2014

TR-2014011: Real Polynomial Root-Finding: New Advances

Victor Y. Pan

Real Polynomial Root-finding: New Advances

Victor Y. Pan

Departments of Mathematics and Computer Science
Lehman College and the Graduate Center of the City University of New York
Bronx, NY 10468 USA
victor.pan@lehman.cuny.edu,
home page: http://comet.lehman.cuny.edu/vpan/
(Supported by NSF Grant CCF 1116736)

Abstract. Univariate polynomial root-finding is both classical and 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 typically nonreal roots are much more numerous than the real ones. We dramatically accelerate the known algorithms in this case by exploiting the correlation between the computations with matrices and polynomials, extending the techniques of the matrix sign iteration, and exploiting the structure of the companion matrix of the input polynomial.

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

1 Introduction

Assume a univariate polynomial of degree n with real coefficients,

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

which has r real roots $x_1, \dots, x_r$ and $s = (n-r)/2$ pairs of nonreal complex conjugate roots. In some applications, e.g., to algebraic and geometric optimization, one seeks just the r real roots, which make up just a small fraction of all roots. This is a well studied subject (see [13, Chapter 15], [21], [25], and the bibliography therein), but we dramatically accelerate the known algorithms by combining and extending the techniques of [23] and [20]. At first our iterative Algorithm 1 reduces the original problem to the same problem for an auxiliary polynomial of degree r having r real roots. Our iterations converge with quadratic rate, and so we need only $k = O(b+d)$ iterations, assuming the tolerance 2^{-b} to the error norm of the approximation to the auxiliary polynomial (we denote it $v_k(x)$) and the minimal distance 2^{-d} of the nonreal roots from the real axis. The values d and k are large for the input polynomials with nonreal roots lying very close to the real axis, but our techniques of Remark 4 enable us to handle such harder

inputs as well. The known algorithms approximate the roots of $v_k(x)$ at a low arithmetic cost, and then we recover the r real roots of the input polynomial $p(x)$. Overall we perform $O(kn\log(n))$ arithmetic operations. This arithmetic cost bound is quite low, but in the case of large degree n, the algorithm is prone to numerical problems, and so we devise dual Algorithm 2 and propose some special recipes in Remark 8 to avoid the latter problems. This works quite well according to our test results, but the formal study of the issue and of the Boolean complexity of the algorithm is left as a research challenge.

Let us comment briefly on the techniques involved and the complexity of the algorithm. It performs computations in the Frobenius matrix algebra generated by the companion matrix of the input polynomial. By using FFT and exploiting the structure of the matrices in the algebra, we operate with them as fast as with polynomials. Real polynomial root-finding is reduced to real eigen-solving for the companion matrix. Transition to matrices and the randomization techniques, extended from [20, Section 5], streamline and simplify the iterative process of Algorithm 1. Now this process outputs an auxiliary $r \times r$ matrix L whose eigenvalues are real and approximate the r real eigenvalues of the companion matrix. It remains to apply the QR algorithm to the matrix L, at the arithmetic cost $O(r^3)$ (cf. [9, page 359]), dominated if $r^3 = O(kn\log(n))$.

We engage, extend, and combine the number of efficient methods available for complex polynomial root-finding, particularly the ones of [23] and [20], but we also propose new techniques and employ some old methods in novel and nontrivial ways. Our Algorithm 1 streamlines and substantially modifies [23, Algorithm 9.1] by avoiding the stage of root-squaring and the application of the Cayley map. Some techniques of Algorithm 2 are implicit in [20, Section 5], but we specify a distinct iterative process, employ the Frobenius matrix algebra, extend the matrix sign iteration to real eigen-solving, employ randomization and the QR algorithm, and include the initial acceleration by scaling. Our Algorithm 3 naturally extends Algorithms 1 and 2, but we show that this extension is prone to the problems of numerical stability, and our finding can be applied to the similar iterations of [3] and [7] as well. Algorithm 4 can be linked to Algorithm 1 and hence to [20, Section 5], but incorporates some novel promising techniques. Our simple recipe for real root-finding by means of combining the root radii algorithm with Newton's iteration in Algorithm 5 and even the extension of our Algorithm 2 to the approximation of real eigenvalues of a real matrix are also novel and promising. Some of our algorithms take advantage of combining the power of operating with matrices and polynomials (see Remarks 11 and 13). Finding their deeper synergistic combinations is another natural research challenge, traced back to [14] and [2]. Our exploitation of the complex plane geometry, various transforms of the variable and of the roots (including simple but apparently novel Theorem 1) can be of independent interest.

Hereafter "ops" stands for "arithmetic operations", "lc(p)" stands for "the leading coefficient of $p(x)$". $D(X,r) = \{x : |x - X| \le r\}$ and $C(X,r) = \{x : |x - X| = r\}$ denote a disc and a circle on the complex plane, respectively. We write $||\sum_i v_i x^i||_q = (\sum_i |v_i|^q)^{1/q}$ for $q = 1, 2$ and $||\sum_i v_i x^i||_\infty = \max_i |v_i|$. A

function is in $\tilde{O}(f(bc))$ if it is in $O(f(bc))$ up to polylogarithmic factors in b and c. $\mathrm{agcd}(u, v)$ denotes the *approximate greatest common divisor* of two polynomials $u(x)$ and $v(x)$ (see [1] on definitions and algorithms).

2 Some Basic Results for Polynomial Computations

Next we cover the auxiliary algorithms that map polynomial roots, approximate their absolute values, split a polynomial into two factors, and approximate its roots if it has only real roots (the two latter algorithms are the two basic blocks of our first algorithm, cited in the introduction).

Theorem 1. *(*Mapping the Roots*.) Assume a polynomial $p(x)$ of (1) and a rational function $y(x) = w(x) + u(x)/v(x)$ for three polynomials $w(x)$, $u(x)$, and $v(x)$ of degree k, l, and m, respectively, where $l < m$. Assume n values $s_1, \dots, s_n$ such that the values $y(s_1), \dots, y(s_n)$ are distinct. (They are distinct with probability 1 if $s_1, \dots, s_n$ are independent Gaussian random variables.) Then it is sufficient to apply $O(m \log^2(n) + n \log^2(m))$ ops in order to compute the polynomial $q(x) = \prod_{j=1}^{n}(x - y_j)$ such that $y_j = y(x_j)$ for $j = 1, \dots, n$.*

Proof. Compute the values $q(y(s_j)) = p(s_j)$ for $j = 1, \dots, n$, interpolate to the polynomial $q(x)$, and recall the known cost estimates for polynomial evaluation and interpolation (cf. [17, Sections 3.1 and 3.3]).

