NEW ALGORITHMS FOR COMPUTING THE REAL STRUCTURED PSEUDOSPECTRAL ABSCISSA AND THE REAL STABILITY RADIUS OF LARGE AND SPARSE MATRICES

MINGHAO W. ROSTAMI

Abstract. We present two new algorithms for investigating the stability of large and sparse matrices subject to real perturbations. The first algorithm computes the real structured pseudospectral abscissa and is based on the algorithm for computing the pseudospectral abscissa proposed by Guglielmi and Overton [SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1166–1192]. It entails finding the rightmost eigenvalues for a sequence of large matrices, and we demonstrate that these eigenvalue problems can be solved in a robust manner by an unconventional eigenvalue solver. We also develop an algorithm for computing the real stability radius of a real and stable matrix, which utilizes a recently developed technique for detecting the loss of stability in a large dynamical system. Both algorithms are tested on large and sparse matrices.

Key words. real structured pseudospectrum, abscissa, eigenvalue, stability radius, sparse matrix, Lyapunov equation

AMS subject classifications. 15A18, 15A24, 65F15, 65F50

DOI. 10.1137/140975413

1. Introduction. The aim of this paper is to develop efficient algorithms for studying the stability of large, sparse matrices under real perturbations. Let $A \in \mathbb{C}^{n \times n}$ and $\alpha(A)$ denote its spectral abscissa, i.e., the real part of the rightmost eigenvalue of $A$. If $\alpha(A) < 0$, then $A$ is said to be stable; otherwise, it is unstable. It is well known that the pseudospectral abscissa of $A$ is a more robust measurement of stability since the eigenvalues of $A$ can be very sensitive to small perturbations. Let $\Lambda_\varepsilon(A)$ (where $\varepsilon > 0$) denote the $\varepsilon$-pseudospectrum of $A$, i.e.,

$$\Lambda_\varepsilon(A) = \{ z \in \mathbb{C} \mid z \text{ is an eigenvalue of } A + \Delta, \text{ where } \Delta \in \mathbb{C}^{n \times n} \text{ and } \| \Delta \| \leq \varepsilon \}.$$ 

(Throughout this paper, $\| \cdot \| = \| \cdot \|_2$, though $\| \cdot \|$ could be any matrix norm in principle.) The $\varepsilon$-pseudospectral abscissa of $A$ is the real part of the rightmost point of $\Lambda_\varepsilon(A)$, i.e.,

$$\alpha_\varepsilon(A) = \max \{ \Re(z) \mid z \in \Lambda_\varepsilon(A) \}.$$ 

An example where $\varepsilon$-pseudospectral abscissa is more reliable than spectral abscissa is stability analysis of a linear dynamical system $u_t = Au$. Eigenvalue analysis indicates that if $A$ is stable, then the steady state of this dynamical system will be stable as well (that is, any finite perturbation introduced to the steady state will eventually die down). However, there are cases where $A$ is stable and yet the steady state has been observed to be unstable in practice (see section 20 of [34] for several examples from fluid mechanics). It turns out that in these cases, very small values of $\varepsilon$ will cause $\alpha_\varepsilon(A) > 0$.

*Received by the editors July 1, 2014; accepted for publication (in revised form) June 1, 2015; published electronically October 29, 2015.


†Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609-2280 (mwu@wpi.edu).
When \( A \) is small, \( \alpha_\varepsilon(A) \) can be computed using the \textit{criss-cross algorithm} developed in [4]. This method is guaranteed to find \( \alpha_\varepsilon(A) \); however, it requires computing the complete sets of eigenvalues of a few \( 2n \times 2n \) Hamiltonian matrices and thus is not applicable to large \( A \). A method intended for large and sparse matrices has been proposed by Guglielmi and Overton [20]. It is able to identify \( \alpha_\varepsilon(A) \) for a large number of examples by introducing a sequence of perturbations to \( A \). Each perturbation is chosen based on standard eigenvalue perturbation theory to increase the spectral abscissa of the previous perturbed \( A \) as much as possible (see section 2 for a more precise statement). This method entails computing the rightmost eigenvalues of a sequence of matrices, for which an iterative eigenvalue solver such as the \textit{implicitly restarted Arnoldi method} (see [31]) is needed.

In some applications (see section 45 of [34] and the references therein), instead of taking into account all the possible perturbations, we want to impose a certain structure on the perturbations introduced to \( A \), such as requiring them to be real or to have the same sparsity pattern as \( A \). In this paper, we will focus on the real perturbations. The \textit{real structured} \( \varepsilon \)-pseudospectrum of \( A \) is

\[
\Lambda_\varepsilon^R(A) = \{ z \in \mathbb{C} \mid z \text{ is an eigenvalue of } A + \Delta, \text{ where } \Delta \in \mathbb{R}^{n \times n} \text{ and } \| \Delta \| \leq \varepsilon \},
\]

and accordingly, the \textit{real structured} \( \varepsilon \)-pseudospectral abscissa of \( A \) is the real part of the rightmost point of \( \Lambda_\varepsilon^R(A) \), i.e.,

\[
\alpha_\varepsilon^R(A) = \max \{ \Re(z) \mid z \in \Lambda_\varepsilon^R(A) \}.
\]

It is obvious that \( \Lambda_\varepsilon^R(A) \subseteq \Lambda_\varepsilon(A) \) and therefore, \( \alpha_\varepsilon^R(A) \leq \alpha_\varepsilon(A) \).

Inspired by the algorithm introduced in [20], we hope to develop an algorithm for computing \( \alpha_\varepsilon^R(A) \) for large and sparse \( A \) in a similar fashion. Therefore, we need to find a sequence of \textit{real} perturbations to introduce to \( A \), each of which increases the spectral abscissa of the previous perturbed \( A \) as much as possible. The issue is that when the perturbations are restricted to be real, it is difficult to see what these perturbations should be. We prove a theorem that describes them explicitly. The new algorithm again requires finding the rightmost eigenvalues for a series of matrices. An algorithm for computing \( \alpha_\varepsilon^R(A) \) for large and sparse \( A \) has already been developed in previous work [18], which is similar to the new algorithm in structure but uses a different approach to generate the perturbations. A comparison of the two methods is given in section 3.

Although our algorithm works for any \( A \in \mathbb{C}^{n \times n} \), we are particularly interested in the more common case where \( A \) is real. Under this assumption, all the eigenvalue problems arising from the computation of \( \alpha_\varepsilon^R(A) \) are also real. We will show that these eigenvalue problems can be solved in a robust manner using the method by Elman and Wu [13]. Traditional iterative eigenvalue solvers such as Arnoldi’s method and its variants (see [29, 33]) are very reliable when eigenvalues near a point \( \sigma \in \mathbb{C} \) (called a \textit{shift}) are sought. However, when the rightmost eigenvalue is wanted but an estimate for it is not available (which is usually the case), these methods may converge rather slowly or even to a wrong eigenvalue. The approach developed in [13], on the other hand, does not compute the rightmost eigenvalue directly; instead, it obtains the rightmost eigenvalue of the original problem by computing the eigenvalue with smallest modulus (i.e., closest to zero) of an alternative eigenvalue problem in the form of a \textit{Lyapunov equation}. This new eigenvalue problem can be solved using the \textit{Lyapunov inverse iteration} proposed in [25], which is a version of the well-known inverse iteration (or inverse power method) (see Chapter 2 of [33], for example).
Another problem that we will investigate in this paper is how to compute the real stability radius or distance to instability caused by real perturbations of a real and stable matrix $A$, i.e.,

$$r^R(A) = \min \left\{ \|\Delta\| \; | \; \alpha(A + \Delta) \geq 0, \; \text{where} \; \Delta \in \mathbb{R}^{n \times n} \right\}.$$ 

That is, it is the minimum of the norms of all the real matrices that make $A$ unstable. A formula for $r^R(A)$ was given in [27] in the form of an optimization problem in two variables:

$$r^R(A) = \min_{s \in \mathbb{R}} \max_{\gamma \in (0,1)} \sigma_{2n-1} \left( \begin{bmatrix} A & -\gamma sI \\ \gamma^{-1}sI & A \end{bmatrix} \right),$$

where $\sigma_{2n-1}(\cdot)$ denotes the $(2n-1)st$ largest singular value of a matrix. Several methods have been developed for solving (1.1). An approach based on level set methods was presented in [32], which requires computing the complete sets of eigenvalues of several $4n \times 4n$ Hamiltonian matrices. More recently, Freitag and Spence [15] proposed an algorithm that solves (1.1) using the implicit determinant method [14] which is a Newton-based approach. It entails solving linear systems of order $4n + 1$.

A method that is more suitable for large and sparse matrices has been developed in [19]. It has the following inner-outer structure: the outer iteration is a Newton step that approximates $r^R(A)$ and the inner iteration is to compute $\alpha^R_\varepsilon(A)$ for some $\varepsilon$. This algorithm is developed based on eigenvalue perturbation theory and its main cost is solving a number of eigenvalue problems of order $n$. The method that we develop in the current paper has a similar structure, the main difference being that the estimate to $r^R(A)$ is obtained by solving an eigenvalue problem. We first make the observation that finding $r^R(A)$ is similar to a problem considered in previous work [9, 25], where the quantity of interest was the critical value of a physical parameter $\xi$ (such as the Reynolds number) at which the steady state of the parameter-dependent dynamical system $u_t = A(\xi)u$ loses stability (see section 5.1 for a more detailed description). Following the same argument given in [25], we then show that $r^R(A)$ is the eigenvalue with smallest modulus of an eigenvalue problem with the structure of a Lyapunov equation, which can again be found by Lyapunov inverse iteration.

The rest of this paper is organized as follows. In section 2, we give a brief review of the algorithm for computing $\alpha_\varepsilon(A)$ proposed in [20]. In section 3, we introduce the algorithm for computing $\alpha^R_\varepsilon(A)$ and test it on several large and sparse matrices. In section 4, we demonstrate how Lyapunov inverse iteration can be applied to solve the eigenvalue problems arising from the algorithm described in the previous section. In section 5, we describe the algorithm for computing $r^R(A)$ and apply it to a few examples. Some concluding remarks are given in section 6.

The following definition is adopted from [20] and will be referred to in this paper as well: two vectors $u, v \in \mathbb{C}^n$ are RP-compatible if $\|u\| = \|v\| = 1$ and $u^*v$ is real and positive. For any pair of vectors $(x, y)$ that satisfies $y^*x \neq 0$, a pair of RP-compatible vectors $(\tilde{x}, \tilde{y})$ can be obtained by scaling $x$ and $y$ in the following way:

$$\tilde{x} = \frac{x}{\|x\|}, \quad \tilde{y} = \frac{y^*x}{\|y^*x\|}, \quad \frac{y}{\|y\|}.$$

2. Review of the algorithm for computing the $\varepsilon$-pseudospectral abscissa. According to Lemma 1.1 in [20], there exists a rank-1, norm-1 matrix $L_{opt} = u_{opt}v_{opt}^*$ where $u_{opt}, v_{opt} \in \mathbb{C}^n$ and $\|u_{opt}\| = \|v_{opt}\| = 1$ such that the rightmost eigenvalue of $A + \varepsilon L_{opt}$ is the rightmost point of $\Lambda_\varepsilon(A)$. An algorithm was then proposed
in [20] that generates a sequence of rank-1, norm-1 matrices \( \{ L_k = u_k v_k^* \} \) which almost always converge to \( L_{opt} \) in numerical experiments. It requires the solution of a series of eigenvalue problems for their rightmost eigenvalues. In this section, we give a brief review of this algorithm. Its numerical results for an example arising from fluid dynamics are given in Appendix A.

