.77 \times 10^{-9}   |
| 100 | 16 | 11.39          | 1.70          | 1.37 \times 10^{-10}  | 2.39 \times 10^{-10}  |
| 150 | 16 | 11.62          | 1.78          | 1.49 \times 10^{-11}  | 4.580 \times 10^{-11} |
| 200 | 16 | 11.88          | 1.32          | 1.04 \times 10^{-12}  | 2.09 \times 10^{-12}  |
| 250 | 16 | 12.54          | 1.51          | 3.41 \times 10^{-11}  | 1.08 \times 10^{-10}  |
4.2 Tests for the Stabilized Matrix Sign Iteration (Algorithm 3.3) 
Applied to Polynomials

We tested Algorithm 3.3 on various modified benchmark polynomials from the website of MPSolve (http://numpi.dm.unipi.it/mpsolve-2.2/). Wherever a family of benchmark polynomials tended to have only trivial real roots, we multiplied it by Chebyshev polynomials, which have only real roots.

Having generated such an input polynomial \( p = p(x) \) and its companion matrix \( C_p \), we computed the condition numbers of the matrices \( M_k = C_p + 2^{r+k}I_n \) with \( k = 1,2,\ldots \) and selected an integer \( k \) such that \( \kappa(M_k) < 10^5 \). Clearly, this is ensured for sufficiently large integers \( k \) defining diagonally dominant matrices \( M_k \), but in our tests \( k \) was less than 5 in most cases.

Having fixed \( k \) and \( M_k \) and following the description of Algorithm 3.3 we computed at first the matrices \( Y_1 = \alpha I_n + M_k \) and \( Y_2 = \alpha I_n - M_k \), for \( \alpha = 0.0001\sqrt{-1} \), and then successively the matrices \( Y_{i+1,j} = \frac{1}{\kappa}(Y_{i,j} - Y_{i+1,j}^{-1}) \) with \( Y_{0,j} = Y_j \) for \( j = 1,2 \) (cf. Algorithms 3.1 and 3.3).

We have observed that with our real shifting by \( 2^r+kI_n \) at the initial stage, non-real eigenvalues of \( Y_1 \) and \( Y_2 \) were never close to \( \pm \sqrt{-1} \) at the first \( 7+k \) iterations. So we began checking convergence only when we have performed these initial iterations, and since that moment we checked convergence in every 5 iterations. As soon as we observed that \( \text{rank}(Y'_i) = r \) for \( Y'_i = Y_{i-1} + Y_{i,2} \) and \( r \) denoting the number of distinct real roots of \( p(x) \), we stopped the iteration loop and moved to the final stage of the algorithm, that is, approximated the real eigenvalues of matrix \( C_p \), equal to the real roots of the polynomial \( p(x) \).

We run numerical tests on polynomials of five types having degree \( n = 64,128,256,512,1024 \), and we compared our results with the outputs of MATLAB function "roots()”:

I. \( p(x) = p_1(x)p_2(x) \), where \( p_1(x) \) is the \( r \)-th degree Chebyshev polynomial, \( r = 8,12,16, p_2(x) = x^n-r-1. \)

II. \( p(x) = p_1(x)p_2(x) \), where \( p_1(x) \) is the \( r \)-th degree Chebyshev polynomial, \( r = 8,12,16, p_2(x) = 1 + 2x + 3x^2 + \cdots + (n-r+1)x^{n-r}. \)

III. \( p(x) = p_1(x)p_2(x) \), where \( p_1(x) \) is the \( r \)-th degree Chebyshev polynomial, \( r = 8,12,16, p_2(x) = (x+1)(x+a)(x+a^2)\cdots(x+a^{n-r-1}), \) with \( a = \frac{1}{100}. \)

IV. \( p(x) = x^n-r - (ax-1)^3, \) where \( a = 60,80,100. \)

V. \( p(x) = p_1(x)p_2(x) \), where \( p_1(x) \) is the \( r \)-th degree Chebyshev polynomial, \( r = 8,12,16, p_2(x) = \sum_{k=0}^{n} a_kx^k, \) with \( a_0,\ldots,a_n \) being i.i.d standard random variables.

Tables 4.3-4.6 display the number of iterations and the maximum error bounds for the polynomials of Types I–IV. Table 4.7 shows the average error bounds and the average numbers of iterations in 50 tests with the polynomials of Type V.
Table 4.3: Number of Iterations and Error Bounds for Algorithm 3.3 on Type I Polynomials

| n   | r  | Iterations | Errors   |
|-----|----|------------|----------|
| 64  | 8  | 10         | $1.03E - 10$ |
| 64  | 12 | 23         | $1.32E - 08$ |
| 64  | 16 | 23         | $3.97E - 06$ |
| 128 | 8  | 10         | $1.60E - 10$ |
| 128 | 12 | 23         | $4.91E - 04$ |
| 128 | 16 | 23         | $2.22E - 03$ |
| 256 | 8  | 10         | $6.18E - 06$ |
| 256 | 12 | 28         | $1.75E - 09$ |
| 256 | 16 | 28         | $3.54E - 06$ |
| 512 | 8  | 15         | $8.05E - 13$ |
| 512 | 12 | 28         | $1.71E - 08$ |
| 512 | 16 | 28         | $2.78E - 05$ |
| 1024| 8  | 15         | $2.33E - 12$ |
| 1024| 12 | 28         | $1.27E - 09$ |
| 1024| 16 | 28         | $2.19E - 05$ |

