Finding Roots of Complex Analytic Functions via Generalized Colleague Matrices

Hanwen Zhang and Vladimir Rokhlin

We present a scheme for finding all roots of an analytic function in a square domain in the complex plane. The scheme can be viewed as a generalization of the classical approach to finding roots of a function on the real line, by first approximating it by a polynomial in the Chebyshev basis, followed by diagonalizing the so-called "colleague matrices". Our extension of the classical approach is based on several observations that enable the construction of polynomial bases in compact domains that satisfy three-term recurrences and are reasonably well-conditioned. This class of polynomial bases gives rise to "generalized colleague matrices", whose eigenvalues are roots of functions expressed in these bases. In this paper, we also introduce a special-purpose QR algorithm for finding the eigenvalues of generalized colleague matrices, which is a straightforward extension of the recently introduced componentwise stable QR algorithm for the classical cases (See [6]). The performance of the schemes is illustrated with several numerical examples.

Finding roots of complex analytic functions via generalized colleague matrices

H. Zhang[†◇], V. Rokhlin[† ⊕]
August 24, 2023

† Dept. of Mathematics, Yale University, New Haven, CT 06511

Contents

1 Introduction

In this paper, we introduce a numerical procedure for finding all roots of an analytic function f over a square domain in the complex plane. The method first approximates the function to a pre-determined accuracy by a polynomial

$$p(z) = c_0 + c_1 P_1(z) + c_2 P_2(z) + \cdots + c_n P_n(z) , \tag{1}$$

where $P_0, P_1, \ldots, P_n, \ldots$ is a polynomial basis. For this purpose, we introduce a class of bases that (while not orthonormal in the classical sense) are reasonably well-conditioned, and are defined by three-term recurrences of the form

$$zP_j(z) = \beta_j P_{j-1}(z) + \alpha_{j+1} P_j(z) + \beta_{j+1} P_{j+1}(z), \tag{2}$$

given by complex coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $\beta_1, \beta_2, \ldots, \beta_n$. Subsequently, we calculate the roots of the obtained polynomial by finding the eigenvalues of a "generalized colleague matrix" via a straightforward generalization of the scheme of [6]. The resulting algorithm is a complex orthogonal QR algorithm that requires order $O(n^2)$ operations to determine all n roots, and numerical evidence strongly indicates that it is *componentwise* backward stable; it should be observed that at this time, our stability analysis is incomplete. The square domain makes adaptive rootfinding possible, and is suitable for functions with nearby singularities outside of the square. The performance of the scheme is illustrated via several numerical examples.

The approach of this paper can be seen as a generalization of the classical approach – finding roots of polynomials on the real line by computing eigenvalues of *companion matrices*, constructed from polynomial coefficients in the monomial basis. Since the monomial basis is known to be ill-conditioned on the real line, it is often desirable to use the Chebyshev polynomial basis [8]. By analogy with the companion matrix, roots of polynomials can be obtained as eigenvalues of "colleague matrices", which are formed from the Chebyshev expansion coefficients. Both the companion and colleague matrices have special structures that allow them to be represented by only a small number of vectors, rather than a dense matrix, throughout the QR iterations used to calculate all eigenvalues of the matrix. Based on such representations, special-purpose QR algorithms that are proven *componentwise* backward stable have been designed (see [1] and [2] for the companion matrix, and [6] for the colleague matrix). This work extends the classical approaches to square domains in the complex plane. For this purpose, we construct polynomial bases in a square region, similar to orthogonal polynomials on the interval, where they satisfy three-term recurrence relations. The resulting bases are *not* orthogonal in the classical sense but are still reasonably well-conditioned. The bases naturally lead to generalized colleague matrices very similar to the classical cases. We then extend the componentwise backward stable QR algorithm of [6] to the generalized colleague matrices.

The structure of this paper is as follows. Section 2 contains the mathematical preliminaries. Section 3 contains the numerical apparatus to be used in the remainder of the paper. Section 4 builds the complex orthogonal version of the QR algorithm in [6]. Section 5 contains descriptions of our rootfinding algorithms for analytic functions in a square. Results of several numerical experiments are shown in Section 6. Finally, generalizations and conclusions can be found in Section 7.

2 Mathematical preliminaries

2.1 Notation

We will denote the transpose of a complex matrix A by A^T, and emphasize that A^T is the *transpose* of A, as opposed to the standard practice, where A^* is obtained from A by

transposition *and* complex conjugation. Similarly, we denote the *transpose* of a complex vector b by b^T, as opposed to the standard b^* obtained by transposition *and* complex conjugation. Furthermore, we denote standard l^2 norms of the complex matrix A and the complex vector b by $\|A\|$ and $\|b\|$ respectively.

2.2 Maximum principle

Suppose $f(z)$ is an analytic function in a compact domain $D \subset \mathbb{C}$. Then the maximum principle states the maximum modulus $|f(z)|$ in D is attained on the boundary ∂D. The following is the famous maximum principle for complex analytic functions (see, for example, [7]).

Theorem 1. *Given a compact domain D in $\mathbb{C}$ with boundary ∂D, the modulus $|f(z)|$ of a non-constant analytic function f in D attains its maximum value on ∂D.*

2.3 Complex orthogonalization

Given two complex vectors $u, v \in \mathbb{C}^m$, we define a complex inner product $[u, v]_w$ of u, v with respect to complex weights $w_1, w_2, \ldots, w_m \in \mathbb{C}$ by the formula

$$[u, v]_w = \sum_{j=1}^m w_j u_j v_j \,. \tag{3}$$

By analogy with the standard inner product, the complex one in (3) admits notions of norms and orthogonality. Specifically, the complex norm of a vector $u \in \mathbb{C}^m$ with respect to w is given by

$$[u]_w = \sqrt{\sum_{i=1}^m w_i u_i^2} \,. \tag{4}$$

All results in this work are independent of the branch chosen for the square root in (4).

We observe that the complex inner product behaves very differently from the standard inner product $\langle u, v \rangle = \sum_{i=j}^m \overline{u}_j v_j$, where $\overline{u}_i$ is the complex conjugate of u_i. The elements in w serving as the weights are complex instead of being positive real, and complex orthogonal vectors u, v with $[u, v]_w = 0$ are not necessarily linearly independent. Unlike the standard norm of u that is always nonzero unless u is a zero vector, the complex norm in (4) can be zero even if u is nonzero (see [3] for a detailed discussion).

Despite this unpredictable behavior, in many cases, complex orthonormal bases can be constructed via the following procedure. Given a complex symmetric matrix $Z \in \mathbb{C}^{m \times m}$ (i.e. $Z = Z^T$), a complex vector $b \in \mathbb{C}^m$ and the initial conditions $q_{-1} = 0, \beta_0 = 0$ and $q_0 = b/[b]_w$, the complex orthogonalization process

$$v = Zq_j, \tag{5}$$

$$\alpha_{j+1} = [q_j, v]_w \tag{6}$$

$$v = Zq_j - \beta_j q_{j-1} - \alpha_{j+1} q_j \,, \tag{7}$$

$$\beta_{j+1} = [v]_w \,, \tag{8}$$

$$q_{j+1} = v/\beta_{j+1} \,, \tag{9}$$

for $j = 0, 1, \ldots, n$ $(n \le m - 1)$, produces vectors $q_0, q_1, \ldots, q_n$ together with two complex coefficient vectors $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_n)^T$, provided the iteration does not break down (i.e. none of the denominators β_j in (9) is zero). It is easily verified that these constructed vectors are complex orthonormal:

$$[q_i]_w = 1 \quad \text{for all } i, \quad [q_i, q_j]_w = 0 \quad \text{for all } i \ne j. \tag{10}$$

Furthermore, substituting β_{j+1} and q_{j+1} defined respectively in (8) and (9) into (7) shows that the vectors q_j satisfy a three-term recurrence

$$Zq_j = \beta_j q_{j-1} + \alpha_{j+1} q_j + \beta_{j+1} q_{j+1}, \tag{11}$$

defined by coefficient vectors α and β. The following lemma summarizes the properties of the vectors generated via this procedure.

Lemma 2. *Suppose $b \in \mathbb{C}^m$ is an arbitrary complex vector, $w \in \mathbb{C}$ is an m-dimensional complex vector, and $Z \in \mathbb{C}^{m \times m}$ with $Z = Z^T$ is an $m \times m$ complex symmetric matrix. Provided the complex orthogonalization from (5) to (9) does not break down , there exists a sequence of $n + 1$ $(n \le m - 1)$ complex orthonormal vectors $q_0, q_1, \ldots, q_n \in \mathbb{C}^m$ with*

$$[q_i]_w = 1 \quad \text{for all } i, \quad [q_i, q_j]_w = 0 \quad \text{for all } i \ne j, \tag{12}$$

that satisfies a three-term recurrence relation, defined by two complex coefficient vectors $\alpha, \beta \in \mathbb{C}^n$,

$$Zq_j = \beta_j q_{j-1} + \alpha_{j+1} q_j + \beta_{j+1} q_{j+1}, \quad \text{for } 0 \le j \le n - 1, \tag{13}$$

with the initial conditions

$$q_{-1} = 0, \beta_0 = 0 \quad \text{and} \quad q_0 = b/[b]_w. \tag{14}$$

2.4 Linear algebra

The following lemma states that if the sum of a complex symmetric matrix A and a rank-1 update pq^T is lower Hessenberg (i.e. entries above the superdiagonal are zero), then the matrix A has a representation in terms of its diagonal, superdiagonal and the vectors p, q. It is a simple modification of the case of a Hermitian A (see, for example, [4]).

Lemma 3. *Suppose that $A \in \mathbb{C}^{n \times n}$ is complex symmetric (i.e. $A = A^T$), and let d and τ denote the diagonal and superdiagonal of A respectively. Suppose further that $p, q \in \mathbb{C}^{n \times n}$ and $A + pq^T$ is lower Hessenberg. Then*

$$a_{ij} = \begin{cases} -p_i q_j, & \text{if } j > i + 1, \\ \tau_i & \text{if } j = i + 1, \\ d_i & \text{if } j = i, \\ \tau_i & \text{if } j = i - 1, \\ -q_i p_j, & \text{if } j < i - 1, \end{cases} \tag{15}$$

where a_{ij} is the (i, j)-th entry of A.

Proof. Since the matrix $A + pq^T$ is lower Hessenberg, elements above the superdiagonal are zero, i.e. $A_{ij} + p_i q_j = 0$ for for all i, j such that $j > i + 1$. In other words, we have $A_{ij} = -p_i q_j$ for $j > i + 1$. By the complex symmetry of A, we have $A_{ji} = -p_i q_j$ for all i, j such that $j > i + 1$, or $A_{ij} = -q_i p_j$ for $i > j + 1$ by exchanging i and j. We denote the ith diagonal and superdiagonal (which is identical the subdiagonal) of A by d_i and τ_i, and obtain the form of A in (15). ∎

The following lemma states if a sum of a matrix B and a rank-1 update pq^T is lower triangular, the upper Hessenberg part of B (i.e. entries above the subdiagonal) has a representation entirely in terms of its diagonal, superdiagonal elements and the vectors p, q. It is straightforward modification of an observation in [6].

Lemma 4. *Suppose that $B \in \mathbb{C}^{n \times n}$ and let $d \in \mathbb{C}^n$, $\gamma \in \mathbb{C}^{n-1}$ denote the diagonal and subdiagonal of B respectively. Suppose further that $p, q \in \mathbb{C}^n$ and the matrix $B + pq^T$ is lower triangular. Then*

$$
b_{ij} = \begin{cases} -p_i q_j, & \text{if } j > i, \\ d_i & \text{if } j = i, \\ \gamma_i & \text{if } j = i - 1. \end{cases} \tag{16}
$$

Proof. Since the matrix $B + pq^T$ is lower triangular, elements above the diagonal are zero, i.e. $B_{ij} + p_i q_j = 0$ for all i, j such that $j > i$. In other words, $B_{ij} = -p_i q_j$ whenever $j > i$. We denote the ith diagonal and subdiagonal by vectors d_i and γ_i, and obtain the representation of the upper Hessenberg part of B in (16). ∎

The following lemma states the form of a 2×2 complex orthogonal matrix Q that eliminates the first component of an arbitrary complex vector $x \in \mathbb{C}^2$. We observe that elements of the matrix Q can be arbitrarily large, unlike the case of classical SU(2) rotations.

Lemma 5. *Given $x = (x_1, x_2)^T \in \mathbb{C}^2$ such that $x_1^2 + x_2^2 \neq 0$, suppose we define $c = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}$ and $s = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}$, or $c = 1$ and $s = 0$ if $\|x\| = 0$. Then the 2×2 matrix Q defined by*

$$
\begin{align}
Q_{11} = c \quad Q_{12} = -s \tag{17} \\
Q_{21} = s \quad Q_{22} = c \tag{18}
\end{align}
$$

eliminates the first component of x:

$$
Qx = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \sqrt{x_1^2 + x_2^2} \end{pmatrix}. \tag{19}
$$

Furthermore, Q is a complex orthogonal matrix (i.e. $Q^T = Q^{-1}$).

3 Numerical apparatus

In this section, we develop the numerical apparatus used in the remainder of the paper. We describe a procedure for constructing polynomial bases that satisfy three-term recurrence

relations in complex domains. Once a polynomial is expressed in one of the constructed bases, we use the expansion coefficients to form a matrix C, whose eigenvalues are the roots of the polynomial. Section 3.2 contains the construction of such bases. Section 3.1 contains the expressions of the matrix C. We refer to matrices of the form C as the "generalized colleague matrices", since they have structures similar to those of the classical "colleague matrices".

3.1   Generalized colleague matrices

Suppose $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T, \beta = (\beta_1, \beta_2, \ldots, \beta_n)^T \in \mathbb{C}^n$ are two complex vectors, and $P_0, P_1, P_2, \ldots, P_n$ are polynomials such that the order of P_j is j. Furthermore, suppose those polynomials satisfy a three-term recurrence relation

$$zP_j(z) = \beta_j P_{j-1}(z) + \alpha_{j+1} P_j(z) + \beta_{j+1} P_{j+1}(z) \quad \text{for all} \quad 0 \le j \le n-1 \,, \tag{20}$$

with the initial condition $P_{-1} = 0, \beta_0 = 0$. Given a polynomial p of order n defined by the formula

$$p(z) = \sum_{j=0}^{n} c_j P_j(z) \,, \tag{21}$$

the following theorem states the n roots of (21) are eigenvalues of an $n \times n$ "generalized colleague matrix". The proof is virtually identical to that of classical colleague matrices, which can be found in [8], Theorem 18.1.

Theorem 6. *The roots of the polynomial in (21) are the eigenvalues of the matrix C defined by the formula*

$$C = A + e_n q^T \,, \tag{22}$$

where A is a complex symmetric tridiagonal matrix

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix} \,, \tag{23}$$

and $e_n, q \in \mathbb{C}^n$ are defined, respectively, by the formulae

$$e_n = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}^T \,, \tag{24}$$

and

$$q = -\beta_n \begin{pmatrix} \frac{c_0}{c_n} & \frac{c_1}{c_n} & \frac{c_2}{c_n} & \cdots & \frac{c_{n-1}}{c_n} \end{pmatrix}^T \,. \tag{25}$$

We will refer to the matrix C in (22) as a *generalized colleague matrix*. Obviously, the matrix C is lower Hessenberg. It should be observed that the matrix A in Theorem 6 consists of complex entries, so it is complex symmetric, as opposed to Hermitian in the classical colleague matrix case. Thus, the matrix C has the structure of a complex symmetric tridiagonal matrix plus a rank-1 update.