The algorithm proposed in [20] for computing \( \alpha_\varepsilon(A) \) where \( A \) is large and sparse is restated in Algorithm 1. (It is referred to as Algorithm PSA0 in [20].)

**Algorithm 1: Compute the \( \varepsilon \)-pseudospectral abscissa of \( A \).**

1. Initialization:
   1.1. Compute a rightmost eigenvalue \( z_0 \) of \( A \) and its corresponding RP-compatible right and left eigenvectors \( x_0 \) and \( y_0 \).
   1.2. Set \( L_0 = y_0 x_0^* \) and \( B_1 = A + \varepsilon L_0 \).
2. For \( k = 1, 2, \ldots \) until convergence:
   2.1. Compute the rightmost eigenvalue \( z_k \) of \( B_k \) (if there is more than one, choose the one closest to \( z_{k-1} \)) and its corresponding RP-compatible right and left eigenvectors \( x_k \) and \( y_k \).
   2.2. Set \( L_k = y_k x_k^* \) and \( B_{k+1} = A + \varepsilon L_k \).

The design of this algorithm utilizes the following standard perturbation result for eigenvalues (see Chapter 6 of [21], for example).

**Lemma 2.1.** Let \( t \in \mathbb{R} \), and consider the \( n \times n \) matrix family \( C(t) = C_0 + t C_1 \). Let \( \mu(t) \) be an eigenvalue of \( C(t) \) converging to a simple eigenvalue \( \mu_0 \) of \( C_0 \) as \( t \to 0 \). Then \( y_0^* x_0 \neq 0 \) and \( \mu(t) \) is analytic near \( t = 0 \) with

\[
\left. \frac{d\mu(t)}{dt} \right|_{t=0} = \frac{y_0^* C_1 x_0}{y_0^* x_0},
\]

where \( x_0 \) and \( y_0 \) are, respectively, right and left eigenvectors of \( C_0 \) corresponding to \( \mu_0 \), that is, \( (C_0 - \mu_0 I)x_0 = 0 \) and \( y_0^*(C_0 - \mu_0 I) = 0 \).

The rationale behind choosing \( L_k = y_k x_k^* \) in Algorithm 1 is as follows. For \( k = 0, 1, 2, \ldots \), consider the matrix family \( (A + \varepsilon L_{k-1}) + t(L - L_{k-1}) \), where \( L \in \mathbb{C}^{n \times n} \) and \( \|L\| = 1 \). \( (L - L_{k-1}) \) is taken to be the \( n \times n \) zero matrix.) At \( t = 0 \), \( (A + \varepsilon L_{k-1}) + t(L - L_{k-1}) = A + \varepsilon L_{k-1} \), and when \( t = \varepsilon \), \( (A + \varepsilon L_{k-1}) + t(L - L_{k-1}) = A + \varepsilon L \), which is a norm-\( \varepsilon \) perturbation away from \( A \). Assume \( \mu_0 \) is any simple eigenvalue of \( A + \varepsilon L_{k-1} \) and \( (v_0, u_0) \) is a pair of RP-compatible right and left eigenvectors of \( A + \varepsilon L_{k-1} \) corresponding to \( \mu_0 \). Let \( \mu(t) \) denote the eigenvalue of \( (A + \varepsilon L_{k-1}) + t(L - L_{k-1}) \) converging to \( \mu_0 \) as \( t \to 0 \). By Lemma 2.1,

\[
\Re \left. \frac{d\mu(t)}{dt} \right|_{t=0} = \Re \frac{u_0^*(L - L_{k-1}) v_0}{u_0^* v_0} = \Re \frac{u_0^* L v_0 - u_0^* L_{k-1} v_0}{u_0^* v_0} \\
\leq \left| 1 - \Re \frac{u_0^* L v_0}{u_0^* v_0} \right|
\]

and the equality is attained when \( L = u_0 v_0^* \), i.e., the real part of \( \mu(t) \) grows most rapidly at \( t = 0 \) for this choice of \( L \). (Since \( u_0^* L_{k-1} v_0 \leq \|u_0\| \cdot \|L_{k-1}\| \cdot \|v_0\| = 1 \) in (2.2), this maximum rate is always nonnegative.) If the rightmost eigenvalue \( z_k \) of \( A + \varepsilon L_{k-1} \) is simple, (2.2) suggests choosing \( \mu_0 = z_k \) and consequently \( L_k = u_0 v_0^* \).

\(^1\)Stated here is the version of this result given by Lemma 2.1 in [20].
$y_kx_k^*$ to be the new guess of $L_{\text{opt}}$, where $(x_k, y_k)$ is a pair of RP-compatible right and left eigenvectors associated with $z_k$. That is, we choose $L_k$ to maximize the instantaneous rate of change of $\text{Re}(\mu(t))$ at $t = 0$, where $\mu(t)$ denote the eigenvalue of $(A + \varepsilon L_{k-1}) + t(L - L_{k-1})$ converging to $z_k$ as $t \to 0$. For this choice of $L_k$ and sufficiently small $\varepsilon$,

\begin{equation}
\text{Re} (\mu(\varepsilon)) \approx \text{Re} (\mu(0)) + \varepsilon \left( \text{Re} \left. \frac{d\mu(t)}{dt} \right|_{t=0} \right) \geq \text{Re} (\mu(0)) = \text{Re} (z_k),
\end{equation}

that is, $\mu(\varepsilon)$ and therefore the rightmost eigenvalue $z_{k+1}$ of $A + \varepsilon L_k$ will lie to the right of $z_k$. The success of Algorithm 1 has been reported for extensive numerical experiments in [20]; furthermore, for the majority of these examples, the sequence $\{\text{Re}(z_k) = \alpha(A + \varepsilon L_k)\}$ increases monotonically to $\alpha_\varepsilon(A)$.

At each iteration of Algorithm 1, we need to compute the rightmost eigenvalue for $B_k = A + \varepsilon L_{k-1}$. When $n$ is large, this requires an iterative eigenvalue solver, which usually involves matrix-vector products and/or solves with $B_k$. Although $B_k$ is a dense matrix in general, due to the low-rank structure of $L_k$ and the sparsity of $A$, applying it to a vector is cheap, and for small $\varepsilon$, preconditioners developed for $A$ can be used to precondition $B_k$ as well. In addition, note that $B_k$ is generally not real.

3. An algorithm for computing the real structured pseudospectral abscissa. By Theorem 3.1 in [18], there exists a real, rank-2, and norm-1 matrix $L^\mathbb{R}_{\text{opt}} = U_{\text{opt}} V_{\text{opt}}^T$ where $U_{\text{opt}}, V_{\text{opt}} \in \mathbb{R}^{n \times 2}$ consist of orthonormal columns such that the rightmost eigenvalue of $A + \varepsilon L^\mathbb{R}_{\text{opt}}$ is the rightmost point of $\Lambda^\mathbb{R}(A)$. In this section, we present an algorithm for computing the real structured $\varepsilon$-pseudospectral abscissa for a large and sparse $A$, which generates a sequence of real, rank-2, and norm-1 perturbation matrices $\{L^\mathbb{R}_k = U_k V_k^T\}$.

3.1. The new algorithm. As described in section 2, in Algorithm 1, we seek

\begin{equation}
L_k = \arg \left( \max_{L \in \mathbb{C}^{n \times n}, \|L\| = 1} \left\{ \text{Re} \left. \frac{d\mu(t)}{dt} \right|_{t=0} = \frac{\text{Re}(y_k^* L x_k) - \text{Re}(y_k^* L_{k-1} x_k)}{y_k^* x_k} \right\} \right)
\end{equation}

for $k = 0, 1, 2, \ldots$, where $\mu(t)$ is the eigenvalue of $(A + \varepsilon L_{k-1}) + t(L - L_{k-1})$ converging to the rightmost eigenvalue $z_k$ of $A + \varepsilon L_{k-1}$ as $t \to 0$, and $(x_k, y_k)$ is a pair of RP-compatible right and left eigenvectors of $A + \varepsilon L_{k-1}$ corresponding to $z_k$. That is, we look for the perturbation matrix $L_k$ that maximizes the local growth rate of the real part of $\mu(t)$ at $t = 0$. It is not difficult to see that $L_k = y_k x_k^*$ since it maximizes the quantity Re($y_k^* L x_k$) in (3.1) among all $L \in \mathbb{C}^{n \times n}$ with unit norm. By (2.3), for sufficiently small $\varepsilon$, this choice guarantees that the next iterate $z_{k+1}$ will lie to the right of the current iterate $z_k$.

Our aim is to develop an algorithm for computing $\alpha^\mathbb{R}_\varepsilon(A)$ in a similar fashion. Since only real perturbations of $A$ are allowed, instead of (3.1), we want to find

\begin{equation}
L^\mathbb{R}_k = \arg \left( \max_{L \in \mathbb{R}^{n \times n}, \|L\| = 1} \left\{ \text{Re} \left. \frac{d\mu(t)}{dt} \right|_{t=0} = \frac{\text{Re}(y_k^* L x_k) - \text{Re}(y_k^* L^\mathbb{R}_{k-1} x_k)}{y_k^* x_k} \right\} \right)
\end{equation}

for $k = 0, 1, 2, \ldots$, where $\mu(t)$ is the eigenvalue of $(A + \varepsilon L^\mathbb{R}_{k-1}) + t(L - L^\mathbb{R}_{k-1})$ converging to the rightmost eigenvalue $z_k$ of $A + \varepsilon L^\mathbb{R}_{k-1}$ as $t \to 0$, and $(x_k, y_k)$ is a pair of RP-compatible right and left eigenvectors of $A + \varepsilon L^\mathbb{R}_{k-1}$ corresponding to $z_k$. ($L^\mathbb{R}_{k-1}$ is the
$n \times n$ zero matrix.) Note that finding $L_k^R$ is equivalent to finding an $L \in \mathbb{R}^{n \times n}$ with unit norm that maximizes $\text{Re}(y_k^* L x_k)$. Thus, we first prove the following theorem.

**Theorem 3.1.** Let $x = c + id$, $y = a + ib \in \mathbb{C}^n$ ($a, b, c, d \in \mathbb{R}^n$) and

$$U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V^T$$

be the truncated singular value decomposition (SVD) of $ac^T + bd^T$, i.e., $U, V \in \mathbb{R}^{n \times 2}$ consist of orthonormal columns and $\sigma_1 \geq \sigma_2 \geq 0$. Then

$$\max_{L \in \mathbb{R}^{n \times n}, \|L\|=1} \{\text{Re}(y^* L x)\} = \sigma_1 + \sigma_2$$

and one maximizer is $L = UV^T$.