Table 4.4: Number of Iterations and Error Bounds for Algorithm 3.3 on Type II Polynomials

| n   | r  | Iterations | Errors   |
|-----|----|------------|----------|
| 64  | 8  | 10         | $1.53E - 11$ |
| 64  | 12 | 23         | $1.30E - 07$ |
| 64  | 16 | 23         | $1.40E - 05$ |
| 128 | 8  | 28         | $9.42E - 11$ |
| 128 | 12 | 10         | $7.51E - 08$ |
| 128 | 16 | 28         | $2.27E - 04$ |
| 256 | 8  | 28         | $1.92E - 11$ |
| 256 | 12 | 28         | $2.21E - 07$ |
| 256 | 16 | 28         | $1.69E - 03$ |
| 512 | 8  | 28         | $3.68E - 12$ |
| 512 | 12 | 28         | $2.17E - 06$ |
| 512 | 16 | 33         | $1.53E - 02$ |
| 1024| 8  | 28         | $2.96E - 11$ |
| 1024| 12 | 33         | $5.00E - 07$ |
| 1024| 16 | 33         | $3.58E - 03$ |
Table 4.5: Number of Iterations and Error Bounds for Algorithm 4.3 on Type III Polynomials

| n   | r   | Iterations | Errors       |
|-----|-----|------------|--------------|
| 64  | 8   | 28         | 4.63E−11     |
| 64  | 12  | 23         | 1.69E−07     |
| 64  | 16  | 28         | 7.36E−06     |
| 128 | 8   | 28         | 3.83E−12     |
| 128 | 12  | 23         | 1.45E−08     |
| 128 | 16  | 28         | 1.68E−05     |
| 256 | 8   | 28         | 1.58E−12     |
| 256 | 12  | 23         | 1.02E−04     |
| 256 | 16  | 28         | 6.50E−04     |
| 512 | 8   | 28         | 7.69E−13     |
| 512 | 12  | 23         | 5.00E−09     |
| 512 | 16  | 28         | 8.60E−06     |
| 1024| 8   | 28         | 9.90E−14     |
| 1024| 12  | 23         | 1.45E−09     |
| 1024| 16  | 28         | 2.64E−05     |

Table 4.6: Number of Iterations and Error Bounds Algorithm 4.3 on Type IV Polynomials

| n   | a   | Iterations | Errors       |
|-----|-----|------------|--------------|
| 64  | 60  | 41         | 2.43E−04     |
| 64  | 80  | 42         | 7.98E−04     |
| 64  | 100 | 43         | 1.72E−05     |
| 128 | 60  | 41         | 1.12E−03     |
| 128 | 80  | 42         | 4.43E−04     |
| 128 | 100 | 43         | 1.31E−04     |
| 256 | 60  | 41         | 2.10E−04     |
| 256 | 80  | 42         | 1.91E−04     |
| 256 | 100 | 43         | 1.34E−04     |
| 512 | 60  | 41         | 3.37E−04     |
| 512 | 80  | 42         | 1.80E−04     |
| 512 | 100 | 43         | 8.33E−05     |
| 1024| 60  | 36         | 1.10E−01     |
| 1024| 80  | 42         | 1.16E−04     |
| 1024| 100 | 43         | 1.76E−04     |
Table 4.7: Number of Iterations and Error Bounds for Algorithm 4.3 on Type V Polynomials

| n  | r  | Iterations | Errors       |
|----|----|------------|--------------|
| 128| 8  | 22.3       | 5.33E-06     |
| 128| 12 | 24.6       | 4.85E-05     |
| 128| 16 | 24.94      | 3.59E-03     |
| 256| 8  | 26.02      | 1.11E-06     |
| 256| 12 | 27.01      | 2.37E-05     |
| 256| 16 | 30.18      | 1.80E-03     |
| 512| 8  | 27.54      | 2.73E-08     |
| 512| 12 | 28.00      | 2.27E-06     |
| 512| 16 | 38.18      | 2.39E-03     |
4.3 Tests for the Stabilized Matrix Sign Iteration (Algorithm 3.3) on Gaussian Random Matrices

We tested Algorithm 3.3 on randomly generated matrices of two types:

Type I: Gaussian random tridiagonal matrices of dimension \( n = 64, 128, 256, 512, 1024 \). We generated each entry in the tridiagonal part independently by using standard Gaussian distribution and set the other entries to 0. Our tables show the error bounds equal to the maximal difference of the outputs of our algorithm and MATLAB function \( \text{eig()} \). We generated 100 matrices for each \( n \) and recorded the mean and standard deviation of the error bounds and of the numbers of iterations.

