IMA Journal of Numerical Analysis (2016) **36**, 985–1014
doi:10.1093/imanum/drv046
Advance Access publication on September 15, 2015

Hybrid expansion–contraction: a robust scaleable method for approximating the H_∞ norm

Tim Mitchell and Michael L. Overton*
Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA
*Corresponding author: tim.mitchell@cims.nyu.edu overton@cims.nyu.edu

[Received on 23 October 2014; revised on 28 April 2015]

We present a new scaleable algorithm for approximating the H_∞ norm, an important robust stability measure for linear dynamical systems with input and output. Our spectral-value-set-based method uses a novel hybrid expansion–contraction scheme that, under reasonable assumptions, is guaranteed to converge to a stationary point of the optimization problem defining the H_∞ norm, and, in practice, typically returns local or global maximizers. We prove that the hybrid expansion–contraction method has a quadratic rate of convergence that is also confirmed in practice. In comprehensive numerical experiments, we show that our new method is not only robust but exceptionally fast, successfully completing a large-scale test set 25 times faster than an earlier method by Guglielmi, Gürbüzbalaban & Overton (2013, *SIAM J. Matrix Anal. Appl.*, **34**, 709–737), which occasionally breaks down far from a stationary point of the underlying optimization problem.

Keywords: complex stability radius; pseudospectra; robust stability.

1. Introduction

Consider the continuous-time linear dynamical system with input and output defined by

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bw(t), \\ z(t) &= Cx(t) + Dw(t), \end{aligned} \tag{1.1}$$

and the discrete-time analogue

$$\begin{aligned} x_{k+1} &= Ax_k + Bw_k, \\ z_k &= Cx_k + Dw_k, \end{aligned} \tag{1.2}$$

where $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times p}$, $C \in \mathbb{C}^{m\times n}$, $D \in \mathbb{C}^{m\times p}$ and w is a disturbance feedback depending linearly on the output z (Hinrichsen & Pritchard, 2005, p. 538). We assume that the matrix A is Hurwitz stable (all its eigenvalues are in the open left half-plane) in the continuous-time case or Schur stable (all its eigenvalues are in the open unit disc) in the discrete-time case.

In this paper, we present a new spectral-value-set-based algorithm for approximating the H_∞ norm of the transfer function associated with these systems, which to compute exactly is tantamount to finding a global optimum of a nonconvex and nonsmooth optimization problem. Our new method is intended to be used for large and sparse systems where it is not feasible to use the standard Boyd–Balakrishnan–Bruinsma–Steinbuch (BBBS) algorithm (Boyd & Balakrishnan, 1990; Bruinsma & Steinbuch, 1990) due to its cubic cost per iteration. There has been a recent flurry of interest in approximating the H_∞

norm for large and sparse systems without reducing the dimensions of the original matrices, notably by Guglielmi *et al.* (2013) and, for related descriptor systems, by Benner & Voigt (2014) and Freitag *et al.* (2014). Though the last of these algorithms relies on solving linear systems, and is thus unlikely to scale as well as the first two methods, all three methods are much faster than the BBBS algorithm for large problems and are able to compute good approximations to the H_∞ norm.

In the continuous-time case, the algorithm of Guglielmi *et al.* (2013), which we call the GGO method, consists of an inner iteration to approximate the so-called ε-spectral value set abscissa (defined in Section 2) for a given $\varepsilon > 0$, and an outer iteration to vary ε using a Newton-bisection method to approximate the complex stability radius (also defined below), which is the reciprocal of the H_∞ norm. Because there is no guarantee that the inner iteration will return the correct value of the spectral value set abscissa, the authors state that the method is an idealized algorithm (Guglielmi *et al.*, 2013, p. 729). The crux of the matter, however, is that even if the inner iteration does deliver locally optimal values of the spectral value set abscissa subproblem (the most that can be reasonably assumed), the outer Newton-bisection method is still not guaranteed to converge to a stationary point of the underlying optimization problem. In practice, the GGO algorithm occasionally breaks down far from such a stationary point, leading to a poor approximation of the H_∞ norm.

While our new spectral-value-set-based algorithm also makes use of the ε-spectral value set abscissa approximation subroutine of the GGO method, we instead propose a novel hybrid expansion–contraction iteration which not only provides provably breakdown-free convergence but also, generically, has a quadratic rate of convergence. Furthermore, we present improvements to the main subroutine shared by both methods, accelerating its linear convergence and extending it to systems where both p and m may be large.

The paper is organized as follows. In the next section, we define spectral value sets and then explain their fundamental properties and their relationship to the H_∞ norm. In Section 3, we outline the algorithm given in Guglielmi *et al.* (2013) and explain how it may break down. Then in Section 4, we present our new method along with its theoretical guarantees. In Section 5, we present a key lemma underpinning the convergence rate analysis of our method given in Section 4.2. In Section 6, we show how both methods can be extended to handle systems with a large number of inputs and/or outputs and accelerated via vector extrapolation methods. We provide implementation notes in Section 7 and present numerical results in Section 8.

