Real Polynomial Root-Finding by Means of Matrix and Polynomial Iterations

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
http://comet.lehman.cuny.edu/vpan/

Abstract. Frequently one seeks approximation to all r real roots of a polynomial of degree n with real coefficients, which also has nonreal roots. We split a polynomial into two factors, one of which has degree r and has r real roots. We approximate them at a low cost, and then decrease the arithmetic time of the known algorithms for this popular problem by roughly a factor of n/k, if k iterations prepare splitting. k is a small integer unless some nonreal roots lie close to the real axis, but even if there nonreal roots near the real axis, we substantially accelerate the known algorithms. We also propose a dual algorithm, operating with the associated structured matrices. At the price of minor increase of the arithmetic time, it facilitates numerical implementation. Our analysis and tests demonstrate the efficiency of our approach.

Keywords: polynomials, real roots, matrices, matrix sign iteration, companion matrix, real eigenvalues, Frobenius algebra, square root iteration, root squaring.

1 Introduction

In some applications, e.g., to algebraic and geometric optimization, one seeks real roots of a univariate polynomial

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

of degree n that has real coefficients, r real roots $x_1, \dots, x_r$, and $s = (n-r)/2$ pairs of nonreal complex conjugate roots $x_{r+1}, \dots, x_n$ (typically $r \ll n$). This is a well studied subject (see [13, Chapter 15], [21], [25], and the bibliography therein), but we propose new efficient algorithms by extending and combining the techniques of [23] and [20]. We combine the two known low cost steps, recalled in Section 2, that is, splitting a polynomial into two factors whose two sets of roots are isolated from one another, and real root-finding when all n roots of the input polynomial are real. Namely, our iterative processes split out the factor

V.P. Gerdt et al. (Eds.): CASC Workshop 2014, LNCS 8660, pp. 335–349, 2014.

$s(x) = \prod_{j=1}^{r}(x-x_j)$ of degree r that shares with the input polynomial $p(x)$ all its real roots, and as soon as this factor has been computed, we readily approximate its r roots at a low computational cost. As a result, we yield the solution at the arithmetic cost $O(kn\log(n))$, provided that k iterations prepare splitting of the factor. Our iterative algorithms converge exponentially fast (with quadratic or cubic rates), and so $k = O(b+d)$, assuming the tolerance 2^{-b} to the errors of the output approximation and the minimal distance 2^{-d} of the nonreal roots from the real axis. Usually this bound on k is not large, except for the inputs having nonreal roots that lie very close to the real axis. In Remark 4, we discuss some techniques for handling even such harder inputs. According to our preliminary considerations (cf., e.g., Remark 8) and the test results, our algorithms can be implemented with a reasonably bounded precision of computing, but we leave the formal study of this subject and of the Boolean complexity of our algorithms as a challenge for further research.

We devise dual iterations with polynomials generated from the input polynomial $p(x)$ of (1) and with matrices generated from the companion matrix of this polynomial. In the latter case, we seek real eigenvalues of this matrix, extend the matrix sign classical iteration toward this goal, and employ the known results and techniques in this well developed area. Dealing with matrices one can engage efficient packages of subroutines available for numerical matrix computations with the IEEE standard double precision. The highly structured companion matrix generates the Frobenius matrix algebra, in which one can perform FFT-based computations in nearly linear time, that is, as fast as the similar operations with polynomials. In some cases, we take advantage of combining the power of operating with matrices and polynomials (see Remark 13). Finding their deeper synergistic combinations is another natural research challenge.

We present a number of promising algorithms. Algorithms 2 and 5 have the lowest estimated arithmetic cost $O(kn\log(n))$, which increases to $O(kn\log^2(n))+c(n,r)$ for Algorithms 3 and 4. Here $c(n,r)$ is the overhead due to randomization for Algorithm 3 and to computing approximate polynomial GCDs for Algorithm 4. We include these algorithms since they use some promising techniques and since Algorithm 3 showed superior numerical stability in our tests.