We can compute some important maps of the roots at a little lower cost.

Theorem 2. *(*Root Inversion, Shift and Scaling*, cf. [17].) Given a polynomial $p(x)$ of (1) and two scalars a and b, we can compute the coefficients of the polynomial $q(x) = p(ax + b)$ by using $O(n \log(n))$ ops. We need only $2n - 1$ ops if $b = 0$. Reversing a polynomial inverts all its roots involving no ops, that is, $p_{\mathrm{rev}}(x) = x^n p(1/x) = \sum_{i=0}^{n} p_i x^{n-i} = p_n \prod_{j=1}^{n}(1 - x x_j)$.*

Theorem 3. *(*Root Squaring*, cf. [10].) (i) Assume that a polynomial $p(x)$ of (1) is monic. Then the map $q(x) = (-1)^n p(\sqrt{x})p(-\sqrt{x})$ squares the roots, that is, $q(x) = \prod_{j=1}^{n}(x - x_j^2)$, and (ii) one can evaluate $p(x)$ at the k-th roots of unity for $k > 2n$ and then interpolate to $q(x)$ by using $O(n \log(n))$ ops.*

Theorem 4. *(*The Cayley Maps*, cf. [9].) The maps $y = (x - \sqrt{-1})/(x + \sqrt{-1})$ and $x = \sqrt{-1}(y + 1)/(y - 1)$ send the real axis $\{x : \ x \text{ is real}\}$ onto the unit circle $C(0, 1) = \{y : \ |y| = 1\}$, and vice versa.*

Theorem 5. *(*Möbius Map*.) (i) The maps $\widehat{y} = \frac{1}{2}(\widehat{x} + 1/\widehat{x})$, $\widehat{x} = \widehat{y} \pm \sqrt{\widehat{y}^2 - 1}$ and $y = \frac{1}{2}(x - 1/x)$, $x = y \pm \sqrt{y^2 + 1}$ send the unit circle $C(0, 1) = \{x : \ |x = 1|\}$ into the line intervals $[-1, 1] = \{\widehat{y} : \ \Im(\widehat{y}) = 0, \ -1 \le \widehat{y} \le 1\}$ and $[-\sqrt{1}, \sqrt{-1}] = \{y : \ \Re(y) = 0, \ -1 \le y\sqrt{-1} \le 1\}$, and vice versa. (ii) Write $\widehat{y} = \frac{1}{2}(\widehat{x} + 1/\widehat{x})$, $\widehat{y}_j = \frac{1}{2}(\widehat{x}_j + 1/\widehat{x}_j)$, $y = \frac{1}{2}(x - 1/x)$, and $y_j = \frac{1}{2}(x_j - 1/x_j)$, for $j = 1, \dots, n$. Then $\widehat{q}(\widehat{y}) = p(\widehat{x})p(1/\widehat{x}) = \widehat{q}_n \prod_{j=1}^{n}(\widehat{y} - \widehat{y}_j)$ (cf. [3, equation (14)]) and $q(y) = p(x)p(-1/x) = q_n \prod_{j=1}^{n}(y - y_j)$. (iii) Given a polynomial $p(x)$ of (1), one can interpolate to the polynomials $\widehat{q}(y)$ and $q(y)$ by using $O(n \log(n))$ ops.*

Proof. Follow the proof of Theorem 1 for $s_j = \exp(2\pi j\sqrt{-1}/n)$, $j = 0, \dots, n-1$. Use FFT for the evaluation. Note that $\frac{1}{2}(x + 1/x) = \cos(\phi)$ and $\frac{1}{2}(x - 1/x) = \sin(\phi)$ for $x = \exp(\phi\sqrt{-1})$ and real ϕ, and extend the algorithms of [16] for the interpolation (cf. [3, Section 2]).

Theorem 6. *(*Error Bounds of the Möbius Iteration*.) Fix a complex $x = x^{(0)}$ and define the iterations*

$$x^{(h+1)} = \frac{1}{2}(x^{(h)} + 1/x^{(h)}) \text{ and } \gamma = \sqrt{-1} \text{ for } h = 0, 1, \dots, \tag{2}$$

$$x^{(h+1)} = \frac{1}{2}(x^{(h)} - 1/x^{(h)}) \text{ and } \gamma = 1 \text{ for } h = 0, 1, \dots \tag{3}$$

If $x^{(0)}\gamma$ is real, then $x^{(h)}\gamma$ are real for all h. Otherwise $|x^{(h)} - \text{sign}(x)\sqrt{-1}/\gamma| \le \frac{2\tau^{2^h}}{1-\tau^{2^h}}$ for $\tau = |\frac{x-\text{sign}(x)}{x+\text{sign}(x)}|$ and $h = 0, 1, \dots$

Proof. Under (2), for $\gamma = \sqrt{-1}$, the bound is from [3, page 500]). It is readily extended to the case of (3), for $\gamma = 1$.

Theorem 7. *(*Root Radii Approximation*, cf. [24], [13, Section 15.4], [5].) Assume a polynomial $p(x)$ of (1) and two real scalars $c > 0$ and d. Define the n* root radii *$r_j = |x_{k_j}|$ for $j = 1, \dots, n$ and $r_1 \ge r_2 \ge \cdots \ge r_n$. Then we can compute approximations $\tilde{r}_j$ such that $\tilde{r}_j \le r_j \le (1 + c/n^d)\tilde{r}_j$, for $j = 1, \dots, n$, by using $O(n\log^2(n))$ ops.*

By combining Theorems 7 and 2 we can move the roots of a polynomial into a fixed disc, e.g., $D(0, 1) = \{x : |x| \le 1\}$.