2. Spectral value sets and their relationship to the H_∞ norm

This section follows the development in Hinrichsen & Pritchard (2005, Section 5.1) and Guglielmi *et al.* (2013, Section 2). We use the following notation: $\mathbb{R}^+$ denotes the set of non-negative real numbers while $\mathbb{R}^{++}$ denotes the set of strictly positive real numbers, $\sigma(\cdot)$ is the spectrum of a matrix and $\|\cdot\|$ is the 2-norm. The dimension of the identity matrix I depends on the context.

Given matrices A, B, C, D defining the linear dynamical system (1.1), consider the *perturbed system matrix*[1]

$$M(\Delta) = A + B\Delta(I - D\Delta)^{-1}C \quad \text{for } \Delta \in \mathbb{C}^{p\times m}, \tag{2.1}$$

[1] To motivate this formula, write $w = \Delta z$ and observe that it then follows from (1.1) that $\dot{x} = M(\Delta)x$ or from (1.2) that $x_{k+1} = M(\Delta)x_k$ (Hinrichsen & Pritchard, 2005, p. 538).

assuming $I - D\Delta$ is invertible, and the associated *transfer matrix* (Hinrichsen & Pritchard, 2005, p. 549)

$$G(\lambda) = C(\lambda I - A)^{-1}B + D \quad \text{for } \lambda \in \mathbb{C}\backslash\sigma(A).$$

DEFINITION 2.1 Let $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon\|D\| < 1$, and define the *spectral value set*

$$\sigma_\varepsilon(A, B, C, D) = \bigcup\{\sigma(M(\Delta)) : \Delta \in \mathbb{C}^{p\times m}, \|\Delta\| \leqslant \varepsilon\}.$$

The sets σ_ε are called spectral value sets in Hinrichsen & Pritchard (2005) and Karow (2003), and are also sometimes known as structured pseudospectra. In the special case $B = I$, $C = I$, $D = 0$, the sets σ_ε are called pseudospectra (Trefethen & Embree, 2005). In contrast to the references just mentioned, our use of nonstrict inequalities above implies that the set $\sigma_\varepsilon(A, B, C, D)$ is compact for fixed ε.

In fact, Δ above can be taken to have rank 1, as stated in the next result relating eigenvalues of a perturbed system matrix to singular values of an associated transfer matrix. This lemma follows from Hinrichsen & Pritchard (2005, Theorem 5.2.9 and Remark 5.2.20 (iii)) or Guglielmi *et al.* (2013, Theorem 2.1 and Corollary 2.4). See Guglielmi *et al.* (2013, Theorem 2.9) for a discussion of the relationship between the eigenvectors of the perturbed system matrix and singular vectors of the transfer matrix.

LEMMA 2.2 Let $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon\|D\| < 1$. Then

$$\sigma_\varepsilon(A, B, C, D)\backslash\sigma(A) \equiv \bigcup\{\sigma(M(\Delta)) : \Delta \in \mathbb{C}^{p\times m}, \|\Delta\| \leqslant \varepsilon, \operatorname{rank}(\Delta) = 1\} \tag{2.2}$$

$$\equiv \bigcup\{\lambda \in \mathbb{C}\backslash\sigma(A) : \|G(\lambda)\| \geqslant \varepsilon^{-1}\}. \tag{2.3}$$

Furthermore, given $\lambda \in \mathbb{C}\backslash\sigma(A)$ with $\|G(\lambda)\| = \varepsilon^{-1}$, we can obtain Δ with rank 1 such that $\|\Delta\| = \varepsilon$ and $\lambda \in \sigma(M(\Delta))$ by setting $\Delta = \varepsilon uv^*$, where $\varepsilon G(\lambda)u = v$ and $\varepsilon v^* G(\lambda) = u^*$, that is, u and v are, respectively, right and left singular vectors of $G(\lambda)$ corresponding to its largest singular value ε^{-1}.

DEFINITION 2.3 Suppose that $\lambda \in \mathbb{C}\backslash\sigma(A)$ is given with $\|G(\lambda)\| = \varepsilon^{-1}$. We say that the *simplicity condition* holds at λ with respect to ε if

(1) the largest singular value ε^{-1} of $G(\lambda)$ is simple;

(2) letting u and v be corresponding right and left singular vectors and setting $\Delta = \varepsilon uv^*$, the eigenvalue λ of $M(\Delta)$ is simple.

DEFINITION 2.4 The *spectral abscissa* of the matrix A is

$$\alpha(A) = \max\{\operatorname{Re}(\lambda) : \lambda \in \sigma(A)\}.$$

For $\varepsilon \in \mathbb{R}^+$ with $\varepsilon\|D\| < 1$, the *spectral value set abscissa* is

$$\alpha_\varepsilon(A, B, C, D) := \max\{\operatorname{Re}(\lambda) : \lambda \in \sigma_\varepsilon(A, B, C, D)\} \tag{2.4}$$

$$\equiv \max\{\operatorname{Re}(\lambda) : \lambda \in \sigma(A) \text{ or } \|G(\lambda)\| \geqslant \varepsilon^{-1}\}, \tag{2.5}$$

with the equivalence following by Lemma 2.2 and $\alpha_0(A, B, C, D) = \alpha(A)$.