Type II: Random matrices \( A \) with a fixed number of real eigenvalues. At first we generated a diagonal matrix \( \Sigma \) with \( r \) diagonal entries under standard real Gaussian distribution and \( n - r \) diagonal entries under standard complex Gaussian distribution for \( n = 64, 128, 256, 512, 1024 \) and \( r = 8, 12, 16 \). Then we generated orthonormal standard Gaussian matrix \( Q \). We generated 100 such matrices \( A \) for each pair of \( n \) and \( r \) and recorded the mean and standard deviation of the error bounds and of the numbers of iterations.

The following two tables summarize the performance, showing a low number of iterations required for ensuring a reasonable precision of the approximation of the eigenvalues.

Table 4.8: Number of Iterations and Error Bounds for Root-finding Algorithm 3.3 on Type I matrices

| \( n \) | Iteration-mean | Iteration-std | Error-mean | Error-std |
|---|---|---|---|---|
| 64 | 10.70 | 2.36 | 1.78 \( E - 06 \) | 1.14 \( E - 05 \) |
| 128 | 12.16 | 3.34 | 5.68 \( E - 07 \) | 4.49 \( E - 06 \) |
| 256 | 12.97 | 3.97 | 3.26 \( E - 06 \) | 1.35 \( E - 05 \) |
| 512 | 15.46 | 9.82 | 8.80 \( E - 04 \) | 8.44 \( E - 03 \) |
| 1024 | 16.52 | 10.26 | 2.43 \( E - 03 \) | 2.25 \( E - 02 \) |

Table 4.9: Number of Iterations and Error Bounds for Algorithm 3.3 on Type II matrices

| \( n \) | \( r \) | Iteration-mean | Iteration-std | Error-mean | Error-std |
|---|---|---|---|---|---|
| 64 | 8 | 11.65 | 2.47 | 3.69 \( E - 08 \) | 2.29 \( E - 07 \) |
| 64 | 12 | 11.75 | 2.50 | 3.98 \( E - 10 \) | 2.71 \( E - 09 \) |
| 64 | 16 | 11.60 | 2.45 | 4.10 \( E - 09 \) | 3.88 \( E - 08 \) |
| 128 | 8 | 13.75 | 2.79 | 1.17 \( E - 08 \) | 7.56 \( E - 08 \) |
| 128 | 12 | 13.70 | 2.90 | 4.41 \( E - 09 \) | 2.73 \( E - 08 \) |
| 128 | 16 | 13.65 | 2.55 | 1.23 \( E - 07 \) | 1.34 \( E - 06 \) |
| 256 | 8 | 14.55 | 3.26 | 5.59 \( E - 09 \) | 5.58 \( E - 08 \) |
| 256 | 12 | 14.15 | 3.70 | 1.38 \( E - 07 \) | 1.38 \( E - 06 \) |
| 256 | 16 | 14.70 | 2.54 | 3.06 \( E - 11 \) | 1.93 \( E - 10 \) |
| 512 | 8 | 13.65 | 5.59 | 5.08 \( E - 10 \) | 4.88 \( E - 09 \) |
| 512 | 12 | 15.65 | 9.47 | 7.46 \( E - 04 \) | 7.46 \( E - 03 \) |
| 512 | 16 | 16.55 | 10.26 | 2.78 \( E - 03 \) | 5.47 \( E - 03 \) |
| 1024 | 8 | 18.20 | 15.35 | 2.33 \( E - 10 \) | 1.22 \( E - 09 \) |
| 1024 | 12 | 20.85 | 17.60 | 1.27 \( E - 07 \) | 3.36 \( E - 07 \) |
| 1024 | 16 | 24.35 | 19.56 | 2.19 \( E - 03 \) | 4.33 \( E - 03 \) |

4.4 Tests for a Hybrid Matrix Algorithm on Benchmark Polynomials

We performed numerical tests of a hybrid algorithm. We began with Algorithm 5.1 and after sufficiently many iterations continued with its variation avoiding matrix inversion.
Namely, we first applied a real shift $\beta I$ to the companion matrix $C_p$, such that the matrix $M = C_p + \beta I$ had condition number less than $10^5$. For such inputs, we expected (based on our preliminary tests) that at least $T = \log_2 \beta$ iterations $M_{i+1} = \frac{1}{2}(M_i - M_i^{-1})$ would be required in order to move the complex nonreal eigenvalues close enough to $\pm\sqrt{-1}$. After the first $T$ iterations, we periodically (in every 5 iterations) applied two iterations $M_{i+1} = \frac{1}{2}(M_i^3 + 3M_i)$, which converged with cubic rate provided that all complex eigenvalues have distance less than $\frac{1}{2}$ from $\sqrt{-1}$ or $-\sqrt{-1}$. Before switching to iterations of the second type, we performed the following transformation in order to avoid problems of numerical stability:

Step 1: Compute $P = \frac{0.5M + \sqrt{-1}}{0.5M + \sqrt{-1}}$, which maps the real line into the unit circle.

Step 2: Compute $Y = \frac{2\sqrt{-1}}{3}(P - P^{-1})$, which maps the unit circle onto the interval $[-2/3, 2/3]$. Note that these two maps together keep the value $\pm\sqrt{-1}$ unmoved.

