SIAM J. SCI. COMPUT.
Vol. 31, No. 1, pp. 624–643

STATISTICAL CONDITION ESTIMATION FOR THE ROOTS OF POLYNOMIALS*

A. J. LAUB† AND J. XIA‡

Abstract. This paper presents fast and reliable condition estimates for the roots of a real polynomial based on the method of statistical condition estimation (SCE) by Kenney and Laub. Using relative perturbations, we provide both a basic implementation of SCE and an advanced one via the condition of the eigenvalues of the corresponding companion matrix. New results for estimating the condition of eigenvalues are given. Fast structured calculations of the eigenvalues of the companion matrix are used, and fast structured manipulations of the Schur decomposition and the invariant subspaces of the companion matrix are presented. The overall process is based on fast operations for sequentially semiseparable structures with small off-diagonal ranks. The cost of obtaining the condition estimates for all of the roots is $O(n^2)$, where n is the degree of the polynomial. Numerical examples are used to demonstrate the accuracy and efficiency of the proposed methods.

Key words. statistical condition estimation, polynomial roots, companion matrix, sequentially semiseparable structure, structured QR iterations, invariant subspace

AMS subject classifications. 65H05, 65F15, 65F30, 65F35

DOI. 10.1137/070702242

1. Introduction. The problem of computing the roots of real polynomials numerically arises in many different applications. It is important to assess the accuracy of the numerical roots. In this paper, we are interested in the condition of the roots of real polynomials, that is, the sensitivity of the roots with respect to small perturbations in the polynomial coefficients. A lot of theoretical discussions have been given for polynomials with known roots (see, e.g., [10], [11], [20], [22], [23]).

Here, we use the method of statistical condition estimation (SCE) [16] to numerically estimate the condition of polynomial roots. The SCE method has been shown to be both reliable and efficient for many problems including linear systems [17], linear least squares problems [18], eigenvalue problems [13], matrix functions [16], control problems [14], etc. SCE provides a way to estimate the componentwise local sensitivity of any differentiable function at a given point. It is generally flexible and accommodates a wide range of perturbation types such as structured perturbations. Thus SCE often gives less conservative estimates than methods that do not respect such structure. In SCE, a small random perturbation is introduced to the input, and the change in the output, with appropriate scaling, is used to produce the condition estimate. Explicit bounds on the probability of the accuracy of the estimate exist.

The idea of SCE can be illustrated with a general real-valued function $f : \mathbb{R}^m \to \mathbb{R}$. By Taylor's theorem we have

$$f(p + \delta z) - f(p) = \delta d^T z + \mathrm{O}(\delta^2), \tag{1.1}$$

where δ is small, $\|z\|_2 = 1$, and $d^T \equiv \nabla f(p)$ is the gradient of f. It is clear that the

*Received by the editors September 7, 2007; accepted for publication (in revised form) June 19, 2008; published electronically October 31, 2008.
http://www.siam.org/journals/sisc/31-1/70224.html
†Department of Electrical Engineering and Department of Mathematics, University of California, Los Angeles, CA 90095 (laub@ucla.edu).
‡Department of Mathematics, University of California, Los Angeles, CA 90095. Current address: Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu).

following is true up to first order in δ:

$$|f(p+\delta z)-f(p)| \approx \delta\|d\|_2. \tag{1.2}$$

That is, $\|d\|_2$ can be considered as a local condition number which gives a good approximation error $\delta\|d\|_2$ in numerical computations of $f(p)$ (with δ on the order of the machine epsilon $\varepsilon_{\text{mach}}$). According to [16], if z is selected uniformly and randomly from the unit m-sphere S_{m-1} (denoted $z \in \mathrm{U}(S_{m-1})$), then the expected value $\mathrm{E}(|d^T z|/\omega_m)$ is $\|d\|_2$, where ω_m is the Wallis factor [16]. We can then use

$$\nu = \left|d^T z\right|/\omega_m$$

as a condition estimator. The accuracy of the estimate is given by the probability relationship [16]

$$\Pr(\|d\|_2/\gamma \le \nu \le \gamma\|d\|_2) \approx 1 - \frac{2}{\gamma\pi},\ \gamma > 1.$$

We can use multiple samples of z, denoted $z^{(j)}$, to increase the accuracy [16]. For example, a 2-sample condition estimator is given by $\nu^{(2)} = \frac{\omega_2}{\omega_m}\sqrt{|d^T z^{(1)}|^2 + |d^T z^{(2)}|^2}$, where $[z^{(1)}, z^{(2)}]$ is orthonormalized after $z^{(1)}$ and $z^{(2)}$ are selected uniformly and randomly from $\mathrm{U}(S_{m-1})$. The accuracy of $\nu_i^{(2)}$ is given by

$$\Pr\left(\|d\|_2/\gamma \le \nu^{(2)} \le \gamma\|d\|_2\right) \approx 1 - \frac{\pi}{4\gamma^2},\ \gamma > 1.$$

Usually, a few samples are sufficient for good accuracy. These results can be conveniently extended to vector-valued functions.

Since the roots x of a polynomial $\mathbf{p}(x)$ depend continuously on the coefficients, the general idea of SCE also applies to polynomial root problems. In a basic implementation of SCE, we perturb the nonzero coefficients (one type of structured perturbations) and solve the perturbed problem. If we consider relative perturbations in both the coefficients and the roots, we get estimates of the condition numbers in [10], [11].

To avoid any possible difficulty in choosing an appropriate amount of perturbation δ or in finding the correct correspondence between the original roots and the perturbed roots, we propose an advanced estimation scheme. In this scheme, the roots of the polynomial $\mathbf{p}(x)$ are considered as the eigenvalues of the companion matrix C corresponding to $\mathbf{p}(x)$, and the condition of the roots is transferred to the structured condition of the eigenvalues of C. We give new formulas for the condition of both single eigenvalues and complex conjugate eigenpairs.

The companion matrix is a special sparse structured matrix. This fact, firstly, leads to the development of fast eigensolvers [2], [3], [4], [8], and secondly, leads to an SCE implementation with structured perturbations. Those fast companion matrix eigensolvers cost only $\mathrm{O}(n^2)$ flops, in general, as compared with the traditional $\mathrm{O}(n^3)$ complexity for a general matrix, where n is the degree of $\mathbf{p}(x)$. The estimate of the sensitivity of average eigenvalues has been discussed in [13]. In that paper, only the average eigenvalue of the leading diagonal block of T is considered, where $C = UTU^T$ is a Schur decomposition of C. Here, for problems with distinct eigenvalues, we derive specific condition estimates for both single real eigenvalues and complex pairs of eigenvalues. What's more, to consider all of the eigenvalues, certain manipulations of T and the invariant subspaces corresponding to the eigenvalues need to be considered, or more specifically, the diagonal blocks of T may need to be swapped [9]. We provide

some theoretical results for the invariant subspaces. The manipulations of the Schur decomposition can be costly (usually, $O(n^3)$) for a general matrix. However, we can take advantage of the special structure of C and reduce the cost needed for estimating the condition of its eigenvalues to $O(n^2)$. This is possible since the matrices involved in the QR iterations for C have special rank structures and thus can be represented by some compact low-rank matrices [8]. These low-rank structures were previously used to quickly solve for the eigenvalues and can also be used here to efficiently transform U and T so that we can compute the condition estimates to the eigenvalues using the technique in [13].

The low-rank structures we use are called sequentially semiseparable (SSS) structures [6], [7]. When a matrix has small off-diagonal ranks, it can be represented by a compact SSS matrix, which makes many matrix algorithms very efficient. For example, we can solve a compact SSS linear system in linear complexity. In fact, SSS representations have been used to develop the fast eigensolvers in [2], [8].

With SSS structures, our SCE methods are very efficient. Especially, the second SCE scheme (based on the condition of eigenvalues) provides a fast and reliable way of estimating the condition of the polynomial roots. The swappings of related diagonal blocks and the solutions of related systems are all in fast structured forms. There is no need to choose the perturbation δ in SCE. What's more, for each additional sample in SCE, the extra cost is insignificant. That means we can gain high accuracy without increasing the cost much. Only real operations are involved in the estimations. Our numerical experiments indicate that the method accurately estimates the condition numbers defined in [10], [11], and it provides results which are less conservative than other condition estimators. For convenience, in this paper we focus on real polynomials with distinct (real or complex) roots. There is a good potential for similar techniques to be extended to a general real polynomial or other structured eigenvalue problems (see subsection 4.7).

We also give some theoretical results for the invariant subspaces corresponding to the eigenvalues, especially complex eigenpairs. They are used to construct certain matrices in the condition estimation.