DEFINITION 2.5 The *spectral radius* of the matrix A is

$$\rho(A) = \max\{|\lambda| : \lambda \in \sigma(A)\}.$$

For $\varepsilon \in \mathbb{R}^+$ with $\varepsilon\|D\| < 1$, the *spectral value set radius* is

$$\rho_\varepsilon(A,B,C,D) := \max\{|\lambda| : \lambda \in \sigma_\varepsilon(A,B,C,D)\} \tag{2.6}$$

$$\equiv \max\{|\lambda| : \lambda \in \sigma(A) \text{ or } \|G(\lambda)\| \geqslant \varepsilon^{-1}\}, \tag{2.7}$$

with the equivalence following by Lemma 2.2 and $\rho_0(A,B,C,D) = \rho(A)$.

We now define the stability radius (Hinrichsen & Pritchard, 2005, Section 5.3) (often known as the complex stability radius: 'complex' because complex perturbations are admitted even if the data are real, and 'radius' in the sense of the perturbation space, not the complex plane). It is the largest ε such that the spectral value set $\sigma_\varepsilon(A,B,C,D)$ is defined and contained in the left half-plane (for the continuous-time system (1.1)) or in the unit disc (for the discrete-time system (1.2)).

DEFINITION 2.6 The *stability radius* for (A,B,C,D) is

$$\varepsilon_\star := \begin{cases} \sup\{\varepsilon : \varepsilon\|D\| < 1 \text{ and } \alpha_\varepsilon(A,B,C,D) < 0\} & \text{(continuous-time case)}, \\ \sup\{\varepsilon : \varepsilon\|D\| < 1 \text{ and } \rho_\varepsilon(A,B,C,D) < 1\} & \text{(discrete-time case)}. \end{cases}$$

When $B = C = I$ and $D = 0$ this quantity is also known as the *distance to instability* (Hinrichsen & Pritchard, 2005, Section 5.3.5; VanLoan, 1985) for the matrix A.

Now we define the H_∞ norm.

DEFINITION 2.7 The H_∞ *norm of the transfer matrix function* G is

$$\|G\|_\infty := \begin{cases} \sup\limits_{\omega\in\mathbb{R}} \|G(\mathrm{i}\omega)\| & \text{(continuous-time case)}, \\ \sup\limits_{\theta\in[0,2\pi]} \|G(e^{\mathrm{i}\theta})\| & \text{(discrete-time case)}. \end{cases} \tag{2.8}$$

Both the algorithm of Guglielmi *et al.* (2013) and the one presented here exploit the following well-known property.

LEMMA 2.8 The stability radius and the H_∞ norm are reciprocals of each other, that is,

$$\varepsilon_\star = \|G\|_\infty^{-1}. \tag{2.9}$$

For a proof, see Guglielmi *et al.* (2013, Lemma 2.17). Note that since we assumed that A is stable, the stability radius is always positive and the H_∞ norm is always finite.

2.1 *Locally rightmost and outermost points of spectral value sets*

We now consider locally rightmost or locally outermost points of spectral value sets and how they relate to the norm of the transfer function.

DEFINITION 2.9 A *rightmost* (*outermost*) point of a set $S \subset \mathbb{C}$ is a point where the maximal value of the real part (modulus) of the points in S is attained. A *locally rightmost* (*locally outermost*) point of a set $S \subset \mathbb{C}$ is a point λ which is a rightmost (outermost) point of $S \cap \mathcal{N}$ for some neighborhood $\mathcal{N}$ of λ.

REMARK 2.10 Since $\sigma_\varepsilon(A, B, C, D)$ is compact, its locally rightmost or locally outermost points, that is, the local maximizers of the optimization problems in (2.4) and (2.6), lie on its boundary. There can be only a finite number of these; otherwise, the boundary would need to contain an infinite number of points with the same real part or modulus, which can be ruled out by an argument similar to Guglielmi & Overton (2011, Lemma 2.5), exploiting Hinrichsen & Pritchard (2005, Lemma 5.3.30).

In the following, we make use of the simplicity condition of Definition 2.3. This lemma is taken from Guglielmi *et al.* (2013, Lemma 2.21), although the assumptions are stated differently.

LEMMA 2.11 Suppose that $\lambda \in \mathbb{C} \backslash \sigma(A)$ is given with $\|G(\lambda)\| = \varepsilon^{-1}$ and that the simplicity condition holds at λ with respect to ε. A necessary condition for λ to be a local maximizer of the optimization problem in (2.5) is

$$v^* C(\lambda I - A)^{-2} B u \in \mathbb{R}^{++}, \tag{2.10}$$

where u and v are, respectively, right and left singular vectors corresponding to the largest singular value ε^{-1} of $G(\lambda)$.

We have not seen the following lemma stated explicitly before, although it is clear from a geometrical perspective.

LEMMA 2.12 Suppose that $\lambda \in \mathbb{C} \backslash \sigma(A)$ is given with $\|G(\lambda)\| = \varepsilon^{-1}$ and that the simplicity condition holds at λ with respect to ε. If λ satisfies the first-order necessary condition of Lemma 2.11 and $\mathrm{Re}(\lambda) = 0$, then $\mathrm{Im}(\lambda)$ is a stationary point of $\|G(\mathrm{i}\omega)\|$, and furthermore, if λ is a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ with $\mathrm{Re}(\lambda) = 0$, then $\mathrm{Im}(\lambda)$ is a local maximizer of $\|G(\mathrm{i}\omega)\|$.