We tested polynomials of Types II and IV of the previous section. For polynomials of Types I, III, and V the test results were similar to those for polynomials of Type II, apparently due to the shared Chebyshev factors. The test results on Type IV polynomials indicate the strength of this algorithm in the case of clustered roots.

The number of iterations required and the error bound are displayed in the tables below.

### Table 4.10: Number of Iterations and Error Bounds for Hybrid Algorithm on Type II Polynomials

| n  | r  | Iterations | Errors         |
|----|----|------------|----------------|
| 64 | 8  | 10         | 3.69E-10       |
| 64 | 12 | 23         | 4.96E-08       |
| 64 | 16 | 23         | 4.97E-03       |
| 128| 8  | 10         | 2.28E-11       |
| 128| 12 | 28         | 1.97E-07       |
| 128| 16 | 23         | 8.68E-02       |
| 256| 8  | 28         | 6.56E-12       |
| 256| 12 | 28         | 3.64E-07       |
| 256| 16 | 28         | 3.82E-04       |
| 512| 8  | 15         | 8.05E-13       |
| 512| 12 | 28         | 1.71E-08       |
| 512| 16 | 28         | 2.78E-05       |
| 1024|8  | 28        | 3.72E-11       |
| 1024|12 | 28        | 1.09E-08       |
| 1024|16 | 33        | 2.19E-05       |

### Table 4.11: Number of Iterations and Error Bounds for Hybrid Algorithm on Type IV Polynomials

| n  | Iterations | Errors         |
|----|------------|----------------|
| 64 | 33         | 7.32E-05       |
| 128| 33         | 6.12E-06       |
| 256| 38         | 1.60E-05       |
| 512| 38         | 1.08E-04       |
| 1024|38       | 9.19E-01       |
Table 4.12 displays the test results for Algorithm 3.4 that is, the iterations $f_{i+1}(x) \equiv \frac{1}{2}(f_i(x) - f_i(x)^{-1}) \mod p(x)$, applied to polynomials of Types I and II. The polynomial inverses modulo $p(x)$ were computed by means of solving the associated Sylvester linear systems of equations. A small number of the iterations have produced a polynomial $f_i(x)$ whose roots approximated quite closely the complex roots of the polynomial $p(x)$ of (1.1). The approximate quotient $[p(x)/\gcd]$ had degree $r$ and had $r$ real roots, all shared with $p(x)$.

After each iteration, the roots of polynomial $f_i(x)$ have been computed and compared to the complex roots of the input polynomial $p(x)$ computed with MATLAB function "roots()". The iterations stopped and the number of iterations was recorded when the maximum difference was less than a pre-determined tolerance bound $\epsilon = 10^{-5}$.

Table 4.12 displays our test results for Algorithm 3.4 that is, the iterations $f_{i+1}(x) \equiv \frac{1}{2}(f_i(x) - f_i(x)^{-1}) \mod p(x)$, where we computed polynomial inverses modulo $p(x)$ by solving the associated Sylvester linear systems of equations. We applied the tests to polynomials of Types I and II.

In the tests a small number of these iterations were sufficient to produce a polynomial $f_i(x)$ whose roots approximated quite closely the complex roots of the polynomial $p(x)$ of (1.1). The approximate quotient $[p(x)/\gcd]$ had degree $r$ and had $r$ real roots, all shared with $p(x)$.

After each iteration, we computed the roots of polynomial $f_i(x)$ and compare them to the complex roots of the input polynomial $p(x)$ computed with MATLAB function "roots()". The iteration stopped and the number of iteration was recorded when the maximum difference was less than a pre-determined tolerance bound $\epsilon = 10^{-5}$.

Table 4.12: Number of Iterations for Algorithm 3.4 on Polynomials of Types I and II

| n  | r  | Type I | Type II |
|----|----|--------|---------|
| 64 | 8  | 9      | 14      |
| 64 | 12 | 4      | 16      |
| 64 | 16 | 2      | 17      |
| 128| 8  | 9      | 14      |
| 128| 12 | 12     | 16      |
| 128| 16 | 2      | 17      |
| 256| 8  | 9      | 14      |
| 256| 12 | 12     | 16      |
| 256| 16 | 8      | 17      |
| 512| 8  | 9      | 14      |
| 512| 12 | 12     | 16      |
| 512| 16 | 8      | 17      |
| 1024| 8 | 10     | 14     |
| 1024| 12| 12     | 16     |
| 1024| 16| 11     | 17     |

Acknowledgements: This work has been supported by NSF Grant CCF–1116736 and PSC CUNY Award 67699-00 45.