**Proof.** For any $L \in \mathbb{R}^{n \times n}$, since $a, b, c, d \in \mathbb{R}^n$,

$$\text{Re}(y^* L x) = a^T L c + b^T L d = (L c)^T a + (L d)^T b = \sum_{j=1}^n (c_j L_j)^T a + \sum_{j=1}^n (d_j L_j)^T b$$

$$= \sum_{j=1}^n L_j^T (ac_j + bd_j) = \text{trace} \left( L^T (ac^T + bd^T) \right),$$

where $(\cdot)_j$ denotes the $j$th column of a matrix or the $j$th entry of a vector. Let $P \Sigma Q^T$ be the full SVD of $ac^T + bd^T$, i.e.,

$$ac^T + bd^T = P \Sigma Q^T = \begin{bmatrix} U & \tilde{U} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} V & \tilde{V} \end{bmatrix}^T = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V^T,$$

and let $G = P^T L Q$ so that $L = PGQ^T$. Since $\|G\| = \|L\| = 1$, by (3.3) and (3.4),

$$\text{Re}(y^* L x) = \text{trace} \left( QG^T P^T P \Sigma Q^T \right) = \text{trace} \left( G^T \Sigma \right) = G_{11} \sigma_1 + G_{22} \sigma_2 \leq \sigma_1 + \sigma_2,$$

and the equality is attained if $G_{11} = G_{22} = 1$, for instance, when

$$L = \begin{bmatrix} U & \tilde{U} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} V & \tilde{V} \end{bmatrix}^T = UV^T. \quad \square$$

By Theorem 3.1, $L_k^R = U_k V_k^T$ in (3.2), where $U_k$ and $V_k$ contain, respectively, the left and right singular vectors associated with the two leading singular values of the matrix $\text{Re}(y_k)(\text{Re}(x_k)^T + \text{Im}(y_k)(\text{Im}(x_k))^T$. For this choice of $L_k^R$ and sufficiently small $\varepsilon$, we can again obtain the inequality (2.3), and therefore, the rightmost eigenvalue of $A + \varepsilon L_k^R$ will lie to the right of the rightmost eigenvalue of $A + \varepsilon L_{k-1}^R$. The analysis above leads to Algorithm 2.

As in Algorithm 1, we need to compute the rightmost eigenvalues for a sequence of matrices in Algorithm 2; the main difference is that these matrices will always be real as long as $A$ is real. Compared to Algorithm 1, an extra truncated SVD is required at every step of Algorithm 2, but it is cheap due to the low-rank structure of $\text{Re}(y_k)(\text{Re}(x_k))^T + \text{Im}(y_k)(\text{Im}(x_k))^T$. To see this, let

$$Y = [\text{Re}(y_k), \text{Im}(y_k)], \ X = [\text{Re}(x_k), \text{Im}(x_k)],$$

and let $Q_y R_y$ be the “skinny” QR decomposition of $Y$ and $Q_x R_x$ that of $X$. In addition, let $U \Sigma \tilde{V}^T$ be the SVD of $R_y R_x^T$ which is $2 \times 2$. Then $(Q_y U) \tilde{\Sigma} (Q_x \tilde{V})^T$ is the truncated SVD of $YX^T$ or, equivalently, $\text{Re}(y_k)(\text{Re}(x_k))^T + \text{Im}(y_k)(\text{Im}(x_k))^T$. 

\[\text{S452} \quad \text{MINGHAO W. ROSTAMI}\]
Algorithm 2: Compute the real structured $\varepsilon$-pseudospectral abscissa of $A$.

1. Initialization:
   1.1. Compute a rightmost eigenvalue $z_0$ of $A$ and its corresponding RP-compatible right and left eigenvectors $x_0$ and $y_0$.
   1.2. Compute the truncated SVD $U_0\Sigma_0V_0^T$ of $\text{Re}(y_0)(\text{Re}(x_0))^T + \text{Im}(y_0)(\text{Im}(x_0))^T$, where $U_0, V_0 \in \mathbb{R}^{n \times 2}$ and $\Sigma_0 \in \mathbb{R}^{2 \times 2}$.
   1.3. Set $L_0^R = U_0V_0^T$ and $B_1 = A + \varepsilon L_0^R$.

2. For $k = 1, 2, \ldots$ until convergence:
   2.1. Compute the rightmost eigenvalue $z_k$ of $B_k$ (if there is more than one, choose the one closest to $z_{k-1}$) and its corresponding RP-compatible right and left eigenvectors $x_k$ and $y_k$.
   2.2. Compute the truncated SVD $U_k\Sigma_kV_k^T$ of $\text{Re}(y_k)(\text{Re}(x_k))^T + \text{Im}(y_k)(\text{Im}(x_k))^T$, where $U_k, V_k \in \mathbb{R}^{n \times 2}$ and $\Sigma_k \in \mathbb{R}^{2 \times 2}$.
   2.3. Set $L_k^R = U_kV_k^T$ and $B_{k+1} = A + \varepsilon L_k^R$.

3.2. Numerical results. In this section, we present numerical results of Algorithm 2 applied to a few examples and compare them with the results produced by an existing method in [18].

The following set of real, large, and sparse examples will be considered here: “obstacle” ($n = 9, 512$), “PDE” ($n = 2, 961$), “rdbrusselator” ($n = 3, 200$), “tolosa” ($n = 4, 000$), and “tubular” ($n = 1, 000$), the latter four of which are test problems provided in EigTool and have been studied in [20] as well. (These four matrices are also available from the Matrix Market library [3], where they are labeled PDE2961, RDB3200L, TOLS4000, and TUB1000, respectively.) They all arise from spatial discretization of partial differential equations (PDEs) and have a complex conjugate pair of rightmost eigenvalues. Numerical results of both Algorithms 1 and 2 (with $\varepsilon = 10^{-2}$) applied to these problems are summarized in Table 1, and the sequence of iterates $\{z_k\}$ that they generate are plotted in Figure 1. For both algorithms, we continue to use (A.4) as the stopping criterion and MATLAB routine \texttt{eigs} (with appropriate parameters) as the eigenvalue solver. The version of MATLAB used is R2014a and all the numerical experiments in this study are run on an Intel Xeon E5-2680 v2 CPU.

As shown in Table 1, it takes Algorithms 1 and 2 the same number of iterations to converge in all but one example. The quantity $\eta_\varepsilon(A)$ in the last column of this table is introduced in order to compare the efficiencies of real and complex perturbations at
increasing $\alpha(A)$. For any $\varepsilon > 0$, by definition, $0 < \eta_\varepsilon(A) \leq 1$; in addition, the closer $\eta_\varepsilon(A)$ is to 1, the more efficient real perturbations are. It can be seen from both Table 1 and Figure 1 that the efficiency of real perturbations can be very different for different problems. In the obstacle and PDE examples, $\eta_\varepsilon(A) > 0.97$, indicating that real perturbations are almost as efficient as complex perturbations, whereas for the tolosa example, $\eta_\varepsilon(A) < 0.02$, which suggests that real perturbations are far less efficient than their complex counterparts. In particular, $\alpha_\varepsilon^2(A) < 0$ and $\alpha_\varepsilon(A) > 0$ in this case, i.e., although $A$ will remain stable under any real perturbation of size $10^{-2}$,
some complex perturbations of the same size will cause $A$ to lose its stability. The rdbrusselator and tubular examples are between these two extremes.

Remark. The estimates for $\alpha_\epsilon(A)$ computed by Algorithm 1 have been validated by EigTool. However, at the time this paper was written, we were not aware of any existing code that produces the exact $\alpha_\epsilon^R(A)$. Therefore, in Table 1, the estimates found by Algorithm 2 are only guaranteed to be lower bounds of $\alpha_\epsilon^R(A)$.

An algorithm for computing $\alpha_\epsilon(A)$ for large and sparse $A$ has already been developed in [18]. (It is also referred to as Algorithm 2 in [18].) We will compare this algorithm and Algorithm 2 below. It has a similar structure as Algorithm 2: at each step, it finds a real, rank-2, and norm-1 perturbation matrix $L_k^R$ followed by the rightmost eigenvalue of $A + \epsilon L_k^R$. Each perturbation $L_k^R$ in Algorithm 2 is a solution to the optimization problem (3.2) and can be found by computing the truncated SVD of a rank-2 matrix according to Theorem 3.1. Every perturbation $L_k^R$ in the method of [18] is chosen to be a point on the trajectory of an ordinary differential equation (ODE) and can be approximated using the forward Euler’s method. Such an approach is based on the observation that $L_{opt}^R$ is the stationary point of a certain ODE (see Theorem 3.6 of [18]). In fact, Algorithm 2 can be interpreted as a special discretization of the ODEs considered in [18], where a constant step size 1 is used. One advantage of the method in [18] is that it is guaranteed to produce a monotonically increasing sequence of estimates to $\alpha_\epsilon^R(A)$. The estimates computed by Algorithm 2 are not guaranteed to be monotonically increasing, although they are in all the cases considered in this section. For both algorithms, the costs of generating the perturbations are negligible and their main cost is therefore the eigenvalue computation.

We apply Algorithm 2 of [18] to the same set of problems and compare the two algorithms in Table 2. The parameters of the algorithm in [18] are chosen as follows: the initial guess of $L_{opt}^R$ is a random, rank-2, and norm-1 matrix, the initial step size of Euler’s method is 1, and the increase/reduction factor of the step size, $\gamma$, is 2. The same stopping criterion (A.4) is used for both algorithms. In Algorithm 2, we need to find the rightmost eigenvalues for $A$ in the initialization step and for a perturbed $A$ in each subsequent iteration. In every iteration of the method in [18], we may need to compute the rightmost eigenvalues for multiple matrices. Therefore, instead of reporting their iteration counts, we list the total numbers of eigenvalue problems arising from the two algorithms. As shown in Table 2, the estimates to $\alpha_\epsilon^R(A)$ produced by the two algorithms coincide in all the examples. Algorithm 2 requires fewer eigenvalue solves than the method of [18] in all but one example. In particular, when applied to the rdbrusselator and tubular examples, Algorithm 2 requires 50% fewer eigenvalue solves.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Estimated $\alpha_\epsilon^R(A)$</th>
<th>Eig. solves in Algorithm 2</th>
<th>Estimated $\alpha_\epsilon^R(A)$</th>
<th>Eig. solves in Algorithm 2 of [18]</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td>-0.23832400</td>
<td>6</td>
<td>-0.23832400</td>
<td>9</td>
</tr>
<tr>
<td>PDE</td>
<td>9.95239250</td>
<td>8</td>
<td>9.95239251</td>
<td>11</td>
</tr>
<tr>
<td>rdbrusselator</td>
<td>0.11662268</td>
<td>3</td>
<td>0.11662268</td>
<td>6</td>
</tr>
<tr>
<td>tubular</td>
<td>-0.80900864</td>
<td>5</td>
<td>-0.80900864</td>
<td>11</td>
</tr>
</tbody>
</table>

We also note that neither Algorithm 2 nor the algorithm of [18] is guaranteed to converge to $\alpha_\epsilon^R(A)$. Consider again the PDE example and let $\epsilon = 0.1$ instead. Both
algorithms estimate $\alpha_R^\varepsilon(A)$ to be 10.17583935 (the sequence of iterates generated by Algorithm 2 is plotted in Figure 2; see also Figure 5.5 in [18]). On the other hand, the rightmost point of $\Lambda_\varepsilon(A)$ computed by EigTool is a real number: 10.20376724 (see again Figure 2). Lemma 1.1 of [20] implies that the rightmost point of $\Lambda_R^\varepsilon(A)$ and thus $\alpha_R^\varepsilon(A)$ should be 10.20376724 as well, not 10.17583935.