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. E.g., our Algorithm 2 streamlines and substantially modifies [23, Algorithm 9.1] by avoiding the stage of root-squaring and the application of the Cayley map, and similar comments apply to our adjustment of the matrix sign classical iteration to real eigen-solving. Most of the techniques of Algorithm 3 are implicit in [20, Section 5], but we specify the algorithm in some detail, include initial scaling, substantially modify the recovery of the eigenvalues, and combine it with Algorithm 2. Algorithms 4 and 5 are new, in spite of some links to Algorithms 2 and 3 and hence to [20, Section 5] and [23, Section 9]. Our interplay with matrix and polynomial computations to the benefit of both subjects (this idea can be traced back to [14] and [2]) as well as our exploitation of the complex

plane geometry and of various transforms of the variable can be of independent interest. Our simple recipe for real root-finding by means of combining the root radii algorithm with Newton's iteration in Algorithm 1 works for a large class of inputs, and even the extension of our approach to the approximation of real eigenvalues of a real matrix can be of some potential 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. agcd(u, v) denotes the *approximate greatest common divisor* of two polynomials $u(x)$ and $v(x)$ (see [1] on definitions and algorithms).

2 Basic Results for Polynomials

Next we present some building blocks for our root-finders. Besides the two cited results, used as the main blocks of our algorithms (that is, inexpensive splitting of a polynomial into two factors and fast real root-finding for a polynomial that has only real roots) we recall scaling, shifting, inverting and squaring the roots, their mapping from the real axis or from a real line interval into a fixed circle and back, and the approximation of the absolute values of all roots $|x_1|, \dots, |x_n|$. All these operations can be performed at a low arithmetic cost as well.

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

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, because $p_{\rm rev}(x) = x^n p(1/x) = \sum_{i=0}^n p_i x^{n-i} = p_n \prod_{j=1}^n (1 - x x_j)$.*

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

Theorem 3. *(*Root Squaring*, cf. [10].) (i) Assume a monic polynomial $p(x)$ of (1), $p_n = 1$. 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 $\{x : \ x$ is real$\}$ into the unit circle $C(0,1) = \{x : \ |x| = 1\}$, and vice versa.*

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

Proof. Follow [3, Section 2]. Apply the algorithms of [16] to interpolate to the polynomial $q(y)$ from its values at the Chebyshev knots at the cost $O(n\log(n))$.

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

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

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

If $x^{(0)}\gamma$ *is real, then* $x^{(h)}\gamma$ *are real for all* h*. Otherwise* $|x^{(h)} - \mathrm{sign}(x)\sqrt{-1}/\gamma| \le \frac{2\tau^{2^h}}{1-\tau^{2^h}}$ *for* $\tau = |\frac{x-\mathrm{sign}(x)}{x+\mathrm{sign}(x)}|$ *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-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 with superlinear convergence rate and uses* $O(n)$ *ops per iteration. Consequently the algorithm approximates all* n *roots within* $\epsilon = 1/2^b$ *by using* $O(\log(b))$ *iteration loops, performing* $\tilde{O}(n\log(b))$ *ops overall. This cost bound is optimal and is also supported by the alternative algorithms of [6] and [4].*

Algorithm 1. *(*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 1). (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 concurrent step or a few steps, performed at a low arithmetic and Boolean cost (cf. [22]), should exclude the other extraneous candidates and refine the remaining approximations to the real roots, as long as these roots are well isolated from the nonreal roots.)*

Theorem 8. *(*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 in the disc* $D(0,\rho)$ *and* $n-r$ *roots outside the disc* $D(0,R)$ *for* $R/\rho \ge 1 + 1/n$*. Let* $\epsilon = 1/2^b$ *for* $b \ge n$*. Then 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 the circle. The algorithm performs* $O((\log^2(n)+\log(b))n\log(n))$ *ops (that is,* $O(n\log^3(n))$ *ops for* $\log(b) = O(\log^2(n))$*), with a precision of* $O(b)$ *bits.*

Remark 1. (*Increasing Isolation by Means of Repeated Squaring.*) Let the assumptions of Theorem 8 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. 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 8.