Remark 3.1. Similar to the case of classical colleague matrices, all non-zero elements in the matrix A in (22) are of order 1. However, the matrix $e_n q^T$ in (22) may contain elements orders of magnitude larger. Indeed, when the polynomial p in (21) approximates an analytic function f to high accuracy, the leading coefficient c_n is likely to be small. In extreme cases, some of the elements of the vector q in (25) will be of order $1/u$, with u the machine zero. Due to this imbalance in the sizes of the two parts in the matrix C, the usual backward stability of QR algorithms is insufficient to guarantee the precision of eigenvalues computed. As a result, special-purpose QR algorithms are required. We refer readers to [6] for a more detailed discussion and the componentwise backward stable QR for classical colleague matrices.

3.2 Complex orthogonalization for three-term recurrences

In this section, we describe an empirical observation at the core of our rootfinding scheme, enabling the construction of a class of bases in simply connected compact domain in the complex plane. Not only are these bases reasonably well-conditioned, they obey three-term recurrence relations similar to those of classical orthogonal polynomials. Section 3.2.1 contains a procedure for constructing polynomials satisfying three-term recurrences in the complex plane, based on the complex orthogonalization of vectors in Lemma 2. It also contains the observation that the constructed bases are reasonably well-conditioned for most practical purposes if random weights are used in defining the complex inner product. This observation is illustrated numerically in Section 3.2.2.

3.2.1 Polynomials defined by three-term recurrences in complex plane

Given m points $z_1, z_2, \ldots, z_m$ in the complex plane, and m complex numbers $w_1, w_2, \ldots, w_m$, we form a diagonal matrix $Z \in \mathbb{C}^{m \times m}$ by the formula

$$Z = \operatorname{diag}(z_1, z_2, \ldots, z_m), \tag{26}$$

(obviously, Z is complex symmetric, i.e. $Z = Z^T$), and a complex inner product in $\mathbb{C}^m$ by choosing $\{w_j\}_{j=1}^m$ as weights in (3). Suppose further that we choose the initial vector $b = (1, 1, \ldots, 1)^T \in \mathbb{C}^m$. The matrix Z and the vector b define a complex orthogonalization process described in Lemma 2. Provided the complex orthogonalization process does not break down, by Lemma 2 it generates complex orthonormal vectors $q_0, q_1, \ldots, q_n \in \mathbb{C}^m$, along with two vectors $\alpha, \beta \in \mathbb{C}^n$ (See (13)); vectors $q_0, q_1, \ldots, q_n$ satisfy the three-term recurrence relation

$$z_i(q_j)_i = \beta_j(q_{j-1})_i + \alpha_{j+1}(q_j)_i + \beta_{j+1}(q_{j+1})_i, \tag{27}$$

for $j = 0, 1, \ldots, n-1$ and for each component $i = 1, 2, \ldots, m$, with initial conditions of the form (14). In particular, all components of q_0 are identical. The vector q_{j+1} is constructed as a linear combination of vectors Zq_j, q_j and q_{j-1} by (7). Moreover, the

vector Zq_j consists of components in q_j multiplied by $z_1, z_2, \ldots, z_m$ respectively (See (26) above). By induction, components of the vector q_j are values of a polynomial of order j at points $\{z_i\}_{i=1}^m$ for $j = 0, 1, \ldots, n$. We thus define polynomials $P_0, P_1, P_2, \ldots, P_n$, such that P_j is of order j with $P_j(z_i) = (q_j)_i$ for $i = 1, 2, \ldots, m$. As a result, the relation (27) becomes a three-term recurrence for polynomials $\{P\}_{i=0}^n$ restricted to points $\{z_i\}_{i=1}^m$. Since the polynomials are defined everywhere, we extend the polynomials and the three-term recurrence to the entire the complex plane. We thus obtain polynomials $P_0, P_1, P_2, \ldots, P_n$ that satisfy a three-term recurrence relation in (20). We summarize this construction in the following lemma.

Lemma 7. *Let $z_1, z_2, \ldots, z_m$ and $w_1, w_2, \ldots, w_m$ be two sets of complex numbers. We define a diagonal matrix $Z = \mathrm{diag}(z_1, z_2, \ldots, z_m)$, and a complex inner product of the form (3) by choosing $w_1, w_2, \ldots, w_m$ as weights. If the complex orthogonalization procedure defined above in Lemma 2 does not break down, there exist polynomials $P_0, P_1, P_2, \ldots, P_n$ ($n \leq m - 1$) such that P_j is of order j, with two vectors $\alpha, \beta \in \mathbb{C}^n$ (See (13)). In particular, the polynomials satisfy a three-term recurrence relation of the form (20) in the entire complex plane.*

Remark 3.2. We observe that the matrix Z in (26) is complex symmetric, i.e. $Z = Z^T$, as opposed to being Hermitian. Thus, if we replace the complex inner product used in the construction in Lemma 7 with the standard inner product (with complex conjugation), it is necessary to orthogonalize Zq_j in (7) against not only q_j and q_{j-1} but also all remaining preceding vectors $q_{j-2}, \ldots, q_0$, since the complex symmetry of Z is unable to enforce orthogonality automatically, destroying the three-term recurrence. As a result, the choice of complex inner products is necessary for constructing three-term recurrences.

Although the procedure in Lemma 7 provides a formal way of constructing polynomials satisfying three-term recurrences in $\mathbb{C}$, it will not in general produce a practical basis. The sizes of the polynomials tend to grow rapidly as the order increases, even at the points where the polynomials are constructed. Thus, viewed as a basis, they tend to be extremely ill-conditioned. In particular, when the points $\{z_i\}_{i=1}^m$ are Gaussian nodes of each side of a square (with weights $\{w_i\}_{i=1}^m$ the corresponding Gaussian weights), the procedure in Lemma 7 breaks down due to the complex norm $[q_1]_w$ of the first vector produced being zero. This problem of constructed bases being ill-conditioned is partially solved by the following observations.

We observe that, if the weights $\{w_i\}_{i=1}^m$ in defining the complex inner product are chosen to be *random* complex numbers, the growth of the size of polynomials at points of construction $\{z_i\}_{i=1}^m$ is suppressed – the polynomials form reasonably well-conditioned bases and the condition number grows almost linearly with the order. Moreover, this phenomenon is relatively insensitive to locations of points $\{z_i\}_{i=1}^m$ and distributions from which numbers $\{w_i\}_{i=1}^m$ are drawn. For simplicity, we choose $\{w_i\}_{i=1}^m$ to be real random numbers uniformly drawn from $[0, 1]$ for all numerical examples in the remaining of this paper. Thus, this observation combined with Lemma 7 enables the construction of practical bases satisfying three-term recurrences. Furthermore, when such a basis is constructed based on points $\{z_i\}_{i=1}^m$ chosen on the boundary ∂D of a compact domain $D \subset \mathbb{C}$, we observe that both the basis and the three-term recurrence extends *stably* to the entire domain D. It is not completely understood why the choice of random weights leads to well-conditioned polynomial bases and why their extension over a compact domain is stable. Both questions are under vigorous investigation.

9

Remark 3.3. For rootfinding purposes in a compact domain D, since the three-term recurrences extend stably over the entire D automatically, we only need values of the basis polynomials at points on its boundary ∂D. In practice, those points are chosen to be identical to those where the polynomials are constructed (See Lemma 7 above), so their values at points of interests are usually readily available. Their values elsewhere, if needed, can be obtained efficiently by evaluating their three-term recurrences.

3.2.2 Condition numbers of $\{P_j\}$

In this section, we evaluate numerically condition numbers of polynomial bases constructed via Lemma 7. Following the observations discussed above, random numbers are used as weights in defining the complex inner product for the construction. We construct such polynomials at m nodes $\{z_j\}_{j=1}^m$ on the boundary of a square, a triangle, a snake-like domain and a circle in the complex plane. Clearly, the domains in the first three examples are polygons. For choosing nodes $\{z_j\}_{j=1}^m$ on the boundaries of these polygons, we parameterize each edge by the interval $[-1, 1]$, by which each edge is discretized with equispaced or Gaussian nodes. We choose the number of discretization points on each edge to be the same (so the total node number m is different depending on the number of edges). In Figure 2, discretizations with Gaussian nodes are illustrated for the first three examples, and that with equispaced nodes is shown in the last one. Plots of polynomials P_2, P_5 and P_{20} are shown in Figure 1, and they are constructed with Gaussian nodes on the square shown in Figure 2a (for a particular realization of random weights). Roots of polynomial P_{300} are also shown in Figure 1 for all examples constructed with Gaussian nodes, where most roots of P_{300} are very close to the exterior boundary on which the polynomials are constructed.

In the first three examples, to demonstrate the method's insensitivity to the choice of $\{z_j\}_{j=1}^m$, we construct polynomials at points $\{z_j\}_{j=1}^m$ formed by both equispaced and Gaussian nodes on each edge. In the last (circle) example, the boundary is only discretized with m equispaced nodes. We measure the condition numbers of the polynomial bases by that of the $m \times (n+1)$ basis matrix, denoted by G, formed by values of polynomials $\{P_j\}_{j=0}^n$ at Gaussian nodes $\{\tilde{z}_j\}_{j=1}^m$ on each edge (not necessarily the same as $\{z_j\}_{j=1}^m$), scaled by the square root of the corresponding Gaussian weights $\{\tilde{w}_j\}_{j=1}^m$ – except for the last example, where the nodes $\{\tilde{z}_j\}_{j=1}^m$ are equispaced, and all weights $\{\tilde{w}_j\}_{j=1}^m$ are $2\pi/m$. More explicitly, the (i,j) entry of the matrix G is given by the formula

$$G_{ij} = \sqrt{\tilde{w}_i} P_j(\tilde{z}_i). \tag{28}$$

It should be observed that the weights $\{\tilde{w}_j\}_{j=1}^m$ in G are different from those for constructing the polynomials – the former are good quadrature weights while the latter are random ones. Condition numbers of the first three examples shown Figure 2, labeled "Gaussian 1" and "Gaussian 2", are those of the bases constructed at Gaussian nodes on each edge. Thus the construction points coincide with those for condition number evaluations, and the matrix G is readily available. Condition numbers labeled "Equispaced" in the first three examples are those of the bases constructed at equispaced nodes on each edge. In these cases, the corresponding three-term recurrences are used to obtain the values of the bases at Gaussian nodes, from which G is obtained. In the last example, the points for basis construction also coincide with those for condition number evaluations, so G is available directly.

Since the weights used in defining complex inner products during construction are random, the condition numbers are averaged over 10 realizations for each basis of order n. It can be seen in Figure 2 that the condition number grows almost linearly with the order n, an indication of the stability of the constructed bases and the method's domain independence. In all numerical experiments involving rootfinding in this paper, we do not use basis of order larger than $n = 100$, so the condition number is no larger than 1000.
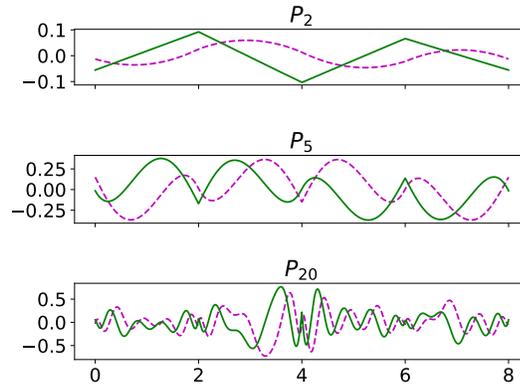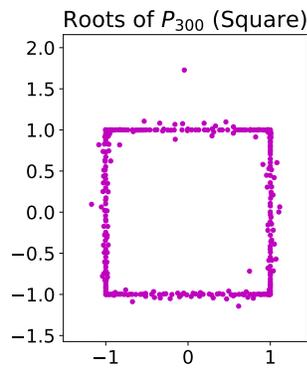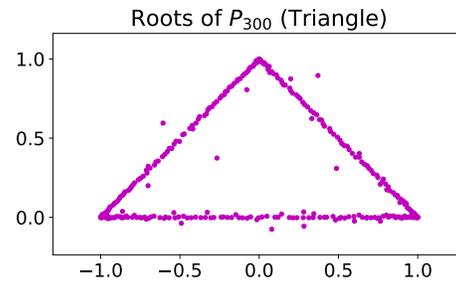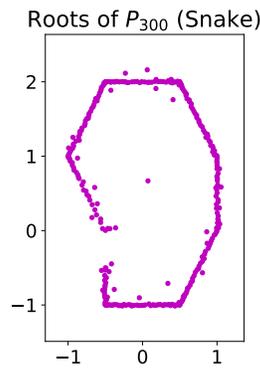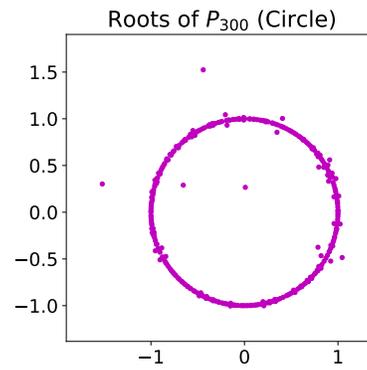
Figure 1: The graphs of the polynomials P_2, P_5 and P_{20} are shown in (a), constructed on the boundary of a square domain shown in Figure 2a. The four sides of the square are mapped to the interval $[0, 8]$, on which the polynomials are plotted. Solid lines are the real part and dotted lines the imaginary part. Roots of P_{300} for all examples are shown. All roots tend to be near the exterior boundary of domains.
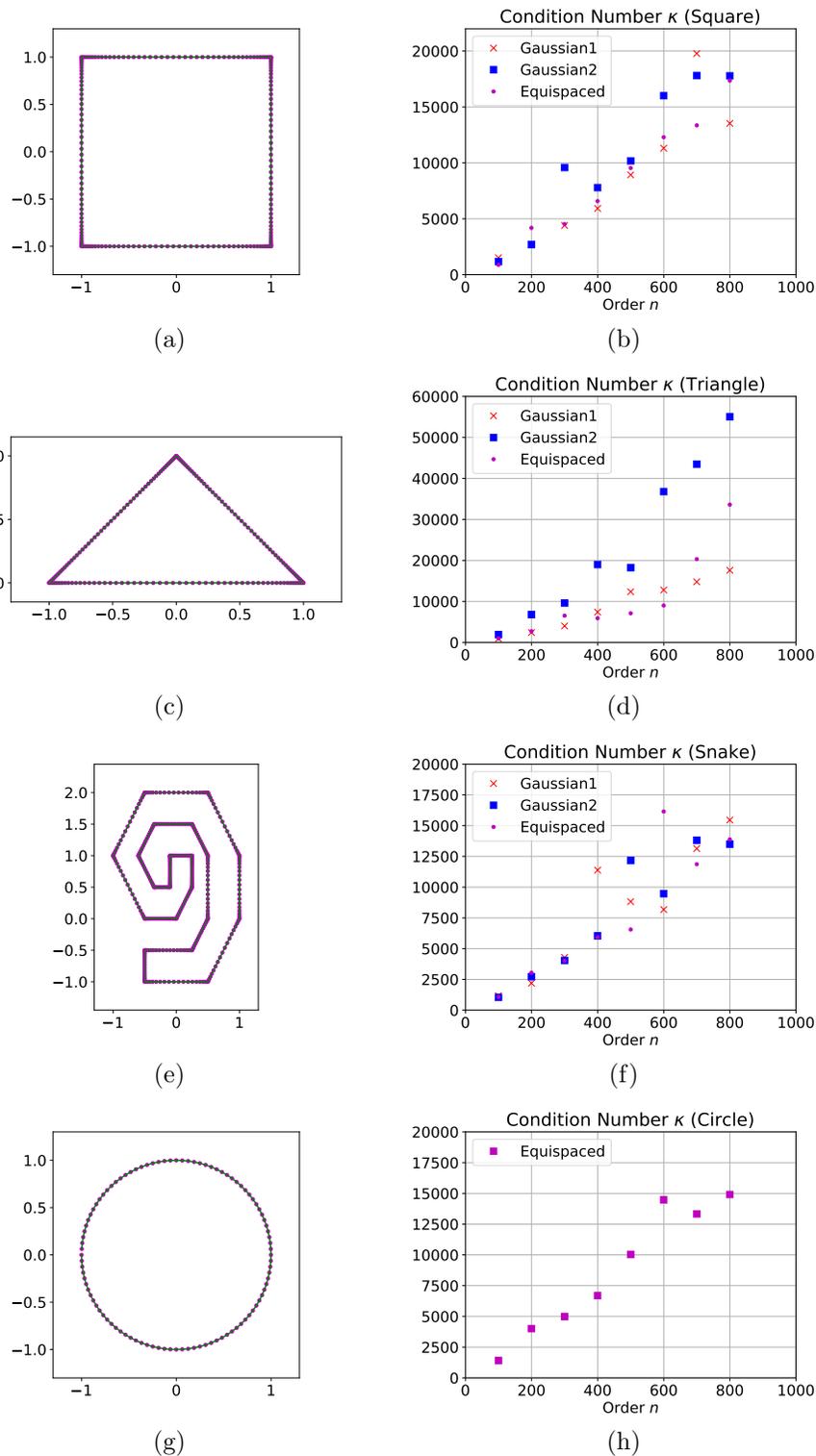
Figure 2: The condition numbers of polynomial bases constructed on the boundary of various domains are shown. Data points labeled by "Gaussian 1" for order n represent condition numbers of $\{P_j\}_{j=0}^n$ constructed with n Gaussian nodes on each side; "Gaussian 2" represents those with $n/2 + 10$ Gaussian nodes. In both cases, the points where the polynomials are constructed coincide with those where the conditioned numbers of the polynomial bases are evaluated. "Equispaced" represents those constructed with n equispaced nodes on each side, except for the triangle example where $4n$ equispaced node are used. In the last example, in total $2n$ equispaced nodes are used.