Proof. Under the assumptions, it is straightforward to differentiate $\|G(\mathrm{i}\omega)\|$ and show that the derivative is zero at $\omega = \mathrm{Im}(\lambda)$ using the techniques given in the proof of Guglielmi *et al.* (2013, Lemma 2.21). For the second part, suppose that λ is a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ with $\mathrm{Re}(\lambda) = 0$ and write $\lambda_\mathrm{I} = \mathrm{Im}(\lambda_\mathrm{I})$. Suppose λ_I is not a local maximizer of $\|G(\mathrm{i}\omega)\|$, that is, there exists $\delta \in \mathbb{R}^{++}$ such that $\|G(\mathrm{i}\lambda_\mathrm{I})\| < \|G(\mathrm{i}(\lambda_\mathrm{I} + t\delta))\|$ for all $t \in [0, 1]$. By Lemma 2.2, all points $\mathrm{i}(\lambda_\mathrm{I} + t\delta) \in \sigma_\varepsilon(A, B, C, D)$ and since $\lambda = \mathrm{i}\lambda_\mathrm{I}$ is a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ by assumption, for some sufficiently small value $\hat{t} \in \mathbb{R}^{++}$ we have that the points $\mathrm{i}(\lambda_\mathrm{I} + t\delta)$ for $0 \leqslant t < \hat{t}$ must also be locally rightmost. Thus, $\sigma_\varepsilon(A, B, C, D)$ must have an infinite number of locally rightmost points, contradicting Remark 2.10. □

The next lemma, the discrete-time variant of Lemma 2.11, follows from Guglielmi *et al.* (2013, Lemma 2.29).

LEMMA 2.13 Suppose that $\lambda \in \mathbb{C} \backslash \sigma(A)$ is given with $\|G(\lambda)\| = \varepsilon^{-1}$ and that the simplicity condition holds at λ with respect to ε. A necessary condition for λ to be a local maximizer of the optimization problem in (2.7) is

$$\lambda(v^* C(\lambda I - A)^{-2} B u) \in \mathbb{R}^{++}, \tag{2.11}$$

where u and v are, respectively, right and left singular vectors corresponding to the largest singular value ε^{-1} of $G(\lambda)$.

LEMMA 2.14 Suppose that $\lambda \in \mathbb{C} \backslash \sigma(A)$ is given with $\|G(\lambda)\| = \varepsilon^{-1}$ and that the simplicity condition holds at λ with respect to ε. If λ satisfies the first-order necessary condition of Lemma 2.13 and

$|\lambda| = 1$, then $\angle\lambda$ is a stationary point of $\|G(e^{i\theta})\|$, and furthermore, if λ is a locally outermost point of $\sigma_\varepsilon(A, B, C, D)$ with $|\lambda| = 1$, then $\angle\lambda$ is a local maximizer of $\|G(e^{i\theta})\|$. Here $\angle$ denotes complex argument.

The proof is similar to the proof of Lemma 2.12.

3. The algorithm of Guglielmi, Gürbüzbalaban and Overton

Since the spectral value set abscissa and radius, $\alpha_\varepsilon(A, B, C, D)$ and $\rho_\varepsilon(A, B, C, D)$, are monotonically increasing functions of ε, the stability radius (the reciprocal of $\|G\|_\infty$ by Lemma 2.8) can be found by finding the root of

$$g(\varepsilon) := \begin{cases} \alpha_\varepsilon(A, B, C, D) & \text{(continuous-time case)}, \\ \rho_\varepsilon(A, B, C, D) - 1 & \text{(discrete-time case)}. \end{cases} \tag{3.1}$$

However, like the standard BBBS algorithm for computing the H_∞ norm, computing either $\alpha_\varepsilon(A, B, C, D)$ or $\rho_\varepsilon(A, B, C, D)$ to guaranteed precision[2] apparently requires a cubic order of operations and thus the benefit of this alternative formulation of calculating the H_∞ norm is not immediately apparent. In order to scale to large-dimensional problems, the GGO algorithm introduces two related methods, SVSA1 and SVSR1, to instead, respectively *approximate* $\alpha_\varepsilon(A, B, C, D)$ and $\rho_\varepsilon(A, B, C, D)$, by generalizing the fast methods of Guglielmi & Overton (2011) for approximating the pseudospectral abscissa and radius of a large sparse matrix. The benefit is that the GGO algorithm can forgo computing the norm of the transfer function, which involves the potentially expensive calculation of $(\lambda I - A)^{-1}B$ at any iterate λ, and instead requires only the computation of a rightmost (or outermost) eigenvalue of $M(\Delta)$ evaluated for a sequence of rank 1 matrices, for which sparse eigenvalue solvers based on matrix–vector products such as Lehoucq & Sorensen (1996) may be used.

Then, using SVSA1 or SVSR1 as a subroutine to attempt to compute either $\alpha_\varepsilon(A, B, C, D)$ or $\rho_\varepsilon(A, B, C, D)$ respectively, the GGO algorithm uses a hybrid Newton-bisection algorithm to try to find the root $\varepsilon_\star$ of g. However, as both SVSA1 and SVSR1 often converge to points that are, respectively, only *locally* rightmost or outermost, there is no guarantee that the Newton-bisection method will converge to $\varepsilon_\star$, although, when it does not, it still typically converges to a local maximizer of (2.8). Unfortunately, as we will describe in Section 3.2, the GGO method can also sometimes critically break down before convergence to even a local maximizer is reached.