3 Root-Finding as Eigen-Solving and Basic Results for Matrix Computations

3.1 Companion Matrix, Its Maps, and Maps of Its Eigenvalues

$$C_p = \begin{pmatrix} 0 & & & -p_0/p_n \\ 1 & \ddots & & -p_1/p_n \\ & \ddots & \ddots & \vdots \\ & & \ddots\ 0 & -p_{n-2}/p_n \\ & & 1 & -p_{n-1}/p_n \end{pmatrix}$$

denotes the *companion matrix* of a polynomial $p(x)$ of (1). $p(x) = c_{C_p}(x) = \det(xI_n - C_p)$ is its *characteristic polynomial*. Its roots form the *spectrum* of C_p, and so our problem can be restated as the problem of real eigen-solving for the companion matrix C_p. Next we recall that operations with this matrix are as inexpensive as with polynomials and restate the maps for the variable x of the polynomials in terms of maps of the matrix C_p, playing the role of this variable.

Theorem 9. *(*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.*

3.2 Some Fundamental Matrix Computations

To study the eigen-solving for C_p, next we recall some fundamentals of matrix computations. In the next subsection we focus on the basic properties of eigenvalues and eigenspaces of matrices, that we use in our algorithms.

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

A matrix Q is *unitary* if $Q^HQ = I$ or $QQ^H = I$. Let $(Q, R) = (Q(M), R(M))$ for an $m \times n$ matrix M of rank n denote 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 an orthogonal matrix M. $M^{(I)} = M^{-1}$ 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.

3.3 Eigenspaces and Eigenvalues

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 $d = 1$. The set $\Lambda(M)$ of all eigenvalues of the matrix M is called its* spectrum.

Our next goal is to limit eigen-solving for the matrix C_p to the study of its invariant space of dimension r associated with the r real eigenvalues. The following theorem is basic for this step.

Theorem 10. *(*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 an orthogonal matrix, then $L = U^HMU$.*

To facilitate the computation of the desired invariant space of C_p, we reduce the task to the case of an appropriate matrix function, for which the solution is simpler, but we still solve our problem, because, by virtue of the following theorem, a matrix function shares its invariant spaces with the matrix C_p.

Theorem 11. *(*Reduction of the Eigenproblem for a Matrix to That for a Matrix 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 $f(M)$.*

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 11 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 10). Empirically the QR algorithm uses $O(r^3)$ ops at the latter stage.

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 Theorem 11 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

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 $\Lambda(f(C_p)) = \{f(x_i)\}_{i=1}^n$, by virtue of Theorem 11. 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 the 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 $y = y(x)$ denote 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. (3)) 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. Here $y_h = y_h(x)$ are the characteristic polynomials of the matrices $M_0 = C_p$, $M_{h+1} = 0.5(M_h \pm M_h^{-1})$ and $M_0 = C_p$, $M_{h+1} = aM_h + bM_h^{-1}$, $h = 0, 1, \ldots$, respectively.

4 Real Root-Finders

4.1 Möbius Iteration

Theorem 6 implies that right from the start of iteration (3) the values $x^{(h)}$ converge to $\pm\sqrt{-1}$ exponentially fast 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 to 0 exponentially fast unless $x^{(0)}$ is real, in which

case all values $y^{(h)}$ are real and exceed 1. Write $q_h(y) = \prod_{j=1}^{n}(y - (x_j^{(h)})^2 - 1)$ for $h = 1, 2, \ldots$ and $u_h(y) = \prod_{j=1}^{r}(y - (x_j^{(h)})^2 - 1)$. The roots of the polynomials $q_h(y)$ and $u_h(y)$ 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 $y^{(h)} = (x^{(h)})^2 + 1$. Therefore $q_h(y) \approx y^{2s}u_h(y)$ for large integers h where the polynomial $u_h(y)$ has degree r and has exactly r real roots, all of them exceeding 1. Hence for sufficiently large integers h, we can closely approximate the polynomial $y^{2s}u_h(y)$ simply by the sum of the $r + 1$ leading terms of the polynomial $q_h(y)$. To verify that the $2s$ trailing coefficients nearly vanish, we need just $2s$ comparisons. The above argument shows correctness of the following algorithm.