References

[1] Ben-Or, M., Tiwari, P.: Simple algorithms for approximating all roots of a polynomial with real roots. J. Complexity, 6(4), 417–442 (1990)

[2] Bini, D.: Numerical computation of polynomial zeros by means of Aberth’s method. Numerical Algorithms, 13, 179–200 (1996)
[3] Bini, D. A.: private communication (2014)

[4] Bini, D.A., Boito, P.: A fast algorithm for approximate polynomial GCD based on structured matrix computations. In: Operator Theory: Advances and Applications, vol. 199, pp. 155–173. Birkhäuser Verlag, Basel (2010)

[5] Bini, D. A., Fiorentino, G.: Design, analysis, and implementation of a multiprecision polynomial rootfinder. Numerical Algorithms, 23, 127–173 (2000)

[6] Bini, D., Pan, V. Y.: Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms. Birkhäuser, Boston (1994)

[7] Bini, D., Pan, V.Y.: Graeffe’s, Chebyshev, and Cardinal’s processes for splitting a polynomial into factors. J. Complexity, 12, 492–511 (1996)

[8] Bini, D., Pan, V.Y.: Computing matrix eigenvalues and polynomial zeros where the output is real. SIAM J. on Computing 27(4), 1099–1115 (1998) Proc. version in SODA’1991, pp. 384-393, ACM Press, New York, and SIAM Publications, Philadelphia (1991)

[9] Bini, D.A., Robol, L.: Solving secular and polynomial equations: a multiprecision algorithm. J. Computational and Applied Mathematics, 272, 276–292 (2014)

[10] Boito, P., Ruatta, O.: Extended companion matrix for approximate GCD. Proc. SNC 2011, pp. 74-80 (2011)

[11] Carstensen, C.: Inclusion of the roots of a polynomial based on Gerschgorin theorem. Numerische Math. 59, 349–360 (1991)

[12] Cardinal, J.P.: On two iterative methods for approximating the roots of a polynomial. In: Lectures in Applied Mathematics, vol. 32, pp. 165–188, AMS (1996)

[13] R. M. Corless, S. M. Watt, L. Zhi, QR factoring to compute the GCD of univariate approximate polynomials, IEEE Transactions on Signal Processing, 52, pp. 3394–340 (2004).

[14] Du, Q., Jin, M., Li, T.Y., Zeng, Z.: The quasi-Laguerre iteration. Math. Computation, 66(217), 345–361 (1997)

[15] Eidelman, Y., Gohberg, I., Haimovici, I.: Separable Type Representations of Matrices and Fast Algorithms, volumes 1 and 2. Birkhäuser (2013)

[16] Emiris, I. Z., Pan, V. Y., Tsigaridas, E.: Algebraic algorithms. Chapter 10 of Computing Handbook (Third edition), Volume I: Computer Science and Software Engineering (Allen B. Tucker, Teo Gonzales, and Jorge L. Diaz-Herrera, editors). Taylor and Francis Group (2014). Available at arXiv 1311.3731 [cs.DS]

[17] Fortune, S.: An iterated eigenvalue algorithm for approximating roots of univariate polynomials. J. of Symbolic Computation, 33(5), 627–646 (2002)

[18] Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edition. The Johns Hopkins University Press, Baltimore, Maryland (2013)

[19] Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review 53(2), 217–288 (2011)

[20] Higham, N.J.: Functions of Matrices, SIAM, Philadelphia (2008)

[21] Householder, A.S.: Dandelin, Lobachevskii, or Graeffe. Amer. Math. Monthly, 66, 464–466 (1959)
[22] E. Kaltofen, Z. Yang, L. Zhi, Structured Low Rank Approximation of a Sylvester Matrix, Proc. 2nd Intern. Workshop on Symbolic-Numerical Algorithms (SNC 05), 188–201 (Dongming Wang and Lihong Zhi, editors), July 2005, Xi’an, China (2005).

[23] Malajovich, G., Zubelli, J. P.: Tangent Graeffe iteration. Numerische Mathematik, 89 (4), 749–782 (2001)

[24] McNamee, J.M.: Numerical Methods for Roots of Polynomials. Part 1 (XIX + 354 pages), Elsevier (2007)

[25] McNamee, J.M., Pan, V.Y.: Numerical Methods for Roots of Polynomials. Part 2 (XXII + 718 pages), Elsevier (2013)

[26] Pan, V.Y.: Complexity of computations with matrices and polynomials. SIAM Review 34(2), 225–262 (1992)

[27] Pan, V.Y.: 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 (1995)

[28] Pan, V.Y.: Approximating complex polynomial zeros: modified quadtree (Weyl’s) construction and improved Newton’s iteration. J. of Complexity, 16(1), 213-264 (2000)

[29] Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, Boston, and Springer, New York (2001)

[30] Pan, V.Y.: A new proximity test for polynomial zeros. Computers and Math. (with Applications), 41(12), 1559-1560 (2001)

[31] Pan, V.Y.: Numerical Computation of a Polynomial GCD and Extensions. Information and Computation, 167, 2, 71–85 (2001).