The rest of this paper is organized as follows. In section 2 we show the condition numbers of polynomial roots and their SCE estimation. Section 3 provides the basic implementation of SCE, which gives estimates to the theoretical condition numbers. In section 4 the more advanced scheme based on SCE for the eigenvalues of companion matrices is presented. Numerical examples are shown in section 5 to demonstrate the reliability and efficiency of the estimators.

2. SCE with relative perturbations and condition of polynomial roots.

We first discuss SCE with relative perturbations, and then we show how SCE can be used to estimate the condition numbers of polynomial roots.

2.1. SCE with relative perturbations.

SCE with relative perturbations is an extension of SCE with absolute perturbations in [16]. Consider a real-valued function $f(p) : \mathbb{R}^m \to \mathbb{R}$, with $p = (p_j)$. To measure the sensitivity of the relative error in terms of the relative input, we perturb an entry p_j to $\tilde{p}_j$ and assume that the perturbed output is $\tilde{f}$. Then define

$$\kappa_j = \lim_{\tilde{p}_j \to p_j} \left| \frac{\tilde{f} - f}{f} \Big/ \frac{\tilde{p}_j - p_j}{p_j} \right|,$$

and we use

$$\kappa = \|\kappa_{1:m}\|_2$$

as the condition number. Replace (1.1) by

$$\frac{f(p+\delta Sz)-f(p)}{f(p)} = \delta\frac{d^T Sz}{f(p)} + \mathrm{O}(\delta^2), \tag{2.1}$$

where $S = \mathrm{diag}(p)$. Then the expected value $\mathrm{E}(|d^T Sz|/\omega_m)$ is $\|d^T S\|_2$. On the other hand, we notice that

$$\kappa = \frac{1}{f(p)}\left\|d^T S\right\|_2. \tag{2.2}$$

Thus, we can use

$$\nu = \frac{1}{\omega_m}\left|\frac{d^T Sz}{f(p)}\right|$$

as an estimate to the condition number (2.2). The results can be extended to vector-valued functions $f : \mathbb{R}^m \to \mathbb{R}^n$. Define

$$\kappa_{i,j} = \lim_{\tilde{p}_j \to p_j}\left|\frac{\tilde{f}_i - f_i}{f_i}\Big/\frac{\tilde{p}_j - p_j}{p_j}\right|,\ \kappa_i = \|\kappa_{i,1:m}\|_2, \tag{2.3}$$

where $\kappa \equiv (\kappa_i)$ is the condition vector. Then we can use

$$\nu_i = \frac{1}{\omega_m}\left|\frac{d_i^T Sz}{f_i(p)}\right| \tag{2.4}$$

as an estimate to the condition number κ_i of f_i, where $d_i^T \equiv \nabla f_i(p)$. Note that the original SCE mainly focuses on absolute perturbations to the input, and the corresponding condition estimator is $\frac{1}{\omega_m}\frac{|d_i^T z|}{f_i(p)}$.

2.2. Condition of polynomial roots. Consider a real polynomial

$$\mathbf{p}(x) = \sum_{k=0}^{n} a_k x^k = \prod_{i=1}^{n}(x - x_i),$$

where we assume that $a_n \equiv 1$. For simplicity, we assume that $\mathbf{p}(x)$ has distinct roots. Let f be the function given by the problem of finding the roots of $\mathbf{p}(x)$. Then $m = n$. Discussions on the condition of the roots of polynomials can be found in [11], [20], [21]. The condition number $\kappa_{i,j}$ in (2.3) can be shown to have the following form:

$$\kappa_{i,j} = \left|\frac{a_j x_i^{j-1}}{\mathbf{p}'(x_i)}\right| = \left|\frac{a_j x_i^{j-1}}{\prod_{k\neq i}(x_i - x_k)}\right|. \tag{2.5}$$

In SCE, (2.4) is one way to estimate the condition number $\kappa_i = \|\kappa_{i,1:n}\|_2$.

3. Basic SCE scheme for polynomial roots. There are two ways to estimate $\kappa_{i,j}$. One is to find the roots of a perturbed polynomial and to estimate the condition of the roots by (2.4). The other way is to consider the condition of the eigenvalues of the companion matrix corresponding to $\mathbf{p}(x)$. The second way will be discussed in the next section. In the first way, we solve for the roots x_i of $\mathbf{p}(x)$ and also the roots $\tilde{x}_i$ of the perturbed polynomial $\tilde{\mathbf{p}}(x)$, where each coefficient $\tilde{a}_j = a_j(1 + \delta z_j)$, with $z \equiv (z_j) \in \mathrm{U}(S_{n-1})$. Then an SCE condition estimator for x_i is

$$\nu_i = \frac{1}{\omega_n} \frac{|\tilde{x}_i - x_i|}{\delta |x_i|},$$

where we have dropped an $\mathrm{O}(\delta)$ term. The cost for solving a polynomial root problem is $\mathrm{O}(n^2)$ by using the fast eigensolver in [8]. This eigensolver computes the roots by finding the eigenvalues of the companion matrix corresponding to $\mathbf{p}(x)$:

$$C = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \cdots & -a_0 \\ 1 & 0 & \cdots & 0 \\ & \ddots & \ddots & \vdots \\ 0 & & 1 & 0 \end{bmatrix}. \tag{3.1}$$

Hessenberg QR iterations are used to find the eigenvalues of C. In the meantime, the eigensolver takes advantage of a low-rank property [2], [3], [8].

THEOREM 3.1. *All off-diagonal blocks of the QR iterates which are orthogonally similar to C have ranks no larger than* 3.

Based on this property, the fast eigensolver in [8] performs structured bulge chasing by representing all QR factors of the QR iterates with compact semiseparable matrices (subsection 4.2). When the QR iterates converges, we obtain a block upper-triangular matrix T with 1×1 or 2×2 diagonal blocks. The matrix T also has off-diagonal ranks no larger than 3 and can be related to C by $C = UTU^T$, where U is an accumulation of Givens rotations used in the structured QR iterations.

We summarize this SCE method in Algorithm 1, where $\mathbf{x}$ denotes the vector of the roots of $\mathbf{p}(x)$.

ALGORITHM 1 (k-sample SCE for the roots of $\mathbf{p}(x)$).

1. Solve for the roots $\mathbf{x}$ of $\mathbf{p}(x)$ using the fast eigensolver in [8].
2. Generate $z^{(1)}, \ldots, z^{(k)} \in \mathrm{U}(S_{n-1})$. Orthonormalize $[z^{(1)}, \ldots, z^{(k)}]$, say, by a QR factorization.
3. Choose δ.
4. For $j = 1, \ldots, k$,
   (a) Solve for the roots $\tilde{\mathbf{x}}$ of $\tilde{\mathbf{p}}(x)$, where $\tilde{\mathbf{p}}(x)$ has coefficients $\tilde{a}_n \equiv 1, \tilde{a}_j = a_i(1 + \delta z_i^{(j)})$.
   (b) Order $\tilde{\mathbf{x}}$ so that the perturbed roots match the original ones.
   (c) Calculate $\nu^{(j)} = |\tilde{\mathbf{x}} - \mathbf{x}|/(\delta|\mathbf{x}|))$ (componentwise division).
5. Calculate the k-sample componentwise SCE condition vector

$$\kappa_{\mathrm{SCE}} = \frac{\omega_k}{\omega_n} \sqrt{|\nu^{(1)}|^2 + \cdots + |\nu^{(k)}|^2}.$$

Remark 3.2. The potential shortcomings of this algorithm include the ambiguity in deciding the correspondence between the original roots and the perturbed ones (especially when clustered roots exist) and the difficulty in choosing the small parameter δ to get accurate results. The choice of δ is generally a complicated issue. It may

not be too small. We usually choose δ to be $\sqrt{\|\mathbf{x}\|\varepsilon_{\text{mach}}}$, where $\varepsilon_{\text{mach}}$ is the machine epsilon. This choice approximately balances the approximation error and the rounding error. See, e.g., [15, Chapter 3] for further discussions. In practice, this choice of δ has worked well in SCE implementations. On the other hand, Algorithm 2 below does not involve δ explicitly and does not need to match the roots.

4. SCE for polynomial roots via the condition of eigenvalue problems. Alternatively, for a given polynomial $\mathbf{p}(x)$, we can avoid the problem of choosing δ and consider the condition of its roots via the condition of the eigenvalues of the corresponding companion matrix (3.1).