Algorithm 2. 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 $p(x)$.*
INITIALIZATION: *Write $p_0(x) = p(-x\sqrt{-1})$.*
COMPUTATIONS:

1. *Recursively compute the polynomials $p_{h+1}(y) = p_h(x)p_h(1/x)$ for $y = (x + 1/x)/2$ and $h = 0, 1, \ldots$ (Part (ii) of Theorem 5 and Theorem 6 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 $q_k(z) = p_k(z)/\mathrm{lc}(p_k)$ (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, approximate the factor $v_k(x)$ of the polynomial $t_k(x)$ that has r real roots on the ray $\{x : x \leq -1\}$ (see above).
3. *Apply one of the algorithms of [6], [4], and [8] (cf. Theorem 7) to approximate the r roots $z_1, \ldots, z_r$ of the polynomial $v_k(x)$.*
4. *Extend the descending process from [15], [18] and [3] to recover approximations to the r roots $-x_i\sqrt{-1}$, $i = 1, \ldots, r$, of the polynomial $p_0(x) = p(-x\sqrt{-1})$. First approximate $2r$ candidates for r roots of the polynomial $q_k(y)$ lying on the imaginary axis and select r of them on which the polynomial $q_k(y)$ nearly vanishes. Similarly define from these r roots $2r$ candidates for approximating the r roots of $p_{k-1}(x)$ lying on the imaginary axis. Recursively descend down to the r roots of $p_0(x)$ lying on the imaginary axis. This process is not ambiguous because only r roots of the polynomial $p_h(x)$ lie on that axis for each h, by virtue of Theorem 6.*
5. *Having approximated the r roots $-x_i\sqrt{-1}$, $i = 1, \ldots, r$, output the approximations to the real roots $x_1, \ldots, x_r$ of the polynomial $p(x)$.*

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, this holds for Gaussian random variable s, but alternatively we can approximate the root radii of the polynomial $p(x)$ (cf. Theorem 1) to find a shift scalar s such that $q(x)$ has no roots near 0 as well.

Remark 3. (Saving the Recursive Steps of Stage 1.) We would decrease the parameter k of the cost estimate, if we approximate the factor $v_k(x)$ of the polynomial $t_k(x)$ for a smaller integer k. Theorem 8 enables us to do this (at a reasonable cost) if its assumptions are satisfied for $t(x) = t_k(x)$. We can verify if the assumptions hold by applying the root radii algorithm of Theorem 1. 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 integer k is large, but we can periodically test just selected integers k, by applying binary search.

Remark 4. (Handling the Nearly Real Roots.) The integer parameter k and the overall cost of performing the algorithm are large if $2^{-d} = \min_{j=r+1}^{n} |\Im(x_j)|$ is small. To counter this deficiency, we can split out a factor $v_{k,+}(x)$ of the polynomial $p(x)$ having a degree $r_+ > r$ and having r_+ real and nearly real roots such that the other nonreal roots lie sufficiently far from the real axis. Indeed our convergence analysis and the techniques for splitting out the factor $v_k(x)$ can be readily extended to splitting out the factor $v_{k,+}(x)$. Having this factor approximated, we can tentatively apply to it the modified Laguerre algorithm of [8], expecting fast convergence to the r_+ roots of the polynomial $v_{k,+}(x)$ if all its roots lie on or sufficiently close to the real axis.

Remark 5. (The Number of Real Roots.) We assume that we are given the number r of the real roots (e.g., computed by means of non-costly techniques of computer algebra if the roots are distinct and simple), but we can compute this number as by-product of Stage 2, and similarly for our other algorithms. Moreover with a proper try-and-test policy we can apply our algorithm for at most $2 + 2\lceil \log(r) \rceil$ tentative choices of integers k in the range $[0, 2k - 1]$ to detect r.

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 2 is prone to numerical stability problems already for moderately large integers k.

4.2 Adjusted Matrix Sign Iteration

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

$$Z_h = 0.5(Z_h + Z_h^{-1}) \text{ for } h = 0, 1, \ldots \tag{4}$$

It maps the eigenvalues of the matrix Z_0 according to (2). Therefore, by virtue of part (ii) of Theorem 5, Stage 1 of Algorithm 2 maps the characteristic polynomials of the above matrices Z_h. Unlike the case of the latter map, working

with matrices enables two minor implications: (i) we recover the desired real eigenvalues of the matrix C_p by means of our recipes of Section 3, without recursive descending, and (ii) we avoid scaling by $\sqrt{-1}$ and just slightly modify the iteration to keep the computations in the field of real numbers.