4. Lyapunov inverse iteration applied to the eigenvalue problems. As shown in the previous section, in order to compute the real structured pseudospectral abscissa of $A$, we need to compute the rightmost eigenvalue for $A$ as well as for a sequence of matrices in the form of $A$ plus a real matrix. We will show that when $A$ is real, these eigenvalue problems can be solved in a reliable way using Lyapunov inverse iteration. We focus on real $A$ since this is the case in most applications that we have seen in the literature. For reasons that will soon become clear, we will treat $A$ and the perturbed $A$ separately in the following two subsections.

4.1. Computing the rightmost eigenvalue of the original matrix. We first give a brief description of how Lyapunov inverse iteration can be applied to compute the rightmost eigenvalue for a real and stable matrix $A$. This approach is based on the following theorem, which is equivalent to Theorems 2.1 and 2.2 in [13] combined.

**Theorem 4.1.** Let $A \in \mathbb{C}^{n \times n}$ be diagonalizable and $\{(\mu_j, x_j)\}_{j=1}^n$ denote its eigenpairs. Then

1. the eigenvalues of
   \[
   SZ + ZS^T + \lambda (2SZS^T) = 0,
   \]
   where $S = A^{-1}$ are given by $\{\lambda_{i,j} = -\frac{1}{2}(\mu_i + \mu_j)\}_{i,j=1}^n$, and $Z_{i,j} = c_1 x_i x_j^T + c_2 x_j x_i^T \in \mathbb{C}^{n \times n}$ for any $c_1, c_2 \in \mathbb{C}$ is an “eigenvector” of (4.1) associated with $\lambda_{i,j}$; in addition,

2. if $A$ is also real and stable (i.e., $\alpha(A) < 0$), then the eigenvalue of (4.1) with smallest modulus is $-\alpha(A)$ (or equivalently, $-\text{Re}(z_0)$, where $z_0$ is a rightmost eigenvalue of $A$).
If a real, stable $A$ has no other rightmost eigenvalue besides $z_0$ and $\overline{z_0}$, then by Theorem 4.1, in the subspace of real and symmetric matrices, $Z_{sm} = x_0x_0^T + \overline{x_0}\overline{x_0}^T$ where $x_0$ is the eigenvector of $A$ corresponding to $z_0$ is the unique (up to a scalar multiplier) eigenvector of (4.1) associated with $-\text{Re}(z_0)$. Let $VDV^T$ be the truncated eigenvalue decomposition of $Z_{sm}$ which is of rank 1 or 2. Since $\text{span}\{V\} = \text{span}\{x_0, \overline{x_0}\}$, once $Z_{sm}$ is known, $(\mu_0, x_0)$ can be obtained easily by computing the eigenpair(s) of the $1 \times 1$ or $2 \times 2$ matrix $V^TAV$. Unlike for the rightmost eigenvalue, many methods are reliable for computing the eigenvalue with smallest modulus, such as inverse iteration (see Chapter 2 of [33]). Lyapunov inverse iteration is a version of inverse iteration proposed in [25] for eigenvalue problems similar in structure to (4.1). It was designed by exploiting the low-rank structure of the target eigenvector $Z_{sm}$. An outline of this method is given in Algorithm 3.

**Algorithm 3: Lyapunov inverse iteration for (4.1).**

1. Given $W_0 \in \mathbb{R}^n$ with $\|W_0\|_2 = 1$. Let $Z_0 = W_0W_0^T$.
2. For $\ell = 1, 2, \ldots$ until convergence:
   2.1. Solve for $Y_\ell$ from
   $$(4.2) \quad SY_\ell + Y_\ell S^T = -2SZ_{\ell-1}S^T$$
   in factored form: $Y_\ell = W_\ell X_\ell W_\ell^T$, where $W_\ell \in \mathbb{R}^{n \times d_\ell}$ and $X_\ell \in \mathbb{R}^{d_\ell \times d_\ell}$ with $d_\ell \ll n$.
   2.2. Rank reduction: set $\tilde{S} = W_\ell^T SW_\ell$ and solve
   $$(4.3) \quad \tilde{S} \tilde{Z} + \tilde{Z} \tilde{S}^T + \tilde{\lambda} \left(2\tilde{S} \tilde{Z} \tilde{S}^T\right) = 0$$
   for the eigenvalue with smallest modulus, $\tilde{\lambda}_{sm}$, and its eigenvector $\tilde{Z}_{sm} = \tilde{W}\tilde{X}\tilde{W}^T$, where $\tilde{W} \in \mathbb{R}^{d_\ell \times q}$, $\tilde{X} \in \mathbb{R}^{q \times q}$ and $\|\tilde{X}\|_F = 1$ with $q = 1$ or 2.
   2.3. Set $\lambda_\ell = \tilde{\lambda}_{sm}$ and $Z_\ell = W_\ell\tilde{X}\tilde{W}^T$, where $W_\ell = W_\ell\tilde{W}$.

A few implementation details of this algorithm are discussed as follows.

**The initial guess.** The initial guess $Z_0$ of $Z_{sm}$ is taken to be a real, symmetric, and rank-1 matrix $W_0W_0^T$, where $W_0$ is a random vector in $\mathbb{R}^n$. This choice guarantees that all subsequent iterates $\{Z_\ell\}$ will also be real and symmetric as desired and that the right-hand side of the first instance of the Lyapunov equation (4.2) is of rank 1.

**Solving the Lyapunov equations.** When $n$ is large, at each iteration of Lyapunov inverse iteration, a large-scale Lyapunov equation (4.2) whose right-hand side is symmetric and of rank 1 or 2 needs to be solved. Its solution is symmetric and has low-rank approximation (see [1, 17, 23, 26]), i.e., $Y_\ell \approx W_\ell X_\ell W_\ell^T$ where $W_\ell \in \mathbb{R}^{n \times d_\ell}$ has orthonormal columns and $X_\ell \in \mathbb{R}^{d_\ell \times d_\ell}$ with $d_\ell \ll n$, which can be computed by Krylov-type methods such as the standard Krylov subspace method [22, 28], the extended Krylov subspace method [30], and the rational Krylov subspace method [6]. They all entail solving linear systems in the forms of $Ay = x$ and/or $(I - sA)y = x$ where $s \in \mathbb{C}$. It has been shown in [13] that Algorithm 3 applied to (4.1) converges quickly, requiring only a few solves of (4.2).

**The “rank reduction” step.** When converged, $Z_\ell$ will be of rank 1 or 2 due to Theorem 4.1; however, at the early stage of the algorithm, it may be of rank much
larger than 2 (though still much smaller than $n$). The purpose of the rank reduction step is twofold: it ensures that $Z_\ell$ is always of rank 1 or 2 so that the right-hand side of the Lyapunov equation (4.2) is of rank 1 or 2 as well and can be solved efficiently using an iterative method; in addition, it forces the residual

$$R_{\ell}^{\text{eig}} = SZ_\ell + Z_\ell S^T + \lambda_\ell (2SZ_\ell S^T)$$

associated with the eigenvalue problem (4.1) to satisfy the Galerkin condition

$$\text{trace} \left( R_{\ell}^{\text{eig}} X^T \right) = 0$$

for any $X$ of the form $W_\ell GW_\ell^T$ and hence increases the accuracy of $(\lambda_\ell, Z_\ell)$. The small eigenvalue problem (4.3) can be solved using a version of Algorithm 3 without rank reduction and with the Bartels–Stewart algorithm [2] as the solver for (4.2). The eigenvector $\tilde{Z}_{\text{sm}}$ of (4.3) is of rank 1 or 2 again due to Theorem 4.1. Moreover, the matrix $\tilde{S} = W_\ell^T SW_\ell$ can be obtained at no additional cost from the Krylov-type method used to solve (4.2).

Low-rank representation of the iterates. In Algorithm 3, we never form the iterate $Z_\ell$ or $Y_\ell$; instead, we only store the factors $W_\ell$, $W_\ell$, $X_\ell$, and $\tilde{X}_\ell$.

Finding the rightmost eigenvalue. Algorithm 3 computes the eigenvalue of (4.1) with smallest modulus (i.e., $-\text{Re}(z_0)$) and its corresponding eigenvector, but our real goal is to compute the rightmost eigenvalue $z_0$ of $A$. To do this, we solve the $1 \times 1$ or $2 \times 2$ eigenvalue problem $(W_\ell^T AW_\ell)y = \mu y$ at the end of each iteration of Algorithm 3, producing an estimate $(\mu, W_\ell y)$ to the eigenpair $(z_0, x_0)$ that we want.

Among the matrices considered in the previous section, the ones from the obstacle (at $R = 200$), tolosa, and tubular examples are stable and will therefore be considered here. (Recall that $A = M^{-1}A$ in the obstacle case.) Numerical results of Algorithm 3 applied to the three examples can be found in Table 3. In this table, $\ell$ denotes the iteration count and $\lambda_\ell$ is the estimate of the eigenvalue of (4.1) with smallest modulus, i.e., $-\text{Re}(z_0)$. As mentioned earlier, at the end of each iteration of Algorithm 3, an estimate for $z_0$ can be obtained as a by-product, which is denoted by $\mu_\ell$ in Table 3. In addition, $R_{\ell}^{\text{eig}}, R_{\ell}^{\text{lyap}} \in \mathbb{R}^{n \times n}$ are the residuals associated with the eigenvalue problem (4.1) and the Lyapunov equation (4.2), respectively, and $d_\ell$ denotes rank of the approximate solution to (4.2). The stopping criteria used are $\| R_{\ell}^{\text{eig}} \|_F < 10^{-8}$ for the eigenvalue computation and $\| R_{\ell}^{\text{lyap}} \|_F < 10^{-9}$, $\| 2SZ_{\ell-1}S^T \|_F$ for the Lyapunov solve, where $\| \cdot \|_F$ denotes the Frobenius norm.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$\lambda_\ell$</th>
<th>$\mu_\ell$</th>
<th>$| R_{\ell}^{\text{eig}} |_F$</th>
<th>$| R_{\ell}^{\text{lyap}} |_F$</th>
<th>$d_\ell$</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td>1</td>
<td>0.32883763</td>
<td>-0.32883763 + 2.16392646i</td>
<td>1.40629293 $10^{-6}$</td>
<td>3.45880983 $10^{-9}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.32883791</td>
<td>-0.32883791 + 2.16392728i</td>
<td>1.10970153 $10^{-8}$</td>
<td>3.91468415 $10^{-10}$</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.32883792</td>
<td>-0.32883792 + 2.16392727i</td>
<td>1.75987585 $10^{-10}$</td>
<td>4.17757558 $10^{-12}$</td>
</tr>
<tr>
<td>tolosa</td>
<td>1</td>
<td>0.15600000</td>
<td>-0.15600000 + 155.99992i</td>
<td>2.38116623 $10^{-11}$</td>
<td>2.66679374 $10^{-10}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tubular</td>
<td>1</td>
<td>0.82761133</td>
<td>-0.82761133 + 1.59388650i</td>
<td>5.40234302 $10^{-12}$</td>
<td>3.95759830 $10^{-12}$</td>
</tr>
</tbody>
</table>

As can be seen from Table 3, Lyapunov inverse iteration converges rapidly in all three cases, requiring only one or three iterations. The Lyapunov solver used
is a version of the rational Krylov subspace method proposed in [11], which entails approximately \( d \ell \) linear solves of the form \((I - s_j A)y = x\) where the shifts \( \{s_j\}_{j=1}^{d \ell} \) are real, positive, and adaptively chosen.\(^2\) Therefore, the total costs of computing the rightmost eigenvalue of \( A \) for the three examples are roughly 180, 480, and 30 such solves, respectively. As observed in [11], though the shifts vary considerably and so do the costs of solving \((I - s_j A)y = x\), they are on average significantly easier to solve than \( Ay = x \). Our strategy for solving them is preconditioned GMRES with the ILU(0) preconditioner (for the tolosa and tubular examples) or the least squares commutator (LSC) preconditioner (for the obstacle example, as in Appendix A).