4.1. Condition estimation for eigenvalue problems. Use structured SCE for eigenvalue problems [13] by considering the companion matrix (3.1). Assume that we have a block Schur decomposition of C:

$$C = UTU^T,\ U = [U_1, U_c],\ T = \begin{bmatrix} T_1 & H \\ 0 & T_c \end{bmatrix}, \tag{4.1}$$

where U is orthogonal and T_1 and T_c have orders n_1 and $n - n_1$, respectively. When the spectra of T_1 and T_c are well separated, the sensitivity of the average eigenvalue of T_1 is well defined. Denote the average eigenvalue by

$$\mu(T_1) = \frac{\text{trace}(T_1)}{n_1}.$$

In SCE, we introduce an $n \times n$ structured perturbation matrix

$$\delta E = \begin{bmatrix} \delta e^T \\ 0 \end{bmatrix} \begin{matrix} 1 \\ n-1 \end{matrix}, \tag{4.2}$$

where $e^T = [-a_{n-1}z_{n-1}, -a_{n-2}z_{n-2}, \ldots, -a_0 z_0]$. That is, δE is a relative perturbation matrix. As derived in [13], when δ is small enough, the average eigenvalue of T_1 is perturbed to

$$\mu\big(\tilde{T}_1\big) \approx \mu(T_1 + \delta B) = \mu(T_1) + \delta\mu(B), \tag{4.3}$$

where $B = U_1^T E U_1 + Y^T U_c^T E U_1$ and Y is an $(n - n_1) \times n_1$ matrix that satisfies a Sylvester equation

$$H = T_1 Y^T - Y^T T_c. \tag{4.4}$$

Based on (4.3), SCE provides a relative condition estimate to $\mu(T_1)$ in the following form:

$$\nu = \frac{1}{\omega_n |\mu(T_1)|} \left|\mu\left(U_1^T E U_1 + Y^T U_c^T E U_1\right)\right|. \tag{4.5}$$

Using these results, for a problem with distinct eigenvalues, we can further derive specific estimates for real single eigenvalues and complex eigenpairs.

4.1.1. Condition estimation for single real eigenvalues. When T_1 is a 1×1 block (eigenvalue), U_1 is a column vector, and we denote it by u_1. We write T_1 as t_1 and Y as y. Then (4.4) becomes a linear system

$$\left(t_1 I - T_c^T\right) y = H^T. \tag{4.6}$$

The estimator ν in (4.5) can also be simplified, since EU_1 is a multiple of the first unit vector of dimension n. That is,

$$\begin{aligned} B &= \gamma u_{11} + \gamma y^T r_c, \\ \nu &= \frac{1}{\omega_n |t_1|} |B|, \end{aligned} \tag{4.7}$$

where $\gamma = e^T u_1$, the scalar u_{11} is the first element of u_1, and r_c^T is the first row of U_c.

4.1.2. Condition estimation for complex conjugate pairs of eigenvalues. When T_1 is a 2×2 block, we need to solve the Sylvester equation (4.4). Using Kronecker products, we can write the equation as

$$\left(I_{n-2} \otimes T_1 - T_c^T \otimes I_2\right) \mathrm{vec} Y^T = \mathrm{vec} H, \tag{4.8}$$

where vec$H$ denotes the column vector formed by stacking the columns of H from left to right. The computation of $B = U_1^T E U_1 + Y^T U_c^T E U_1$ can also be simplified as

$$B = \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} \begin{bmatrix} e^T u_1 & e^T u_2 \end{bmatrix} + Y^T r_c \begin{bmatrix} e^T u_1 & e^T u_2 \end{bmatrix}, \tag{4.9}$$

where $U_1 = [u_1, u_2]$ with its first row $[u_{11}, u_{12}]$.

To get condition estimates for the eigenvalues of T_1, we assume that T_1 has a complex conjugate pair of eigenvalues λ_1 and λ_2, and that δE perturbs λ_1 and λ_2 to $\lambda_1 + \delta_1$ and $\lambda_2 + \delta_2$, respectively. That is, $\lambda_1 + \delta_1$ and $\lambda_2 + \delta_2$ are the eigenvalues of $T_1 + \delta B$ as in (4.3). Clearly, $\lambda_1 + \delta_1$ and $\lambda_2 + \delta_2$ are complex conjugate. Let

$$T_1 = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}, \; B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

We have

$$\begin{aligned} (\lambda_1 + \delta_1) + (\lambda_2 + \delta_2) &= \mathrm{trace}(T_1 + \delta B), \\ (\lambda_1 + \delta_1)(\lambda_2 + \delta_2) &= \det(T_1 + \delta B). \end{aligned}$$

Using the facts that $\lambda_1 + \lambda_2 = \mathrm{trace}(T_1)$ and $\lambda_1 \lambda_2 = \det(T_1)$, we can easily get

$$\begin{aligned} \delta_1 + \delta_2 &= \delta\alpha, \\ \lambda_2 \delta_1 + \lambda_1 \delta_2 &\approx \delta\beta, \end{aligned}$$

where we have dropped the δ^2 term in the approximation and $\alpha = b_{11} + b_{22}$, $\beta = t_{11} b_{22} + t_{22} b_{11} - t_{12} b_{21} - t_{21} b_{12}$. Clearly, we have

$$\delta_1 = \delta \frac{\alpha\lambda_1 - \beta}{\lambda_1 - \lambda_2}, \quad \delta_2 = \delta \frac{\beta - \alpha\lambda_2}{\lambda_1 - \lambda_2} \left(= \bar{\delta}_1\right),$$

which are first order approximations to the perturbations in λ_1 and λ_2, respectively. Therefore, in SCE we can use

$$\begin{aligned} \nu &\equiv \frac{1}{\omega_n |\lambda_1|} \left| \frac{\alpha\lambda_1 - \beta}{\lambda_1 - \lambda_2} \right| \left(= \frac{1}{\omega_n |\lambda_2|} \left| \frac{\beta - \alpha\lambda_2}{\lambda_1 - \lambda_2} \right| \right) \\ &= \frac{1}{\omega_n \sqrt{\det(T_1)}} \sqrt{\frac{\alpha^2 \det(T_1) - \alpha\beta\,\mathrm{trace}(T_1) + \beta^2}{[\mathrm{trace}(T_1)]^2 - 4\det(T_1)}} \end{aligned} \tag{4.10}$$

as the (relative) condition estimator for λ_1 and λ_2. We see that only real operations are involved in (4.10).

By considering the condition for the eigenvalues of C, we can avoid the difficulty of choosing δ, as can be seen from (4.7) and (4.10). However, the cost of computing Y and ν can be more than $O(n^2)$. This makes the total cost for all eigenvalues to be at least $O(n^3)$. In addition, to obtain the sensitivity for other eigenvalues, we need to swap the diagonal entries of T so as to bring a particular diagonal block to the upper left position. Noticing that C is structured and so are T and U, we can reduce the cost for each ν to $O(n)$ and thus the total cost for all eigenvalues to $O(n^2)$. We review certain matrix structures in the next subsection.

4.2. Review of SSS structures. The computations of the estimators (4.7) and (4.10) involve matrix operations such as block permutations and Sylvester equation solutions. To efficiently perform these operations, we use certain compact low-rank structures, called sequentially semiseparable, or SSS structures [6], [7]. These structures take advantage of the low-rank property of matrices such as those in (4.1). An SSS matrix looks like

$$\begin{bmatrix} \mathcal{D}_1 & \mathcal{U}_1\mathcal{V}_2^T & \mathcal{U}_1\mathcal{W}_2\mathcal{V}_3^T & \mathcal{U}_1\mathcal{W}_2\mathcal{W}_3\mathcal{V}_4^T \\ \mathcal{P}_2\mathcal{Q}_1^T & \mathcal{D}_2 & \mathcal{U}_2\mathcal{V}_3^T & \mathcal{U}_2\mathcal{W}_3\mathcal{V}_4^T \\ \mathcal{P}_3\mathcal{R}_2\mathcal{Q}_1^T & \mathcal{P}_3\mathcal{Q}_2^T & \mathcal{D}_3 & \mathcal{U}_3\mathcal{V}_4^T \\ \mathcal{P}_4\mathcal{R}_3\mathcal{R}_2\mathcal{Q}_1^T & \mathcal{P}_4\mathcal{R}_3\mathcal{Q}_2^T & \mathcal{P}_4\mathcal{Q}_3^T & \mathcal{D}_4 \end{bmatrix}, \tag{4.11}$$

where the matrices $\{\mathcal{D}_i\}, \{\mathcal{U}_i\}, \{\mathcal{W}_i\}, \{\mathcal{V}_i\}, \{\mathcal{P}_i\}, \{\mathcal{R}_i\}$, and $\{\mathcal{Q}_i\}$ are called (SSS) generators. (For convenience, we also use $\mathcal{D}_i(T)$, $\mathcal{U}_i(T)$, etc., to denote the generators of an SSS matrix T.) In the SSS form (4.11), the upper off-diagonal block numbers $1, 2$, and 3 are represented by

$$\mathcal{U}_1 \begin{pmatrix} \mathcal{V}_2^T & \mathcal{W}_2\mathcal{V}_3^T & \mathcal{W}_2\mathcal{W}_3\mathcal{V}_4^T \end{pmatrix}, \quad \begin{pmatrix} \mathcal{U}_1\mathcal{W}_2 \\ \mathcal{U}_2 \end{pmatrix} \begin{pmatrix} \mathcal{V}_3^T & \mathcal{W}_3\mathcal{V}_4^T \end{pmatrix}, \quad \begin{pmatrix} \mathcal{U}_1\mathcal{W}_2\mathcal{W}_3 \\ \mathcal{U}_2\mathcal{W}_3 \\ \mathcal{U}_3 \end{pmatrix} \mathcal{V}_4^T,$$

respectively. Thus, SSS representations can clearly reflect the off-diagonal rank structures of the matrix. An SSS matrix is said to be compact if the sizes of all $\mathcal{W}_i$ and $\mathcal{R}_i$ are small and are close to the maximum off-diagonal rank of the matrix. SSS matrices can be used to efficiently represent matrices whose off-diagonal blocks have small ranks. For these matrices, algorithms such as system solution, matrix addition, and matrix multiplication are very efficient. In fact, those algorithms often have linear complexity.