Algorithm 3. Matrix sign iterations modified for real eigen-solving.
INPUT AND OUTPUT *as in Algorithm 2, except that FAILURE can be output with a probability close to 0.*
COMPUTATIONS:

1. *Write* $Y_0 = C_p$ *and recursively compute the matrices*

$$Y_{h+1} = 0.5(Y_h - Y_h^{-1}) \text{ for } h = 0, 1, \dots. \tag{5}$$

   (For sufficiently large integers h*, the* $2s$ *eigenvalues of the matrix* Y_h *lie near the points* $\pm\sqrt{-1}$*, whereas the* r *other eigenvalues are real, 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 *to 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*. (Otherwise 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 an orthogonal 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 11,* Q *is a matrix basis for the invariant space of the matrix* C_p *associated with its* r *real eigenvalues. Therefore, by virtue of Theorem 10, the matrices* C_p *and* L *share these eigenvalues.)*

Stages 1 and 2 involve $O(kn\log^2(n))$ ops by virtue of Theorem 9. This exceeds the estimate for Algorithm 2 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 $O(nr^2)$ 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 recursive process of the algorithm. So the process involves ill conditioned matrices if and only if the images of some real eigenvalues of C_p lie close to 0. We can detect that this has occurred if it is hard to invert the matrix Y_h of (5) or by computing the smallest singular value of that matrix (e.g., by applying the Lanczos efficient, cf. [9, Proposition 9.1.4]). As soon as we detect an ill conditioned matrix Y_h, we would shift it (and hence shift its eigenvalues) by adding the matrix sI for a reasonably small real scalar s, which we can select by applying Theorem 1, heuristic, or randomization.

Remark 9. (Acceleration by Using Random Circulant Multiplier.) We can decrease the cost of performing Stage 3 to $a_{n+r} + O(n \log(n))$ by replacing an $n \times r$ standard Gaussian random multiplier by the product $\Omega C P$ 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 supporting probability and cost estimates. They are only slightly less favorable than in the case of a Gaussian random multiplier.) The overall arithmetic cost bound would change into $O(kn \log^2(n) + nr^2) + a_{r+n}$.

Remark 10. (Acceleration by Means of Scaling.) We can dramatically accelerate the initial convergence of Algorithm 3 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 $\nu = 1/|\det(Y_0)|^{1/n} = |p_n^{(k)}/p_0^{(k)}|$, $Y_0 = C_{p^{(k)}}$, and $p^{(k)}(x) = \sum_{i=0}^{n} p_i^{(k)} x^i$.

Remark 11. (Hybrid Matrix and Polynomial Algorithms.) Can we modify Algorithm 3 to keep its advantages but to decrease the computational cost of its Stage 1 to the level $kn \log(n)$ of Algorithm 2? 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 $h = 0, 1, \ldots$ use $O(n \log(n))$ ops per loop. Right from the start they send the nonreal roots lying in these discs to the two points $\pm\sqrt{-1}$ with quadratic and cubic convergence rates, respectively (extend the proof of [3, Proposition 4.1]), while keeping the real roots real. This suggests the following policy. Heuristically or by applying Theorem 1 choose a proper integer h and run Algorithm 2 until all or almost all nonreal roots of $p(x)$ are moved into the discs $D(\pm\sqrt{-1}, 1/2)$. Then apply one of the two latter inversion-free variants of Algorithm 3 to the polynomial $q_h(x)$ produced by Algorithm 2. Descend from the output roots to the real roots of the polynomial $p(x)$. The hybrid algorithm combines the benefits of both Algorithms 2 and 3 when the above integer h is not large.

4.3 Adjusted Modular Square Root Iteration

The polynomial version of Algorithm 3 is known as the square root iteration. It mimics Algorithm 3, but replaces all rational functions in the matrix C_p by the

same rational functions in the variable x, and then reduces all these functions 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 3.