Theorem 8. *(*Root-finding Where All Roots Are Real*). The modified Laguerre algorithm of [8] converges to all roots of a polynomial $p(x)$ of (1) right from the start, uses $O(n)$ ops per iteration, and therefore approximates all n roots within $\epsilon = 1/2^b$ by using $O(\log(b))$ iterations and performing $\tilde{O}(n\log(b))$ ops overall. This asymptotic cost bound is optimal and is also supported by the alternative algorithms of [6] and [4].*

Theorem 9. *(*Splitting a Polynomial into Two Factors Over a Circle*, cf. [24] or [13, Chapter 15].) Suppose a polynomial $t(x)$ of degree n has r roots inside the circle $C(0, \rho)$ and $n - r$ roots outside the circle $C(0, R)$ for $R/\rho \ge 1 + 1/n$. Let $\epsilon = 1/2^b$ for $b \ge n$. Then by performing $O((\log^2(n) + \log(b))n\log(n))$ ops (that is, $O(n\log^3(n))$ ops for $\log(b) = O(\log^2(n))$), with a precision of $O(b)$ bits, we can compute two polynomials $\tilde{f}$ and $\tilde{g}$ such that $||p - \tilde{f}\tilde{g}||_q \le \epsilon||p||_q$ for $q = 1, 2$ or ∞, the polynomial $\tilde{f}$ of degree r has r roots inside the circle $C(0, 1)$, and the polynomial $\tilde{g}$ of degree $n - r$ has $n - r$ roots outside this circle.*

Remark 1. (*Increasing Isolation by Means of Repeated Squaring.*) Let the assumptions of Theorem 9 hold, except that $R/\rho = 1 + c/n^d < 1 + 1/n$, for two positive constants c and d. Then the map of Theorem 3 squares the ratio R/ρ, and so $d = O(\log(n))$ applications of this map (using $O(n\log^2(n))$ ops overall) increase the ratio above $1 + 1/n$, which supports the application of Theorem 9.

3 Root-finding As Eigen-solving and Basic Definitions and Results for Matrix Computations

3.1 Some Basic Definitions for Matrix Computations

$M^T = (m_{ji})_{i,j=1}^{n,m}$ is the transpose of a matrix $M = (m_{ij})_{i,j=1}^{m,n}$. M^H is its Hermitian transpose. $I = I_n = (\mathbf{e}_1 \mid \mathbf{e}_2 \mid \dots \mid \mathbf{e}_n)$ is the $n \times n$ identity matrix, whose columns are the n coordinate vectors $\mathbf{e}_1$, $\mathbf{e}_2, \dots, \mathbf{e}_n$. $J = J_n = (\mathbf{e}_n \mid \mathbf{e}_{n-1} \mid \dots \mid \mathbf{e}_1)$ is the $n \times n$ reversion matrix, $J(v_i)_{i=1}^n = (v_i)_{i=n}^1$. $\mathrm{diag}(b_j)_{j=1}^s = \mathrm{diag}(b_1, \dots, b_s)$ is the $s \times s$ diagonal matrix with the diagonal entries $b_1, \dots, b_s$.

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 [9, Theorem 5.2.2].

M^+ is the Moore–Penrose pseudo inverse of M [9, Section 5.5.4]. An $n \times m$ matrix $X = M^{(I)}$ is a left (resp. right) inverse of an $m \times n$ matrix M if $XM = I_n$ (resp. if $MY = I_m$). $M^{(I)} = M^+$ for a matrix M of full rank. $M^{(I)} = M^H$ for a unitary matrix M. $M^{(I)} = M^+ = M^{-1}$ for a nonsingular matrix M.

$\mathcal{R}(M)$ 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.

Definition 1. *$\mathcal{S}$ is the* invariant subspace *of a square matrix M if $M\mathcal{S} = \{M\mathbf{v} : \mathbf{v} \in \mathcal{S}\} \subseteq \mathcal{S}$. A scalar λ is an* eigenvalue *of a matrix M associated with an* eigenvector $\mathbf{v}$ *if $M\mathbf{v} = \lambda\mathbf{v}$. All eigenvectors associated with an eigenvalue λ of M form an eigenspace $\mathcal{S}(M, \lambda)$, which is an invariant space. Its dimension d is the* geometric multiplicity *of λ. The eigenvalue is simple if its multiplicity is* 1. *The set $\Lambda(M)$ of all eigenvalues of a matrix M is called its* spectrum.

3.2 The Companion Matrix and the Frobenius Algebra

$$C_p = \begin{pmatrix} 0 & & & -p_0/p_n \\ 1 & \ddots & & -p_1/p_n \\ & \ddots & \ddots & \vdots \\ & & \ddots\ 0 & -p_{n-2}/p_n \\ & & 1 & -p_{n-1}/p_n \end{pmatrix} \text{ and } C_{p_{\mathrm{rev}}} = \begin{pmatrix} 0 & & & -p_n/p_0 \\ 1 & \ddots & & -p_{n-1}/p_0 \\ & \ddots & \ddots & \vdots \\ & & \ddots\ 0 & -p_2/p_0 \\ & & 1 & -p_1/p_0 \end{pmatrix}$$

for $p_0 \neq 0$ denote the *companion matrices* of the polynomials $p(x)$ of (1) and $p_{\mathrm{rev}}(x)$ of Theorem 2, respectively, $C_p^{-1} = JC_{p_{\mathrm{rev}}}J$. $p(x) = c_{C_p}(x) = \det(xI_n - C_p)$ is the *characteristic polynomial* of $p(x)$. Its roots form the spectrum of C_p, and so real root-finding for $p(x)$ turns into real eigen-solving for the matrix C_p.

Theorem 10. *(*The Cost of Computations in the Frobenius Matrix Algebra*, cf. [7].) The companion matrix $C_p \in \mathbb{C}^{n \times n}$ of a polynomial $p(x)$ of (1) generates the Frobenius matrix algebra $\mathcal{A}_p$. One needs $O(n)$ ops for addition, $O(n \log(n))$ ops for multiplication, and $O(n \log^2(n))$ ops for inversion in this algebra. One needs $O(n \log(n))$ ops to multiply a matrix in this algebra by a vector.*

3.3 Decreasing the Size of an Eigenproblem

Next we reduce eigen-solving for the matrix C_p to the study of its invariant space generated by the r eigenspaces associated with the r real eigenvalues. The following theorem is basic for this step.

Theorem 11. *(*Decreasing the Eigenproblem Size to the Dimension of an Invariant Space*, cf. [26, Section 2.1].) Let $U \in \mathbb{C}^{n\times r}$, $\mathcal{R}(U) = \mathcal{U}$, and $M \in \mathbb{C}^{n\times n}$. Then $\mathcal{U}$ 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 $L = U^{(I)}MU$. The matrix L is unique (that is independent of the choice of the left inverse $U^{(I)}$) if U is a matrix basis for the space $\mathcal{U}$. (Hence $MU\mathbf{v} = \lambda U\mathbf{v}$ if $L\mathbf{v} = \lambda\mathbf{v}$, $\Lambda(L) \subseteq \Lambda(M)$, and if U is a unitary matrix, then $L = U^H MU$.)*

By virtue of the following theorem, a matrix function shares its invariant spaces with the matrix C_p, and so we can facilitate the computation of the desired invariant space of C_p if we reduce the task to the case of an appropriate matrix function, for which the solution is simpler.