4 Complex orthogonal QR algorithm

This section contains the complex orthogonal QR algorithm used in this paper for computing the eigenvalues of the generalized colleague matrix C in the form of (22). Section 4.1 contains an overview of the algorithm, and Section 4.2 and 4.3 contain the detailed description.

The complex orthogonal QR algorithm is a slight modification of Algorithm 4 in [6]. Similar to the algorithm in [6], the QR algorithm here takes $O(n^2)$ operations, and numerical results suggest the algorithm is componentwise backward stable. The major difference is that A in (23) here is complex symmetric, whereas A in [6] is Hermitian. In order to maintain the complex symmetry of A, we replace SU(2) rotations used in the Hermitian case with complex orthogonal transforms in Lemma 5. (A discussion of similarity transforms via complex orthogonal matrices can be found in [5].) Except for this change, the algorithm is very similar to its Hermitian counterpart. Due to the use of complex orthogonal matrices, the proof of backward stability in [6] cannot be extended to this work directly. The numerical stability of our QR algorithm is still under investigation. Thus we here describe the algorithm in exact arithmetic, except for the section describing an essential correction to stabilize large rounding error when the rank-1 update has large norm.

4.1 Overview of the complex QR

The complex orthogonal QR algorithm we use is designed for a class of lower Hessenberg matrices $\mathcal{F} \subset \mathbb{C}^{n \times n}$ of the form

$$A + pq^T, \tag{29}$$

where $A \in \mathbb{C}^{n \times n}$ is complex symmetric and $p, q \in \mathbb{C}^{n \times n}$. Due to Lemma 3, the matrix A is determined entirely by:

1. The diagonal entries $d_i = a_{i,i}$, for $i = 1, 2, \ldots, n$,

2. The superdiagonal entries $\beta_i = a_{i,i+1}$ for $i = 1, 2, \ldots, n-1$,

3. The vectors p, q.

Following the terminology in [6], we refer to the four vectors d, β, p and q as the *basic elements* or *generators* of A. Below, we introduce a complex symmetric QR algorithm applied to a lower Hessenberg matrix of the form $C = A + pq^T$, defined by its generators d, β, p and q. In a slight abuse of terminology, we call complex orthogonal transforms "rotations" as in [6]. However, these complex orthogonal transforms are not rotations in general.

First, we eliminate the superdiagonal of the lower Hessenberg C. Let the matrix $U_n \in \mathbb{C}^{n \times n}$ be the complex orthogonal matrix (i.e. $U_n^T = U_n^{-1}$) that rotates the $(n-1, n)$-plane so that the superdiagonal in the $(n-1, n)$ entry is eliminated:

$$(U_n C)_{n-1,n} = 0. \tag{30}$$

We obtain the matrix U_n from an $n \times n$ identity matrix, replacing its $(n-1, n)$-plane block with the 2×2 complex orthogonal Q_n computed via Lemma 5 for eliminating

14

$C_{n-1,n}$ in the vector $(C_{n-1,n}, C_{n,n})^T$. Similarly, by the replacing $(n-2, n-1)$-plane block of an $n \times n$ identity matrix by the corresponding Q_{n-1}, we form the complex orthogonal matrix $U_{n-1} \in \mathbb{C}^{n \times n}$ that rotates the $(n-2, n-1)$-plane to eliminate the superdiagonal in the $(n-2, n-1)$ entry:

$$(U_{n-1} U_n C)_{n-2,n-1} = 0. \tag{31}$$

We can repeat this process and use complex orthogonal matrices $U_{n-2}, U_{n-3}, \ldots, U_2$ to eliminate the superdiagonal entries in the $(n-3, n-2), (n-4, n-3), \ldots, (1, 2)$ entry of the matrices $(U_{n-1} U_n C), (U_{n-2} U_{n-1} U C), \ldots, (U_3 \cdots U_{n-1} U_n C)$, respectively. We will denote the product $U_2 U_3 \cdots U_n$ of complex orthogonal transforms by U:

$$U := U_2 U_3 \cdots U_n. \tag{32}$$

Obviously, the matrix U is also complex orthogonal (i.e. $U^T = U^{-1}$). As all superdiagonal elements of C are eliminated by U, the matrix UC is lower triangular, and is given by the formula

$$UC = UA + (Up)q^T. \tag{33}$$

We denote the matrix UA by B, and the vector Up by $\underline{p}$:

$$UA := B, \quad Up := \underline{p}. \tag{34}$$

Due to Lemma 4, the upper Hessenberg part of the matrix B is determined entirely by:

1. The diagonal entries $\underline{d}_i = b_{i,i}$, for $i = 1, 2, \ldots, n$,

2. The subdiagonal entries $\underline{\gamma}_i = b_{i+1,i}$, for $i = 1, 2, \ldots, n-1$,

3. The vectors $\underline{p}, q$.

Similarly, the four vectors $\underline{d}, \underline{\gamma}, \underline{p}$ and q are the generators of (the upper Hessenberg part of) B.

Next, we multiply UC by U^T on the right to bring the lower triangular UC back to a lower Hessenberg form. Thus we obtain UCU^T via the formula

$$UCU^T = UAU^T + Up(Uq)^T. \tag{35}$$

We denote the matrix UAU^T by $\underline{A}$, and the vector Uq by $\underline{q}$:

$$UAU^T := \underline{A} = BU^T, \quad Uq := \underline{q}. \tag{36}$$

This completes one iteration of the complex orthogonal QR algorithm. Clearly, the matrix UCU^T is similar to C as U is complex orthogonal, so they have the same eigenvalues.

It should be observed that the matrix $UCU^T = \underline{A} + \underline{p}\underline{q}^T$ in (35) is still a sum of a complex symmetric matrix and a rank-1 update. Moreover, multiplying U^T on the right of the lower triangular matrix B brings it back to the lower Hessenberg form, since U^T is the product of $U_n^T, U_{n-1}^T, \ldots, U_2^T$ (See (32) above), where U_k^T only rotates column k and column $k-1$ of B. As a result, the matrix UCU^T is lower Hessenberg, thus still in the

class $\mathcal{F}$ in (29). Again, due to Lemma 3, the matrix $\underline{A}$ can be represented by its four generators.

As we will see, in the next two sections, that the generators of B can be obtained directly from generators of A, and the generators of $\underline{A}$ can be obtained from those of B. Consequently, it suffices to only rotate generators of A from (29) to (33), followed by only rotating generators of B to go from (33) to (36). Thus, each QR iteration only costs $O(n)$ operations. Moreover, since UCU^T remains in the class $\mathcal{F}$, the process can be iterated to find all n eigenvalues of the matrix A in $O(n^2)$ operations, provided all complex orthogonal transforms Q_k are defined (See Lemma 5). The next two sections contain the details of the complex orthogonal QR algorithm outlined above.

4.2 Eliminating the superdiagonal (Algorithm 2)

In this section, we describe how the QR algorithm eliminates all superdiagonal elements of the matrix $A + pq^T$, proceeding from (29) to (33). Suppose we have already eliminated the superdiagonal elements in the positions $(n-1, n), (n-2, n-1), \ldots, (k, k+1)$. We denote the vector $U_{k+1}U_{k+2} \cdots U_n p$ and the matrix $U_{k+1}U_{k+2} \cdots U_n A$, for $k = 1, 2, \ldots, n-1$, by $p^{(k+1)}$ and $B^{(k+1)}$ respectively:

$$p^{(k+1)} := U_{k+1}U_{k+2} \cdots U_n p \quad \text{and} \quad B^{(k+1)} := U_{k+1}U_{k+2} \cdots U_n A. \tag{37}$$

For convenience, we define $p^{(n+1)} = p$ and $B^{(n+1)} = A$. Thus the matrix we are working with is given by

$$B^{(k+1)} + p^{(k+1)}q^T . \tag{38}$$

Since we have rotated a part of the superdiagonal elements of A in (29) to produce part of the subdiagonal elements of B in (33), the upper Hessenberg part of $B^{(k+1)}$ is represented by the generators

1. The diagonal elements $d_i^{(k+1)} = b_{i,i}^{(k+1)}$, for $i = 1, 2, \ldots, n$,

2. The superdiagonal elements $\beta_i^{(k+1)} = b_{i,i+1}^{(k+1)}$, for $i = 1, 2, \ldots, k-1$,

3. The subdiagonal elements $\gamma_i^{(k+1)} = b_{i+1,i}^{(k+1)}$, for $i = 1, 2, \ldots, n-1$,

4. The vectors $p^{(k+1)}$ and q from which the remaining elements in the upper Hessenberg part are inferred.

First we compute the complex orthogonal matrix Q_k via Lemma 5 that eliminates the superdiagonal element in the $(k-1, k)$ position of the matrix (38), given by the formula,

$$B_{k-1,k}^{(k+1)} + (p^{(k+1)}q^T)_{k-1,k} = \beta_{k-1}^{(k+1)} + p_{k-1}^{(k+1)}q_k, \tag{39}$$

by rotating row k and row $k-1$ (See Figure 3). Next, we apply the complex orthogonal matrix Q_k separately to the generators of $B^{(k+1)}$ and to the vector $p^{(k+1)}$. Since we are only interested in computing the upper Hessenberg part of $B^{(k)}$ (See Lemma 4), we only need to update the diagonal element $d_k^{(k+1)}, d_{k-1}^{(k+1)}$ in the (k, k) and $(k-1, k-1)$ position and the subdiagonal element $\gamma_{k-1}^{(k+1)}$ and $\gamma_{k-2}^{(k+1)}$ in the $(k, k-1)$ and $(k-1, k-2)$

$$\begin{pmatrix} \ddots & \gamma_{k-3}^{(k+1)} & d_{k-2}^{(k+1)} & \beta_{k-2}^{(k+1)} & -p_{k-2}^{(k+1)}q_k & -p_{k-2}^{(k+1)}q_{k+1} & \cdots \\ \cdots & \times & \gamma_{k-2}^{(k+1)} & d_{k-1}^{(k+1)} & \beta_{k-1}^{(k+1)} & -p_{k-1}^{(k+1)}q_{k+1} & \cdots \\ \cdots & \times & b_{k,k-2}^{(k+1)} & \widehat{\gamma}_{k-1}^{(k+1)} & d_k^{(k+1)} & -p_k^{(k+1)}q_{k+1} & \cdots \\ \cdots & \times & \times & \times & \gamma_k^{(k+1)} & d_{k+1}^{(k+1)} & \ddots \end{pmatrix}$$

Figure 3: The $(k-1)$-th and k-th rows of $B^{(k+1)}$, represented by its generators.

position. They are straightforward to update except for $\gamma_{k-2}^{(k+1)}$ since updating it requires the sub-subdiagonal element $b_{k,k-2}^{(k+1)}$ in the $(k, k-2)$ plane. This element can inferred via the following observation.

Suppose we multiply the matrix (38) by the rotations $U_n^T, U_{n-1}^T, \cdots, U_{k+1}^T$ from the right, and the result is given by the formula

$$B^{(k+1)}U_n^T U_{n-1}^T \cdots U_{k+1}^T + p^{(k+1)}(U_{k+1}U_{k+2}\cdots U_n q)^T. \tag{40}$$

Since multiplying U_j^T from the right only rotates column j and $j-1$ of a matrix, applying $U_n^T, U_{n-1}^T, \cdots, U_{k+1}^T$ rotates the matrix (38) back to a lower Hessenberg one. For the same reason, the desired element $b_{k,k-2}^{(k+1)}$ is not affected by the sequence of transforms $U_n^T U_{n-1}^T \cdots U_{k+1}^T$. In other words, $b_{k,k-2}^{(k+1)}$ is also the $(k, k-2)$ entry of $B^{(k+1)}U_n^T U_{n-1}^T \cdots U_{k+1}^T$:

$$\left(B^{(k+1)}U_n^T U_{n-1}^T \cdots U_{k+1}^T\right)_{k,k-2} = b_{k,k-2}^{(k+1)}. \tag{41}$$

Moreover, the matrix $B^{(k+1)}U_n^T U_{n-1}^T \cdots U_{k+1}^T$ is complex symmetric since $B^{(k+1)} = U_{k+1}U_{k+2}\cdots U_n A$ (See (37) above)) and A is complex symmetric. As a result, the matrix in (40) is a complex symmetric matrix with a rank-1 update. Due to Lemma 3, the sub-subdiagonal element in (41) of the complex symmetric part can be inferred from the $(k-2, k)$ entry of the rank-1 part $p^{(k+1)}(U_{k+1}U_{k+2}\cdots U_n q)^T$. This observation shows that it is necessary to compute the following vector, denoted by $\tilde{q}^{(k+1)}$:

$$\tilde{q}^{(k+1)} := U_{k+1}U_{k+2}\cdots U_n q. \tag{42}$$

Then the sub-subdiagonal $b_{k,k-2}^{(k+1)}$ is computed by the formula

$$b_{k,k-2}^{(k+1)} = -\tilde{q}_k^{(k+1)}p_{k-2}^{(k+1)}. \tag{43}$$

(See Line 14 of Algorithm 2.)