Algorithm 4. Square root modular iteration modified for real root-finding.
INPUT AND OUTPUT *as in Algorithm 2.*
COMPUTATIONS:

1. *Write* $y_0 = x$ *and* $Y_0 = C_p$ *and (cf. (5)) compute the polynomials*
$$y_{h+1} = (y_h - y_h^{-1})/2 \mod p(x). \tag{6}$$
2. *Periodically, for selected integers* k, *compute the polynomials* $t_k = y_k^2 + 1 \mod p(x)$ *and* $g_k(x) = \text{agcd}(p, t_k)$.
3. *If* $\deg(g_k)) = n - r = 2s$, *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 [6], [4], and [8] (cf. Theorem 7) to approximate the* r *roots* $y_1, \ldots, y_r$ *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 3. Hence the values of the polynomials t_k at the nonreal roots converge to 0 as $k \to \infty$, whereas their values at the real roots stay far from 0. Therefore, for sufficiently large integers k, $\text{agcd}(p, t_k)$ 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 $\text{agcd}(p, t_k)$ and choosing the integer k (see our next remark).

Remark 12. Compared to Algorithm 3, the latter algorithm reduces real root-finding essentially to the computation of $\text{agcd}(p, t_k)$, 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 apparently 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 3 and 4. The information about numerical rank at Stage 3 of Algorithm 3 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 4. Having the polynomial v_k available, Algorithm 4 produces the approximations to the real roots more readily than Algorithm 3 does this at its Stage 4.

5 Cayley Map and Root-Squaring

The following algorithm is somewhat similar to Algorithm 2, but employs repeated squaring of the roots instead of mapping them into their square roots.

Algorithm 5. 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 $q(x) = p((x+\sqrt{-1})(x-\sqrt{-1})^{-1}) = \sum_{i=0}^{n} q_i x^i$. (This is the Cayley map, cf. 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 $h = 1, \dots, k-1$. (These steps keep the images of the real roots of $p(x)$ on the circle $C(0,1)$ for any 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 $v_k(x) = \sqrt{-1}(u_k(x)+1)(u_k(x)-1)^{-1}$. (This is the inverse Cayley map. It 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 $v_k(x)$ (cf. Theorem 7).

7. Apply the Cayley map $w_j = (z_j + \sqrt{-1})(z_j - \sqrt{-1})^{-1}$ for $j = 1, \dots, r$ to extend Stage 6 to approximating the r roots $w_1, \dots, w_r$ 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 2) 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. Apply the inverse Cayley map to approximate the r real roots $x_j = (x_j^{(0)} + \sqrt{-1})(x_j^{(0)} - \sqrt{-1})^{-1}$ of the polynomials $p(x)$.

Our analysis of Algorithm 2 (including its complexity estimates and the comments and recipes in Remarks 2–6) can be extended to Algorithm 5. 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. A more tricky modification, based on binomial factorization, promises to produce a working matrix iteration. We postpone its presentation.

6 Numerical Tests

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

In the first series of tests, Algorithm 3 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 3 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 they randomly generated polynomials $p(x)$ of degree $n = 50, 100, 150, 200, 250$ as the product $p(x) = f_1(x)f_2(x)$. They generated the polynomials $f_1(x)$ and $f_2(x)$ where $f_1(x) = \prod_{j=1}^{r}(x - x_j)$, $f_2(x) = \sum_{i=0}^{n-r} a_i x^i$, and x_i and a_j were i.i.d. standard Gaussian random variables, for $j = 1, \dots, r$, $i = 0, \dots, n - r$, and $r = 4, 8, 12, 16$. Hence the polynomial $p(x) = f_1(x)f_2(x)$ had at least r real roots. Then Algorithm 3 (performed with double precision) was applied to 100 randomly generated polynomials $p(x)$ for each pair of n and r, and the output data were recorded, namely, the numbers of iterations and the maximum difference of the output values of the roots from their values produced by MATLAB root-finding function "roots()". The test results were similar to the case of the Mignotte polynomials (see the Journal version of the paper).

Acknowledgement. I am grateful to NSF, for the support under Grant CCF 1116736, and to the reviewers, for their thoughtful and valuable comments.

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. Fundamental Algorithms, vol. 1. 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)
7. Cardinal, J.P.: On two iterative methods for approximating the roots of a polynomial. Lectures in Applied Mathematics 32, 165–188 (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, 3rd edn. The Johns Hopkins University Press, Baltimore (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. Springer, New York (2001)
18. Pan, V.Y.: Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. J. Symb. Computations 33(5), 253–267 (2002); Proc. version in ISSAC 2001, pp. 253–267, ACM Press, New York (2001)
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), 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., Koepf, W., Mayr, E.W., Vorozhtsov, E.V. (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), 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 (2007)