Theorem 12. *(*The Eigenproblems for a Matrix and Its Function.*) Suppose M is a square matrix, a rational function $f(\lambda)$ is defined on its spectrum, and $M\mathbf{v} = \lambda\mathbf{v}$. Then (i) $f(M)\mathbf{v} = f(\lambda)\mathbf{v}$. (ii) Let $\mathcal{U}$ be the eigenspace of the matrix $f(M)$ associated with its eigenvalue μ. Then this is an invariant space of the matrix M generated by its eigenspaces associated with all its eigenvalues λ such that $f(\lambda) = \mu$. (iii) The space $\mathcal{U}$ is associated with a single eigenvalue of M if μ is a simple eigenvalue of $f(M)$.*

Proof. We readily verify part (i), which implies parts (ii) and (iii).

Suppose we have computed a matrix basis $U \in \mathbb{C}^{n\times r}$ for an invariant space $\mathcal{U}$ of a matrix function $f(M)$ of an $n \times n$ matrix M. By virtue of Theorem 12, this is a matrix basis of an invariant space of the matrix M. We can first compute a left inverse $U^{(I)}$ 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 $L = U^{(I)}MU = Q^H MQ$ (cf. Theorem 11).

Given an approximation $\tilde{\mu}$ to a simple eigenvalue of a matrix function $f(M)$, we can compute an approximation $\tilde{\mathbf{u}}$ to an eigenvector $\mathbf{u}$ of the matrix $f(M)$ associated with this eigenvalue, recall from part (iii) of Theorem 12 that this is also an eigenvector of the matrix M, associated with its simple eigenvalue, and approximate this eigenvalue by the Rayleigh Quotient $\frac{\tilde{\mathbf{u}}^T M\tilde{\mathbf{u}}}{\tilde{\mathbf{u}}^T\tilde{\mathbf{u}}}$.

3.4 Some Maps in the Frobenius Matrix Algebra

Part (i) of Theorem 12 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 spectrum $\Lambda(f(C_p)) = \{f(x_i)\}_{i=1}^n$. In particular, the maps

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

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

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

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

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

where $y = y(x)$ denotes polynomials. Apply the two latter maps recursively, to define two iterations with polynomials modulo $p(x)$ as follows, $y_0 = x$, $y_{h+1} = 0.5(y_h + 1/y_h) \mod p(x)$ (cf. (2)) and $y_0 = x$, $y_{h+1} = 0.5(y_h - 1/y_h) \mod p(x)$ (cf. (3)), $h = 0, 1, \ldots$. 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.

4 Real Root-finders

4.1 Möbius Iteration

Theorem 6 implies that right from the start of iteration (3) the values $x^{(h)}$ converge fast to $\pm\sqrt{-1}$ unless the initial value $x^{(0)}$ is real, in which case all iterates $x^{(h)}$ are real. It follows that right from the start the values $y^{(h)} = (x^{(h)})^2 + 1$ converge fast to 0 unless $x^{(0)}$ is real, whereas all values $y^{(h)}$ are real and exceed 1 if $x^{(0)}$ is real. Write $t_h(y) = \prod_{j=1}^{n}(y - (x_j^{(h)})^2 - 1)$ for $h = 1, 2, \ldots$ and $v_h(y) = \prod_{j=1}^{r}(y - (x_j^{(h)})^2 - 1)$. The roots of the polynomials $t_h(y)$ and $v_h(y)$ are the images of all roots and of the real roots of the polynomial $p(x)$ of (1), respectively, produced by the composition of the maps (3) and $y^{(h)} = (x^{(h)})^2 + 1$. Hence $t_h(y) \approx y^{2s} v_h(y)$ for large integers h where the polynomial $v_h(y)$ has degree r and exactly r real roots, all exceeding 1, and so for large integers h, the sum of the $r + 1$ leading terms of the polynomial $t_h(y)$ closely approximates the polynomial $y^{2s} v_h(y)$. (To verify that the $2s$ trailing coefficients nearly vanish, we need just $2s$ comparisons.) The above argument shows correctness of the following algorithm. (One can similarly employ and analyze iteration (2).)

Algorithm 1. Möbius iteration for real root-finding.
INPUT: *two integers n and r, $0 < r < n$, and the coefficients of a polynomial $p(x)$ of equation (1) where $p(0) \neq 0$.*
OUTPUT: *approximations to the real roots $x_1, \ldots, x_r$ of $p(x)$.*
COMPUTATIONS:

1. *Write $p_0(x) = p(x)$ and recursively compute the polynomials $p_{h+1}(y)$ such that $p_{h+1}(y) = p_h(x)\ p_h(-1/x)$ for $y = (x - 1/x)/2$ and $h = 0, 1, \ldots$ (Part (ii) of Theorem 5 and Theorem 6 combined define the images of the real and nonreal roots of the polynomial $p(x)$ for all h.)*

2. *Periodically, at some selected Stages k, compute the polynomials*

$$t_h(y) = (-1)^n q_k(\sqrt{y+1})\ q_h(-\sqrt{y+1})$$

where $q_k(y) = p_k(y)/\mathrm{lc}(p_k)$ (cf. Theorems 2 and 3). When the integer k becomes large enough, so that $2s$ trailing coefficients of the polynomial $q_k(x)$ nearly vanish, compute an approximate factor $v_k(x)$ of the polynomial $t_k(x)$ that has degree r and has r real roots on the ray $\{x:\ x \geq 1\}$.

3. *Apply one of the algorithms of [6], [4], and [8] (cf. Theorem 8) to approximate the r roots of the polynomial $v_k(x)$.*
4. *Extend the descending process from [15], [18] and [3] to recover approximations to the r roots $x_1, \ldots, x_r$ of the polynomial $p_0(x) = p(x)$. At first having the r roots w_j of the polynomial $v_k(x)$, compute the $2r$ candidates $\pm\sqrt{w_j - 1}$, $j = 1, \ldots, r$, for being approximations to the r real roots of the polynomials $q_k(y)$ and $p_k(y)$. Then select r of them on which the polynomials $q_k(y)$ vanishes or nearly vanishes. Apply part (i) of Theorem 5 to define from these r roots the $2r$ candidates for approximating the r real roots of $p_{k-1}(x)$. Recursively descend down to the r real roots of $p_0(x) = p(x)$. This process is not ambiguous because only r roots of the polynomial $p_h(x)$ are real for each h, by virtue of Theorem 6.*

Like lifting Stage 1, descending Stage 4 involves order of $kn\log(n)$ ops, which also bounds the overall cost of performing the algorithm.