3.1 *Approximating the spectral value set abscissa and radius*

The SVSA1 and SVSR1 algorithms are based on the following observation: by (2.2), $\sigma_\varepsilon(A, B, C, D)\backslash\sigma(A)$ may be characterized solely by rank-1 perturbations and thus for $\Delta = \varepsilon uv^*$, by (2.1) and Guglielmi *et al.* (2013, Lemma 2.8),

$$M(\Delta) = A + B\tilde{\Delta}C \quad \text{where } \tilde{\Delta} = \frac{\varepsilon uv^*}{1 - \varepsilon v^* Du}, \tag{3.2}$$

where $\varepsilon \in \mathbb{R}^{++}$ with $\varepsilon\|D\| < 1$ and vectors $u \in \mathbb{C}^p$, $v \in \mathbb{C}^m$ are normalized so that $\|u\| = \|v\| = 1$. So, focusing on the continuous-time case, the idea of the SVSA1 algorithm is to construct a sequence of rank-1 perturbations $u^j(v^j)^*$ that 'push' the rightmost eigenvalue of $M(\Delta)$ as far to the right as possible. At each step of the iteration, using the current u^j and v^j, the algorithm computes the rightmost

[2] For the case $B = C = I$, $D = 0$, see Burke *et al.* (2003) and Mengi & Overton (2005).

eigenvalue of $M(\varepsilon u^j (v^j)^*)$ as well as its corresponding right and left eigenvectors, which together comprise what we call an *eigentriple*. Then, as explained in Guglielmi *et al.* (2013, Section 3), the next iterates u^{j+1} and v^{j+1} are obtained by explicitly solving a nonlinear maximization problem defined by the eigentriple information. If necessary, a line search is then invoked to modify u^{j+1} and v^{j+1} to ensure that $\alpha(M(\varepsilon u^{j+1}(v^{j+1})^*)) > \alpha(M(\varepsilon u^j (v^j)^*))$. The line search is explained in Guglielmi *et al.* (2013, Section 3.5); some improvements for accelerating the line search and increasing its robustness are given in Mitchell (2014, Sections 2.2 and 4.3, Procedures 2, 3 and 4).

REMARK 3.1 For the SVSA1 and SVSR1 algorithms, let λ^j denote the rightmost or outermost eigenvalue of $M(\varepsilon u^j (v^j)^*)$, respectively. The algorithms as they appear in Guglielmi *et al.* (2013) are not provided with initial perturbation vectors u^0 and v^0 defining an initial λ^0, but instead with eigenvector information that initializes the process. The result is that, although the line search ensures that, for $j = 1, 2, \ldots, \mathrm{Re}(\lambda^{j+1}) > \mathrm{Re}(\lambda^j)$ or $|\lambda^{j+1}| > |\lambda^j|$ respectively, no guarantee is made with respect to an initial eigenvalue λ^0. We therefore modify the algorithms so that they can instead be initialized with ε, u^0, v^0 and λ^0 (though λ^0 can be recomputed from ε, u^0 and v^0); the line search then ensures that monotonicity also holds with respect to λ^0. Though this seemingly minor modification usually has no significant impact, negative or positive, on the GGO algorithm, and does not prevent its potential breakdown case, ensuring monotonicity from a given starting point λ^0 is a crucial property needed for the new method we propose in this paper. As such, and for brevity, in the context of the GGO algorithm, we will use SVSAR to refer to SVSA1 or SVSR1 respectively, or, in the context of our new method, to refer to their fully monotonic modified versions described in this remark, including the additional line search improvements referenced above.

Despite its monotonicity properties, SVSAR is known to converge, linearly, only to rightmost points of $\sigma_\varepsilon(A, B, C, D)$ if ε is sufficiently small, and there is no guarantee that such points will be globally rightmost. In practice though, without assuming ε is small, we observe that SVSAR does often converge to a globally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ (in the cases where we have been able to test this) and furthermore, in cases where we have observed that it does not, it still typically finds locally rightmost points which provide good approximations to the globally optimal value. Assuming this is always the case would be too strong, so instead we make the following assumption, which we will use in the next section.

ASSUMPTION 3.2 Suppose the SVSAR algorithm is given as input $\varepsilon > 0$ satisfying $\varepsilon\|D\| < 1$ and unit-norm vectors u^0 and v^0 with λ^0 the rightmost or outermost eigenvalue of $M(\varepsilon^0 u^0 (v^0)^*)$. Then it returns unit-norm vectors u and v and $\lambda \in \mathbb{C}$ such that $\|G(\lambda)\| = \varepsilon^{-1}$ with corresponding right and left singular vectors u and v, with λ satisfying the simplicity condition with respect to ε, and such that

(1) in the continuous-time case, λ is a rightmost eigenvalue of $M(\varepsilon u v^*)$ and the first-order necessary condition of Lemma 2.11 holds;

(2) in the discrete-time case, λ is an outermost eigenvalue of $M(\varepsilon u v^*)$ and the first-order necessary condition of Lemma 2.13 holds.

Of course, the SVSAR and GGO algorithms as well as our new algorithm all depend on a much more basic assumption, which was implicit in Guglielmi & Overton (2011) and Guglielmi *et al.* (2013), but which we state here for completeness.