As an example, the cost of solving a compact order-n SSS system is only $O(n)$ [7]. Here, instead of reviewing the complicated general SSS solver, we briefly present a forward substitution algorithm for a lower-triangular SSS system $Ly = b$, where the lower-triangular matrix L has compact SSS representation. This is sufficient for solving related linear systems in our condition estimation for polynomial roots. We

TABLE 4.1
SSS generators of L in terms of the generators of T, where $n_{i+1} = 1$ or 2 is the order of $\mathcal{D}_{i+1}(T)$.

$\mathcal{D}_i(L)$	$\mathcal{P}_i(L)$	$\mathcal{Q}_i(L)$	$\mathcal{R}_i(L)$
$t_1 I_{n_{i+1}} - \mathcal{D}_{i+1}(T)^T$	$-v_{i+1}$	$-u_{i+1}$	$-w_{i+1}^T$

TABLE 4.2
SSS generators of $T_c^T \otimes I_2$ in terms of the generators of T_c^T, where n_i is the order of $\mathcal{D}_i(T_c^T)$ and $\mathcal{P}_i(T_c^T) = \begin{bmatrix} \mathcal{P}_{i,1}(T_c^T) \\ \mathcal{P}_{i,2}(T_c^T) \end{bmatrix}$, $\mathcal{Q}_i(T_c^T) = \begin{bmatrix} \mathcal{Q}_{i,1}(T_c^T) \\ \mathcal{Q}_{i,2}(T_c^T) \end{bmatrix}$.

n_i	$\mathcal{D}_i(T_c^T \otimes I_2)$	$\mathcal{P}_i(T_c^T \otimes I_2)$	$\mathcal{Q}_i(T_c^T \otimes I_2)$	$\mathcal{R}_i(T_c^T \otimes I_2)$
1	$\mathcal{D}_i(T_c^T) \otimes I_2$	$I_2 \otimes \mathcal{P}_i(T_c^T)$	$I_2 \otimes \mathcal{Q}_i(T_c^T)$	$I_2 \otimes \mathcal{R}_i(T_c^T)$
2	$\mathcal{D}_i(T_c^T) \otimes I_2$	$\begin{bmatrix} I_2 \otimes \mathcal{P}_{i,1}(T_c^T) \\ I_2 \otimes \mathcal{P}_{i,2}(T_c^T) \end{bmatrix}$	$\begin{bmatrix} I_2 \otimes \mathcal{Q}_{i,1}(T_c^T) \\ I_2 \otimes \mathcal{Q}_{i,2}(T_c^T) \end{bmatrix}$	$I_2 \otimes \mathcal{R}_i(T_c^T)$

assume the system to be

$$\begin{bmatrix} \mathcal{D}_1 & & & & 0 \\ \mathcal{P}_2\mathcal{Q}_1^T & \mathcal{D}_2 & & & \\ \mathcal{P}_3\mathcal{R}_2\mathcal{Q}_1^T & \mathcal{P}_3\mathcal{Q}_2^T & \mathcal{D}_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \mathcal{P}_l\mathcal{R}_{l-1}\cdots\mathcal{R}_2\mathcal{Q}_1^T & \mathcal{P}_l\mathcal{R}_{l-1}\cdots\mathcal{R}_3\mathcal{Q}_2^T & \cdots & \mathcal{P}_l\mathcal{Q}_{l-1}^T & \mathcal{D}_l \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_l \end{bmatrix}$$

$$= \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_l \end{bmatrix} - \begin{bmatrix} 0 \\ \mathcal{P}_2 \\ \mathcal{P}_3\mathcal{R}_2 \\ \vdots \\ \mathcal{P}_l\mathcal{R}_{l-1}\cdots\mathcal{R}_2 \end{bmatrix} \xi, \tag{4.12}$$

where the generators have sizes O(1) and the parameter ξ is initially a zero vector. In the ith step of the substitution process, we solve a block lower diagonal system

$$\mathcal{D}_i y_i = b_i - \mathcal{P}_i \xi$$

for y_i and update ξ by $\mathcal{R}_i\xi + \mathcal{Q}_i^T y_i$, where $\mathcal{P}_1$ is assumed to be a zero matrix. Each such step costs O(1). Thus the total cost of the solver is O(n).

4.3. Solving the Sylvester equation (4.4).

4.3.1. Solving (4.6) for single eigenvalues. When $T_1 \equiv t_1$ is a single eigenvalue, (4.4) becomes a linear system (4.6) with coefficient matrix $L = t_1 I - T_c^T$. Since T has a compact SSS form with off-diagonal ranks no larger than 3, the matrix L is also an SSS matrix with off-diagonal ranks no larger than 3, and we can easily identify its generators. See Table 4.1. Thus, we can solve (4.6) with the forward substitution algorithm in subsection 4.2.

4.3.2. Converting (4.4) to a triangular SSS system for double blocks. When T_1 is a 2×2 block, the matrix T_c^T in (4.8) is a lower-triangular SSS matrix. We can easily verify that $T_c^T \otimes I_2$ is also a lower-triangular SSS matrix. Its generators are shown in Table 4.2.

Therefore, the coefficient matrix of (4.8), $I_{n-2} \otimes T_1 - T_c^T \otimes I_2$, has the same generators as $-T_c^T \otimes I_2$ except the diagonal generator corresponding to $\mathcal{D}_i(-T_c^T \otimes I_2)$

is $T_1 - \mathcal{D}_i(T_c^T) \otimes I_2$ if $n_i = 1$, and $I_2 \otimes T_1 - \mathcal{D}_i(T_c^T) \otimes I_2$ if $n_i = 2$. Again, the forward substitution strategy in subsection 4.2 can be used to solve (4.8). The total cost is still $O(n)$.

4.4. Swapping diagonal blocks. To estimate the condition of the eigenvalues of T_i other than T_1, we can reorder the diagonal blocks of T by successively swapping adjacent diagonal blocks (see, e.g., [1], [9], [19]). In general, consider a submatrix

$$\begin{bmatrix} T_i & H_i \\ 0 & T_j \end{bmatrix},$$

where T_i and T_j have orders n_i and n_j, respectively, with $n_i, n_j = 1$ or 2, and T_i and T_j have no eigenvalue in common. As discussed in [1], [9], [19], find an orthogonal matrix G_i which is a product of one or two Givens matrices such that

$$G_i \begin{bmatrix} -X \\ I_{n_j} \end{bmatrix} = \begin{bmatrix} M_j \\ 0 \end{bmatrix}, \tag{4.13}$$

where X is the unique solution to

$$T_i X - X T_j = H_i. \tag{4.14}$$

Then

$$G_i \begin{bmatrix} T_i & H_i \\ 0 & T_j \end{bmatrix} G_i^T = \begin{bmatrix} M_j T_j M_j^{-1} & \bar{H}_i \\ 0 & M_i T_i M_i^{-1} \end{bmatrix},$$

where M_i is the bottom $n_j \times (n_i + n_j)$ submatrix of $G_i \begin{bmatrix} I_{n_i} \\ 0 \end{bmatrix}$. Thus T_i and T_j have been swapped. In order to bring T_i to the $(1,1)$ block position, we need $i-1$ orthogonal matrices $G_k, k = i-1, \ldots, 2, 1$. Each G_k is used to swap T_k with T_{k+1}. Each G_k is applied to the kth and $(k+1)$st block rows of T, and G_k^T is applied to the kth and $(k+1)$st block columns of T. Since T is in SSS form, the application of G_k to T can be very efficient. However, since it is usually not straightforward to get a new SSS form for T after each G_k is applied, we simply use the SSS form of T to compute all G_k and then combine all of these transformation matrices into another SSS matrix. Then we find the SSS form of the product of three SSS matrices

$$\tilde{T} = GTG^T$$

using the SSS multiplication formulas in [5], [8].