Remark 2. (Countering Degeneracy.) If $p(0) = p_0 = \cdots = p_m = 0 \neq p_{m+1}$, then we should output the real root $x_0 = 0$ of multiplicity m and apply the algorithm to the polynomial $p(x)/x^m$ to approximate the other real roots. Alternatively we can apply the algorithm to the polynomial $q(x) = p(x - s)$ for a shift value s such that $q(0) \neq 0$. With probability 1, $q(0) \neq 0$ for Gaussian random variable s, but we can approximate the root radii of the polynomial $p(x)$ (cf. Theorem 7) and then deterministically find a scalar s such that $q(x)$ has no roots near 0. Moreover, we can avoid both degeneracy problem and numerical problems by extending our recipes of Remark 8, proposed for dual Algorithm 2.

Remark 3. (Saving the Recursive Steps of Stage 1.) The basic step of the algorithm is the computation of an approximate factor $v_k(x)$ of $t_k(x)$, having degree r and r real roots. We would decrease the overall computational cost if we compute such a polynomial $v_k(x)$ for a smaller integer k. Theorem 9 enables us to do this (at a reasonable cost) if its assumptions are satisfied for $t(x) = t_k(x)$, which we can verify by applying the root radii algorithm of Theorem 7. For a fixed k this requires $O(n\log^2(n))$ ops, so even the verification for all integers k in the range is not costly, unless the range is large, but we can apply the binary search for the minimum integer k satisfying Theorem 9, and then we would need only $O(\log(n))$ approximations of the n root radii, by using $O(n\log^3(n))$ ops.

Remark 4. (Handling the Nearly Real Roots.) The integer parameter k and the overall arithmetic cost of performing the algorithm are large if the value $2^{-d} =$

$\min_{j=r+1}^{n} |\Im(x_j)|$ is small. We can counter this deficiency by splitting out from the polynomial $t_k(x)$ its factor $v_{k,+}(x)$ of degree $r_+ > r$ that has r_+ real and nearly real roots such that the other nonreal roots lie sufficiently far from the real axis. Our convergence analysis and the techniques for splitting out the factor $v_k(x)$ can be readily extended. At a low cost we can compute its roots (r of which are real) if the integer r_+ is small. For any r, we can apply the modified Laguerre algorithm of [8], expecting that it still converges fast to the r_+ roots of the polynomial $v_{k,+}(x)$ if all of them lie on or close enough to the real axis.

Remark 5. (The Number of Real Roots.) We assume that we know the number r of the real roots (e.g., supplied by noncostly algorithms of computer algebra), but we can compute this number as by-product of Stage 2, and similarly for our other algorithms. With a proper policy we can compute the integer r by testing at most $2 + 2\lceil \log(r) \rceil$ candidates in the range $[0, 2r - 1]$.

Remark 6. The known upper bounds on the condition numbers of the roots of the computed polynomials $p_k(y)$ grow exponentially as k grows large (cf. [3, Section 3]). So, unless these bounds are overly pessimistic, Algorithm 1 is prone to numerical stability problems already for moderately large integers k.

4.2 Extended Matrix Sign Iteration

To avoid the latter potential deficiency, we replace the polynomial iteration at Stages 1 and 2 by the dual iteration

$$Y_{h+1} = 0.5(Y_h - Y_h^{-1}) \text{ for } h = 0, 1, \dots, \tag{4}$$

which extends the *matrix sign* iteration $\widehat{Y}_{h+1} = 0.5(\widehat{Y}_h + \widehat{Y}_h^{-1})$ for $h = 0, 1, \dots$ (cf. (2), (3), part (ii) of our Theorem 5, and [11]) and maps the eigenvalues of the matrix Y_0 according to (3). Therefore Stage 1 of Algorithm 1 maps the characteristic polynomials of the above matrices Y_h. Unlike the case of the latter map, working with matrices enables us to recover the desired real eigenvalues of the matrix C_p by means of our recipes of Section 3, without recursive descending.

Algorithm 2. Matrix sign iteration modified for real eigen-solving.
INPUT AND OUTPUT *as in Algorithm 1, except that FAILURE can be output with a probability close to 0.*
COMPUTATIONS:

1. *Write $Y_0 = C_p$ and recursively compute the matrices Y_{h+1} of (4) for $h = 0, 1, \dots$ (2s eigenvalues of the matrix Y_h converge to $\pm\sqrt{-1}$ as $h \to \infty$, whereas its r other eigenvalues are real for all h, by virtue of Theorem 6.)*
2. *Fix a sufficiently large integer k and compute the matrix $Y = Y_k^2 + I_n$. (The map $Y_0 = C_p \to Y$ 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 the randomized algorithms of [12] to compute the numerical rank of the matrix Y. Suppose it equals r. (It is at least r, and if it exceeds r, then go back to Stage 1.) Generate a standard Gaussian random $n \times r$ matrix G and compute the matrices $H = YQ(G)$ and $Q = Q(H)$. (The analysis of preprocessing with Gaussian random multipliers in [12, Section 4], [19, Section 5.3] shows that, with a probability close to 1, the columns of the matrix Q closely approximate a unitary basis of the invariant space of the matrix Y associated with its r absolutely largest eigenvalues, which are the images of the real eigenvalues of the matrix C_p. Having this approximation is equivalent to having a small upper bound on the residual norm $||Y - QQ^HY||$ [12], [19].) Verify the latter bound. In the unlikely case where the verification is failed, output FAILURE and stop the computations.*
4. *Otherwise compute and output approximations to the r eigenvalues of the $r \times r$ matrix $L = Q^H C_p Q$. They approximate the real roots of the polynomial $p(x)$. (Indeed, by virtue of Theorem 12, Q is an approximate matrix basis for the invariant space of the matrix C_p associated with its r real eigenvalues. Therefore, by virtue of Theorem 11, the r eigenvalues of the matrix L approximate the r real eigenvalues of the matrix C_p.)*

Stages 1 and 2 involve $O(kn\log^2(n))$ ops by virtue of Theorem 10. This exceeds the estimate for Algorithm 1 by a factor of $\log(n)$. Stage 3 adds $O(nr^2)$ ops and the cost a_{rn} of generating $n \times r$ standard Gaussian random matrix. The cost bounds are $O(r^3)$ at Stage 4 and $O((kn\log^2(n) + nr^2) + a_{rn}$ overall.

Remark 7. (Counting Real Eigenvalues.) If the number of real eigenvalues is not given, we can apply binary search to compute it as the numerical rank of the matrices $Y_k^2 + I$ when this rank stabilizes.