The rightmost eigenvalues of all three matrices considered above can also be found by the MATLAB routine `eigs` with appropriate parameters. However, different parameters are usually needed for different problems and it is difficult to determine them beforehand. We explore the effect of various parameter values on the performance of `eigs` below. For simplicity, only the type and the number of eigenvalues wanted will be varied and all other parameters of `eigs` are set to their default values. In order to find the rightmost eigenvalue of \( A \), we usually demand that `eigs` return \( k \) (\( \ll n \)) eigenvalues with algebraically largest real parts (option `lr`) or smallest moduli (option `sm`). As explained in Appendix A, the first choice works well if the rightmost eigenvalue has large modulus and/or is well-separated from other eigenvalues of \( A \); the second choice should be used when the reciprocal of the rightmost eigenvalue has large modulus and/or is well-separated from other eigenvalues of \( A^{-1} \), which is not uncommon. The first choice entails matrix-vector products with \( A \), whereas the second requires solving linear systems with coefficient matrix \( A \). The values of \( k \) that we consider in our experiments are 1 and integer multiples of 10 that are no greater than 100. Table 4 displays the results of these experiments, in which “fails” means that `eigs` either converges to a wrong eigenvalue or fails to find any eigenvalue estimate that is accurate enough, and “succeeds” means that one of the converged eigenvalues is a rightmost eigenvalue. As shown in this table, there is not a single combination of the type and the number of eigenvalues wanted that works for all three problems.

### Table 4

The performance of `eigs` applied to the same set of examples.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td><code>lr</code> and ( k = 1 ) to 100</td>
<td>fails</td>
</tr>
<tr>
<td></td>
<td><code>sm</code> and ( k = 1 )</td>
<td>fails</td>
</tr>
<tr>
<td></td>
<td><code>sm</code> and ( k = 10 ) to 100</td>
<td>succeeds</td>
</tr>
<tr>
<td>tolosa</td>
<td><code>lr</code> and ( k = 1 ) to 50</td>
<td>fails</td>
</tr>
<tr>
<td></td>
<td><code>lr</code> and ( k = 60 ) to 100</td>
<td>succeeds</td>
</tr>
<tr>
<td></td>
<td><code>sm</code> and ( k = 1 ) to 100</td>
<td>fails</td>
</tr>
<tr>
<td>tubular</td>
<td><code>lr</code> and ( k = 1 ) and 10</td>
<td>fails</td>
</tr>
<tr>
<td></td>
<td><code>lr</code> and ( k = 20 ) to 100</td>
<td>succeeds</td>
</tr>
<tr>
<td></td>
<td><code>sm</code> and ( k = 1 ) to 100</td>
<td>succeeds</td>
</tr>
</tbody>
</table>

4.2. Computing the rightmost eigenvalues of the perturbed matrices.

We would like to apply Lyapunov inverse iteration to the subsequent eigenvalue problems arising from Algorithm 2 as well. Recall that at the \( k \)th iteration of this algorithm, we need to compute the rightmost eigenvalue \( z_k \) of \( A + \varepsilon L_{k-1}^R \), where

\(^2\) Throughout this paper, \( d \ell \) is always an integer multiple of 10 since we only check the convergence of the Lyapunov solve (4.2) every 10 iterations of the rational Krylov subspace method in order to reduce computational costs (see [11] for details).
\( L_{k-1}^R \in \mathbb{R}^{n \times n} \) is rank-2. The issue here is that even if \( A \) is stable, \( A + \varepsilon L_{k-1}^R \) may not be stable anymore, in which case Lyapunov inverse iteration cannot be applied directly. (To be more precise, if \( A \) is replaced by \( A + \varepsilon L_{k-1}^R \) in Theorem 4.1, the first conclusion still holds, but the second one may not be true.)

The remedy we propose is to choose a “guard shift” \( s_{gd}^k > \text{Re}(z_k) \) such that the shifted matrix

\[
A + \varepsilon L_{k-1}^R - s_{gd}^k I
\]

is stable and apply Algorithm 3 to find the rightmost eigenvalue \( \omega_k \) of (4.4) instead. The rightmost eigenvalue of \( A + \varepsilon L_{k-1}^R \) can then be recovered easily by \( z_k = \omega_k + s_{gd}^k \).

Our strategy for choosing \( s_{gd}^k \) is described below. Recall that \( z_{k-1} \) and \( z_k \) are the rightmost eigenvalues of the parameterized matrix

\[
(A + \varepsilon L_{k-2}^R) + t (L_{k-1}^R - L_{k-2}^R)
\]

at \( t = 0 \) and \( t = \varepsilon \), respectively. Let \( \mu(t) \) be the eigenvalue of (4.5) converging to \( z_{k-1} \) as \( t \) approaches 0. By Lemma 2.1, for \( t \) close to zero,

\[
\text{Re}(\mu(t)) \approx \text{Re}(z_{k-1}) + t \frac{\text{Re}(y_{k-1}^* (L_{k-1}^R - L_{k-2}^R) x_{k-1})}{y_{k-1}^* x_{k-1}} = \text{Re}(z_{k-1}) + t \delta_{k-1},
\]

where \((x_{k-1}, y_{k-1})\) is an RP-compatible pair of right and left eigenvectors of \( A + \varepsilon L_{k-2}^R \) associated with \( z_{k-1} \), and \( \delta_{k-1} \) denotes the local growth rate of \( \text{Re}(\mu(t)) \) at \( t = 0 \). If \( z_k = \mu(\varepsilon) \), that is, if \( \mu(t) \) will evolve from \( z_{k-1} \) to \( z_k \) as \( t \) grows from 0 to \( \varepsilon \) in (4.5), then for sufficiently small \( \varepsilon \), by (4.6),

\[
\text{Re}(z_k) = \text{Re}(\mu(\varepsilon)) \approx \text{Re}(z_{k-1}) + \varepsilon \delta_{k-1}.
\]

The quantity \( \delta_{k-1} \) can be computed easily using the iterates \( x_{k-1}, y_{k-1}, L_{k-1}^R \), and \( L_{k-2}^R \) from previous iterations of Algorithm 2. In light of (4.7), we propose the following choice of \( s_{gd}^k \):

\[
s_{gd}^k = \max \{ \text{Re}(z_{k-1}) + \varepsilon (\gamma \delta_{k-1}) \},
\]

where \( \gamma > 1 \). Since \( \delta_{k-1} \geq 0 \) by Theorem 3.1, this choice ensures that \( s_{gd}^k \) is no less than either \( \text{Re}(z_{k-1}) \) or the estimated \( \text{Re}(z_k) \) given by (4.7). Furthermore, the rightmost eigenvalue \( z_k \) of \( A + \varepsilon L_{k-1}^R \) may have evolved from an eigenvalue of \( A + \varepsilon L_{k-2}^R \) other than \( z_{k-1} \), in which case \( \text{Re}(z_{k-1}) + \varepsilon \delta_{k-1} \) may not be a good estimate for \( \text{Re}(z_k) \). Therefore, a larger \( \gamma \) increases the likelihood of \( s_{gd}^k > \text{Re}(z_k) \).

In our computation, \( \gamma \) is chosen to be 2.

Table 5 displays the computational results of Algorithm 3 applied to the shifted matrix (4.4) arising from the first iteration of Algorithm 2, where \( k = 1 \), \( \varepsilon = 10^{-1} \), and \( s_{gd}^k \) is determined by (4.8). We continue to use the same stopping criteria for the eigenvalue problem (4.1) and the Lyapunov solve (4.2), where \( S = (A + \varepsilon L_{k-1}^R - s_{gd}^k I)^{-1} \). The Lyapunov solver used is again the method proposed in [11], and the resulting linear systems are solved using preconditioned GMRES as before. LSC is again the preconditioner for the “obstacle” problem, whereas ILU(0) is used for the other four problems. In all five examples, the chosen guard shift makes (4.4) stable as desired. In Table 5, \( \mu_1 \) denotes the estimated \( \omega_1 \) and thus, \( \mu_1 + s_{gd}^k \) provides an estimate for \( z_1 \). For instance, the estimated \( z_1 \) obtained for the “obstacle” example is

\[
(-2.34687840 + 2.16787839i) + 2.35957555 = 0.01269715 + 2.16787839i,
\]

which has crossed the imaginary axis already.
Algorithm 3 applied to (4.4) arising from the first iteration of Algorithm 2, where $k = 1$ and $\varepsilon = 10^{-1}$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\lambda_t$</th>
<th>$\mu_t$</th>
<th>$|R^\text{lyap}_t|_F$</th>
<th>$|R^\text{lyap}_t|_F$</th>
<th>$d_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.42868360</td>
<td>-2.42868360+2.34410997i</td>
<td>7.01569259 10^{-2}</td>
<td>3.04824478 10^{-9}</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>2.34714935</td>
<td>-2.34714935+2.16811022i</td>
<td>1.94619867 10^{-3}</td>
<td>2.48023343 10^{-11}</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>2.34884784</td>
<td>-2.34884784+2.16788643i</td>
<td>5.4064151 10^{-5}</td>
<td>8.47787700 10^{-13}</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>2.34887417</td>
<td>-2.34887417+2.16788600i</td>
<td>1.48340711 10^{-5}</td>
<td>5.90476268 10^{-11}</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>2.34887925</td>
<td>-2.34887925+2.16787750i</td>
<td>1.38269262 10^{-6}</td>
<td>2.95296638 10^{-12}</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>2.34887859</td>
<td>-2.34887859+2.16787821i</td>
<td>4.24908453 10^{-7}</td>
<td>9.28992188 10^{-11}</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>2.34887380</td>
<td>-2.34887380+2.16787846i</td>
<td>1.16260310 10^{-7}</td>
<td>2.4430422 10^{-11}</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>2.34887389</td>
<td>-2.34887389+2.16787840i</td>
<td>3.1092454 10^{-8}</td>
<td>7.38695320 10^{-12}</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>2.34887840</td>
<td>-2.34887840+2.16787839i</td>
<td>9.16834770 10^{-9}</td>
<td>1.53824230 10^{-12}</td>
<td>20</td>
</tr>
</tbody>
</table>

| pde ($s^4_{gd} = 10.9704975$) | 0.86144338 | -0.86144338+0.56591285i | 5.06769961 10^{-5} | 2.16042817 10^{-17} | 20  |
| tolusa ($s^4_{gd} = 0.30662268$) | 0.10000000 | -0.1000000+1.90115453i | 1.95291975 10^{-11} | 3.61599084 10^{-11} | 50  |
| tubular ($s^4_{gd} = 0$) | 0.14801641 | -0.14801641+156.007099i | 5.10536526 10^{-9} | 2.5451261 10^{-10} | 480 |

Once in the convergent regime of Algorithm 2, we can switch to shifted inverse iteration with $z_{k-1}$ as the shift to compute $z_k$. The problem with using this approach in the early stages of Algorithm 2 is that $z_k$ and $z_{k-1}$ can still be distant from each other since $z_k$ may have evolved from an eigenvalue of $A + \varepsilon L_{k-2}$ other than $z_{k-1}$. In this scenario, the strategy proposed in this section is more robust.