The detailed procedure is as follows. We consider T in the following form:

$$T = \begin{bmatrix} \cdots & \cdots & u_1 w_2 \cdots w_{k-1} v_k^T & u_1 w_2 \cdots w_k v_{k+1}^T & \cdots \\ \ddots & \vdots & \vdots & \vdots & \\ & T_{k-1} & H_k & u_{k-1} w_k v_{k+1}^T & \vdots \\ & & T_k & H_{k+1} & \\ & & & T_{k+1} & \\ 0 & & & & \ddots \end{bmatrix},$$

where $H_k = u_{k-1} v_k^T$, $H_{k+1} = u_k v_{k+1}^T$, and each T_k has order n_k. (For notational convenience, u_1 and u_2 are used in the SSS representation of T in this subsection and

should not be confused with those anywhere else.) A transformation matrix G_k can be computed for $\left[\begin{smallmatrix} T_k & H_{k+1} \\ 0 & T_{k+1} \end{smallmatrix}\right]$ as discussed in (4.13)–(4.14).

Apply G_k to the kth and $(k+1)$st block rows of T. It is sufficient to update the following blocks:

$$G_k \begin{bmatrix} T_k & H_{k+1} \\ 0 & T_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{T}_k & \hat{H}_k \\ \hat{H}'_k & \hat{T}_{k+1} \end{bmatrix}.$$

Then apply G_k^T to the kth and $(k+1)$st block columns of T:

$$\begin{bmatrix} u_1w_2\cdots w_{k-1}v_k^T & u_1w_2\cdots w_kv_{k+1}^T \\ \vdots & \vdots \\ H_k & u_{k-1}w_kv_{k+1}^T \\ \hat{T}_k & \hat{H}_k \\ \hat{H}'_k & \hat{T}_{k+1} \end{bmatrix} G_k^T = \begin{bmatrix} \begin{bmatrix} u_1w_2\cdots w_{k-1} \\ \vdots \\ u_{k-2}w_{k-1} \\ u_{k-1} \end{bmatrix} \begin{bmatrix} v_k^T & w_kv_{k+1}^T \end{bmatrix} G_k^T \\ \begin{bmatrix} \tilde{T}_k & \tilde{H}_{k+1} \\ 0 & \tilde{T}_{k+1} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} u_1w_2\cdots w_{k-1} \\ \vdots \\ u_{k-2}w_{k-1} \\ u_{k-1} \end{bmatrix} \begin{bmatrix} \hat{v}_k^T & \hat{v}_{k+1}^T \end{bmatrix} \\ \begin{bmatrix} \tilde{T}_k & \tilde{H}_{k+1} \\ 0 & \tilde{T}_{k+1} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} u_1w_2\cdots w_{k-1}\hat{v}_k^T & u_1w_2\cdots w_k\hat{v}_{k+1}^T \\ \vdots & \vdots \\ \hat{H}_k & u_{k-1}w_k\hat{v}_{k+1}^T \\ \tilde{T}_k & \tilde{H}_{k+1} \\ 0 & \tilde{T}_{k+1} \end{bmatrix},$$

where $\left[\begin{smallmatrix} \hat{v}_k \\ \hat{v}_{k+1} \end{smallmatrix}\right]_{n_k}^{n_{k+1}} = G_k \left[\begin{smallmatrix} v_k \\ v_{k+1}w_k^T \end{smallmatrix}\right]$, $\hat{H}_k = u_{k-1}^T\hat{v}_k$. Note that n_{k+1} is now the order of $\tilde{T}_k$ and the row dimension of $\hat{v}_k$. That is, we have swapped the diagonal blocks via the SSS generators.

Next, we can compute a transformation G_{k-1} based on the matrix $\left[\begin{smallmatrix} T_{k-1} & \hat{H}_k \\ 0 & \tilde{T}_k \end{smallmatrix}\right]$, and the process repeats. We can see that, to obtain the transformation matrices G_k, we need only to update v_k, H_k, T_k, and T_{k+1}.

Therefore, in order to bring T_i to the upper left position (possibly with similarity transformations), we compute $G_k, k = i-1, \ldots, 2, 1$ with the above procedure. The total cost is no more than $\mathrm{O}(n)$. All of the matrices G_k are then combined into a matrix G, which turns out to have small off-diagonal ranks also. We can conveniently write G in a compact SSS form. There are different situations based on the sizes of the diagonal blocks of T.

4.4.1. Single past single or single past double. Assume that T_i is a scalar and has row or column index ι in T. Then in any swapping process for $\left[\begin{smallmatrix} T_k & H_{k+1} \\ 0 & T_{k+1} \end{smallmatrix}\right]$, the block T_{k+1} is always a scalar.

If T_k is also a scalar, then G_k is simply a Givens rotation matrix

$$G_k = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix},$$

where j is the row or column index of the upper left entry of T_k in T. If T_k is a 2×2 block, then according to (4.13), G_k has the form

$$G_k = \begin{bmatrix} c_j & s_j & 0 \\ -s_j & c_j & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{j+1} & s_{j+1} \\ 0 & -s_{j+1} & c_{j+1} \end{bmatrix}.$$

TABLE 4.3
SSS generators of G.

$\mathcal{D}_i(G)$	$\mathcal{U}_i(G)$	$\mathcal{V}_i(G)$	$\mathcal{W}_i(G)$	$\mathcal{P}_i(G)$	$\mathcal{Q}_i(G)$	$\mathcal{R}_i(G)$
$c_{i-1}c_i$	$c_{i-1}s_i$	c_i	s_i	1	$-s_i$	0

It is clear that we can assemble all G_k into the matrix

$$G = Q_1 Q_2 \cdots Q_\iota,$$

where

$$Q_j = \operatorname{diag}\left(I_{j-1}, \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix}, I_{n-j-1}\right).$$

The matrix G has the following form:

$$G = \begin{pmatrix} c_0c_1 & c_0s_1c_2 & c_0s_1s_2c_3 & \cdots & \cdots & c_0s_1\cdots s_\iota c_{\iota+1} \\ -s_1 & c_1c_2 & c_1s_2c_3 & \cdots & \cdots & c_1s_2\cdots s_\iota c_{\iota+1} \\ & -s_2 & c_2c_3 & \cdots & \cdots & c_2s_3\cdots s_\iota c_{\iota+1} \\ & & \ddots & \ddots & \vdots & \vdots \\ & & & -s_{\iota-1} & c_{\iota-1}c_\iota & c_{\iota-1}s_\iota c_{\iota+1} \\ 0 & & & & -s_\iota & c_\iota c_{\iota+1} \end{pmatrix}, \tag{4.15}$$

where $c_0 = c_n = 1$. Clearly, G is an orthogonal matrix with the ranks of both its upper and lower off-diagonal blocks bounded by 1. This can be verified by examining the off-diagonal blocks, or by the CS decomposition [12, section 2.6]. Obviously, G is also an SSS matrix with generators as shown in Table 4.3 (see also [8]).

Therefore, after all of the swapping steps, the matrix T is transformed into a new matrix $\tilde{T} = GTG^T$, which is a product of three compact SSS matrices. By Theorem 3.1 and the fact that the off-diagonal ranks of G are bounded by 1, the matrix $\tilde{T}$ is a matrix with off-diagonal ranks at most 5. The formulas in [5], [8] for the product of SSS matrices can be used to provide a compact SSS form for $\tilde{T}$.

4.4.2. Double past single or double past double. Assume that T_i is a 2×2 block, and its $(1,1)$ entry has row or column index ι in T. Then in any swapping process for $\left[\begin{smallmatrix} T_k & H_{k+1} \\ 0 & T_{k+1} \end{smallmatrix}\right]$, the block T_{k+1} is always a 2×2 block.

If T_k is a scalar, then according to (4.13), G_k has the form

$$G_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{j+1} & s_{j+1} \\ 0 & -s_{j+1} & c_{j+1} \end{bmatrix} \begin{bmatrix} \hat{c}_j & \hat{s}_j & 0 \\ -\hat{s}_j & \hat{c}_j & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.16}$$

where j is the row or column index of the upper left entry of T_k in T. If T_k is a 2×2 block, then

$$\begin{aligned} G_k = &\begin{bmatrix} 1 & & & 0 \\ & c_{j+1} & s_{j+1} & \\ & -s_{j+1} & c_{j+1} & \\ 0 & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & c_{j+2} & s_{j+2} \\ 0 & & -s_{j+2} & c_{j+2} \end{bmatrix} \\ &\times \begin{bmatrix} \hat{c}_j & \hat{s}_j & & 0 \\ -\hat{s}_j & \hat{c}_j & & \\ & & 1 & \\ 0 & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ & \hat{c}_{j+1} & \hat{s}_{j+1} & \\ & -\hat{s}_{j+1} & \hat{c}_{j+1} & \\ 0 & & & 1 \end{bmatrix}. \end{aligned} \tag{4.17}$$

It is clear that we can assemble all G_k into the matrix

$$G = Q \cdot \hat{Q} \equiv (Q_2 Q_3 \cdots Q_{\iota+1}) \cdot (\hat{Q}_1 \hat{Q}_2 \cdots \hat{Q}_\iota),$$

where

$$Q_j = \operatorname{diag}\left(I_{j-1}, \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix}, I_{n-j-1}\right), \ \hat{Q}_j = \operatorname{diag}\left(I_{j-1}, \begin{bmatrix} \hat{c}_j & \hat{s}_j \\ -\hat{s}_j & \hat{c}_j \end{bmatrix}, I_{n-j-1}\right).$$

This result uses the fact that Q_j (or $\hat{Q}_j$) and Q_k (or $\hat{Q}_k$) commute when $|j-k| > 1$.