After obtaining $b_{k,k-2}^{(k+1)}$, we update the remaining generators affected by the elimination of the superdiagonal (39). The element in the $(k-1, k-2)$ position of $B^{(k+1)}$, represented by $\gamma_{k-2}^{(k+1)}$, is updated in Line 6 of Algorithm 2. Next, the elements in the $(k-1, k-1)$ and $(k, k-1)$ positions of $B^{(k+1)}$, represented by $d_{k-1}^{(k-1)}$ and $\gamma_{k-1}^{(k+1)}$ respectively, are updated

in a straightforward way in Line 8. Finally, the elements in the $(k-1, k)$ and (k, k) positions of $B^{(k+1)}$, represented by $\beta_{k-1}^{(k+1)}$ and $d_k^{(k+1)}$ respectively, are updated in Line 9, and the vector $p^{(k+1)}$ is rotated in Line 10.

4.2.1 Correction for large $\|pq^T\|$

As discussed in Remark 3.1, the size of the vector q in generalized colleague matrices can be orders magnitude larger than those in the complex symmetric part A. These elements are "mixed" when the superdiagonal is eliminated in Section 4.2. When it is done in finite-precision arithmetic, large rounding errors due to the large size of q can occur – they can be as large as the size of elements in A, completely destroying the precision of eigenvalues to be computed. Fortunately, the representation via generators permits rounding errors to be corrected even when large q is present. The corrections are as follows.

In Section 4.2, we described in some detail the process of elimination of the $(k-1, k)$ entry in (39) of the matrix $B^{(k+1)} + p^{(k+1)}q^T$ in (38) by the matrix Q_k. After rotating all the generators, we update $\beta_{k-1}^{(k+1)}$ and $p_{k-1}^{(k+1)}$ by $\beta_{k-1}^{(k)}$ and $p_{k-1}^{(k)}$ respectively. Thus, in exact arithmetic, the updated $(k-1, k)$ entry becomes zero. More explicitly, we have

$$B_{k-1,k}^{(k)} + (p^{(k)}q^T)_{k-1,k} = \beta_{k-1}^{(k)} + p_{k-1}^{(k)}q_k = 0 . \tag{44}$$

Thus we in principle only need to keep $p_{k-1}^{(k)}$ to infer $\beta_{k-1}^{(k)}$. (This is why $\beta_{k-1}^{(k)}$ is not part of the generators, as can be seen from the representation of $B_{k-1,k}^{(k+1)}$ below (38).) However, when rounding error are considered and elements in q are large, inferring $\beta_{k-1}^{(k)}$ from the computed value $\widehat{p}_{k-1}^{(k)}q_k$ of $p_{k-1}^{(k)}q_k$ can lead to errors much greater than the directly computed value $\widehat{\beta}_{k-1}^{(k)}$. This can be understood by first observing that $\beta_{k-1}^{(k)}$ and $p_{k-1}^{(k)}q_k$ are obtained by applying Q_k to the vectors

$$(\beta_{k-1}^{(k+1)}, d_k^{(k+1)})^T, \quad (p_{k-1}^{(k+1)}q_k, p_k^{(k+1)}q_k)^T \tag{45}$$

respectively. Furthermore, the rounding errors are of the size the machine zero u times the norms of these vectors. As a result, the computed errors in $\widehat{\beta}_{k-1}^{(k)}$ and $p_{k-1}^{(k)}q_k$ are of the order

$$\sqrt{|\beta_{k-1}^{(k+1)}|^2 + |d_k^{(k+1)}|^2}\|Q_k\|u, \quad \sqrt{|p_{k-1}^{(k+1)}q_k|^2 + |p_k^{(k+1)}q_k|^2}\|Q_k\|u \tag{46}$$

respectively, where $\|Q_k\|$ is the size of the complex orthogonal transform. (The norm $\|Q_k\|$ is 1 if it is unitary, i.e. a true rotation.) We observe that inferring $\widehat{\beta}_{k-1}^{(k)}$ from $\widehat{p}_{k-1}^{(k)}q_k$ leads to larger computed error when

$$|p_{k-1}^{(k+1)}q_k|^2 + |p_k^{(k+1)}q_k|^2 > |\beta_{k-1}^{(k+1)}|^2 + |d_k^{(k+1)}|^2 , \tag{47}$$

so we apply a correction based on (44) by setting the computed $\widehat{p}_{k-1}^{(k)}$ to be

$$\widehat{p}_{k-1}^{(k)} = -\widehat{\beta}_{k-1}^{(k)}/q_k. \tag{48}$$

(See Line 12 of Algorithm 2.) On the other hand, when

$$|p_{k-1}^{(k+1)}q_k|^2 + |p_k^{(k+1)}q_k|^2 \leq |\beta_{k-1}^{(k+1)}|^2 + |d_k^{(k+1)}|^2, \tag{49}$$

the correction in (48) is not necessary since the error in $\widehat{p}_{k-1}^{(k)}q_k$ will be smaller than the error in $\widehat{\beta}_{k-1}^{(k)}$. The correction described above is essential in achieving componentwise backward stability for the QR algorithm with SU(2) rotations in [6] for classical colleague matrices. We refer readers to [6] for a detailed discussion on the correction's impact on the stability of the algorithm.

This process of eliminating the superdiagonal elements can be repeated, until the upper Hessenberg part of the matrix $B = U_2 U_3 \cdots U_n A$ is obtained, together with the vector $\underline{p} = U_2 U_3 \cdots U_n p$.

4.3 Rotating back to Hessenberg form (Algorithm 3)

In this section, we describe how our QR algorithm rotates the triangular matrix UC in (33) back to lower Hessenberg form UCU^T in (36). Given the matrix B and vectors $\underline{p}$ and q from the preceding section, the sum $B + \underline{p}q^T$ is lower triangular. Suppose that we have already applied the rotation matrix $U_n^T, U_{n-1}^T, \cdots, U_{k+1}^T$ to right of B and q^T. We denote the vector $U_{k+1} U_{k+2} \cdots U_n q$ by $q^{(k+1)}$ and the matrix $B U_n^T U_{n-1}^T \cdots U_{k+1}^T$ by $A^{(k+1)}$:

$$q^{(k+1)} := U_{k+1} U_{k+2} \cdots U_n q, \quad \text{and} \quad A^{(k+1)} := B U_n^T U_{n-1}^T \cdots U_{k+1}^T. \tag{50}$$

For convenience, we define $q^{(n+1)} = q$ and $A^{(n+1)} = B$. Immediately after applying U_{k+1}^T, we have rotated part of the subdiagonal element of B in (33) to produce part of the superdiagonal elements of $\underline{A}$ in (36), so the upper Hessenberg part of $A^{(k+1)}$ is represented by the generators

1. The diagonal entries $d^{(k+1)} = a_{i,i}^{(k+1)}$, for $i = 1, 2, \ldots, n$,

2. The superdiagonal entries $\beta_i^{(k+1)} = a_{i,i+1}^{(k+1)}$ for $i = k, k+1, \ldots, n-1$,

3. The subdiagonal entries $\gamma_i^{(k+1)} = a_{i+1,i}^{(k+1)}$ for $i = 1, 2, \ldots, k-1$,

4. The vectors $\underline{p}$ and $q^{(k+1)}$, from which the remaining elements in the upper triangular part are inferred.

To multiply the matrix $A^{(k+1)} + \underline{p}(q^{(k+1)})^T$ by U_k^T from the right, we apply the complex orthogonal rotation matrix Q_k^T separately to the generators of $A^{(k+1)}$ and to the vector $q^{(k+1)}$. We start by rotating the diagonal and superdiagonal elements in the $(k-1, k-1)$ and $(k-1, k)$ positions of $A^{(k+1)}$, represented by $d_{k-1}^{(k+1)}$ and $-\underline{p}_{k-1}q_k^{(k+1)}$ respectively, in Line 2 of Algorithm 3, saving the superdiagonal element in $\beta_{k-1}^{(k)}$ (See Figure 4). Next, we rotate the elements in the $(k, k-1)$ and (k, k) positions, represented by $\gamma_{k-1}^{(k+1)}$ and $d_k^{(k+1)}$ respectively, in a straightforward way in Line 3; since we are only interested in computing the upper triangular part of $A^{(k)}$, we only update the diagonal entry. Finally, we rotate the vector $q^{(k+1)}$ in Line 4. This process of applying the complex orthogonal transforms is repeated until the matrix $\underline{A}$ and the vector $\underline{q}$ in (36) are obtained.

$$
\begin{pmatrix}
\ddots & \vdots & \vdots & \vdots \\
d_{k-2}^{(k+1)} & -p_{k-2}q_{k-1}^{(k+1)} & -p_{k-2}q_k^{(k+1)} & -p_{k-2}q_{k+1}^{(k+1)} \\
\gamma_{k-2}^{(k+1)} & d_{k-1}^{(k+1)} & -p_{k-1}q_k^{(k+1)} & -p_{k-1}q_{k+1}^{(k+1)} \\
\times & \gamma_{k-1}^{(k+1)} & d_k^{(k+1)} & \beta_k^{(k+1)} \\
\times & \times & \times & d_{k+1}^{(k+1)} \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
$$

Figure 4: The $(k-1)$-th and k-th columns of $A^{(k+1)}$, represented by its generators.

4.4 The QR algorithm with explicit shifts (Algorithm 4)

The elimination of the superdiagonal described in Section 4.2 (Algorithm 2), followed by transforming back to Hessenberg form described in Section 4.3 (Algorithm 3), makes up a single iteration in the complex orthogonal QR algorithm for finding eigenvalues of $A + pq^T$. In principle, it can be iterated to find all eigenvalues. However, unlike the algorithms in [6], complex orthogonal rotations in our algorithm can get large occasionally. (A typical size distribution of the rotations can be found in Section 6.2.) When no shift is applied in QR iterations, the linear convergence rate of eigenvalues will generally require an unreasonable number of iterations to reach a acceptable precision. Although large size rotations are rare, they do sometimes accumulate to sizable numerical error when the iteration number is large. This can cause breakdowns of the complex orthogonal QR algorithm, and such breakdowns have been observed in numerical experiments. As a result, the introduction of shifts is essential in order to accelerate convergence, reducing the iteration number and avoiding breakdowns. In this paper, we only provide the QR algorithm with explicit Wilkinson shifts in Algorithm 4. No breakdowns of Algorithm 4 have been observed once the shifts are introduced; the analysis of its numerical stability, however, is still under vigorous investigation. All eigenvalues in this paper are found by this shifted version of complex orthogonal QR.

5 Description of rootfinding algorithms

In this section, we describe algorithms for finding all roots of a given complex analytic function over a square domain. Section 5.1 contains the precomputation of a polynomial basis satisfying a three-term recurrence on a square domain (Algorithm 1). Section 5.2 contains the non-adaptive rootfinding algorithm (Algorithm 5) on square domains, followed by the adaptive version (Algorithm 6) in Section 5.3.

The non-adaptive rootfinding algorithm's inputs are a square domain $D \subset \mathbb{C}$, specified by its center z_0 and side length $2l$, and a function f analytic in D, whose roots in D are to be found, and two accuracies $\epsilon_{\exp}$ and ϵ_{eig}. The first $\epsilon_{\exp}$ controls the approximation accuracy of the input function f by polynomials on the boundary ∂D; the second ϵ_{eig} specifies the accuracy of eigenvalues of the colleague matrix found by the complex

orthogonal QR. In order to robustly catch all roots very near the boundary ∂D, a small constant $\delta > 0$ is specified, so that all roots within a slightly extended square of length $2l(1 + \delta)$ are retained and they are the output of the algorithm. We refer to this slightly expanded square as the δ-extended domain of D.

The non-adaptive rootfinding algorithm consists of two stages. First, an approximating polynomial in the precomputed basis of the given function f is computed via least squares. From the coefficients of the approximating polynomial, a generalized colleague matrix is formed. Second, the complex orthogonal QR is applied to compute the eigenvalues, which are also the roots of the approximating polynomial (Theorem 6). The roots of the polynomial inside the domain D are identified as the computed roots of the function f.

The adaptive version of the rootfinding algorithm takes an additional input parameter n_{exp}, the order of polynomial expansions for approximating f. It can be chosen to be relatively small so the expansion accuracy ϵ_{exp} does not have to be achieved on the domain D. When this happens, the algorithm continues to divide the square D adaptively into smaller squares until the accuracy ϵ_{exp} is achieved on those smaller squares. Then the non-adaptive algorithm is invoked on each smaller square obtained from the subdivision. After removing redundant roots near boundaries of neighboring squares, all the remaining roots are collected as the output of the adaptive algorithm.

5.1 Basis precomputation (Algorithm 1)

We follow the description in Section 3.2 to construct a reasonably well-conditioned basis of order n on the boundary $\partial\Omega$ of the square Ω, shown in Figure 2a, centered at the origin with side length 2. For convenience, we choose the points where the polynomials are constructed to be Gaussian nodes on each side of the square Ω. (See Section 3.2.2.) In the non-adaptive case, the order n is chosen to be sufficiently large so that the expansion accuracy ϵ_{exp} is achieved; in the adaptive case, n is specified by the user.

1. Choose the number of Gaussian nodes k ($k \geq n/2$) on each side the square. Generate points $z_1, z_2, \ldots, z_m$, consisting of all Gaussian nodes from the four sides (See Figure 2a), where the total number $m = 4k$. Generate m random weights $w_1, w_2, \ldots, w_m$, uniformly drawn from $[0, 1]$ to define a complex inner product in the form of (3).

2. Perform the complex orthogonalization process (See Lemma 7) to obtain a polynomial basis $P_0, P_1, \ldots, P_n$ of order n on the boundary $\partial\Omega$ at the chosen points $z_1, z_2, \ldots, z_m$, and the two vectors $\alpha, \beta \in \mathbb{C}^n$. The polynomials in $\{P_j\}_{j=1}^n$ satisfies the three-term recurrence relation in the form of (20), defined by the vectors α, β.