We summarize this section as follows. The focus of this section is the solution of the eigenvalue problems arising from Algorithm 2 when applied to a real $A$. The eigenvalue solver we consider is the approach developed in [13] which is based on Lyapunov inverse iteration. As the standard inverse iteration, this method has an “inner-outer” structure: the outer iteration is an eigenvalue problem in the form of a Lyapunov equation, and the inner iteration solves a Lyapunov equation. The main advantage of the method of [13] lies in its robustness, and its main cost is solving a few Lyapunov equations, which essentially requires solving linear systems. Efficient ways of solving both have been explored in previous work [11] and are implemented in the numerical experiments here. A disadvantage of this method is that in theory, it is only applicable to real and stable $A$ and yet the perturbed versions of $A$ arising from Algorithm 2 may not be stable even if $A$ is. Again based on eigenvalue perturbation theory, we develop a strategy for shifting the perturbed $A$ so that the shifted matrices are stable and hence Lyapunov inverse iteration can be applied.

We note that in general, the approach described here cannot be applied to the eigenvalue problems arising from Algorithm 1, in which the perturbations $\{L_k\}$ added to $A$ may not be real. This is because the second statement in Theorem 4.1 is only guaranteed to hold for real matrices.

5. An algorithm for computing the real stability radius. In this section, we propose an algorithm for computing the real stability radius of a real and sta-
able matrix. It is inspired by a method for identifying the critical parameter value at which the steady state of a parameter-dependent dynamical system such as (A.1) loses stability. The main tools employed in the new algorithm are Lyapunov inverse iteration and the algorithm for computing the real structured ε-pseudospectral abscissa developed in section 3.

5.1. Detecting the loss of stability of a parameter-dependent dynamical system. A problem that is similar to finding the real stability radius has been considered in [9, 25], where the goal is to find the critical value \( \xi_c \) of a physical parameter \( \xi \in \mathbb{R} \) at which the steady state of a parameter-dependent dynamical system

\[
u_t = A(\xi)u
\]

(5.1)

loses stability. For an example of (5.1), consider again the dynamical system (A.2) arising from the linearization and spatial discretization of the incompressible Navier–Stokes equations (A.1). The Jacobian matrix \( A \) in (A.2) varies with the Reynolds number \( R \). When \( R \) is less than its critical value \( R_c \), the rightmost eigenvalue of the generalized eigenvalue problem (A.3) lies to the left of the imaginary axis, indicating that (A.2) is stable; as \( R \) increases, the rightmost eigenvalue of (A.3) moves toward the imaginary axis, and at \( R = R_c \), the rightmost eigenvalue of (A.3) lies on the imaginary axis, which implies that the steady state of (A.2) has become unstable. For the flow considered in Appendix A, the critical Reynolds number \( R_c \approx 370 \) (see [9]).

Let \( \xi_0 \) be a value of \( \xi \) close to \( \xi_c \) at which \( A(\xi) \) is still stable, i.e., \( \alpha(A(\xi_0)) < 0 \). In the neighborhood of \( \xi_0 \),

\[
A(\xi) \approx A(\xi_0) + (\xi - \xi_0) \frac{dA}{d\xi}
\]

(5.2)

\[
= A_0 + \lambda B_0.
\]

In particular, let \( \lambda_c = \xi_c - \xi_0 \). If we can find \( \lambda_c \), then the critical parameter value \( \xi_c \) can be computed easily as \( \xi_0 + \lambda_c \).

For simplicity, assume \( A(\xi) = A_0 + \lambda B_0 \) in the neighborhood of \( \xi_0 \), where \( A_0, B_0 \), and \( \lambda \) are defined in (5.2). Under this assumption and by the definition of \( \xi_c \) (it is the value of \( \xi \) closest to \( \xi_0 \) such that \( \alpha(A(\xi)) \geq 0 \)), \( \lambda_c \) is the value of \( \lambda \) closest to zero such that \( \alpha(A_0 + \lambda B_0) \geq 0 \). By Theorem 2.1 of [25], \( \lambda_c \) is the real eigenvalue of

\[
SZ + ZS^T + \lambda \left( SZT^T + TZS^T \right) = 0
\]

(5.3)

(where \( S = A_0^{-1} \) and \( T = A_0^{-1}B_0 \)) with smallest modulus. Furthermore, let \( z_c \) be a rightmost eigenvalue of \( A(\xi_c) = A_0 + \lambda_c B_0 \) and assume that there is no other eigenvalue of \( A(\xi_c) \) lying on the imaginary axis besides \( z_c \) and \( \overline{z_c} \). Then by Theorem 2.3 of [25], in the subspace of real and symmetric matrices, \( \lambda_c \) has the unique (up to a scalar multiplier) eigenvector \( Z_c = x_c x_c^T + z_c x_c^T \) which is of rank 1 or 2, where \( x_c \) is the eigenvector of \( A(\xi_c) \) associated with \( z_c \). Therefore, \( \lambda_c \) can be computed using Algorithm 4 outlined below, which is very similar to Algorithm 3. The eigenpair \( (z_c, x_c) \) can also be obtained as a by-product of this algorithm.

5.2. The new algorithm. Throughout this section, the matrix \( A \) is assumed to be a real and stable \( n \times n \) matrix. By the definition of the real stability radius \( r^R(A) \), it is the value of \( \lambda \) closest to zero such that

\[
\alpha^R(\lambda) = \alpha(A + \lambda E) = 0
\]

\[\text{The exact } B_0 \text{ is usually not available and we approximate it using finite difference instead, i.e., } B_0 = \frac{\Delta A(\xi_1) - \Delta A(\xi_0)}{\xi_1 - \xi_0}, \text{ where } \xi_1 \text{ is a value of } \xi \text{ that is close to } \xi_0.\]
Algorithm 4: Lyapunov inverse iteration for (5.3).

1. Given \( \mathcal{W}_0 \in \mathbb{R}^n \) with \( \|\mathcal{W}_0\|_2 = 1 \). Let \( Z_0 = \mathcal{W}_0\mathcal{W}_0^T \).
2. For \( \ell = 1, 2, \ldots \) until convergence:
   2.1. Solve for \( Y_\ell \) from
   \[
   SY_\ell + Y_\ell S^T = -SZ_{\ell-1}T^T - TZ_{\ell-1}S^T
   \]
   in factored form: \( Y_\ell = W_\ell X_\ell W_\ell^T \), where \( W_\ell \in \mathbb{R}^{n \times dl} \) and \( X_\ell \in \mathbb{R}^{dl \times dl} \) with \( dl \ll n \).
   2.2. Rank reduction: let \( \tilde{S} = (W_\ell^T A_0 W_\ell)^{-1} \),
       \( \tilde{T} = (W_\ell^T A_0 W_\ell)^{-1}(W_\ell^T B_0 W_\ell) \), and solve
       \[
       \tilde{S}\tilde{Z} + \tilde{Z}\tilde{S} + \tilde{\lambda}(\tilde{S}\tilde{Z}\tilde{T} + \tilde{T}\tilde{Z}\tilde{S}^T) = 0
       \]
       for the eigenvalue with smallest modulus, \( \tilde{\lambda}_{sm} \), and its eigenvector \( \tilde{Z}_{sm} = \tilde{W}\tilde{X}\tilde{W}^T \), where \( \tilde{W} \in \mathbb{R}^{dl \times q}, \tilde{X} \in \mathbb{R}^{q \times q} \) and \( \|\tilde{X}\|_F = 1 \) with \( q = 1 \) or 2.
   2.3. Set \( \lambda_\ell = \tilde{\lambda}_{sm} \) and \( Z_\ell = W_\ell\tilde{X}W_\ell^T \), where \( W_\ell = W_\ell\tilde{W} \).

for some \( E \in \mathbb{R}^{n \times n} \) with unit norm. Assume \( \alpha_{r^R(A)}(A) = \alpha(A + r^R(A) \cdot \mathcal{E}) = 0 \), where \( \mathcal{E} \in \mathbb{R}^{n \times n} \) and \( \|\mathcal{E}\| = 1 \). (The matrix \( \mathcal{E} \) is not unique.) As a result, for a fixed \( \mathcal{E} \), \( r^R(A) \) is the value of \( \lambda \) closest to zero such that \( \alpha(A + \lambda\mathcal{E}) \geq 0 \). Therefore, finding \( r^R(A) \) is very similar to the problem of finding \( \lambda_c \) described in section 5.1. By an analysis similar to the one given in [25], we can prove the following.

**Theorem 5.1.** Let \( A \in \mathbb{R}^{n \times n} \) be stable (i.e., \( \alpha(A) < 0 \)) and \( \mathcal{E} \in \mathbb{R}^{n \times n} \) be such that \( \|\mathcal{E}\| = 1 \) and
\[
\alpha(A + r^R(A) \cdot \mathcal{E}) = 0.
\]

Then \( r^R(A) \) is the real eigenvalue of
\[
SZ + ZS^T + \lambda(SZT + TZS^T) = 0
\]
(where \( S = A^{-1} \) and \( T = A^{-T}\mathcal{E} \)) with smallest modulus.

Let \( z_c \) denote a rightmost eigenvalue of \( A + r^R(A) \cdot \mathcal{E} \) and let \( x_c \) be its corresponding eigenvector. Assuming there is no other eigenvalue of \( A + r^R(A) \cdot \mathcal{E} \) lying on the imaginary axis besides \( z_c \) and \( z_c^* \), then again by Theorem 2.3 in [25], in the subspace of real and symmetric matrices, \( \lambda_c \) has the unique (up to a scalar multiplier) eigenvector \( Z_c = x_c x_c^* + \tau x_c x_c^T \) which is of rank 1 or 2. Therefore, if \( \mathcal{E} \) is known, we can simply apply Algorithm 4 (with \( A_0 \) changed to \( A \) and \( B_0 \) changed to \( \mathcal{E} \)) to (5.6) to find \( r^R(A) \) and \( (z_c, x_c) \). Unfortunately, \( \mathcal{E} \) is unavailable to us.

In light of the analysis above, we propose an algorithm that alternately generates estimates for \( r^R(A) \) and \( \mathcal{E} \). It is outlined in Algorithm 5 and has the following iterative structure:
(i) it first computes an initial estimate \( E_0 \) for \( \mathcal{E} \), and then at the \( k \)th \( (k \geq 1) \) iteration, it computes
(ii) an estimate \( r_k \) for \( r^R(A) \) based on the previous estimate \( E_{k-1} \) for \( \mathcal{E} \), and
(iii) an estimate \( E_k \) for \( \mathcal{E} \) based on the current estimate \( r_k \) for \( r^R(A) \).
The implementation of each of these three steps is described as follows.

**Steps (i) and (iii).** Since \( \alpha_{r_k}^R(A) = \alpha(A + r_k E_k) \), if \( r_k \) was known, then we could apply Algorithm 2 with \( \varepsilon = r_k \) to find \( E_k \). In step (iii), an estimate \( r_k \) for \( r_k \) is available from step (ii); we therefore apply Algorithm 2 with \( \varepsilon = r_k \) to obtain an estimate \( E_k \) for \( E_0 \) that satisfies \( \alpha_{r_k}^R(A) = \alpha(A + r_k E_k) \). Recall that the initial guess of Algorithm 2 is originally set to a real, rank-2, and norm-1 matrix \( U_0 V_0^T \). where \( U_0, V_0 \in \mathbb{R}^{n \times 2} \) are defined in step 1.2 of Algorithm 2. Here, since an estimate \( E_k \) for \( E_0 \) is available, in order to accelerate the convergence of Algorithm 2, we use \( E_k \) as its initial guess instead.