As an example, we assemble two matrices G_{k-1} and G_k, where G_k has the form (4.17), written as $Q_{j+1}Q_{j+2}\hat{Q}_j\hat{Q}_{j+1}$, and G_{k-1} has the form (4.16) but with k replaced by $k-1$ and j replaced by $j-1$, written as $Q_j\hat{Q}_{j-1}$. The resulting matrix is

$$(Q_j\hat{Q}_{j-1}) \cdot (Q_{j+1}Q_{j+2}\hat{Q}_j\hat{Q}_{j+1}) = (Q_jQ_{j+1}Q_{j+2}) \cdot (\hat{Q}_{j-1}\hat{Q}_j\hat{Q}_{j+1}),$$

where we have used the fact that $\hat{Q}_{j-1}$ and $(Q_{j+1}Q_{j+2})$ commute.

We can see that both Q and $\hat{Q}$ have forms similar to (4.15). Thus, the orthogonal matrix G is a product of two SSS matrices, and, according to the SSS multiplication formulas in [5], [8], the off-diagonal blocks of G have ranks no larger than 2. Again, we can use the SSS multiplication formulas to obtain $\tilde{T} = GTG^T$.

4.5. Computing the first row of U. Denote the first row of U by r^T. Note that the fast eigensolver in [8] provides U in the form of a sequence of 2×2 Givens rotation matrices. The total number of these Givens matrices is $\mathrm{O}(n^2)$. Thus, the application of these matrices on the right to e_1^T, the first unit vector of length n, yields the initial r^T. This costs $\mathrm{O}(n^2)$ operations. Later, for each diagonal block T_i, the vector r^T needs to be updated when T is updated by the swapping process. According to the previous subsection, U is updated to UG^T; thus the updated vector

$$\tilde{r}^T = r^T G^T. \tag{4.18}$$

Since G^T is represented by no more than $2n$ Givens rotation matrices, the computation of $\tilde{r}^T$ for each T_i costs no more than $\mathrm{O}(n)$.

In addition, the inner product of r^T with another vector as in (4.7) costs $\mathrm{O}(n)$.

4.6. Computing U_1. If $T_1 \equiv \lambda$ is a single entry (eigenvalue), then $U_1 \equiv u_1$ is a corresponding eigenvector. It is clear that we can choose

$$u_1 = \tilde{u}_1/\|\tilde{u}_1\|_2, \text{ with } \tilde{u}_1 = (\lambda^{n-1}, \lambda^{n-2}, \ldots, \lambda, 1)^T. \tag{4.19}$$

In the meantime, we need to make sure that the first entry of u_1 matches the first entry of r^T, or, more specifically, we need to match the signs.

If T_1 is a 2×2 block, then U_1 provides an orthonormal basis for the invariant subspace of C corresponding to the eigenvalues contained in T_1. That is,

$$CU_1 = U_1T_1.$$

Assume that the two eigenvalues of T_1 are $\lambda_1, \lambda_2 = \rho(\cos\theta \pm i\sin\theta)$. Since λ_1 and λ_2 have the corresponding eigenvectors $u_1 = (\lambda_1^{n-1}, \ldots, \lambda_1, 1)^T$ and $u_2 = (\lambda_2^{n-1}, \ldots, \lambda_2, 1)^T$, respectively, we can get the following relationship:

$$C\tilde{U}_1 = \tilde{U}_1\tilde{T}_1, \tag{4.20}$$

where

$$\tilde{U}_1 = \begin{bmatrix} \rho^{n-1}\cos(n-1)\theta & \rho^{n-1}\sin(n-1)\theta \\ \vdots & \vdots \\ \rho^2\cos 2\theta & \rho^2\sin 2\theta \\ \rho\cos\theta & \rho\sin\theta \\ 1 & 0 \end{bmatrix}, \ \tilde{T}_1 = \rho \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \tag{4.21}$$

We know the first row of U_1 (from the previous subsection) and can find U_1 based on $\tilde{U}_1$.

LEMMA 4.1. *For the matrices T_1 in (4.1) and $\tilde{T}_1$ in (4.21), which are both 2×2, there exists a similarity transformation matrix F_1 such that $\tilde{T}_1 F_1 = F_1 T_1$. One such example is*

$$F_1 = \begin{bmatrix} 0 & \rho\sin\theta \\ t_{21} & \rho\cos\theta - t_{11} \end{bmatrix}, \tag{4.22}$$

where $[t_{11}, t_{21}]^T$ is the first column of T_1.

This result can be easily verified using the fact that T_1 and $\tilde{T}_1$ have the same eigenvalues, and thus the same traces and determinants. The result enables us to write (4.20) as

$$C\hat{U}_1 = \hat{U}_1 T_1,$$

where

$$\hat{U}_1 = \tilde{U}_1 F_1. \tag{4.23}$$

The matrix F_1 such as (4.22) is invertible, since $\rho\sin\theta \neq 0$, $t_{21} \neq 0$ due to the fact that T_1 has a pair of complex eigenvalues.

LEMMA 4.2. *For the matrices U_1 in (4.1) and $\hat{U}_1$ in (4.23), which are both $n \times 2$, there exists a matrix F_2 such that $\hat{U}_1 F_2 = U_1$.*

Proof. Given r_1^T (the first row of U_1) and $\hat{r}_1^T \equiv \rho^{n-1}[\cos(n-1)\theta, \sin(n-1)\theta]F_1$ (the first row of $\hat{U}_1$), we can find such a matrix F_2. Clearly, F_2 satisfies

$$\hat{r}_1^T F_2 = r_1^T, \ T_1 F_2 = F_2 T_1. \tag{4.24}$$

Since $\lambda_1, \lambda_2 \equiv \rho(\cos\theta \pm i\sin\theta) \neq 0$ and F_1 is invertible, we have $\hat{r}_1^T \neq 0$. Thus, $\hat{r}_1^T \equiv (\hat{r}_{11}, \hat{r}_{12}) \neq 0$.

If $\hat{r}_{11} \neq 0$, we can construct a Sylvester equation based on (4.24):

$$\left(T_1 + \begin{bmatrix} \hat{r}_1^T \\ 0 \end{bmatrix}\right) F_2 - F_2 T_1 = \begin{bmatrix} r_1^T \\ 0 \end{bmatrix}. \tag{4.25}$$

Since $\hat{r}_{11} \neq 0$, we have that $T_1 + \left[\begin{smallmatrix} \hat{r}_1^T \\ 0 \end{smallmatrix}\right]$ and T_1 have different traces. These two matrices are also 2×2. Thus they have different eigenvalues. The above equation then has a unique solution F_2. Similarly, if $\hat{r}_{12} \neq 0$, we can solve a different Sylvester equation

$$\left(T_1 + \begin{bmatrix} 0 \\ \hat{r}_1^T \end{bmatrix}\right) F_2 - F_2 T_1 = \begin{bmatrix} 0 \\ r_1^T \end{bmatrix}. \tag{4.26}$$

∎

This proof gives a way to compute such a matrix F_2.