3. Generate Gaussian weights $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_m$ corresponding to Gaussian nodes $z_1, z_2, \ldots, z_m$ on the four sides. Form the m by $n+1$ basis matrix G given by the formula

$$G = \begin{pmatrix} \sqrt{\tilde{w}_1} P_0(z_1) & \sqrt{\tilde{w}_1} P_1(z_1) & \cdots & \sqrt{\tilde{w}_1} P_n(z_1) \\ \sqrt{\tilde{w}_2} P_0(z_2) & \sqrt{\tilde{w}_2} P_1(z_2) & \cdots & \sqrt{\tilde{w}_2} P_n(z_2) \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{\tilde{w}_m} P_0(z_m) & \sqrt{\tilde{w}_m} P_1(z_m) & \cdots & \sqrt{\tilde{w}_m} P_n(z_m) \end{pmatrix}. \tag{51}$$

21

4. Initialize for solving least squares involving the matrix G in (51), by computing the reduced QR factorization of G

$$G = QR,$$

where $Q \in \mathbb{C}^{m \times (n+1)}$ and $R \in \mathbb{C}^{(n+1) \times (n+1)}$.

The precomputation is relatively inexpensive and only needs to be done once – it can be done on the flight or precomputed before hand and stored. The vectors α, β and the matrices Q, R are the only information required in our algorithms for rootfinding in a square domain.

Remark 5.1. There is nothing special about the choice of QR above for solving least squares. Any standard method can be applied.

Remark 5.2. In the complex orthogonalization process, the constructed vector q_i will automatically be complex orthogonal to any q_j for $j < i - 2$ in exact arithmetic due to the complex symmetry of the matrix Z. In practice, when q_i is computed, reorthogonalizing q_i against all preceding q_j for $j < i$ is desirable to increase numerical stability. After Line 4 of Algorithm 1, we reorthogonalize v via the formula

$$v \leftarrow v - [v, q_i]q_i - [v, q_{i-1}]q_{i-1} - [v, q_{i-2}]q_{i-2} \cdots - [v, q_0]q_0. \tag{52}$$

Doing so will ensure the complex orthonormality of the computed basis $q_0, q_1, q_2, \ldots, q_n$ holds to near full accuracy. The reorthogonalization in (52) can be repeated if necessary to increase the numerical stability further.

5.2 Non-adaptive rootfinding (Algorithm 5)

We first perform the precompution described Section 5.1, obtaining a polynomial basis $\{P_j\}_{j=0}^n$ of order n on the boundary $\partial \Omega$, the vectors α, β defining the three-term recurrence, and the reduced QR of the basis matrix G in (51).

Stage 1: construction of colleague matrices

Since the basis $\{P_j\}_{j=0}^n$ is precomputed on the square Ω (See Section 5.1), we map the input square D to Ω via a translation and a scaling transform. Thus we transform the function f in D accordingly into its translated and scaled version $\tilde{f}$ in Ω, so that we find roots of f in D by first finding roots of $\tilde{f}$ in Ω, followed by transforming those roots back from Ω to D. To find roots of $\tilde{f}$, we form an approximating polynomial p of $\tilde{f}$, given by the formula

$$p(z) = \sum_{j=0}^n c_j P_j(z), \tag{53}$$

whose expansion coefficients are computed by solving a least-squares problem.

1. Translate and scale f via the formula

$$\tilde{f}(z) = f(lz + z_0). \tag{54}$$

22

2. Form a vector $g \in \mathbb{C}^m$ given by the formula

$$g = \begin{pmatrix} \sqrt{\tilde{w}_1} \tilde{f}(z_1) \\ \sqrt{\tilde{w}_2} \tilde{f}(z_2) \\ \vdots \\ \sqrt{\tilde{w}_m} \tilde{f}(z_m) \end{pmatrix}. \tag{55}$$

3. Compute the vector $c = (c_0, c_1, \ldots, c_n)^T \in \mathbb{C}^{n+1}$, containing the expansion coefficients in (53), via the formula

$$c = R^{-1} Q^* g, \tag{56}$$

which is the least-squares solution to the linear system $Gc = g$. This procedure takes $O(mn)$ operations.

4. Estimate the expansion error by $|c_n|/\|c\|$. If $|c_n|/\|c\|$ is smaller than the given expansion accuracy $\epsilon_{\exp}$, accept the expansion and the coefficient vector c. Otherwise, repeat the above procedures with a larger n.

The colleague matrix $C = A + e_n q^T$ in (22) in principle can be formed explicitly – the complex symmetric A can be formed by elements in the vectors α, β, and the vector q can be obtained from the coefficient vector c together with the last element of β. However, this is unnecessary since we only rotate generators of C in the complex orthogonal QR algorithm (See Section 4). The generators of C here are vectors α, β, e_n and q and they will be the inputs of the complex orthogonal QR.

Remark 5.3. It should be observed that the approximating polynomial p of the function $\tilde{f}$ in (53) is only constructed on the boundary of $\partial \Omega$ since the points $z_1, z_2, \ldots, z_m$ used in forming the basis matrix G in (51) are all on $\partial \Omega$. However, due to the maximum principle (Theorem 1), the largest approximation error $\max_{z \in \Omega} |\tilde{f}(z) - p(z)|$ is attained on the boundary $\partial \Omega$ since the difference $\tilde{f} - p$ is analytic in Ω. ($\tilde{f}$ is analytic in Ω since f is analytic in D.) Thus the absolute error of the approximating polynomial p is smaller in Ω. As a result, once the approximation on the boundary $\partial \Omega$ achieves the required accuracy, it holds over the entire Ω automatically.

Remark 5.4. When the expansion accuracy $\epsilon_{\exp}$ is not satisfied for an initial choice of n, it is necessary to recompute all basis for a larger order n. This can be avoided by choosing a generously large order n in the precomputation stage. This could be undesirable since the number m of Gaussian nodeson needed on the boundary also grows accordingly. When a very large n is needed, it is recommended to switch to the adaptive version (Algorithm 6) with a reasonable n for better efficiency and accuracy.

Stage 2: rootfinding by colleague matrices

Due to Theorem 6, we find the roots of p by finding eigenvalues of the colleague matrix C. Eigenvalues of C are computed by the complex orthogonal QR algorithm in Section 4. We only keep roots of p within Ω or very near the boundary $\partial \Omega$ specified by the constant δ. These roots are identified as those of the transformed function $\tilde{f}$. Finally, we translate and scale roots of $\tilde{f}$ in Ω back so that we recover the roots of the input function f in D.

1. Compute the eigenvalues of the colleague matrix C, represented by its generators α, β, e_n and q, by the complex orthogonal QR (Algorithm 4) in $O(n^2)$ operations. The eigenvalues/roots are labeled by $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n$.

2. Only retain a root $\tilde{r}_j$ for $j = 1, 2, \ldots, n$, if it is inside the δ-extended square of the domain Ω:

$$|\mathrm{Re}\,\tilde{r}_j| < 1 + \delta \quad \text{and} \quad |\mathrm{Im}\,\tilde{r}_j| < 1 + \delta. \tag{57}$$

3. Scale and translate all remaining roots back to the domain D from Ω

$$r_j = l\tilde{r}_j + z_0 \tag{58}$$

and all r_j are identified as roots of the input function f in D.

Remark 5.5. The robustness and accuracy of rootfinding can be improved by applying Newton's method

$$r_j \leftarrow r_j - \frac{f(r_j)}{f'(r_j)} \tag{59}$$

for one or two iterations after roots r_i are located at little cost, if the values of f and f' are available inside D.

5.3 Adaptive rootfinding (Algorithm 6)

In this section we describe an adaptive version of the rootfinding algorithm in Section 5.2. The adaptive version has identical inputs as the non-adaptive one, besides one additional input n_{exp} as the chosen expansion order. (Obviously, the input n_{exp} should be no smaller than the order of the precomputed basis n.) The algorithm has two stages. First, it divides the domain D into smaller squares, on which the polynomial expansion of order n_{exp} converge to the pre-specified accuracy ϵ_{exp}. In the second stage, rootfinding is performed on these squares by procedures very similar to those in Algorithm 5, followed by removing duplicated roots near boundaries of neighboring squares.

Stage 1: adaptive domain division

In this stage, we recursively divide the input domain D until the order n_{exp} expansion achieves the specified accuracy ϵ_{exp}. For convenience, we label the input domain D as D_0. Here we illustrate this process by describing the case depicted in Figure 5 where D is divided three times. Initially, we compute the order n_{exp} expansion on input domain D_0, as in Step 2 and 3 in Section 5.2, and the accuracy is not reached. We divide D_0 into four identical squares D_1, D_2, D_3, D_4. We repeat the process to these four squares and the expansion accuracy except on D_4. Then we continue to divide D_4 into four pieces D_5, D_6, D_7, D_8. Again we check expansion accuracy on all newly generated squares $D_5 - D_8$ and the accuracy requirement is satisfied for all squares except for D_5. Then we continue to divide D_5 into $D_9, D_{10}, D_{11}, D_{12}$ We keep all expansion coefficient vectors $c^{(i)} \in \mathbb{C}^{n_{\mathrm{exp}}+1}$ for (implicitly) forming colleague matrices on domains D_i, where the expansion accuracy ϵ_{exp} is reached. As a result, rootfinding in this example only happens
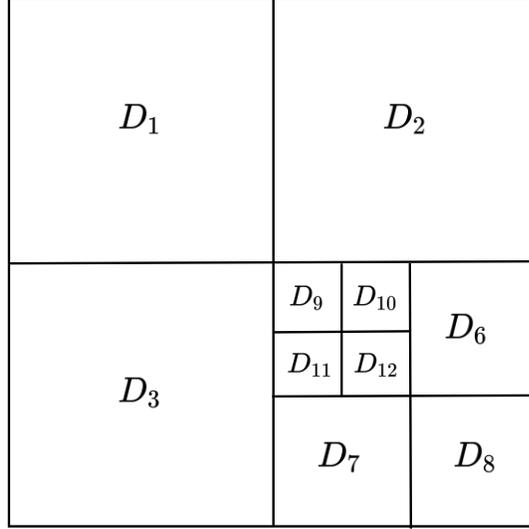
Figure 5: The input domain D, labeled D_0, is divided three times during an adaptive rootfinding by Algorithm 6. Squares D_4 and D_5 are not shown since they are divided further; D_4 is divided in to D_5, D_6, D_7, D_8 and D_5 is divided into $D_9, D_{10}, D_{11}, D_{12}$. In each square shown here, rootfinding is done by finding eigenvalues of the colleague matrix.

on squares D_1, D_4 and $D_5 - D_{12}$, so in total 10 eigenvalue problems of size $n_{\exp}$ by $n_{\exp}$ are solved.

In the following description, we initialize D_0 as the input domain D, and $i = 0$. We also keep track the total number of squares k ever formed; since initially we only have the input domain, the number of squares $k = 1$.

1. For domain D_i centered at $z_0^{(i)}$ with length $2l^{(i)}$, translate and scale the input function $f|_{D_i}$ restricted to D_i to the square Ω via the formula

$$\tilde{f}(z) = f|_{D_i}(l^{(i)}z + z_0^{(i)})\,. \tag{60}$$

2. Form a vector $g^{(i)} \in \mathbb{C}^m$ given by the formula

$$g^{(i)} = \begin{pmatrix} \sqrt{\tilde{w}_1}\tilde{f}(z_1) \\ \sqrt{\tilde{w}_2}\tilde{f}(z_2) \\ \vdots \\ \sqrt{\tilde{w}_m}\tilde{f}(z_m) \end{pmatrix}\,. \tag{61}$$

3. Compute the vector $c^{(i)} \in \mathbb{C}^{n_{\exp}+1}$ containing the expansion coefficients by the formula

$$c^{(i)} = V\Sigma^{-1}U^*g^{(i)}\,, \tag{62}$$

which is the least-squares solution to the linear system $Gc^{(i)} = g^{(i)}$. This procedure takes $O(mn_{\exp})$ operations.

25

4. Estimate the expansion error by $\left|c_{n_{\exp}}^{(i)}\right|/\|c^{(i)}\|$. If $\left|c_{n_{\exp}}^{(i)}\right|/\|c^{(i)}\|$ is smaller than the given expansion accuracy $\epsilon_{\exp}$, accept the expansion and the coefficient vector $c^{(i)}$ and save it for constructing colleague matrices. If the accuracy $\epsilon_{\exp}$ is not reached, continue to divide domain D_i into four square domains $D_{k+1}, D_{k+2}, D_{k+3}, D_{k+4}$ and increment the number of squares k by 4, i.e. $k \leftarrow k + 4$.

5. If i does not equal k, this means there are still squares where expansions may not converge. Increment i by 1 and go to Step 1.

After all necessary subdivisions, all expansions of f clearly achieve the accuracy $\epsilon_{\exp}$. Since the coefficient vector $c^{(i)}$ is saved whenever the expansion converges in the divided domain D_i, the corresponding colleague matrix $C^{(i)}$ can be formed for rootfinding on D_i. As in the non-adaptive case, the matrix $C^{(i)}$ is not formed explicitly but is represented by its generators as inputs to the complex orthogonal QR.

Stage 2: rootfinding by colleague matrices

We have successfully divided D into smaller squares. On each of those domain D_i, where an approximating polynomial of order $n_{\exp}$ converges to the specified accuracy $\epsilon_{\exp}$, we find roots of f in the δ-extended domain of D_i as in Stage 2 of the non-adaptive version in Section 5.2. Then we collect all roots on each smaller squares and remove duplicated roots near edges of neighboring squares. The remaining roots will be identified as roots of f on the δ-extended domain of the input domain D.

1. For all D_i in which the expansion converges to $\epsilon_{\exp}$, form the generator $q^{(i)}$ according to the coefficient vector $c^{(i)}$ and β. Compute the eigenvalues of the colleague matrix $C^{(i)}$, represented by its generators α, β, e_n and $q^{(i)}$, by the complex orthogonal QR (Algorithm 4) in $O(n_{\exp}^2)$ operations. Label eigenvalues for each D_i as $\tilde{r}_j^{(i)}$.

2. For each problem on D_i, only keep a root $\tilde{r}_j^{(i)}$ if it is inside of the square domain slightly extended from Ω:

$$|\operatorname{Re} \tilde{r}_j| < 1 + \delta \quad \text{and} \quad |\operatorname{Im} \tilde{r}_j| < 1 + \delta. \tag{63}$$

3. Scale and translate all remaining roots back to the domain D_i from Ω

$$r_j^{(i)} = l^{(i)} \tilde{r}_j^{(i)} + z_0^{(i)} \tag{64}$$

and all $r_j^{(i)}$ left are identified as roots of the input function $f|_{D_i}$ restricted to D_i.

4. Collect all roots of $r_j^{(i)}$ for all domain D_i where rootfinding is performed. Remove duplicated roots found in neighboring squares. The remaining roots are identified as roots of the input function f in the domain D.

Remark 5.6. In the final step of Stage 2, duplicated roots from neighboring squares are removed. It should be observed that multiple roots can be distinguished from duplicated roots by the accuracy they are computed. When the roots are simple, our algorithm achieves full accuracy determined the machine zero u, especially paired with refinements

by Newton's method (see Remark 5.5). On the other hand, roots with multiplicity m are sensitive to perturbations and can only be computed to the mth root of machine zero $u^{1/m}$. Consequently, one can distinguish if close roots are duplicated or due to multiplicity by the number digits they agree.