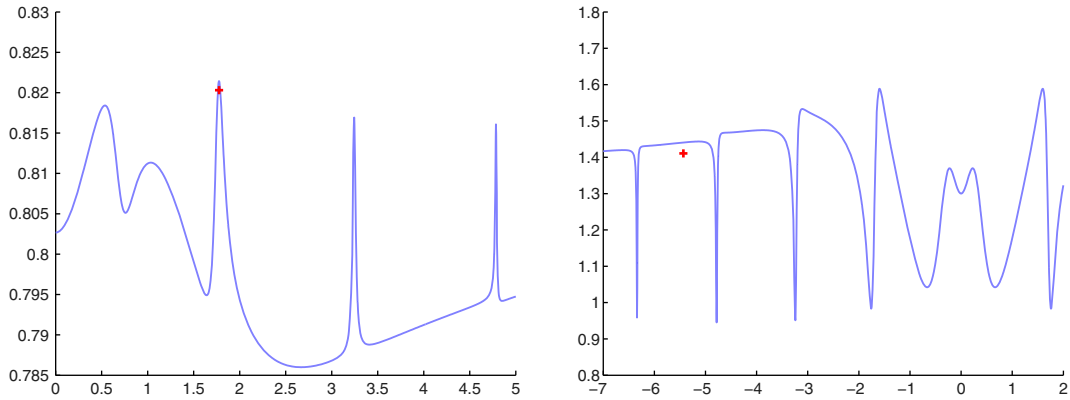

FIG. 1. Left: example CM3. Right: example CM4. Each panel plots the norm of the transfer function evaluated along the imaginary axis with the resulting approximations $(\tilde{\omega}, \tilde{\varepsilon})$ returned by the GGO algorithm as crosses. In CM3, the GGO algorithm appears to have returned an accurate value of $\tilde{\omega}$ to be a maximizer of $\|G(\mathrm{i}\omega)\|$ but its computed value of $\tilde{\varepsilon}$ is inaccurate as $(\tilde{\omega}, \tilde{\varepsilon}^{-1})$ does not lie on the graph of $\|G(\mathrm{i}\omega)\|$. In CM4, this situation is worse as it can clearly be seen that neither $\tilde{\omega}$ or $\tilde{\varepsilon}$ are accurate.

ASSUMPTION 3.3 For any matrix A, an eigenvalue solver (typically sparse) always returns at least one eigenvalue $\lambda \in \sigma(A)$ such that $\mathrm{Re}(\lambda) = \alpha(A)$ or $|\lambda| = \rho(A)$, depending on whether the largest real part or largest modulus is requested, respectively.

3.2 *The breakdown case of the GGO algorithm*

If it were the case that SVSAR always delivered the true value of $\alpha_\varepsilon(A, B, C, D)$ or $\rho_\varepsilon(A, B, C, D)$, the Newton-bisection scheme of the GGO algorithm would accurately deliver the correct value of $\|G\|_\infty = \varepsilon_\star^{-1}$, namely, the maximum value of the norm of the transfer function on the imaginary axis or unit circle, even though (3.1) may be nonsmooth. However, as the approximations found by SVSAR may underestimate $\alpha_\varepsilon(A, B, C, D)$ and $\rho_\varepsilon(A, B, C, D)$, this cannot be assumed and there seems little hope of finding an efficiently scaleable algorithm that computes the spectral value set abscissa and radius accurately. Furthermore, even under Assumption 3.2 or the stronger assumption that SVSAR always converges to locally rightmost or outermost points of $\sigma_\varepsilon(A, B, C, D)$, the GGO algorithm may still break down before finding even a locally maximal value or stationary point of the norm of the transfer function evaluated on the imaginary axis or unit circle. Though this breakdown is briefly discussed in Guglielmi *et al.* (2013, p. 731), we describe the underlying conditions which cause it in more detail here as the first author of the present paper discovered and analysed the breakdown case (Guglielmi *et al.*, 2013, p. 736) and the nature of the deficiency suggests that a new approach is needed. We focus on the continuous-time case.

We first consider the two instances of breakdown on problems CM3 and CM4 reported in Guglielmi *et al.* (2013) where notably only bisection steps were taken and every Newton step was rejected. In Fig. 1, we plot $\|G(\mathrm{i}\omega)\|$ and the approximations produced by the GGO algorithm. Supposing that SVSAR always converges to at least locally rightmost points when the spectral value set abscissa is requested, one would hope that the GGO algorithm would return local, or possibly global, maximizers of the norm of the transfer function evaluated along the imaginary axis. However, here we see that the approximations returned by GGO do not even lie on the graphs of the functions. While the approximation for CM3 seems at least to accurately approximate the vicinity of a local maximizer even though it