Remark 8. (Avoiding Numerical Problems.) The images of nonreal eigenvalues of the matrix C_p converge to $\pm\sqrt{-1}$ in the iteration of Stage 1, but the iteration would involve ill conditioned matrices if the images of some real eigenvalues of C_p lie close to 0. We can detect that the matrix Y_h is ill conditioned by computing its smallest singular value (e.g., by applying the Lanczos algorithm [9, Proposition 9.1.4]) or by encountering difficulty in its numerical inversion. In such cases we can shift the matrix (and hence shift its eigenvalues) by adding the matrix sI for a reasonably small real scalar s, which we can select heuristically or by applying Theorem 7 or randomization. For a more radical recipe, apply Stage 1 of the algorithm to the two matrices $\alpha\sqrt{-1}\, I + C_p$ and $\alpha\sqrt{-1}\, I - C_p$ and then use the sum of the two resulting matrix functions as the matrix Y at Stages 2. Here α is a small positive scalar such that no eigenvalues of the matrix C_p have imaginary parts close to $\pm\alpha\sqrt{-1}$.

Remark 9. (Acceleration by Using Random Circulant Multiplier.) We can decrease the cost of performing Stage 3 to $a_{n+r} + O(n\log(n))$ and the overall arithmetic cost to $O(kn\log^2(n) + nr^2) + a_{r+n}$ if we replace an $n \times r$ standard Gaussian random multiplier by the product ΩCP where Ω and C are $n \times n$ matrices, Ω is the matrix of the discrete Fourier transform, C is a random circulant

matrix, and P is an $n \times l$ random permutation matrix, for a sufficiently large l of order $r \log(r)$. See [12, Section 11], [19, Section 6] for the analysis and for the supporting probability estimates. They are only slightly less favorable than in the case of a Gaussian random multiplier.

Remark 10. (Acceleration by Means of Scaling.) We can dramatically accelerate the initial convergence of Algorithm 2 by applying *determinantal scaling* (cf. [11]), that is, by computing the matrix Y_1 as follows, $Y_1 = 0.5(\nu Y_0 - (\nu Y_0)^{-1})$ for $\nu = 1/|\det(Y_0)|^{1/n} = |p_n/p_0|$, $Y_0 = C_p$.

Remark 11. (Hybrid Matrix and Polynomial Algorithms.) Can we modify Algorithm 2 to keep its advantages but to decrease the arithmetic cost of its Stage 1 to the level $kn \log(n)$ of Algorithm 1? Yes, if all or almost all nonreal roots of the polynomial $p(x)$ lie not too far from the points $\pm\sqrt{-1}$, namely in the discs $D(\pm\sqrt{-1}, 1/2)$. Indeed in this case both iterations $Y_{h+1} = 0.5(Y_h^3 + 3Y_h)$ and $Y_{h+1} = -0.125(3Y_h^5 + 10Y_h^3 + 15Y_h)$ for $h = 0, 1, \dots$ involve no inversions and use $O(n \log(n))$ ops per loop. Right from the start, they send the nonreal roots lying in these discs toward the two points $\pm\sqrt{-1}$ with quadratic and cubic convergence rates, respectively. (To prove this, extend the proof of [3, Proposition 4.1].) Both iterations keep the real roots real. This suggests the following policy. Perform the iterations of Algorithm 1 as long as the outputs are not corrupted by rounding errors. (Choose the number of iterations of Algorithm 1 heuristically and shift the iterates as in Remark 8 to counter numerical problems.) For a large class of inputs, the iterations (even under the above limitation on their number) should still bring the images of the nonreal eigenvalues of C_p into the basin of convergence of the inversion-free matrix iterations above. Then apply one of these inversion-free iterations to the companion matrix C_{q_h} of the polynomial $q_h(x)$ output by Algorithm 1, to approximate the real roots of this polynomial. Descend from them to the real roots of the polynomial $p(x)$ as in Algorithms 1. The hybrid algorithm combines the power of Algorithms 1 and 2.

4.3 Square Root Iteration (a Modified Modular Version)

Our next algorithm is another dual polynomial version of Algorithm 2. It extend the square root iteration $y_{h+1} = \frac{1}{2}(y_h + 1/y_h)$, $h = 0, 1, \dots$. Compared to Algorithm 2, we first replace all rational functions in the matrix C_p by the same rational functions in the variable x and then reduce every function modulo the input polynomial $p(x)$. The reduction does not affect the values of the functions at the roots of $p(x)$, and so these values are precisely the eigenvalues of the rational matrix functions involved in Algorithm 2.

Algorithm 3. Square root modular iteration modified for real root-finding.
INPUT AND OUTPUT *as in Algorithm 1.*
COMPUTATIONS:

1. *Write $y_0 = x$ and (cf. (4)) compute the polynomials*

$$y_{h+1} = \frac{1}{2}(y_h - 1/y_h) \mod p(x), \ h = 0, 1, \dots. \tag{5}$$

2. *Periodically, for selected integers k, compute the polynomials $t_k = y_k^2 + 1 \mod p(x)$ and $g_k(x) = \mathrm{agcd}(p, t_k)$.*
3. *If $\deg(g_k(x)) = n - r = 2s$, 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 [6], [4], and [8] (cf. Theorem 8) to approximate the r roots $y_1, \dots, y_r$ of the polynomial v_k. Output these approximations.*

By virtue of our comments preceding this algorithm, the values of the polynomials t_k at the roots of $p(x)$ equal to the images of the eigenvalues of the matrix C_p in Algorithm 2. Hence the values of the polynomials t_k 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, $\mathrm{agcd}(p, t_k)$ turn into the polynomial $\prod_{j=r+1}^{n}(x - x_j)$. This implies correctness of the algorithm. Its asymptotic computational cost is $O(kn\log^2(n))$ plus the cost of computing $\mathrm{agcd}(p, t_k)$ and choosing the integer k (see our next remark).

Remark 12. Compared to Algorithm 2, the latter algorithm reduces real root-finding essentially to the computation of $\mathrm{agcd}(p, t_k)$, but the complexity of this computation is not easy to estimate [1]. Moreover, the following example exhibits serious problems of numerical stability for this algorithm and for the similar algorithms of [7] and [3]. Consider the case where $r = 0$. Then the polynomial $t(x)$ has degree at most $n - 1$, and its values at the n nonreal roots of the polynomial $p(x)$ are close to 0. This can only occur if $||t_k|| \approx 0$.