Algorithm 1 (Precomputation of basis matrix and three term-recurrence) **Inputs**: The algorithm accepts m nodes $z_1, z_2, \ldots, z_m$ that are Gaussian nodes on each sides of the square domain Ω, along with m Gaussian weights $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_m$ associated with those m nodes. It also accepts another set of m real weights $w_1, w_2, \ldots, w_m \in \mathbb{C}$, drawn uniformly from $[0, 1]$, for the complex inner product in (3). **Outputs**: It returns as output the vectors $\alpha, \beta \in \mathbb{C}^n$ that define the three-term recurrence relation. It also returns the reduced QR factorization of QR of the matrix G in (51).

1: Define $Z = \mathrm{diag}(z_1, z_2, \ldots, z_m)$ and $b = (1, 1, \ldots, 1) \in \mathbb{C}^m$
2: Initialization for complex orthogonalization
$\qquad q_{-1} \leftarrow 0, \beta_0 \leftarrow 0, q_0 \leftarrow b/\, [b]_w$
3: **for** $i = 0, \ldots, n - 1$ **do**
$\qquad v \leftarrow Zq_i$
4: $\quad$ Compute α_{i+1}: $\alpha_{i+1} \leftarrow [q_i, v]_w$
$\qquad v \leftarrow v - \beta_i q_{i-1} - \alpha_{i+1} q_i$
5: $\quad$ Compute β_{i+1}: $\beta_{i+1} \leftarrow [v]_w$
6: $\quad$ Compute q_{i+1}: $q_{i+1} \leftarrow v/\beta_{i+1}$
7: **end for**
8: Set the ith component of q_j as the the value of P_j at z_i: $P_i(z_j) = (q_i)_j$
9: Form the basis matrix $G \in \mathbb{C}^{m \times (n+1)}$ for least squares:
$\qquad G_{ij} \leftarrow \sqrt{\tilde{w}_j} P_i(z_j).$
10: Compute the reduced QR factorization of G:
$\qquad QR \leftarrow G$

Algorithm 2 (A single elimination of the superdiagonal) **Inputs:** This algorithm accepts as inputs two vectors d and β representing the diagonal and superdiagonal, respectively, of an $n \times n$ complex symmetric matrix A, as well as two vectors p and q of length n, where $A + pq$ is lower Hessenberg. **Outputs:** It returns as its outputs the rotation matrices $Q_2, Q_3, \ldots, Q_n \in \mathbb{C}^{2\times 2}$ so that, letting $U_k \in \mathbb{C}^{n \times n}$, $k = 2, 3, \ldots, n$, denote the matrices that rotate the $(k-1, k)$-plane by Q_k, $U_2 U_3 \cdots U_n(A + pq)$ is lower triangular. It also returns the vectors $\underline{d}$, $\underline{\gamma}$, and $\underline{p}$, where $\underline{d}$ and $\underline{\gamma}$ represent the diagonal and subdiagonal, respectively, of the matrix $U_2 \bar{U}_3 \cdots U_n A$, and $\underline{p} = U_2 U_3 \cdots U_n p$.

1: Set $\gamma \leftarrow \beta$, where γ represents the subdiagonal.
2: Make a copy of q, setting $\tilde{q} \leftarrow q$.
3: **for** $k = n, n-1, \ldots, 2$ **do**
4:     Construct the 2×2 rotation matrix complex symmetric Q_k so that

$$\left(Q_k \begin{bmatrix} \beta_{k-1} + p_{k-1} q_k \\ d_k + p_k q_k \end{bmatrix} \right)_1 = 0.$$

5:     **if** $k \neq 2$ **then**
6:         Rotate the subdiagonal and the sub-subdiagonal:

$$\gamma_{k-2} \leftarrow \left(Q_k \begin{bmatrix} \gamma_{k-2} \\ -\tilde{q}_k p_{k-2} \end{bmatrix} \right)_1$$

7:     **end if**
8:     Rotate the diagonal and the subdiagonal: $\begin{bmatrix} d_{k-1} \\ \gamma_{k-1} \end{bmatrix} \leftarrow Q_k \begin{bmatrix} d_{k-1} \\ \gamma_{k-1} \end{bmatrix}$.
9:     Rotate the superdiagonal and the diagonal: $\begin{bmatrix} \beta_{k-1} \\ d_k \end{bmatrix} \leftarrow Q_k \begin{bmatrix} \beta_{k-1} \\ d_k \end{bmatrix}$.
10:     Rotate p: $\begin{bmatrix} p_{k-1} \\ p_k \end{bmatrix} \leftarrow Q_k \begin{bmatrix} p_{k-1} \\ p_k \end{bmatrix}$
11:     **if** $|p_{k-1} q_k|^2 + |p_k q_k|^2 > |\beta_{k-1}|^2 + |d_k|^2$ **then**
12:         Correct the vector p, setting $p_{k-1} \leftarrow -\frac{\beta_{k-1}}{q_k}$
13:     **end if**
14:     Rotate $\tilde{q}$: $\begin{bmatrix} \tilde{q}_{k-1} \\ \tilde{q}_k \end{bmatrix} \leftarrow Q_k \begin{bmatrix} \tilde{q}_{k-1} \\ \tilde{q}_k \end{bmatrix}$
15: **end for**
16: Set $\underline{d} \leftarrow d$, $\underline{\gamma} \leftarrow \gamma$, and $\underline{p} \leftarrow p$.

Algorithm 3 (Rotating the matrix back to Hessenberg form) **Inputs:** This algorithm accepts as inputs $n-1$ rotation matrices $Q_2, Q_3, \ldots, Q_n \in \mathbb{C}^{n \times n}$, two vectors d and γ representing the diagonal and subdiagonal, respectively, of an $n \times n$ complex matrix B, and two vectors $\underline{p}$ and q of length n, where $B + \underline{p}q^T$ is lower triangular. **Outputs:** Letting $U_k \in \mathbb{C}^{n \times n}$, $\overline{k} = 2, 3, \ldots, n$, denote the matrices that rotate the $(k-1, k)$-plane by Q_k, this algorithm returns as its outputs the vectors $\underline{d}$, $\underline{\beta}$, and $\underline{q}$, where $\underline{d}$ and β represent the diagonal and superdiagonal, respectively, of the matrix $\overline{BU_n^T U_{n-1}^T \cdots U_2^T}$, and $\underline{q} = U_2 U_3 \cdots U_n q$.

1: **for** $k = n, n-1, \ldots, 2$ **do**

2:     Rotate the diagonal and the superdiagonal:

$$\begin{bmatrix} d_{k-1} \\ \beta_{k-1} \end{bmatrix} \leftarrow Q_k \begin{bmatrix} d_{k-1} \\ -\underline{p}_{k-1} q_k \end{bmatrix}.$$

3:     Rotate the subdiagonal and the diagonal:

$$d_k \leftarrow \left(Q_k \begin{bmatrix} \gamma_{k-1} \\ d_k \end{bmatrix} \right)_2$$

4:     Rotate q: $\begin{bmatrix} q_{k-1} \\ q_k \end{bmatrix} \leftarrow Q_k \begin{bmatrix} q_{k-1} \\ q_k \end{bmatrix}$

5: **end for**

6: Set $\underline{d} \leftarrow d$, $\underline{\beta} \leftarrow \beta$, and $\underline{q} \leftarrow q$.

Algorithm 4 (Shifted explicit QR) **Inputs:** This algorithm accepts as inputs two vectors d and β representing the diagonal and superdiagonal, respectively, of an $n \times n$ complex symmetric matrix A, as well as two vectors p and q of length n, where $A + pq^T$ is lower Hessenberg. It also accepts a tolerance $\epsilon > 0$, which determines the accuracy the eigenvalues are computed to. **Outputs:** It returns as its output the vector λ of length n containing the eigenvalues of the matrix $A + pq^T$.

1: **for** $i = 1, 2, \ldots, n-1$ **do**

2:     Set $\mu_{\text{sum}} \leftarrow 0$.

3:     **while** $\beta_i + p_i q_{i+1}^T \geq \epsilon$ **do** $\triangleright$ Check if $(A + pq^T)_{i,i+1}$ is close to zero

4:         Compute the eigenvalues μ_1 and μ_2 of the 2×2 submatrix
$$\begin{bmatrix} d_i + p_i q_i^T & \beta_i + p_i q_{i+1}^T \\ \beta_i + p_{i+1} q_i^T & d_{i+1} + p_{i+1} q_{i+1}^T \end{bmatrix}.$$ $\triangleright$ This is just $(A + pq^T)_{i:i+1, i:i+1}$

5:         Set μ to whichever of μ_1 and μ_2 is closest to $d_i + p_i q_i^T$.

6:         Set $\mu_{\text{sum}} \leftarrow \mu_{\text{sum}} + \mu$.

7:         Set $d_{i:n} \leftarrow d_{i:n} - \mu$.

8:         Perform one iteration of QR (one step of Algorithm 2 followed by one step of Algorithm 3) on the submatrix $(A + pq^T)_{i:n, i:n}$ defined by the vectors $d_{i:n}$, $\beta_{i:n-1}$, $p_{i:n}$, and $q_{i:n}$.

9:     **end while**

10:     Set $d_{i:n} \leftarrow d_{i:n} + \mu_{\text{sum}}$.

11: **end for**

12: Set $\lambda_i \leftarrow d_i + p_i q_i^T$, for $i = 1, 2, \ldots, n$.

Algorithm 5 (Non-adaptive rootfinding in square domain D) **Inputs**: This algorithm accepts as inputs an analytic function f on a square domain D centered at z_0 with length $2l$, in which roots of f are to be found, as well as precomputed quantities from Algorithm 1, including vectors α, β defining the three-term recurrence, matrices Q, R constituting the reduced QR of the basis matrix $G \in \mathbb{C}^{m \times (n+1)}$, Gaussian points $z_1, z_2, \ldots, z_m$ and Gaussian weights $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_m$ used in defining G. It also accepts two accuracies $\epsilon_{\exp}$, the accuracy for the expansion, and ϵ_{eig}, the accuracy for eigenvalues, as well as a positive constant δ so that roots within the δ-extension of D are included. **Outputs:** It returns as output the roots of f on the domain D.

1: Scale and translate f in D to $\tilde{f}$ in Ω:
$$\tilde{f}(z) \leftarrow f(lz + z_0) \text{ for } z \in D\,.$$

2: Form the vector g for least squares:
$$g \leftarrow \begin{pmatrix} \sqrt{\tilde{w}_1}\tilde{f}(z_1) & \sqrt{\tilde{w}_2}\tilde{f}(z_2) & \cdots & \sqrt{\tilde{w}_m}\tilde{f}(z_m) \end{pmatrix}^T.$$

3: Compute the coefficient vector c:
$$c \leftarrow V\Sigma^{-1}U^*g\,.$$

4: **if** $|c_n|/\|c\| > \epsilon_{\exp}$ **then**

5:     Precompute a basis with a larger order n and go to Line 2.

6: **end if**

7: Form vector q:
$$q \leftarrow -\beta_n \begin{pmatrix} \frac{c_0}{c_n} & \frac{c_1}{c_n} & \cdots & \frac{c_{n-1}}{c_n} \end{pmatrix}^T.$$

8: Perform Algorithm 4 on the colleague matrix generated by vectors α and $\beta_{1:n-1}$, representing the diagonal and subdiagonal elements of A, as well as vectors e_n and q for the rank-1 part. The accuracy of computed eigenvalues $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n$ is determined by ϵ_{eig}.

9: Keep eigenvalues $\tilde{r}_i$ if $|\mathrm{Re}\,\tilde{r}_i| < 1 + \delta$ and $|\mathrm{Im}\,\tilde{r}_i| < 1 + \delta$.

10: Scale and translate the remaining eigenvalues:
$$r_i \leftarrow l\tilde{r}_i + z_0\,.$$

11: Return all r_i as roots of f on the δ-extended domain of D.

Algorithm 6 (Adaptive rootfinding in square domain D) **Inputs**: This algorithm accepts as inputs an analytic function f on a square domain D centered at z_0 with length $2l$, in which roots of f are to be found, as well as precomputed quantities from Algorithm 1, including vectors α, β defining the three-term recurrence, matrices Q, R constituting the reduced QR of the basis matrix $G \in \mathbb{C}^{m \times (n+1)}$, Gaussian points $z_1, z_2, \ldots, z_m$ and Gaussian weights $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_m$ used in defining G. It also accepts two accuracies $\epsilon_{\exp}$, the accuracy for the expansion, and ϵ_{eig}, the accuracy for eigenvalues, as well as n the order of expansion and a positive constant δ so that roots within the δ-extension of D are included. **Outputs:** It returns as output the roots of f in the domain D.

1: Set $D_0 \leftarrow D$, $l^{(0)} \leftarrow l$ and $z_0^{(0)} \leftarrow z_0$.
2: Set $i \leftarrow 0$.   ▷ Square index
3: Set $k \leftarrow 1$.   ▷ Total number of squares
4: **while** $i < k$ **do**   ▷ If the last square has not been reached
5:      Scale and translate f on D_i to $\tilde{f}$ on Ω:
$$\tilde{f}(z) \leftarrow f(l^{(i)}z + z_0^{(i)}) \text{ for } z \in D_i .$$
6:      Form the vector $g^{(i)}$ for least squares:
$$g^{(i)} \leftarrow \begin{pmatrix} \sqrt{\tilde{w}_1}\tilde{f}(z_1) & \sqrt{\tilde{w}_2}\tilde{f}(z_2) & \cdots & \sqrt{\tilde{w}_m}\tilde{f}(z_m) \end{pmatrix}^T$$
7:      Compute the coefficient vector $c^{(i)}$:
$$c^{(i)} \leftarrow R^{-1}Q^*g^{(i)} .$$
8:      **if** $\left|c_n^{(i)}\right|/\|c^{(i)}\| > \epsilon_{\exp}$ **then**   ▷ Keep dividing if not convergent
9:         Divide D_i into four smaller squares $D_{k+1}, D_{k+2}, D_{k+3}, D_{k+4}$.
10:        Set $k \leftarrow k + 4$.
11:      **else**   ▷ Keep the coefficient vector if convergent
12:        Form vector $q^{(i)}$:
$$q^{(i)} \leftarrow -\beta_n \begin{pmatrix} \frac{c_0}{c_n} & \frac{c_1}{c_n} & \cdots & \frac{c_{n-1}}{c_n} \end{pmatrix}^T .$$
13:      **end if**
14:      Set $i \leftarrow i + 1$.   ▷ Go to the next square
15: **end while**
16: **for** D_i with a convergent expansion **do**
17:      Perform Algorithm 4 on the colleague matrix generated by vectors α and $\beta_{1:n-1}$, representing the diagonal and subdiagonal elements of A, as well as vectors e_n and $q^{(i)}$ for the rank-1 part. The accuracy of computed eigenvalues $\tilde{r}_1^{(i)}, \tilde{r}_2^{(i)}, \ldots, \tilde{r}_n^{(i)}$ is determined by ϵ_{eig}.
18:      Keep eigenvalue $\tilde{r}_j^{(i)}$ if $\left|\text{Re}\,\tilde{r}_j^{(i)}\right| < 1 + \delta$ and $\left|\text{Im}\,\tilde{r}_j^{(i)}\right| < 1 + \delta$.
19:      Scale and translate the remaining eigenvalues:
$$r_j^{(i)} \leftarrow l^{(i)}\tilde{r}_j^{(i)} + z_0^{(i)} .$$
20: **end for**
21: Collect all roots $r_j^{(i)}$ and remove duplicated roots from neighboring squares. Return all the remaining roots as the roots of f in the domain D.