[32] Pan, V.Y.: Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. J. Symb. Computations 33(5), 701–733 (2002). Proc. version in ISSAC’2001, pp. 253–267, ACM Press, New York (2001)

[33] Pan, V.Y.: Amended DSeSC power method for polynomial root-finding. Computers and Math. (with Applications), 49 (9–10), 1515–1524 (2005)

[34] Pan, V.Y.: Univariate polynomial root-finding by arming with constraints, Proc. of the Forth International Workshop on Symbolic-Numerical Computations (SNC ‘11), San Jose, California, June 2011 (edited by Marc Moreno Maza), 112–121, ACM Press, New York (2011)

[35] Pan, V.Y.: Root-refining for a polynomial equation. Proceedings of Workshop on Computer Algebra in Scientific Computing (CASC 2012), (V. P. Gerdt et al. editors), Lecture Notes in Computer Science, vol. 7442, 271–282, Springer, Heidelberg (2012)

[36] Pan, V.Y.: Transformations of Matrix Structures Work Again. Linear Algebra and Its Applications, 465, 1–32 (2015)

[37] Pan, V.Y., Qian, G., Yan, X.: Random multipliers numerically stabilize Gaussian and block Gaussian elimination: proofs and an extension to low-rank approximation. arxiv 1406.5802 math.NA (28 pages), June 23, 2014.

[38] Pan, V.Y., Qian, G., Zheng, A.: Real and complex polynomial root-finding via eigensolving and randomization. Proceedings of Workshop on Computer Algebra in Scientific Computing (CASC 2012), (V. P. Gerdt et al. editors), Lecture Notes in Computer Science, vol. 7442, 283–293. Springer, Heidelberg (2012)
Appendix

A Some Maps of the Variables and the Roots

Some basic maps of the roots of a polynomial can be computed at a linear or nearly linear arithmetic cost.

Theorem A.1. (Root Inversion, Shift and Scaling, cf. [29].)

(i) Given a polynomial \( p(x) \) of \( \ell \) and two scalars \( a \) and \( b \), one can compute the coefficients of the polynomial \( q(x) = p(ax + b) \) by using \( O(n \log(n)) \) flops. This bound decreases to \( 2n - 1 \) multiplications if \( b = 0 \).

(ii) Reversing a polynomial inverts all its roots involving no flops, that is, \( p_{\text{rev}}(x) = x^n p(1/x) = \sum_{i=0}^{\ell} p_i x^{n-i} = p_n \prod_{j=1}^{\ell} (1 - xx_j) \)
Note that by shifting and scaling the variable, we can move all roots of \( p(x) \) into a fixed disc, e.g., \( D(0,1) = \{ x : |x| \leq 1 \} \).

**Theorem A.2.** (Dandelin’s Root Squaring, cf. [21].)  
(i) Let a polynomial \( p(x) \) of (1.1) be monic. Then \( q(x) = (-1)^n p(\sqrt{x}) p(-\sqrt{x}) = \prod_{j=1}^n (x-x_j^2) \).
(ii) One can evaluate \( p(x) \) at the \( k \)-th roots of unity for \( k > 2n \) and then interpolate to \( q(x) \) by using \( O(k \log(k)) \) flops overall.

**Remark A.1.** Recursive root-squaring is prone to numerical stability problems because the coefficients of the iterated polynomials very quickly span many orders of magnitude. Somewhat surprisingly, the Boolean complexity of the recursive root-squaring process is relatively reasonable if high output precision is required [27], [32]. Moreover, one can avoid numerical stability problems and perform all iterations with double precision by applying a special tangential representation of the coefficients and the intermediate results proposed in [23]. In this case the computations involve more general operations than flops, but in terms of the CPU time the computational cost per iteration has the same order as \( n^2 \) flops.

**Theorem A.3.** (The Cayley Maps.)  
(i) The map \( y = (x-a\sqrt{-1})/(x+a\sqrt{-1}) \), for any real nonzero scalar \( a \), sends the real axis \( \{ x : x \text{ is real} \} \) onto the unit circle \( C(0,1) = \{ y : |y| = 1 \} \).
(ii) The converse map \( x = a\sqrt{-1} (1-y)/(y+1) \) sends the unit circle \( C(0,1) \) onto the real axis.

## B Some Functional Iterations for Polynomial Root-finding

In this section we recall two celebrated functional iterations for the approximation of a single root of a polynomial \( p(x) \) of (1.1) and all its roots, respectively. These root-finders are important and popular, but not specialized to our task of approximating real roots. Thus they are not central for our study and only used as auxiliary root-refiners.

Hereafter a disc \( D(X,r) \) is said to be \( \gamma \)-isolated for a polynomial \( p(x) \) and \( \gamma > 1 \) if it contains all roots of the polynomial lying in the disc \( D(X,\gamma r) \). In this case say that the disc has the **isolation ratio** at least \( \gamma \).

Newton’s iterations refine an approximation \( y^{(0)} \) to a single root of a polynomial \( p(x) \) of (1.1),

\[
y_0 = c, \quad y^{(h+1)} = y^{(h)} - p(y^{(h)})/p'(y^{(h)}), \quad h = 0, 1, \ldots
\]  

(Ehrlich–Aberth’s iterations refine \( n \) simultaneous approximations \( z_1^{(0)}, \ldots, z_1^{(n)} \) to all \( n \) roots \( x_1, \ldots, x_n \) of such a polynomial,

\[
z_i^{(h+1)} = z_i^{(h)} - 1/e_i^{(h)} \quad \text{for} \quad e_i^{(h)} = p(z_i^{(h)})/p'(z_i^{(h)}) - \sum_{j \neq i} \frac{1}{z_i^{(h)} - z_j^{(h)}}, \quad i = 1, \ldots, n.
\]  

See [21], [25] for various other functional iterations.