Remark 13. We can concurrently perform Stages 1 of both Algorithms 2 and 3. The information about the numerical rank at Stage 3 of Algorithm 2 can be a guiding rule for the choice of the integer parameter k and computing the polynomials t_k, g_k and v_k of Algorithm 3. Having the polynomial v_k available, Algorithm 3 produces the approximations to the real roots more readily than Algorithm 2 does this at its Stage 4.

4.4 Cayley Map and Root-squaring

The following algorithm is somewhat similar to Algorithm 1, but employs repeated squaring of the roots instead of mapping them into their square roots.

Algorithm 4. Real root-finding by means of repeated squaring.

Assume a polynomial $p(x)$ of (1) with $p(0) \neq \pm\sqrt{-1}$ and proceed as follows.

1. Compute the polynomial $q(x) = p(\sqrt{-1}\,\frac{x+1}{x-1}) = \sum_{i=0}^{n} q_i x^i$. (This is the Cayley map of Theorem 4. It moves the real axis, in particular the real roots of $p(x)$, onto the unit circle $C(0,1)$.)

2. Write $q_0(x) = q(x)/q_n$, choose a sufficiently large integer k, and apply the k squaring steps of Theorem 3, $q_{h+1}(x) = (-1)^n q_h(\sqrt{x})q_h(-\sqrt{x})$ for $h = 0, 1, \dots, k-1$. (These steps keep the images of the real roots of $p(x)$ on the circle $C(0,1)$ for all k, while sending the images of every other root of $p(x)$ toward either the origin or the infinity.)

3. For a sufficiently large integer k, the polynomial $q_k(x)$ approximates the polynomial $x^s u_k(x)$ where the polynomial $u_k(x) = \sum_{i=0}^{r} u_i x^i$ has all its roots lying on the unit circle $C(0,1)$. Extract the approximation to this polynomial $u_k(x)$ from the coefficients of the polynomial $q_k(x)$.

4. Compute the polynomial $w_k(x) = (u_k(\sqrt{-1}\,\frac{x+1}{x-1})$. (This Cayley map sends the images of the real roots of the polynomial $p(x)$ from the unit circle $C(0,1)$ back to the real line.)

6. Apply one of the algorithms of [6], [4], and [8] to approximate the r real roots $z_1, \dots, z_r$ of the polynomial $w_k(x)$ (cf. Theorem 8).

7. Apply the Cayley map $x_j^{(k)} = (z_j + \sqrt{-1})/(z_j - \sqrt{-1})$ for $j = 1, \dots, r$ to extend Stage 6 to approximating the r roots $x_1^{(k)}, \dots, x_r^{(k)}$ of the polynomials $u_k(x)$ and $y_k(x) = x^s u_k(x)$ lying on the unit circle $C(0,1)$.

8. Apply the descending process (similar to the ones of [15], [18], and of our Algorithm 1) to approximate the r roots $x_1^{(h)}, \dots, x_r^{(h)}$ of the polynomials $q_h(x)$ lying on the unit circle $C(0,1)$ for $h = k-1, \dots, 0$.

9. Approximate the r real roots $x_j = \sqrt{-1}(x_j^{(0)} + 1)/(x_j^{(0)} - 1)$, $j = 1, \dots, r$, of the polynomials $p(x)$.

Our analysis of Algorithm 1 (including its complexity estimates and the comments and recipes in Remarks 2–6) can be extended to Algorithm 4. The straightforward matrix version of this numerical algorithm, however, fails because high matrix powers have small numerical rank. Indeed their columns lie near the invariant space associated with the absolutely largest eigenvalues, and as a rule this space has a small dimension. The more tricky modification $M^k - M^{-k} = \prod_{j=0}^{k-1}(M - \omega_k^j M^{-1})$, where $\omega_k = \exp(2\pi\sqrt{-1}/k)$ denotes a primitive kth root of unity, promises to produce a working matrix iteration. Like the Power Method and unlike the repeated squaring of Algorithm 2, it can employ just multiplication of the matrices M and M^{-1} by vectors rather than the operations in the Frobenius matrix algebra.

4.5 A Tentative Approach to Real Root-finding by Means of Root-radii Approximation

Algorithm 5. *(*Real Root-finding via Root Radii Approximation.*)*

1. Compute approximations $\tilde{r}_1, \dots, \tilde{r}_n$ to the root radii of a polynomial $p(x)$ of (1) (see Theorem 7). (This defines $2n$ candidates points $\pm\tilde{r}_1, \dots, \pm\tilde{r}_n$ for the approximation of the r real roots $x_1, \dots, x_r$.)

2. Evaluate the polynomial at these $2n$ points, at a low arithmetic and Boolean cost, to exclude a number of extraneous candidates.

3. Apply Newton's iteration $x^{(h+1)} = x^{(h)} - p(x^{(h)})/p'(x^{(h)})$, $h = 0, 1, \dots$ concurrently at the remaining candidate points. (Its single step or a few step should exclude the other extraneous candidates and refine the remaining approximations to the real roots, as long as these roots are simple and well isolated from the other roots.)

5 Numerical Tests

Three series of numerical tests have been performed in the Graduate Center of the City City University of New York by Ivan Retamoso and Liang Zhao. In both series they tested Algorithm 2, without using the techniques of Remarks 3 and 8, that is, in much weakened form. Still the test results are quite encouraging.

In the first series of tests, Algorithm 2 has been applied to one of the Mignotte benchmark polynomials, namely to $p(x) = x^n + (100x - 1)^3$. 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 2 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 tests, polynomials $p(x)$ of degree $n = 50, 100, 150, 200,$ and 250 have been generated as the products $p(x) = f_1(x)f_2(x)$. Here $f_1(x)$ was the rth degree Chebyshev polynomial (having r real roots) for $r = 8, 12, 16,$ and $f_2(x) = \sum_{i=0}^{n-r} a_i x^i$, a_j being i.i.d. standard Gaussian random variables, for $j = 0, \dots, n - r$. Algorithm 2 (performed with double precision) was applied to 100 such polynomials $p(x)$ for each pair of n and r. Table 1 displays the output data, namely, the average values and 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 tests, Algorithm 2 approximated the real eigenvalues of a random real symmetric matrix $A = U^T \Sigma U$, where U was an orthogonal $n \times n$ standard Gaussian random matrix, $\Sigma = \mathrm{diag}(x_1, \dots, x_r, y_1, \dots, y_{n-r})$, and $x_1, \dots, x_r$ (resp. $y_1, \dots, y_{n-r}$) were r i.i.d. standard Gaussian real (resp. non-real) random variables. Table 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 $r = 8, 12, 16$.