In step (i), however, an estimate for \( r_k \) is not available and therefore the strategy described above for step (iii) cannot be applied. We simply set \( E_0 = U_0 V_0^T \) because this would be the initial guess for \( E_0 \) computed by Algorithm 2 if \( r_k \) was known, according to (3.2) and Theorem 3.1.

**Step (ii).** By Theorem 5.1, if \( E_0 \) was known, then we could identify \( r_k \) by computing the eigenvalue of (5.6) with smallest modulus. Since we only have an estimate \( E_k \) to \( E_0 \), we compute instead the eigenvalue \( r_k \) of

\[
SZ + ZS^T + \lambda \left( SZT_k + T_k ZS^T \right) = 0
\]

(5.7)

where \( S = A^{-1} \) and \( T_k = A^{-1}E_k \) with smallest modulus and take \( r_k \) to be an estimate to \( r_k \). This eigenvalue can be computed by applying Algorithm 4 (with \( A_0 \) changed to \( A \) and \( B_0 \) changed to \( E_k \)) to (5.7).

**Algorithm 5: Compute the real stability radius of a real and stable \( A \).**

1. **Initialization:**
   1.1. Compute a rightmost eigenvalue \( z_0 \) of \( A \) and its corresponding RP-compatible right and left eigenvectors \( x_0 \) and \( y_0 \).
   1.2. Compute the truncated SVD \( U_0 \Sigma_0 V_0^T \) of
      \( \text{Re}(y_0)(\text{Re}(x_0))^T + \text{Im}(y_0)(\text{Im}(x_0))^T \), where \( U_0, V_0 \in \mathbb{R}^{n \times 2} \) and \( \Sigma_0 \in \mathbb{R}^{2 \times 2} \).
   1.3. Set \( E_0 = U_0 V_0^T \).

2. For \( k = 1, 2, \ldots \) until convergence:
   2.1. Apply Algorithm 4 (with \( A_0 \) changed to \( A \) and \( B_0 \) changed to \( E_k \)) to (5.7).
   2.2. Apply Algorithm 2 (with the initial guess \( L_k = E_k \)) to compute
      \( \alpha_{r_k}^R(A) \) and a real, rank-2 and norm-1 matrix \( E_k \) satisfying
      \( \alpha_{r_k}(A) = \alpha(A + r_k E_k) \).

We test Algorithm 5 on the obstacle, tolosa, and tubular examples as well as the shifted PDE example obtained by subtracting 11I from the matrix of the PDE example, a test case considered in [19]. The stopping criterion for Algorithm 5 is

\[
|\alpha_{r_k}^R(A)| < 10^{-8}
\]

(5.8)

and that for Algorithm 2 is again (A.4). The eigenvalue problems arising from Algorithm 2 are solved using \texttt{eigs} with appropriate parameters. As shown in Table 6, Algorithm 5 suggests that in order to make \( A \) unstable, we need to use real perturbations of norm no less than 0.07235615, 0.02843856, 0.15136334, and 0.81945270 for the four problems, respectively. Note that the estimate for \( r_k \) of the tubular
problem is much smaller than the estimate 0.37047 found by Algorithm 4.1 in [15]. Furthermore, the sequence of estimates generated by Algorithm 5 for each example is monotonically decreasing.

The cost of Algorithm 5 is also reported in Table 6. For instance, when applied to the obstacle example, its main cost is solving 13 instances of the Lyapunov equation (5.4) and computing the rightmost eigenvalues for 38 matrices in the form of $A + r_1E_0$. Also note that using $E_k$ as the initial guess for Algorithm 2 considerably reduces its iteration count in the latter stages of Algorithm 5. Consider the obstacle example again. As can be seen in Table 6, although $r_2$ and $r_4$ are very close to each other, as many as 14 iterations of Algorithm 2 are required to find $\alpha_{r_2}(A)$, whereas only 2 of them are needed for computing $\alpha_{r_4}(A)$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$r_k$</th>
<th>Iter. count of Algorithm 4</th>
<th>Estimated $\alpha_{r_k}^R(A)$</th>
<th>Iter. count of Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.09234569</td>
<td>4</td>
<td>5.17250450 $10^{-2}$</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>0.07259973</td>
<td>3</td>
<td>6.62997609 $10^{-4}$</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>0.07235620</td>
<td>3</td>
<td>1.26529855 $10^{-7}$</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0.07235615</td>
<td>3</td>
<td>3.46812659 $10^{-10}$</td>
<td>2</td>
</tr>
<tr>
<td>Total iter. count:</td>
<td>13</td>
<td>38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| tolusa |
| 1 | 0.10541044 | 1 | 1.82152063 $10^{0}$ | 99* |
| 2 | 0.03213267 | 9* | 2.96168073 $10^{-2}$ | 11 |
| 3 | 0.02844256 | 4 | 4.35856415 $10^{-5}$ | 4 |
| 4 | 0.02843856 | 4 | 4.58014904 $10^{-10}$ | 2 |
| Total iter. count: | 18 | 116 |

| tubular |
| 1 | 0.46251460 | 2 | 1.86477194 $10^{0}$ | 9 |
| 2 | 0.15405341 | 1 | 3.60091904 $10^{-2}$ | 5 |
| 3 | 0.15136461 | 1 | 1.73233310 $10^{-5}$ | 4 |
| 4 | 0.15136334 | 1 | 7.57654173 $10^{-10}$ | 2 |
| Total iter. count: | 5 | 20 |

| shifted PDE |
| 1 | 1.09285924 | 3 | 0.28170019 $10^{-1}$ | 35 |
| 2 | 0.82090248 | 2 | 1.54088817 $10^{-3}$ | 19 |
| 3 | 0.81945274 | 2 | 3.8027998 $10^{-8}$ | 4 |
| 4 | 0.81945270 | 2 | 1.09785858 $10^{-9}$ | 2 |
| Total iter. count: | 9 | 60 |

* Maximum iteration count has been reached before the algorithm converges to the to the desired accuracy.

To further illustrate how the estimates $\{r_k\}$ reported in Table 6 are obtained, we display in Table 7 the numerical results of Algorithm 4 (with $A_0$ changed to $A$ and $B_0$ changed to $E_0$) applied to the instance of (5.7) arising from the first iteration of Algorithm 5, where $T_{k-1} = A^{-1}E_0$. In this table, $\lambda_\ell$ denotes the estimated eigenvalue of (5.7) with smallest modulus, namely, $r_1$, and $\mu_\ell$ is the estimated rightmost eigenvalue of $A + r_1E_0$. We use $\|R^\text{eig}_\ell\|_F < 10^{-8}$ as the stopping criterion of the eigenvalue problem (5.7) and the adaptive stopping criterion $\|R^\text{lyap}_\ell\|_F < 10^{-2} \cdot \|R^\text{eig}_{\ell-1}\|_F$ suggested in [9] for the Lyapunov solve (5.4). The Lyapunov solver used is again the version of the rational Krylov subspace method proposed in [11], and all the linear solves that it entails are done by sparse direct methods. As shown in Table 7, $\lambda_\ell$ converges to 0.09234569, 0.10541044, and 0.46251460, respectively, in the obstacle.
tolosa, and tubular examples, and this is how these three values of $r_1$ shown in Table 6 are obtained. For the shifted PDE example, $r_1$ computed by Algorithm 4 is 2.92910421, which is larger than $-\text{Re}(z_0) = 1.09285924$, an upper bound of $r^R(A)$. Thus, $r_1$ is taken to be 1.09285924 in our computation.

**Table 7**

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$\lambda_1$</th>
<th>$\mu_1$</th>
<th>$|R^{\ell,8}_F|_F$</th>
<th>$|R^{\ell,ap}_F|_F$</th>
<th>$d_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td>1</td>
<td>0.29304130</td>
<td>-9.76819600 $10^{-12}$ +2.00640702i</td>
<td>3.38023822 $10^{-1}$</td>
<td>1.32639087 $10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.09234574</td>
<td>-6.50987559 $10^{-12}$ +2.16750080i</td>
<td>9.35867798 $10^{-5}$</td>
<td>8.19901985 $10^{-5}$</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.09234569</td>
<td>3.28181288 $10^{-12}$ +2.16750252i</td>
<td>1.32639044 $10^{-8}$</td>
<td>1.94077671 $10^{-8}$</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.09234569</td>
<td>3.36142521 $10^{-12}$ +2.16750080i</td>
<td>7.27422150 $10^{-12}$</td>
<td>4.74890641 $10^{-12}$</td>
</tr>
<tr>
<td>tolosa</td>
<td>1</td>
<td>0.10541044</td>
<td>1.51829236 $10^{-12}$ +156.007689i</td>
<td>4.83597926 $10^{-14}$</td>
<td>1.30704466 $10^{-2}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.46313013</td>
<td>4.07151321 $10^{-14}$ +0.86045710i</td>
<td>7.63398011 $10^{-3}$</td>
<td>5.5512502 $10^{-7}$</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.46251460</td>
<td>-4.00524310 $10^{-14}$ +0.85629684i</td>
<td>8.59425340 $10^{-13}$</td>
<td>1.53996433 $10^{-13}$</td>
</tr>
<tr>
<td>shifted PDE</td>
<td>1</td>
<td>0.93292783</td>
<td>-4.38206804 $10^{-14}$ +0.98383417i</td>
<td>2.4625027 $10^{-2}$</td>
<td>3.53857425 $10^{-11}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.92910421</td>
<td>8.04629217 $10^{-13}$ +0.98479826i</td>
<td>2.41366915 $10^{-8}$</td>
<td>8.99250051 $10^{-11}$</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.92910421</td>
<td>-6.28598500 $10^{-13}$ +0.98479826i</td>
<td>4.32325191 $10^{-10}$</td>
<td>2.16077876 $10^{-16}$</td>
</tr>
</tbody>
</table>

**Remark.** As shown in Table 6, the convergence of Algorithm 5 appears to be quadratic, though a proof is currently unavailable. Since Algorithm 2 is not guaranteed to find $\alpha^R_{R^\epsilon}(A)$, Algorithm 5 may also fail to identify $r^R(A)$. As a result, the estimate that it converges to is only guaranteed to be an upper bound of $r^R(A)$. In addition, since Theorem 5.1 is only guaranteed to hold when $A$ is real, this algorithm is in general not applicable to matrices that are not real.

Algorithm 5 is similar in structure to Algorithm 1 of [19] and we compare the two algorithms in Table 8. As in Algorithm 5, an estimate $r_k$ to $r^R(A)$ followed by $\alpha^R_{r_k}(A)$ is computed at each iteration of the algorithm developed in [19]. The step for calculating $r_k$ is derived from Lemma 2.1, and $\alpha^R_{r_k}(A)$ is computed using the method of [18]. We apply this algorithm to the same set of problems and report the estimates to $r^R(A)$ produced by both algorithms as well as their run time in Table 8. The stopping criteria for the algorithm of [19] and the algorithm of [18] are (5.8) and (A.4), as for Algorithms 5 and 2. The eigenvalue problems arising from both the algorithm of [18] and Algorithm 2 are solved using $\textit{eigs}$, and the run time is obtained by the $\textit{tic}$ and $\textit{toc}$ pair in MATLAB.