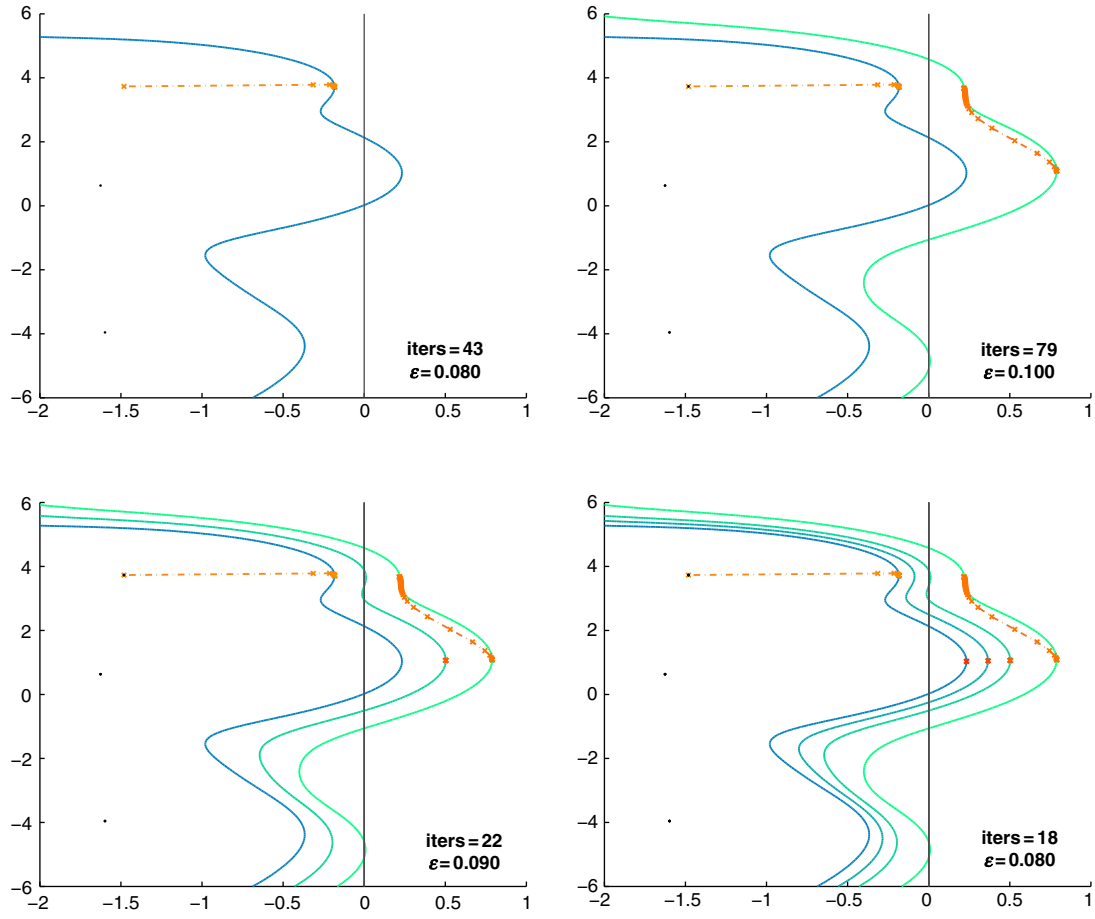

FIG. 2. Progression of breakdown of the GGO algorithm on a synthetic example. Iterates of SVSAR are shown as dashed-dotted lines, spectral value set boundaries are shown as solid curves, and the eigenvalues of A are shown as small dots. Top left: GGO finds, a locally rightmost point λ_1 in the left half-plane for $\varepsilon_1 = 0.08$ and sets $\varepsilon_{\text{lb}} = 0.08$. Top right: GGO attempts $\varepsilon_2 = 0.1$ finds new locally rightmost point λ_2 in the right half-plane and sets $\varepsilon_{\text{ub}} = 0.1$. Bottom left: GGO rejects leftward Newton step ε_2^{N} towards imaginary axis, since $\varepsilon_2^{\text{N}} < \varepsilon_{\text{lb}}$, and instead, via bisection steps, finds a locally rightmost point λ_3 in the right half-plane and updates $\varepsilon_{\text{ub}} = 0.09$. Bottom right: GGO continually rejects leftward Newton steps ε_k^{N} towards the imaginary axis, since $\varepsilon_k^{\text{N}} < \varepsilon_{\text{lb}}$ and erroneously $\text{Re}(\lambda_k) \to \gamma > 0$ and $\varepsilon_k \to 0.08$ as $k \to \infty$, with slow convergence due to only bisection steps being taken, though we depict only two bisection steps and the last step for clarity. The value for ε_k and the number of SVSAR steps to find a locally rightmost point for that level are listed in the bottom right corner of each panel.

underestimates its function value, the approximation for `CM4` seems particularly egregious as it clearly neither locates a maximizer nor approximates a maximal function value well.

As the spectral value sets of `CM3` and `CM4` have quite needle-like shapes that make further visualization difficult, we present an analogous synthetic example in Fig. 2 with the caption describing the progression of breakdown of the GGO algorithm, even when SVSAR always finds locally rightmost points. Even under the unrealistic assumption that SVSAR always converges to globally rightmost points, there is no guarantee that the points found by SVSAR as ε is changed will all lie upon one continuous path of