References

1. 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)
2. Bini, D., Pan, V. Y.: Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms. Birkhäuser, Boston (1994)
3. Bini, D., Pan, V.Y.: Graeffe's, Chebyshev, and Cardinal's processes for splitting a polynomial into factors. J. Complexity 12, 492–511 (1996)
4. 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) (Also in Proc. of SODA'1991.)
5. Bini, D.A., Robol, L.: Solving secular and polynomial equations: A multiprecision algorithm. J. Computational and Applied Mathematics, in press.
6. Ben-Or, M., Tiwari, P.: Simple algorithms for approximating all roots of a polynomial with real roots. J. Complexity 6(4), 417–442 (1990)

Table 1. Number of Iterations and Error Bounds for Root-finding Algorithm on Random Polynomials

n	r	Iter-mean	Iter-std	Bound-mean	Bound-std
50	8	7.44	1.12	4.18×10^{-6}	1.11×10^{-5}
100	8	8.76	1.30	5.90×10^{-6}	1.47×10^{-5}
150	8	9.12	0.88	2.61×10^{-5}	1.03×10^{-4}
200	8	9.64	0.86	1.48×10^{-6}	5.93×10^{-6}
250	8	9.96	0.73	1.09×10^{-7}	5.23×10^{-5}
50	12	7.16	0.85	3.45×10^{-4}	9.20×10^{-4}
100	12	8.64	1.15	1.34×10^{-5}	2.67×10^{-5}
150	12	9.12	2.39	3.38×10^{-4}	1.08×10^{-3}
200	12	9.76	2.52	6.89×10^{-6}	1.75×10^{-5}
250	12	10.04	1.17	1.89×10^{-5}	4.04×10^{-5}
50	16	7.28	5.06	3.67×10^{-3}	7.62×10^{-3}
100	16	10.20	5.82	1.44×10^{-3}	4.51×10^{-3}
150	16	15.24	6.33	1.25×10^{-3}	4.90×10^{-3}
200	16	13.36	5.38	1.07×10^{-3}	4.72×10^{-3}
250	16	13.46	6.23	1.16×10^{-4}	2.45×10^{-4}

7. 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)
8. Du, Q., Jin, M., Li, T.Y., Zeng, Z.: The quasi-Laguerre iteration. Math. Comput. 66(217), 345–361 (1997)
9. Golub, G.H., Van Loan, C.F.: Matrix Computations, third edition. The Johns Hopkins University Press, Baltimore, Maryland (1996)
10. Householder, A.S.: Dandelin, Lobachevskii, or Graeffe. Amer. Math. Monthly 66, 464–466 (1959)
11. Higham, N.J.: Functions of Matrices, SIAM, Philadelphia (2008)
12. 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)
13. McNamee, J.M., Pan, V.Y.: Numerical Methods for Roots of Polynomials, Part 2 (XXII + 718 pages), Elsevier (2013)
14. Pan, V.Y.: Complexity of computations with matrices and polynomials. SIAM Review 34(2), 225–262 (1992)
15. 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)
16. Pan, V.Y.: New fast algorithms for polynomial interpolation and evaluation on the Chebyshev node set. Computers Math. Appls. 35(3), 125–129 (1998)
17. Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, Boston, and Springer-Verlag, New York (2001)
18. 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)

Table 2. Number of Iterations and Error Bounds for Root-finding Algorithm on Random Matrices

n	r	Iter-mean	Iter-std	Bound-mean	Bound-std
50	8	10.02	1.83	5.51×10^{-11}	1.65×10^{-10}
100	8	10.81	2.04	1.71×10^{-12}	5.24×10^{-12}
150	8	14.02	2.45	1.31×10^{-13}	3.96×10^{-13}
200	8	12.07	0.94	2.12×10^{-11}	6.70×10^{-11}
250	8	13.59	1.27	2.75×10^{-10}	8.14×10^{-10}
50	12	10.46	1.26	1.02×10^{-12}	2.61×10^{-12}
100	12	10.60	1.51	1.79×10^{-10}	3.66×10^{-10}
150	12	11.25	1.32	5.69×10^{-8}	1.80×10^{-7}
200	12	12.36	1.89	7.91×10^{-10}	2.50×10^{-9}
250	12	11.72	1.49	2.53×10^{-12}	3.84×10^{-12}
50	16	10.10	1.45	1.86×10^{-9}	5.77×10^{-9}
100	16	11.39	1.70	1.37×10^{-10}	2.39×10^{-10}
150	16	11.62	1.78	1.49×10^{-11}	4.580×10^{-11}
200	16	11.88	1.32	1.04×10^{-12}	2.09×10^{-12}
250	16	12.54	1.51	3.41×10^{-11}	1.08×10^{-10}

19. Pan, V.Y., Qian, G., Yan, X.: Supporting GENP and Low-rank Approximation with Random Multipliers. Technical Report TR 2014008, PhD Program in Computer Science. Graduate Center, CUNY (2014). Available at http://www.cs.gc.cuny.edu/tr/techreport.php?id=472
20. Pan, V.Y., Qian, G., Zheng, A.: Real and complex polynomial root-finding via eigen-solving and randomization. In: Gerdt, V.P. et al. (eds.) CASC 2012. LNCS, vol. 7442, pp. 283–293. Springer, Heidelberg (2012)
21. Pan, V.Y., Tsigaridas, E.P.: On the Boolean Complexity of the Real Root Refinement. Tech. Report, INRIA (2013). url: http://hal.inria.fr/hal-00960896, Proc. version in: M. Kauers (ed.) Proc. Intern. Symposium on Symbolic and Algebraic Computation (ISSAC 2013), pp. 299–306, Boston, MA, June 2013. ACM Press, New York (2013)
22. Pan, V.Y., Tsigaridas, E.P.: Nearly optimal computations with structured matrices. In: SNC 2014. ACM Press, New York (2014). Also April 18, 2014, arXiv:1404.4768 [math.NA] and http://hal.inria.fr/hal-00980591
23. Pan, V.Y., Zheng, A.: New progress in real and complex Ppolynomial root-finding. Computers Math. Applics. 61(5), 1305–1334 (2011)
24. Schönhage, A.: The Fundamental Theorem of Algebra in Terms of Computational Complexity. Math. Department, Univ. Tübingen, Germany (1982)
25. Sagraloff, M., Mehlhorn, K.: Computing Real Roots of Real Polynomials, CoRR, abstract 1308.4088 (2013)
26. Watkins, D.S.: The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods. SIAM, Philadelphia, PA (2007)