6 Numerical results

In this section, we demonstrate the performance of Algorithm 5 (non-adaptive) and Algorithm 16 (adaptive) for finding all roots analytic functions over specified any square domain D. Because the use of complex orthogonal matrix, no proof of backward stability is provided in this paper. Numerical experiments show our complex orthogonal QR algorithm is stable in practice and behaves similarly to the version with unitary rotations in [6].

We apply Algorithm 5 to the first three examples and Algorithm 6 to the rest two. In all experiments in this paper, we do not use Newton's method to refine roots computed by algorithms, although the refinement can be done at little cost to achieve full machine accuracy. We set the two accuracies ϵ_{exp} (for the polynomial expansion) and ϵ_{eig} (for the QR algorithm) both to the machine zero. We set the domain extension parameter δ to be 10^{-6}, determining the relative extension of the square within which roots are kept. For all experiments, we used the same set of polynomials as the basis, precomputed to order $n = 100$, with 60 Gaussian nodes on each side of the square (so in total $m = 240$). The condition number of this basis (See Section 3.2.2) is about 1000. When different expansion orders are used, the number of points on each side is always kept as 60. Given a function f whose roots are to be found, we measure the accuracy of the computed roots $\widehat{z}$ given by the quantity $\eta(\widehat{z})$ defined by

$$\eta(\widehat{z}) := \left| \frac{f(\widehat{z})}{f'(\widehat{z})} \right|, \tag{65}$$

where f' is the derivative of f; it is the size of the Newton step if we were to refine the computed root $\widehat{z}$ by Newton's method.

In all experiments, we report the order n of the approximating polynomial, the number n_{roots} of computed roots, the value of $\max_i \eta(\widehat{z}_i)$, where the maximum is taken over all roots found over the region of interests. For the non-adaptive algorithm, we report the norm $\|q\|$ of the vector q in the colleague matrix (See (25)) and the size of rotations in QR iterations. For the adaptive version, we report the maximum level n_{levels} of the recursive division and the number of times n_{eig} eigenvalues are computed by Algorithm 4. The number n_{eigs} is also the number of squares formed by the recursive division.

We implemented our algorithms in FORTRAN 77, and compiled it using GNU Fortran, version 12.2.0. For all timing experiments, the Fortran codes were compiled with the optimization `-O3` flag. All experiments are conducted on a MacBook Pro laptop, with 64GB of RAM and an Apple M1 Max CPU.

6.1 f_{cosh}: A function with a pole near the square

Here we consider a function given by

$$f_{\text{cosh}}(z) = \frac{\cosh(3\pi z/2)}{z - 2}, \tag{66}$$

over a square centered at 0 with side length 2, with a singularity outside of the square domain. Due to the singularity at $z = 2$, the sizes of the approximating polynomial coefficients decay geometrically, as shown in Figure 6. The results of numerical experiments with order $n = 80$ and 100 are shown in Table 2.

n	$\|q\|$	n_{roots}	$\max_i \eta(\widehat{z}_i)$	Timings (s)
80	$0.11 \cdot 10^{17}$	4	$0.55 \cdot 10^{-11}$	$0.27 \cdot 10^{-1}$
100	$0.18 \cdot 10^{17}$	4	$0.83 \cdot 10^{-11}$	$0.37 \cdot 10^{-1}$

Table 1: The results of computing roots of $f_{\cosh}(z)$ in double precision, using the non-adaptive Algorithm 5 with the polynomial expansion order $n = 80$ and $n = 100$.
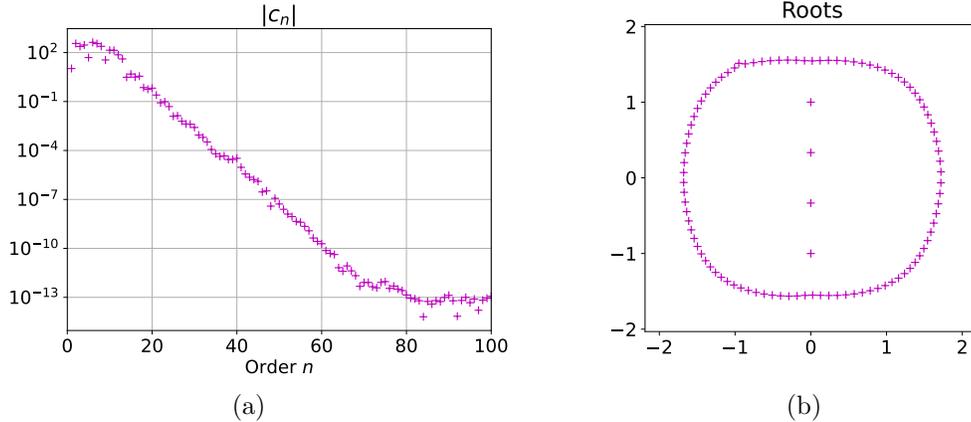

(a) (b)

Figure 6: The magnitude of the leading expansion coefficients $|c_n|$ of $f_{\cosh}$ is plotted in (a). Due to the singularity at $z = 2$, the coefficients decays geometrically. All computed roots of the approximating polynomial of $f_{\cosh}$ are shown in (b).

6.2 f_{poly}: A polynomial with simple roots

Here we consider a polynomial of order 5, whose formula is given by

$$f_{\text{poly}}(z) = (z - 0.5)(z - 0.9)(z + 0.8)(z - 0.7i)(z + 0.1i). \tag{67}$$

All its roots are simple and inside a square of length $l = 2$ centered at $z_0 = 0$. We construct a polynomial of order n and apply Algorithm 5 to find its roots. The results of our experiment are shown in Table 2 (double precision) and Table 3 (extended precision). In Figure 7, the size of complex orthogonal rotations for $n = 40$ in the QR iterations for finding roots of f_{poly} are provided. It is clear that the error in computed roots are insensitive to the size of $\|q\|$ and the order of polynomial approximation used. This is the feature of the unitary version of our QR algorithms in [6], which is proven to be componentwise backward stable. Moreover, in Figure 7 (a), Although no similar proof is provided in this work, results in Table 2 and 3 strongly suggest our algorithm has similar stability properties as the unitary version in [6].
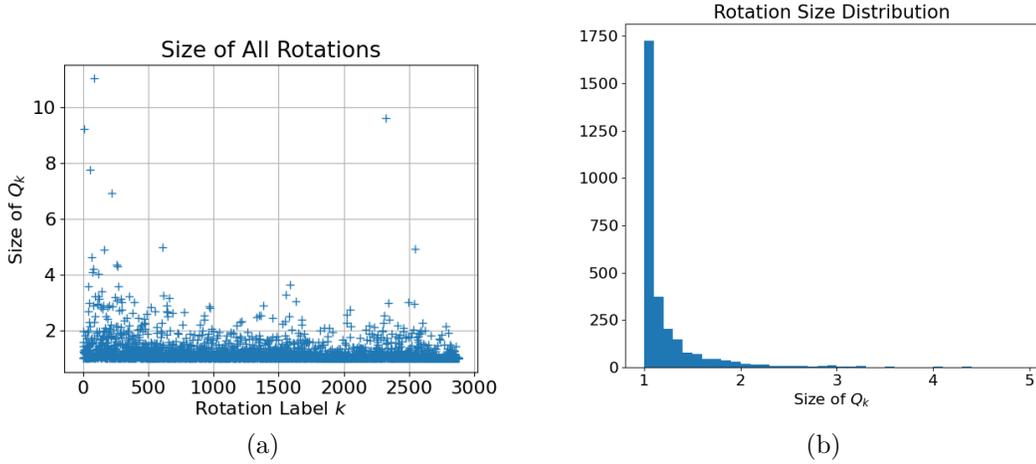
34

Figure 7: The size of all 2882 complex orthogonal rotations defined by $\sqrt{|c|^2 + |s|^2}$, in the QR iterations ($n = 40$) for rooting roots of f_{poly} is shown in (a). The distribution of rotations of size smaller than 5 is shown in (b).

n	$\|q\|$	n_{roots}	$\max_i \eta(\widehat{z}_i)$	Timings (s)
5	$0.52 \cdot 10^1$	5	$0.10 \cdot 10^{-12}$	$0.53 \cdot 10^{-3}$
6	$0.20 \cdot 10^{17}$	5	$0.25 \cdot 10^{-13}$	$0.53 \cdot 10^{-3}$
50	$0.10 \cdot 10^{16}$	5	$0.19 \cdot 10^{-13}$	$0.15 \cdot 10^{-1}$
100	$0.15 \cdot 10^{17}$	5	$0.64 \cdot 10^{-13}$	$0.37 \cdot 10^{-1}$

Table 2: The results of computing roots of f_{poly}, a polynomial of order 5, in double precision, using Algorithm 5 with the polynomial expansion order $n = 5, 6, 50$ and 100.

n	$\|q\|$	n_{roots}	$\max_i \eta(\widehat{z}_i)$	Timings (s)
5	$0.52 \cdot 10^1$	5	$0.90 \cdot 10^{-30}$	$0.12 \cdot 10^{-1}$
6	$0.61 \cdot 10^{34}$	5	$0.78 \cdot 10^{-30}$	$0.12 \cdot 10^{-1}$
50	$0.76 \cdot 10^{33}$	5	$0.94 \cdot 10^{-30}$	$0.85 \cdot 10^{-1}$
100	$0.66 \cdot 10^{34}$	5	$0.94 \cdot 10^{-30}$	$0.27 \cdot 10^{0}$

Table 3: The results of computing roots of $f_{\text{poly}}(z)$, a polynomial of order 5, in extended precision, using Algorithm 5 with the polynomial expansion order $n = 5, 6, 50$ and 100.

6.3 f_{mult}: A polynomial with multiple roots

Here we consider a polynomial of order 12 that has identical roots as f_{poly} above with increased multiplicity:

$$f_{\text{mult}}(z) = (z - 0.5)^5 (z - 0.9)^3 (z + 0.8)(z - 0.7\mathrm{i})(z + 0.1\mathrm{i})^2. \tag{68}$$

We construct a polynomial of order $n = 30$ and apply Algorithm 5 to find its roots in both double and extended precision. All computed roots and their error estimation are

35

shown in Figure 8. The error is roughly proportional to $u^{1/m}$, where u is the machine zero and m is the root's corresponding multiplicity. As discussed in Remark 5.6, this can be used to distinguish multiple roots from redundant roots when removing redundant roots in Algorithm 6.
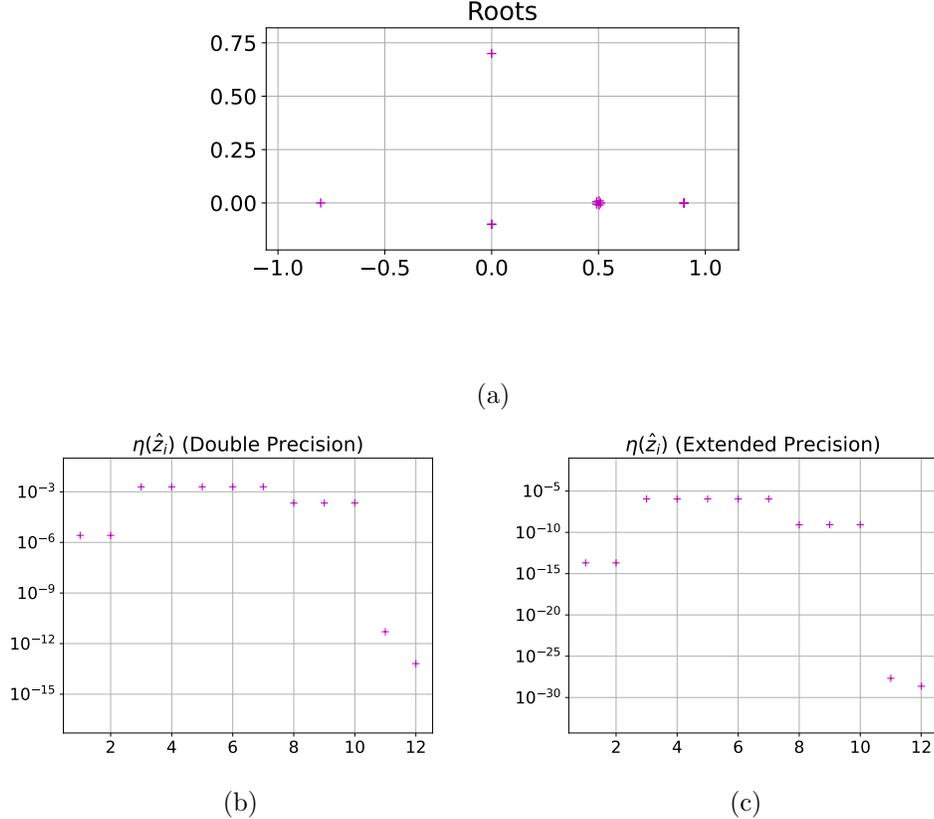

(a)

(b)   (c)

Figure 8: The roots of f_{mult} computed by Algorithm 5 in double precision are shown in (a), and their errors in double and extended precision computation are shown in (b) and (c) respectively. The error is roughly proportional to $u^{1/m}$, where u is the machine zero and m is the multiplicity of the roots.

6.4 f_{clust}: A function with clustering zeros

Here we consider a function given by

$$f_{\mathrm{clust}}(z) = \sin\left(\frac{100}{e^{i\pi/4}z - 2}\right), \tag{69}$$

which has roots clustering around the singularity $z_\star = \sqrt{2} - i\sqrt{2}$ along the line $\theta = -\pi/4$. We apply Algorithm 6 to find roots of f_{clust} within a square of length 2.75 centered at the origin $z_0 = 0$. Near the singularity, the adaptivity of Algorithm 5 allows the size of squares to be chosen automatically, so that the polynomial expansions with a fixed order n can resolve the fast oscillation of f_{clust} near the singularity $z_\star$.

The results of our numerical experiment are shown in Table 4 (in double precision) and Table 5 (in extended precision). Figure 9 contains the roots $\widehat{z}_i$ of f_{clust} found, as well as every center of squares formed by the recursive subdivision and the error estimation $\eta(\widehat{z}_i)$ of roots $\widehat{z}_i$. It is worth noting that the roots found by order $n = 45$ expansions are consistently less accurate than those found by order $n = 30$ ones, as indicated by error estimation shown in Figure 9d and 9e. This can be explained by the following observation. The larger the expansion order n, the larger the square on which the expansion converges, as shown in Figure 9b and 9c. On a larger square, the maximum magnitude of the function f tends to be larger as f is analytic. Since the maximum principle (Theorem 1) only provides bounds for the pointwise *absolute* error $\max_{z \in \partial D} |f(z) - p(z)|$ for any approximating polynomial p, larger f values on the boundary leads larger errors of p inside. This eventually leads to less accurate roots when a larger order n is adopted.