THEOREM 4.3. *Assume that the real matrix C in* (3.1) *has a Schur decomposition as in* (4.1)*, where the leading block T_1 of T is 2×2 and has a complex pair of eigenvalues. Then there exists a unique U_1 with two linearly independent columns and with the first row given such that $CU_1 = U_1T_1$.*

Proof. Assume that there exists another matrix U_1' with two linearly independent columns and with the first row r_1^T being the first row of U_1 such that $CU_1' = U_1'T_1$. Since both U_1 and U_1' are bases for the invariant subspace of C corresponding to the eigenvalues λ_1, λ_2 of T_1, there exists a matrix R such that $U_1' = U_1R$, and

$$r_1^T = r_1^T R. \tag{4.27}$$

Thus,

$$CU_1R = U_1RT_1 = U_1T_1R,$$

which leads to

$$RT_1 = T_1R.$$

Multiplying by r_1^T on the left and using (4.27), we have

$$r_1^T T_1 = r_1^T T_1 R. \tag{4.28}$$

Equations (4.27) and (4.28) indicate that R has an eigenvalue 1 with the corresponding left eigenvectors r_1^T and $r_1^T T_1$. Clearly, r_1^T and $r_1^T T_1$ are linearly independent, since T_1 has a complex pair of eigenvalues. Therefore, $R = I$. ∎

In particular, here U_1 has two orthonormal columns. The previous results give us a procedure for computing U_1 from $\tilde{U}_1$. That is, we first get F_1 in (4.22) and thus $\hat{U}_1$. Then compute F_2 from (4.25) or (4.26). Finally,

$$U_1 = \hat{U}_1F_2 = \tilde{U}_1F_1F_2, \tag{4.29}$$

which costs $O(n)$ with direct computations since F_1 and F_2 are 2×2 matrices.

4.7. Implementation issues and the algorithm. Before presenting the algorithm, we address some implementation issues.

When n is large, a small $|\lambda|$ in (4.19) can make the first entry of r^T small. Thus, it may not be reliable to decide the sign of the first entry of u_1 based on the first entry of r^T. To avoid this problem, we can introduce an auxiliary vector b^T, which is the last row of U. When $|\lambda|$ is small, the first entry of b^T is used to decide the sign of the last entry of u_1 and thus the signs of other entries in u_1.

Similarly, if $|\rho|$ is small, the matrix F_2 computed from (4.25) or (4.26) may be inaccurate. This is because the nonzero entry (or entries) in $\hat{r}_1^T$ is small and the coefficient matrices of F_2 in (4.25) or (4.26) are close to each other. Again, we can use the vector b^T and replace r_1^T in (4.25) and (4.26) by the first two entries of b^T.

So far, we have concentrated on distinct roots. If $\mathbf{p}(x)$ has multiple roots, the previous steps can be simplified, since some swapping steps can be omitted. We also need to modify the condition estimators for multiple roots. Extensions to other structured eigenvalue problems are also possible. More details are expected to appear in future work.

We now summarize the major steps in an algorithm. The complexity of the algorithm is $O(n^2)$, which can be easily verified.

ALGORITHM 2 (k-sample SCE for the roots of $\mathbf{p}(x)$ via the condition of a structured eigenvalue problem).

1. Use the fast eigensolver in [8] to find a Schur decomposition (4.1) for the companion matrix C corresponding to $\mathbf{p}(x)$. Store the SSS generators of T and the first row r^T of U.
2. Generate $z^{(1)}, \ldots, z^{(k)} \in \mathrm{U}(S_{n-1})$. Orthonormalize $[z^{(1)}, \ldots, z^{(k)}]$.
3. For each diagonal block T_i of T,
   (a) If T_i is a single entry,
      i. update T as in subsection 4.4.1 and update r^T as in subsection 4.5;
      ii. solve a lower block triangular SSS system (4.6);
      iii. compute u_1 via (4.19);
      iv. for each $j = 1, \ldots, k$, set $e = (e_i)$ with $e_i = a_i z_i^{(j)}$. Calculate an estimate $\nu_i^{(j)}$ by (4.5);
   (b) Else
      i. update T as in subsection 4.4.2 and update r^T as in subsection 4.5;
      ii. solve a lower block triangular SSS system (4.8);
      iii. compute U_1 via (4.29);
      iv. for each $j = 1, \ldots, k$, set e to be $-a \cdot z^{(j)}$ (componentwise multiplication); calculate an estimate $\nu_i^{(j)}$ by (4.10) using (4.9).
4. Calculate the k-sample componentwise SCE condition vector

$$\kappa_{\mathrm{SCE}} = \frac{\omega_k}{\omega_n} \sqrt{|\nu^{(1)}|^2 + \cdots + |\nu^{(k)}|^2}.$$

5. Numerical experiments. Algorithms 1 and 2 both give satisfactory results for many problems. Algorithm 1 solves two polynomial root problems and is generally straightforward. When solving the perturbed problem, we can use the original roots as the shifts in the structured QR iterations. For each new sample, we need to solve an additional perturbed problem. If each solve costs αn^2, then a k-sample SCE estimate costs about $k\alpha n^2$. The algorithm also has potential shortcomings as mentioned in Remark 3.2.

On the other hand, Algorithm 2 avoids the potential problems of Algorithm 1. There is no δ explicitly involved in the algorithm. We do not need to explicitly solve a perturbed problem or to order the roots. In addition, it can be very efficient for multiple samples, since each additional sample requires about one or two vector inner products (see (4.9)) near the final stage of the condition estimation. Most work is independent of the random perturbations (we call this the precomputation). This makes the total cost of a k-sample SCE estimate with algorithm 2 to be $\hat{\alpha}n^2 + 2kn^2$, where the $\hat{\alpha}n^2$ is the cost of the precomputation. Since $\hat{\alpha}$ is usually much larger than 1, the cost of each additional sample of estimation is insignificant.

We apply the algorithms to some classical examples. The following notations are used in the results:

$\hat{\kappa}_{\mathrm{SCE}}$	SCE by Algorithm 1 (with relative perturbations)
κ_{SCE}	SCE by Algorithm 2 (with relative perturbations)
κ_{exact}	Exact condition numbers by (2.5)
κ_{MATLAB}	Condition estimates by MATLAB function `condeig`

Example 1. Consider Wilkinson's polynomial

$$\mathbf{p}(x) = (x-1)(x-2)\cdots(x-n),$$

where $n = 8$. The roots are $x_j = j,\ j = 1, \ldots, n$.

TABLE 5.1
SCE for Example 1.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
$\hat{\kappa}_{\text{SCE}}$	1.83e1	2.26e2	1.96e3	6.66e3	1.53e4	2.14e4	1.50e4	4.08e3
κ_{SCE}	1.83e1	2.26e2	1.96e3	6.66e3	1.53e4	2.14e4	1.50e4	4.07e3
κ_{exact}	3.58e1	5.87e2	4.21e3	1.57e4	3.28e4	3.85e4	2.37e4	5.97e3

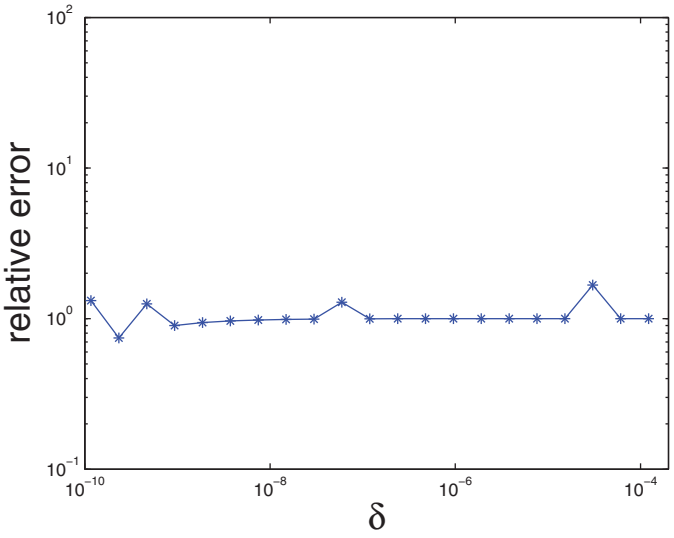

FIG. 5.1. *Relative errors* $\max_j |\frac{\hat{\kappa}_{\text{SCE}}-\kappa_{\text{exact}}}{\kappa_{\text{exact}}}|$ *by Algorithm* 1 *with different* δ *for Example* 1.

The exact condition number for x_j is $\frac{(j+n)!-j^n j!}{(j!)^2(n-j)!}$ [11]. Choose $\delta = 10^{-8}$ in Algorithm 1. Apply SCE with two samples. The same random perturbations $z^{(1)}, \ldots, z^{(k)}$ are used for both SCE algorithms. The condition numbers are in Table 5.1. Both Algorithms 1 and 2 provide very similar results.

For Algorithm 1, we also calculate the relative errors $\max_j |\frac{\hat{\kappa}_{\text{SCE}}-\kappa_{\text{exact}}}{\kappa_{\text{exact}}}|$, with different choices of δ ranging from $2^{-33}(\approx 1.2 \times 10^{-10})$ to $2^{-13}(\approx 1.2 \times 10^{-4})$ so as to demonstrate the sensitivity of the condition estimates in terms of δ. See Figure 5.1. We see that the errors are close to 1 or smaller, and the estimates are pretty accurate. The condition estimate is insensitive to δ over a relatively large range around $\sqrt{\|\mathbf{x}\|\varepsilon_{\text{mach}}}$ (see Remark 3.2).