As we can see next, both iterative algorithms refine very fast the crude initial approximations to simple isolated roots of a polynomial.

**Theorem B.1.** Assume a polynomial \( p = p(x) \) of (1.1) and let \( 0 < 3(n-1) |y_0 - x_j| < |y_0 - x_j| \) for \( j = 2, \ldots, n \). Then Newton’s iteration (B.1) converges to the root \( x_1 \) quadratically right from the start, namely, \( |y_k - x_1| < 2 |y_0 - x_1|/2^k \) for \( k = 0, 1, \ldots \)

*Proof.* See [43] Theorem 2.4, which strengthens [42] Corollary 4.5.  

**Theorem B.2.** (See [43] Theorem 3.3.) Assume a polynomial \( p = p(x) \) of (1.1) and crude initial approximations \( y_j^{(0)} \) to the roots \( x_j \) such that \( 0 < 3\sqrt{n-1} |y_j^{(0)} - x_j| < |y_j^{(0)} - y_i^{(0)}| \) for \( i \neq j \), \( j = 1, \ldots, n \). Then Ehrlich–Aberth’s iteration converges to the roots \( x_j \) with the cubic rate right from the start, namely, \( |y_j^{(k)} - x_j| < |y_j^{(0)} - x_j|/(2^k \sqrt{(n-1)}) \) for \( j = 1, \ldots, n \) and \( k = 0, 1, \ldots \).  

The paper [45] also proves quadratic convergence of the WDK iterations to all \( n \) roots, lying in some given discs with isolation ratios at least \( 3(n - 1)/8 \).

By exploiting the correlations between the coefficients of a polynomial and the power sums of its roots, the paper [40] had weakened the above assumptions on the initial isolation. More precisely, assuming that a simple root lies in the disc \( D(0,1) \) and that the disc has an isolation ratio at least \( s \geq 1 + 1/\log_2(n) \), the paper [40] increased it to \( cn^d \) for any fixed pair of constants \( c \) and \( d \) at the arithmetic cost \( O(n) \), and at the cost \( O(n \log^2(n)) \) in the case of all \( n \) roots isolated in \( n \) discs.

In the case of a single disc, one can allow even an isolation ratio \( s \geq 1 + c' n^{d'} \) for any pair of constants \( c' \) and \( d' \) and then increase it to \( s \geq cn^d \) for any pair of constants \( c \) and \( d \) at the arithmetic cost \( O(n \log^2(n)) \). Indeed one can achieve this by performing \( n \) root-squaring iterations of Theorem \( \ref{A.2} \) for \( h \) of order \( \log(n) \) because each squaring of the roots also squares the isolation ratio. This lifting process ensures the desired isolation for the lifted roots of the new lifted polynomial, but the descending back to the original roots can be also achieved by using \( O(n \log(n)) \) flops [27], [32].

We refer the reader to Remark \( \ref{A.1} \) on the precision growth in these iterations and their Boolean complexity.

Can we completely relax the assumption of the initial isolation? Empirically fast global convergence (that is, convergence right from the start) is very strong over all inputs for the WDK, Ehrlich–Aberth, and some other iterations that approximate simultaneously all \( n \) roots of a polynomial \( p(x) \) of \( \ref{1.1} \). The papers [34], [41], and [35] have challenged the researchers to support this observation with a formal proof, which is still missing, however.

## C Fast Root-finding Where All Roots Are Real

**Theorem C.1.** (Root-finding Where All Roots Are Real).

(i) The modified Laguerre algorithm of \( \ref{14} \) converges to all roots of a polynomial \( p(x) \) of \( \ref{1.1} \) right from the start, uses \( O(n) \) flops per iteration, and therefore approximates all \( n \) roots within \( \epsilon = 1/2^b \) by using \( O(\log(b)) \) iterations and performing \( O(n \log(b)) \) flops.

(ii) The latter asymptotic arithmetic cost bound is optimal and is supported by the alternative algorithms of \( \ref{1} \) and \( \ref{2} \) as well.

(iii) All these algorithms reach the optimal Boolean cost bound up to polylogarithmic factors.

## D Counting the Roots in a Disc. Root Radii, Distances to the Roots, and the Proximity Tests

In this subsection we estimate the distances to the roots of \( p(x) \) from a complex point and the number of the roots in an isolated disc.

The latter task can be solved by using the following result from [42, Lemma 7.1] (cf. also [43, Theorem 14.1]).

**Theorem D.1.** [42, Lemma 7.1] It is sufficient to perform FFT at \( n' = 16 \log_2(n) \) points (using \( 1.5n' \log(n') \) flops) and \( O(n) \) additional flops and comparisons of real numbers with 0 in order to compute the number of roots of a polynomial \( p(x) \) of \( \ref{1.1} \) in a 9-isolated disc \( D(0, r) \).

**Remark D.1.** The algorithm of [42] supporting Theorem \( \ref{1} \) only uses the signs of the real and imaginary parts of the \( n \) output values of FFT. For some groups of the values, the pairs of the signs stay invariant and can be represented by a single pair of signs. Can this observation be exploited in order to decrease the computational cost of performing the algorithm?