Although a larger order expansion leads to less accurate roots, increasing n indeed leads to significantly fewer divided squares, as demonstrated by the numbers of eigenvalue problems n_{eigs} shown in Table 4 and 5. In practice, one can choose a relatively large expansion order n, so that fewer eigenvalue problems need to be solved while the computed roots are still reasonably accurate, achieving a balance between efficiency and robustness. Once all roots are located within reasonable accuracy, one Newton refinement will give full machine accuracy to the computed roots.

n	n_{roots}	$\max_i \eta(\widehat{z}_i)$	n_{levels}	n_{eigs}	Timings (s)
45	565	$0.68 \cdot 10^{-12}$	14	8836	$0.30 \cdot 10^1$
30	565	$0.19 \cdot 10^{-14}$	16	76864	$0.14 \cdot 10^2$

Table 4: The results of computing roots of $f_{\text{clust}}(z)$ in double precision, using the adaptive Algorithm 6 with the polynomial expansion order $n = 45$ and $n = 30$.

n	n_{roots}	$\max_i \eta(\widehat{z}_i)$	n_{levels}	n_{eigs}	Timings (s)
100	565	$0.30 \cdot 10^{-23}$	13	1852	$0.34 \cdot 10^3$
70	565	$0.83 \cdot 10^{-29}$	14	8899	$0.88 \cdot 10^3$

Table 5: The results of computing roots of $f_{\text{clust}}(z)$ in extended precision, using the adaptive Algorithm 6 with the polynomial expansion order $n = 100$ and $n = 70$.

6.5 f_{entire}: An entire function

Here we consider a function given by

$$f_{\text{entire}}(z) = \frac{\sin(3\pi z)}{z - 2} . \tag{70}$$

Since $z = 2$ is a simple zero of the numerator $\sin(3\pi z)$, the function f_{entire} is an entire function. This function only has simple roots on the real line. We apply Algorithm 6 to find roots of f_{entire} within a square of length 50 centered at the origin $z_0 = 10 - 20i$. Although our algorithms are not designed for such an environment, Algorithm 6 is still effective in finding the roots. Figure 10 contains the roots $\widehat{z}_i$ of f found, every center of
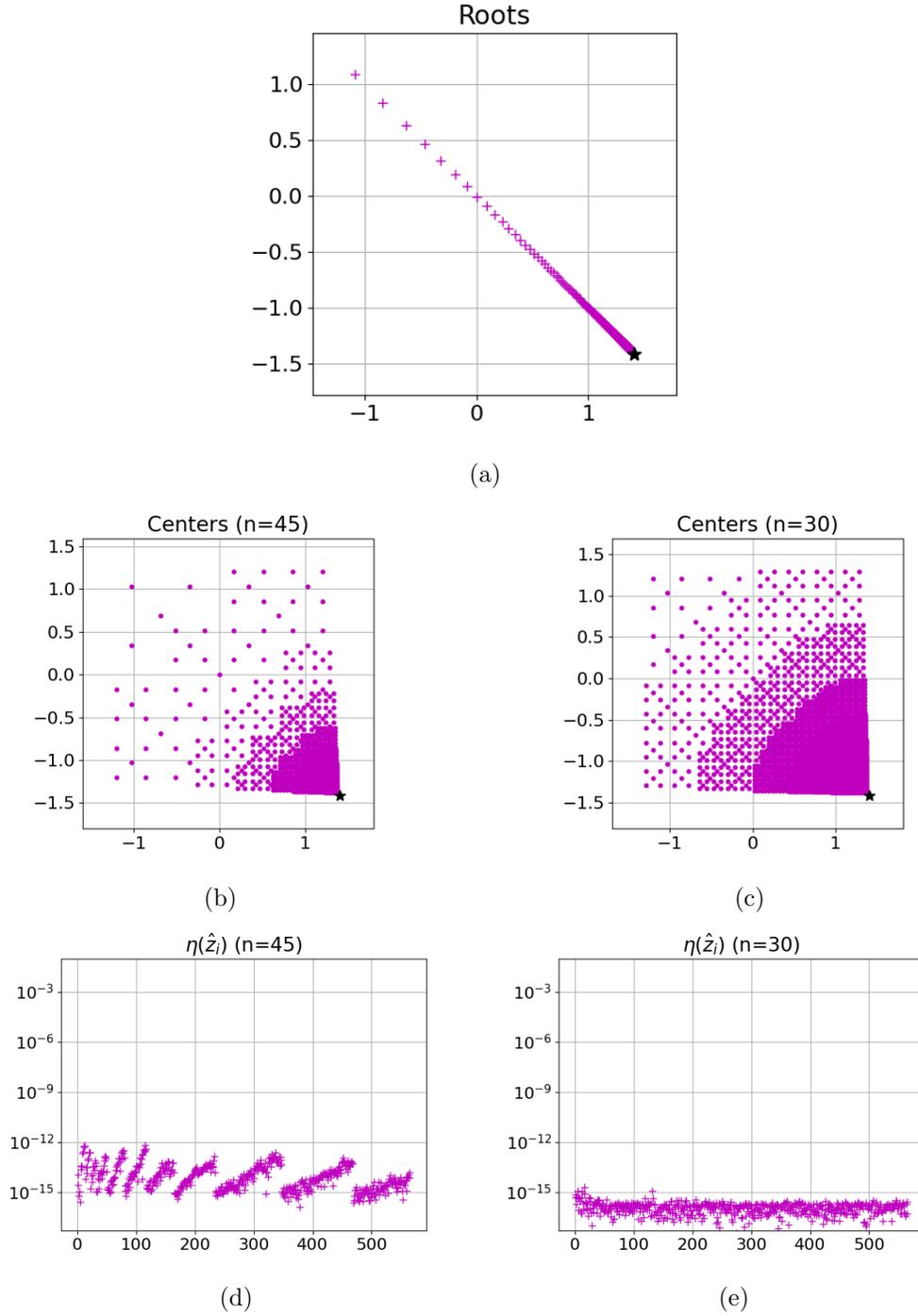
Figure 9: All roots $\widehat{z}_i$ of f_{clust} within a square of length 2.75 centered at $z_0 = 0$ found by Algorithm 6 with polynomials of order $n = 30$ are shown in (a). The centers of all squares during the recursive division with polynomials of order $n = 45$ and $n = 30$ are respectively shown in (b) and (c). The error estimations $\eta(\widehat{z}_i)$ of roots $\widehat{z}_i$ with polynomials of order $n = 45$ and $n = 30$ are shown in (d) and (e). The star $\star$ in (a)-(b) indicates the singularity location at $z_\star = \sqrt{2} - i\sqrt{2}$.

squares formed by the recursive division and the error estimation $\eta(\widehat{z}_i)$ of root $\widehat{z}_i$. The scale of function f_{entire} does not vary much across the whole region of interests, so the algorithm divides the square uniformly.

The results of our numerical experiment are shown in Table 4 (in double precision) and Table 5 (in extended precision). As can be seen from the tables, the larger the expansion order used, the larger the square on which expansions converge, thus the lower the precision of roots computed for the same reason explained in Section 6.4. The most economic approach is to combine a reasonably large expansion order n and a Newton refinement in the end.
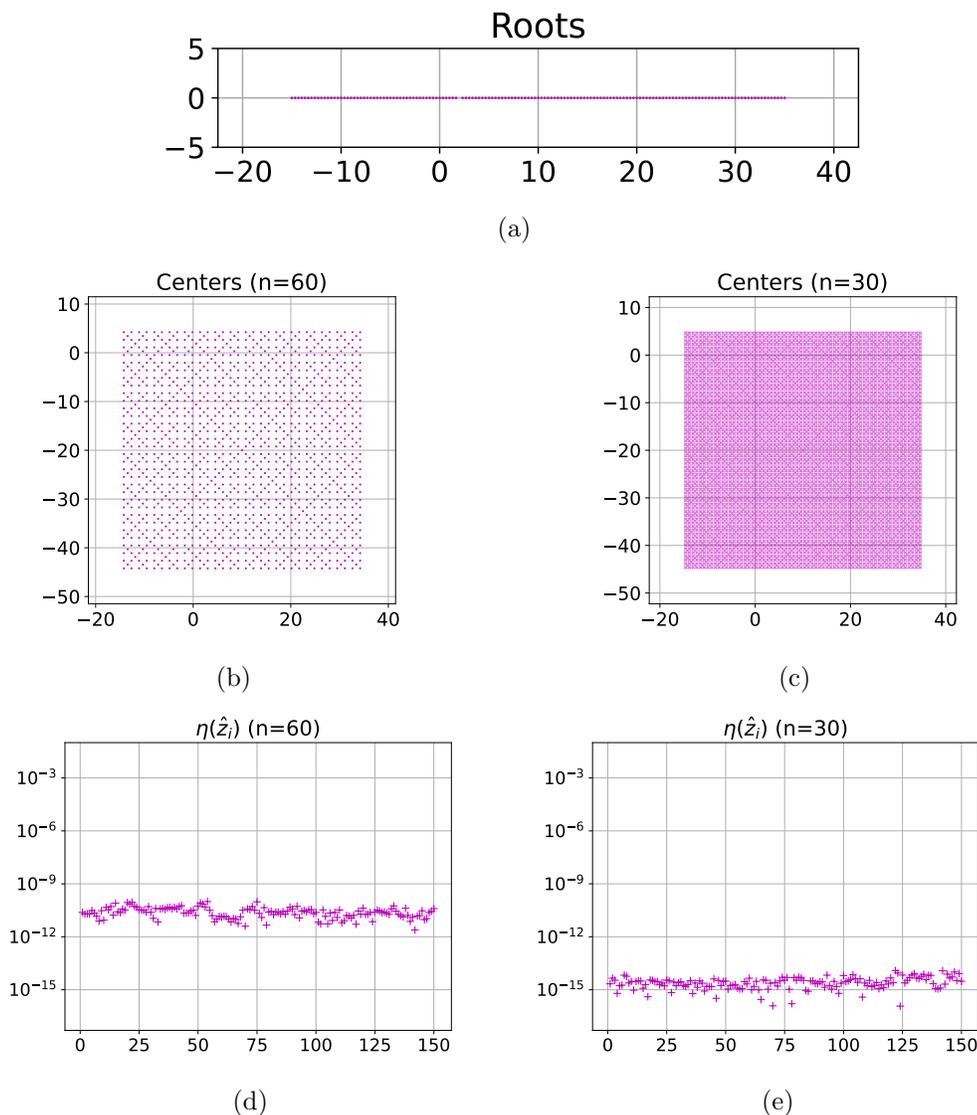

(a)

(b)   (c)

(d)   (e)

Figure 10: All roots $\widehat{z}_i$ of f_{entire} within a square of length 50 centered at $z_0 = 10 - 20i$ found by Algorithm 6 with polynomials of order $n = 30$ are shown in (a). The centers of all squares during the recursive division with polynomials of order $n = 60$ and $n = 30$ are respectively shown in (b) and (c). The error estimations $\eta(\widehat{z}_i)$ of roots $\widehat{z}_i$ with polynomials of order $n = 60$ and $n = 30$ are shown in (d) and (e).

n	n_{roots}	$\max_i \eta(\widehat{z}_i)$	n_{levels}	n_{eigs}	Timings (s)
60	150	$0.99 \cdot 10^{-10}$	6	1024	$0.56 \cdot 10^0$
30	150	$0.22 \cdot 10^{-13}$	8	16384	$0.31 \cdot 10^1$

Table 6: The results of computing roots of $f_{\text{entire}}(z)$ in double precision, using the adaptive Algorithm 6 with the polynomial expansion order $n = 60$ and $n = 30$.

n	n_{roots}	$\max_i \eta(\widehat{z}_i)$	n_{levels}	n_{eigs}	Timings (s)
100	150	$0.14 \cdot 10^{-23}$	5	256	$0.46 \cdot 10^2$
70	150	$0.35 \cdot 10^{-26}$	6	1024	$0.99 \cdot 10^2$
65	150	$0.30 \cdot 10^{-29}$	7	4096	$0.36 \cdot 10^3$

Table 7: The results of computing roots of f_{entire} in extended precision, using the adaptive Algorithm 6 with the polynomial expansion order $n = 100, 70$ and 65.

7 Conclusions

In this paper, we described a method for finding all roots of analytic functions in square domains in the complex plane, which can be viewed as a generalization of rootfinding by classical colleague matrices on the interval. This approach relies on the observation that complex orthogonalizations, defined by complex inner product with random weights, produce reasonably well-conditioned bases satisfying three-term recurrences in compact simply-connected domains. This observation is relatively insensitive to the shape of the domain and locations of points chosen for construction. We demonstrated by numerical experiments the condition numbers of constructed bases scale almost linearly with the order of the basis. When such a basis is constructed on a square domain, all roots of polynomials expanded in this basis are found as eigenvalues of a class of generalized colleague matrices. As a result, all roots of an analytic function over a square domain are found by finding those of a proxy polynomial that approximates the function to a pre-determined accuracy. Such a generalized colleague matrix is lower Hessenberg and consist of a complex symmetric part with a rank-1 update. Based on this structure, a complex orthogonal QR algorithm, which is a straightforward extension of the one in [6], is introduced for computing eigenvalues of generalized colleague matrices. The complex orthogonal QR also takes $O(n^2)$ operations to fine all roots of a polynomial of order n and exhibits componentwise backward stability, as demonstrated by numerical experiments.

Since the method we described is not limited to square domains, this work can be easily generalized to rootfindings over general compact complex domains. Then the corresponding colleague matrices will have structures identical to the one in this work, so our complex orthogonal QR can be applied similarly. However, in some cases, when the conformal map is analytic and can be computed accurately, it is easier to map the rootfinding problem to the unit disk from the original domain, so that companion matrices can be constructed and eigenvalues are computed by the algorithm in [1].

There are several problems in this work that require further study. First, the observation that complex inner products defined by random complex weights lead to well-condition polynomial basis is not understood. Second, the numerical stability of our

QR algorithm is not proved due to the use of complex orthogonal rotations. Numerical evidences about mentioned problems have been provided in this work and they are under vigorous investigation. Their analysis and proofs will appear in future works.

8 Acknowledgements

References

[1] J. L. Aurentz, T. Mach, L. Robol, R. Vandebril, and D. S. Watkins. Fast and backward stable computation of roots of polynomials, part ii: Backward error analysis; companion matrix and companion pencil. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1245–1269, 2018.

[2] J. L. Aurentz, T. Mach, R. Vandebril, and D. S. Watkins. Fast and backward stable computation of roots of polynomials. *SIAM Journal on Matrix Analysis and Applications*, 36(3):942–973, 2015.

[3] B. Craven. Complex symmetric matrices. *Journal of the Australian Mathematical Society*, 10(3-4):341–354, 1969.

[4] Y. Eidelman, L. Gemignani, and I. Gohberg. Efficient eigenvalue computation for quasiseparable hermitian matrices under low rank perturbations. *Numerical Algorithms*, 47(3):253–274, 2008.

[5] F. R. Gantmakher. *The theory of matrices*, volume 131. American Mathematical Soc., 2000.

[6] K. Serkh and V. Rokhlin. A provably componentwise backward stable $O(n^2)$ qr algorithm for the diagonalization of colleague matrices. *arXiv preprint arXiv:2102.12186*, 2021.

[7] E. M. Stein and R. Shakarchi. *Complex analysis*, volume 2. Princeton University Press, 2010.

[8] L. N. Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.