Example 2. Consider

$$\mathbf{p}(x) = \left(x - 2^{-1}\right)\left(x - 2^{-2}\right)\cdots(x - 2^{-n}).$$

The roots are $x_j = 2^{-j},\ j = 1, \ldots, n$.

The exact condition numbers are bounded by $2\prod_{j=1}^{+\infty}(\frac{1+2^{-j}}{1-2^{-j}})^2 \approx 136.32$ [11]. Since both Algorithms 1 and 2 provide very similar results, we only report one set. For $n = 8$, the condition numbers with two samples in SCE are in Table 5.2.

On the other hand, the MATLAB function `condeig` applied to the companion matrix reports large condition numbers for the roots. The condition estimates κ_{MATLAB} are included to roughly show that our algorithms give less conservative results, although it should be emphasized that `condeig` measures a different condition number

TABLE 5.2
SCE for Example 1.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
κ_{SCE}	5.63	14.4	31.8	37.1	11.3	17.6	17.0	7.84
κ_{exact}	8.31	24.8	39.2	46.8	46.8	39.2	25.1	8.99
κ_{MATLAB}	5.68E2	7.24E4	3.18E6	5.81E7	4.73E8	1.72E9	2.64E9	1.33E9

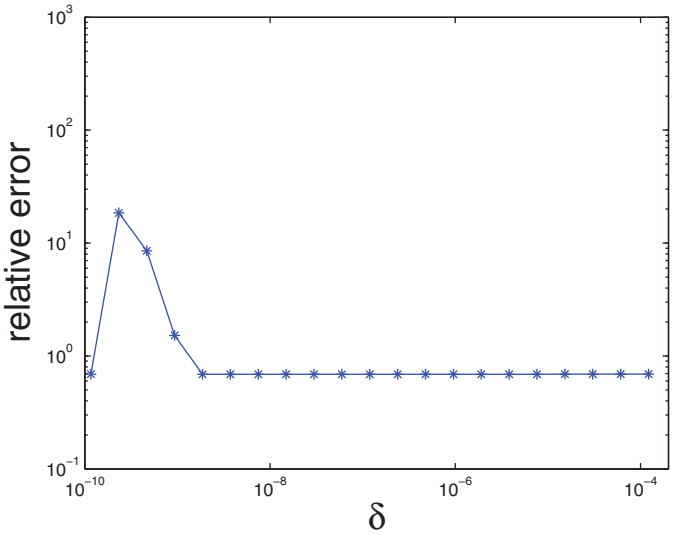

FIG. 5.2. *Relative errors* $\max_j |\frac{\hat{\kappa}_{\text{SCE}}-\kappa_{\text{exact}}}{\kappa_{\text{exact}}}|$ *by Algorithm* 1 *with different* δ *for Example* 2.

and also ignores the structure of the companion matrix. It is generally an $O(n^3)$ process.

The relative errors in the condition estimates by Algorithm 1 with different choices of δ are shown in Figure 5.2. In this example, again, the estimate is very accurate at around $\delta = \sqrt{\|\mathbf{x}\|\varepsilon_{\text{mach}}}$. In both this example and the previous one, the error gets large only when δ is too small or too large. Again, we point out that Algorithm 2 does not have the problem of choosing an appropriate δ.

Example 3. Consider the polynomial

$$\mathbf{p}(x) = x^n - 1.$$

The solutions are the roots of unity $x_i = e^{i2\pi j/n}$, $j = 1, \ldots, n$.

As mentioned in [11], $\kappa_j = \frac{1}{n}$, $j = 1, \ldots, n$ in (2.5). For $n = 8$, applying SCE with two samples, we get an estimate of $\kappa_{\text{SCE}} = 0.133$ compared to the exact condition number $\kappa_{\text{exact}} = 0.125$.

We also apply the 1-sample Algorithm 2 to $\mathbf{p}(x)$ with larger degrees n. For each n, we count the number of floating point operations for the main condition estimation step 2 in Algorithm 2 (denoted by flops_n) and report $\frac{\text{flops}_n}{\text{flops}_{n/2}}$ in the last row of Table 5.3. The ratios $\frac{\text{flops}_n}{\text{flops}_{n/2}}$ are close to 4, which is consistent with the $O(n^2)$ complexity of the algorithm.

In the current preliminary MATLAB implementation of the algorithms, the condition estimation step 2 in Algorithm 2 (mainly the precomputation) takes more time than the root finding step, although they are of the same order. However, each addi-

TABLE 5.3
SCE for Example 3 with different n.

n	32	64	128	256	512	1024
κ_{SCE}	0.0587	0.0212	0.0129	0.0055	0.0023	0.0010
κ_{exact}	0.0313	0.0156	0.0078	0.0039	0.0020	0.0010
$\frac{\text{flops}_n}{\text{flops}_{n/2}}$	4.52	4.23	4.11	4.05	4.03	4.01

tional sample costs little as compared with the precomputation. In addition, although it is possible to test the first two examples with large degrees, the calculations of the coefficients or the roots may be inaccurate.

Acknowledgments. The authors are very grateful to the editor and the referees for their valuable comments.

REFERENCES

[1] Z. Bai, J. W. Demmel, and A. McKenney, *On computing condition numbers for the nonsymmetric eigenproblem*, ACM Trans. Math. Software, 19 (1993), pp. 202–223.
[2] D. Bindel, S. Chandresekaran, J. Demmel, D. Garmire, and M. Gu, *A Fast and Stable Nonsymmetric Eigensolver for Certain Structured Matrices*, Technical report, University of California, Berkeley, CA, 2005.
[3] D. A. Bini, F. Daddi, and L. Gemignani, *On the shifted QR iteration applied to companion matrices*, Electron. Trans. Numer. Anal., 18 (2004), pp. 137–152.
[4] D. A. Bini, Y. Eidelman, L. Gemignani, and I. Gohberg, *Fast QR Eigenvalue Algorithms for Hessenberg Matrices Which are Rank-one Perturbations of Unitary Matrices*, Technical report 1587, Department of Mathematics, University of Pisa, Pisa, Italy, 2005.
[5] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Fast Stable Solvers for Sequentially Semi-separable Linear Systems of Equations and Least Squares Problems*, Technical report, University of California, Berkeley, CA, 2003.
[6] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.
[7] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen, *Fast stable solver for sequentially semi-separable linear systems of equations*, in High Performance Computing-HiPC 2002: 9th International Conference, Lecture Notes in Comput. Sci. 2552, S. Sahni, V. Prasanna, and U. Shakla, eds., Springer-Verlag, Heidelberg, 2002, pp. 545–554.
[8] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu, *A fast eigensolver for companion matrices*, in Oper. Theory Adv. Appl. 179, Birkhauser-Verlag, Basel, Switzerland, 2007, pp. 111–143.
[9] J. J. Dongarra, S. Hammarling, and J. H. Wilkinson, *Numerical considerations in computing invariant subspaces*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 145–161.
[10] W. Gautschi, *On the condition of algebraic equations*, Numer. Math., 21 (1973), pp. 405–424.
[11] W. Gautschi, *Questions of numerical condition related to polynomials*, in Studies in Numerical Analysis, MAA Studies in Math. 24, G. H. Golub, ed., Mathematical Association of America, Washington, DC, 1984, pp. 140–177.
[12] G. Golub and C. V. Loan, *Matrix Computation*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[13] T. Gudmundsson, C. S. Kenney, and A. J. Laub, *Small-sample statistical estimates for the sensitivity of eigenvalue problems*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 868–886.
[14] T. Gudmundsson, C. S. Kenney, A. J. Laub, and M. S. Reese, *Applications of small-sample statistical condition estimation in control*, in Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design, 1996, pp. 164–169.
[15] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008.
[16] C. S. Kenney and A. J. Laub, *Small-sample statistical condition estimates for general matrix functions*, SIAM J. Sci. Comput., 15 (1994), pp. 36–61.
[17] C. S. Kenney, A. J. Laub, and M. S. Reese, *Statistical condition estimation for linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 566–583.
[18] C. S. Kenney, A. J. Laub, and M. S. Reese, *Statistical condition estimation for linear least squares*, SIAM J. Math. Anal. Appl., 19 (1998), pp. 906–923.

[19] G. W. Stewart, *Algorithm* 506 : *HQR*3 *and EXCHNG: Fortran subroutines for calculating and ordering eigenvalues of a real upper Hessenberg matrix*, ACM Trans. Math. Software, 2 (1976), pp. 275–280.
[20] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[21] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon, Oxford, UK, 1965.
[22] J. R. Winkler, *Condition numbers of a nearly singular simple root of a polynomial*, Appl. Numer. Math., 38 (2001), pp. 275–285.
[23] H. Zhang, *Numerical condition of polynomials in different forms*, Electron. Trans. Numer. Anal., 12 (2001), pp. 66–87.