**Corollary D.1.** It is sufficient to perform \( O(h \log(n)) \) flops and \( O(n) \) comparisons of real numbers with 0 in order to compute the number of roots of a polynomial \( p(x) \) of \( \ref{1.1} \) in a \( s \)-isolated disc \( D(0, r) \) for \( s = 9^{1/2^h} \) and for any positive integer \( h \).

**Proof.** Every root-squaring of Theorem \( \ref{A.2} \) squares all root-radii and the isolation ratios of all discs \( D(0, r) \). Suppose \( h \) repeated squaring iterations map a polynomial \( p(x) \) into \( p_h(x) \), for which the
disc $D(0, 1)$ is 9-isolated. Then we can compute the number of roots of $p_h(x)$ in this disc by applying Theorem D.1 which is the same as the number of roots of $p(x)$.

In view of Remark A.1 one must apply the slower operations of [23] or high precision computations in order to support even a moderately long sequence of root-squaring iterations, but in some cases it is sufficient to apply Corollary D.1 for small positive integers $h$. Note that $9^{1/2^h}$ is equal to 1.3160... for $h = 2$, to 1.1472... for $h = 3$, to 1.0710... for $h = 4$ and to 1.0349... for $h = 5$.

We can use the following result if we agree to perform computations with extended precision.

**Theorem D.2.** (The Root Radii Approximation.)

Assume a polynomial $p(x)$ of (1.1) and two real scalars $c > 0$ and $d$. Define the $n$ root radii $r_j = |x_{k_j}|$ for $j = 1, \ldots, n$, distinct $k_1, \ldots, k_n$, and $r_1 \geq r_2 \geq \cdots \geq r_n$. Then, by using $O(n \log^2(n))$ flops, one can compute $n$ approximations $\tilde{r}_j$ to the root radii $r_j$ such that $\tilde{r}_j \leq r_j \leq (1 + c/n^d)\tilde{r}_j$, for $j = 1, \ldots, n$.

**Proof.** (Cf. [43], Section 4.) At first fix a sufficiently large integer $k$ and apply $k$ times the root-squaring of Theorem A.2 which involves $O(k n \log(n))$ flops. Then apply the algorithm of [43] (which uses $O(n)$ flops) in order to approximate within a factor of $2n$ all root radii $r_j^{(k)} = r_j^{2^k}$, $j = 1, \ldots, n$, of the output polynomial $p_k(x)$. By taking the $2^k$-th roots, approximate the root radii $r_1, \ldots, r_n$ within a factor of $(2n)^{1/2^k}$, which is $1 + c/n^d$ for $k$ of order $\log(n)$.

Alternatively we can approximate the root radii by employing the Geršgorin theorem to the companion or generalized companion matrices of a polynomial $p(x)$ (1.1), by applying the heuristic method of [2], used in the packages MPSolve 2000 and 2012 [5], [9], or by recursively applying Theorem D.1 although neither of these techniques support competitive complexity estimates.

The following two theorems define the largest root radius $r_1$ of the polynomial $p(x)$.

**Theorem D.3.** (See [46].) Assume a polynomial $p(x)$ of (1.1). Write $r_1 = \max_{j=1}^n |x_j|$, $r_n = \min_{j=1}^n |x_j|$, and $\gamma^+ = \max_{j=1}^n |p_{n-i}/p_n|$. Then $\gamma^+ / n \leq r_1 \leq 2\gamma^+$.

**Theorem D.4.** (See [30].) For $\epsilon = 1/2^b > 0$, one only needs $a(n, \epsilon) = O(n + b \log(b))$ flops to compute an approximation $r_{1, \epsilon}$ to the largest root radius $r_1$ of $p(x)$ such that $r_{1, \epsilon} \leq r_1 \leq 5(1 + \epsilon)r_{1, \epsilon}$. In particular, $a(n, \epsilon) = O(n)$, for $b = O(n/\log(n))$, and $a(n, \epsilon) = O(n \log(n))$, for $b = O(n)$.

Both theorems can be immediately extended to the approximation of the smallest root radius $r_n$ because it is the largest root radius of the reverse polynomial $p_{\text{rev}}(x) = x^n p(1/x)$ (cf. Theorem A.1). Moreover, by shifting a complex point $c$ into the origin (cf. Theorem A.1), we can turn our estimates for the root radii into the estimates for the distances to the roots from the point $c$. Approximation of the smallest distance from a complex point $c$ to a root of $p(x)$ is called the proximity test at the point. One can perform such a test by applying Theorems D.1, D.3 or D.4.

Alternatively, for heuristic proximity tests by action at a point $c$ or at $n$ points, one can apply Newton’s iterations (B.1) or an appropriate functional iterations, such as the Ehrlich–Aberth iterations (B.2), and estimate the distance to the roots by observing convergence or divergence of the iterations.

Theorems D.3 and all these iterations, including Newton’s, Ehrlich–Aberth’s and WDK’s, can be applied even where a polynomial $p(x)$ is defined by a black box subroutine for its evaluation rather than by its coefficients.