As shown in Table 8, both algorithms produce the same estimate to $r^R(A)$ in every test case. In terms of run time, Algorithm 5 outperforms Algorithm 1 of [19] in all but one case, requiring 30% to 86% less time to converge. The run time of both algorithms for the obstacle and tolosa examples is significantly longer than that for
Table 8
Comparison of Algorithm 5 and the algorithm of [19] for computing \( r^R(A) \).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Estimated ( r^R(A) )</th>
<th>Run time of Algorithm 5</th>
<th>Estimated ( r^R(A) )</th>
<th>Run time of Algorithm 1 in [19]</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstacle</td>
<td>0.07235615</td>
<td>8h 57m</td>
<td>0.07235616</td>
<td>12h 40m</td>
</tr>
<tr>
<td>tolosa</td>
<td>0.02843856</td>
<td>2h 43m</td>
<td>0.02843856</td>
<td>3h 58m</td>
</tr>
<tr>
<td>tubular</td>
<td>0.15136334</td>
<td>1m 19s</td>
<td>0.15136334</td>
<td>9m 24s</td>
</tr>
<tr>
<td>shifted PDE</td>
<td>0.81945270</td>
<td>1m 6s</td>
<td>0.81945271</td>
<td>1m 4s</td>
</tr>
</tbody>
</table>

the other two cases mainly due to the fact that the eigenvalue problems arising from these two examples are much more difficult to solve. To find the rightmost eigenvalue for a matrix in Algorithm 2 or the algorithm of [18], we compute 100 eigenvalues with the smallest moduli in the obstacle problem and 150 eigenvalues with the largest real parts in the tolosa problem, whereas only 20 eigenvalues with the largest real parts are computed in the other two examples.

6. Conclusion. This paper is built on several recent developments in algorithms for pseudospectral analysis [18, 20] and linear stability analysis [9, 11, 13, 25]. Our main contributions are two new algorithms for examining the influence of real perturbations on the stability of large and sparse matrices: one for computing the real structured pseudospectral abscissa and the other for computing the real stability radius. The first algorithm is derived based on eigenvalue perturbation theory and requires computing the rightmost eigenvalues for a sequence of matrices. We show that the method developed in [13] can be applied to obtain reliable solutions to these eigenvalue problems. The algorithm for computing the real stability radius is motivated by the observation that it is the eigenvalue with smallest modulus of an eigenvalue problem in the form of a Lyapunov equation, which is similar to a problem studied in previous work [9, 25]. Its main cost is solving a number of Lyapunov equations and eigenvalue problems for their rightmost eigenvalues. Both algorithms have their advantages compared to existing methods.

Appendix A. Examples arising from incompressible flows. In this section, we apply Algorithm 1 to matrices arising from spatial discretization of the Navier–Stokes equations modeling incompressible flows, i.e.,

\[
\begin{align*}
    u_t - \nu \nabla^2 u + u \cdot \nabla u + \nabla p &= 0, \\
    \nabla \cdot u &= 0,
\end{align*}
\]

subject to appropriate boundary conditions, where \( u, p, \nu \) denote the velocity, pressure, and kinematic viscosity, respectively.

To perform linear stability analysis for a steady state \( \overline{u} \) of (A.1), we linearize a discrete version of (A.1) in a small neighborhood of \( \overline{u} \), which leads to a linear dynamical system

\[
M u_t = A u,
\]

where \( A \in \mathbb{R}^{n \times n} \) is the Jacobian matrix and \( M \in \mathbb{R}^{n \times n} \) is often referred to as the mass matrix. Both \( A \) and \( M \) are large and sparse, and \( A \) is nonsymmetric. Eigenvalue analysis shows that the stability of \( \overline{u} \) is determined by the rightmost eigenvalue of a generalized eigenvalue problem

\[
A z = \mu M z.
\]
If it lies to the left of the imaginary axis, then \( \mathbf{z} \) is stable; otherwise, \( \mathbf{z} \) is unstable. The mass matrix \( \mathbf{M} \) is singular, but a transformation can be applied that makes it nonsingular and also maps all the infinite eigenvalues of (A.3) to any finite number of our choice without affecting the finite eigenvalues of (A.3) (see [5]). In our computation, such a transformation is used that maps all the infinite eigenvalues of (A.3) to \(-100\) which is away from the imaginary axis, and from here on, \( \mathbf{M} \) denotes this nonsingular mass matrix. Therefore, (A.3) and \( \mathbf{M}^{-1}\mathbf{A} \) have the same eigenpairs.

The example we consider is the two-dimensional flow over a square obstacle in a rectangular channel, a test problem provided in the MATLAB package IFISS [10]. A Poiseuille flow profile is imposed on the inflow boundary, and a no-flow condition is imposed on the walls and the obstacle. A Neumann condition is applied at the outflow boundary which automatically sets the mean outflow pressure to zero (see Chapter 7 of [7]). The incompressible Navier–Stokes equations (A.1) are discretized using \( Q_2-Q_1 \) finite elements on a \( 32 \times 128 \) mesh, resulting in a linear dynamical system (A.2) of dimension \( n = 9,512 \). The streamlines of the steady state at Reynolds number \( \mathcal{R} = 350 \) are illustrated in Figure 3. (The Reynolds number is defined to be \( \frac{UL}{\nu} \), where \( U \) is the maximum magnitude of velocity on the inflow and \( L \) is a characteristic length scale for the domain. In this particular example, \( L = 2a \) and \( U = 1 \) so that \( \mathcal{R} = \frac{2}{\nu} \).) Linear stability of this flow was analyzed in [9], which shows that it will lose its stability to a Hopf bifurcation point (see Chapter 4 of [16]) at \( \mathcal{R} \approx 370 \).

Numerical results of Algorithm 1 applied to \( \mathbf{A} = \mathbf{M}^{-1}\mathbf{A} \) at \( \mathcal{R} = 200 \) are summarized as follows. We adopt the stopping criterion used in [20]:

\[
\text{Re}(z_k) - \text{Re}(z_{k-1}) < 10^{-8} \cdot \max(1, \text{Re}(z_{k-1}))
\]

When \( \varepsilon = 10^{-2} \), Algorithm 1 converges in five iterations, generating a sequence of \( z_k \) with monotonically increasing real parts. They are plotted in Figures 4(a) and 4(b) (dots), with the leftmost one being \( z_0 \), the rightmost eigenvalue of \( \mathbf{M}^{-1}\mathbf{A} \). The exact rightmost point of \( \Lambda_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \) computed by EigTool [35] (which implements the criss-cross algorithm [4]) is also plotted in the same set of figures (crosses) for comparison. Moreover, the rightmost part of the boundary of \( \Lambda_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \) produced by EigTool is included in these figures as well.\(^4\) As shown in Figure 4(b), \( z_k \) first moves to the right almost horizontally; once \( \text{Re}(z_k) \) becomes fairly close to \( \alpha_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \), it starts to move vertically toward the rightmost point of \( \Lambda_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \), matching the pattern observed in [20]. When \( \varepsilon = 10^{-1} \), the iterates \( \{z_k\} \) computed by Algorithm 1, the exact rightmost point of \( \Lambda_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \), and the rightmost part of the boundary of \( \Lambda_{\varepsilon}(\mathbf{M}^{-1}\mathbf{A}) \) are illustrated in Figures 4(c) and 4(d). For this larger value of \( \varepsilon \), it takes Algorithm 1 many more iterations (20 instead of 5) to meet the same stopping criterion (A.4). Nonetheless, the convergence behaviors of \( \{z_k\} \) observed in Figures 4(b) and 4(d) are qualitatively similar. Moreover, \( \alpha(\mathbf{M}^{-1}\mathbf{A}) \) and \( \alpha_{0.01}(\mathbf{M}^{-1}\mathbf{A}) \) are both negative, whereas \( \alpha_{0.1}(\mathbf{M}^{-1}\mathbf{A}) > 0 \).

\(^4\)EigTool cannot be applied directly to generalized eigenvalue problems. Here we explicitly form the matrix \( \mathbf{M}^{-1}\mathbf{A} \) and then pass it to EigTool.
Since $\alpha(M^{-1}A) < 0$, eigenvalue analysis predicts that the steady state of (A.2) at $R = 200$ is stable. Pseudospectral analysis, on the other hand, tells us the following: if a perturbation of size no more than $10^{-2}$ has been introduced to $M^{-1}A$, this steady state will indeed be stable in practice; however, if a perturbation of size $10^{-1}$ exists in $M^{-1}A$, in reality, the steady state could be unstable already.

Some implementation details of Algorithm 1 are discussed below. When applying Algorithm 1 to $A = M^{-1}A$, we first need to compute the rightmost eigenvalue $z_0$ of (A.3) and an RP-compatible pair of right and left eigenvectors $(x_0, y_0)$ associated with $z_0$. Then at each iteration, we need to compute the rightmost eigenvalue $z_k$ of

$$(A + \varepsilon ML_{k-1})x = \mu Mx$$

and an RP-compatible pair of right and left eigenvectors $(x_k, y_k)$ corresponding to $z_k$. In [20], MATLAB routine eigs (which implements the implicitly restarted Arnoldi method [31]) with the option lr (short for “largest real part”) or lm (short for “largest modulus”) was used. Under these two choices, the search space of eigs is the Krylov subspace of $A$ which tends to favor the well-separated, extremal eigenvalues of (A.3) (see [29]). However, there are many examples including flow over an obstacle where the rightmost eigenvalue is neither well-separated nor extremal. Thus, this approach may suffer from slow convergence or even convergence to a wrong eigenvalue. A
technique commonly used to enhance the convergence of a target eigenvalue is to apply the eigenvalue solver to a transformation of the original matrix instead (see [24] and the references therein). The shift-invert transformation \((A - \sigma I)^{-1}\), for instance, accelerates the convergence of the eigenvalues of \(A\) near a shift \(\sigma \in \mathbb{C}\) (see Chapter 2 of [33]).

In order to find the rightmost eigenvalue of (A.3) or (A.5), we apply \texttt{eigs} with the option \texttt{sm} (short for “smallest modulus”) to compute 100 eigenvalues of (A.3) or (A.5) with smallest moduli. This approach requires linear solves with \(A\) or \(A + \varepsilon ML_{k-1}\), which are done using preconditioned GMRES: the LSC preconditioner (see [8, 12]) for \(A\) is used to precondition both \(A\) and \(A + \varepsilon ML_{k-1}\), and solves with the preconditioner are approximated by one V-cycle of algebraic multigrid (see Chapter 8 of [7]). The stopping criteria for GMRES when solving \(Ax = b\) and \((A + \varepsilon ML_{k-1})x = b\) are \(\|A\hat{x} - b\| < 10^{-10} \cdot \|b\|\) and \(\|(A + \varepsilon ML_{k-1})\hat{x} - b\| < 10^{-10} \cdot \|b\|\), respectively, where \(\hat{x}\) is the computed solution.

Acknowledgment. The author thanks both referees for their many insightful comments and suggestions, which lead to major improvements of the paper. She also thanks Nicola Guglielmi and Manuela Manetta for sharing their codes and the Department of Mathematical Sciences at Worcester Polytechnic Institute for providing travel funds.

REFERENCES