locally rightmost points. Nonetheless, as long as the points are globally rightmost, the Newton-bisection iteration employed by the GGO algorithm is immune to any inconsistency of the points actually lying upon multiple paths. In practice, we observe that when SVSAR delivers only locally rightmost points, the GGO algorithm often still converges to local maximizers of (2.8) and these often provide good approximations to the H_∞ norm for problems for which this can be tested. However, this convergence is not guaranteed. Indeed, the inconsistency of encountering multiple paths of locally rightmost points can prevent the the GGO algorithm from converging to a local maximizer of (2.8). This inconsistency is caused as much by the inherent behaviour of the SVSAR routine as by the nature of spectral value sets themselves. As shown in Fig. 2, the breakdown of the GGO algorithm is precipitated by the fact that the paths of locally rightmost points encountered do not all exist for every ε_k that the method computes; relatedly, it is not always the case that each path of locally rightmost points necessarily even crosses the imaginary axis. The consequence for the GGO algorithm is that its iterates of lower and upper bounds may, respectively, remain valid only for two different paths of locally rightmost points and thus, the bounds may erroneously converge, typically by bisection steps only, to a value $\tilde{\varepsilon}$ that does not correspond to where either path crosses the imaginary axis. In such a case, we say that the GGO algorithm incurs a *bound mismatch error*.

4. Hybrid expansion–contraction: a breakdown-free algorithm

We now present our new H_∞ norm approximation algorithm, replacing the Newton-bisection outer iteration of the GGO algorithm with a novel hybrid expansion–contraction scheme.

KEY OBSERVATION 4.1 Lower bounds on $\varepsilon_\star$ reported by SVSAR cannot be trusted. However, upper bounds reported by SVSAR always bound $\varepsilon_\star$.

In particular, as shown in the top left panel of Fig. 2, finding a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ in the left half-plane gives no indication of whether or not $\sigma_\varepsilon(A, B, C, D)$ crosses the imaginary axis.

Thus, our new approach to finding a local optimizer of the norm of the transfer function using SVSAR is to forgo the use of any approximations to lower bounds on $\varepsilon_\star$, since they may be unreliable, and to instead focus on monotonically reducing some initial given upper bound $\varepsilon_{\text{ub}} > \varepsilon_\star$ as much as possible, hopefully to $\varepsilon_\star$.

Recalling (3.2), consider the matrix family with respect to the single parameter ε for fixed unit-norm vectors u and v:

$$M_{uv}(\varepsilon) := M(\varepsilon uv^*) = A + B\tilde{\Delta}_{uv}(\varepsilon)C, \quad \text{where } \tilde{\Delta}_{uv}(\varepsilon) := \frac{\varepsilon uv^*}{1 - \varepsilon v^* Du} \tag{4.1}$$

with $\varepsilon \in \mathbb{R}^+$, $\varepsilon\|D\| < 1$. Note that by definition, all eigenvalues of $M_{uv}(\varepsilon)$ lie in $\sigma_\varepsilon(A, B, C, D)$.

KEY OBSERVATION 4.2 In the continuous-time case, let $\varepsilon_{\text{ub}} \in \mathbb{R}^{++}$, $\varepsilon_{\text{ub}}\|D\| < 1$ and fixed vectors $u \in \mathbb{C}^p$ and $v \in \mathbb{C}^m$ with unit norm be given such that $\alpha(M_{uv}(\varepsilon_{\text{ub}})) > 0$. Then since $M_{uv}(0) = A$ is Hurwitz stable, by continuity of $\alpha(M_{uv}(\varepsilon))$ there exists $\hat{\varepsilon}$ such that $0 < \hat{\varepsilon} < \varepsilon_{\text{ub}}$ and $\alpha(M_{uv}(\hat{\varepsilon})) = 0$. Similarly, in the discrete-time case, if $\rho(M_{uv}(\varepsilon_{\text{ub}})) > 1$ then since $M_{uv}(0) = A$ is Schur stable, by continuity of $\rho(M_{uv}(\varepsilon))$ there exists $\hat{\varepsilon}$ such that $0 < \hat{\varepsilon} < \varepsilon_{\text{ub}}$ and $\rho(M_{uv}(\hat{\varepsilon})) = 1$.

Thus, given ε_{ub}, u and v which demonstrate that $\varepsilon_{\text{ub}} > \varepsilon_\star$, that is, the matrix $M_{uv}(\varepsilon_{\text{ub}})$ has at least one eigenvalue in the right half-plane or outside the unit circle respectively, it is clear that we may always

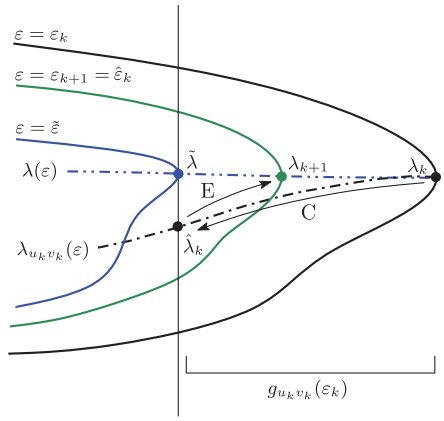

FIG. 3. A single contraction and expansion iteration of the HEC (hybrid expansion-contraction) method, along with its limit point $\tilde{\lambda}$, a locally rightmost point of $\sigma_{\tilde{\varepsilon}}(A, B, C, D)$, which provides a local maximizer to $\|G(i\omega)\|$. The solid curves are the boundaries of $\sigma_\varepsilon(A, B, C, D)$ for ε_k, $\varepsilon_{k+1} = \hat{\varepsilon}_k$ and $\tilde{\varepsilon}$, the vertical line is the imaginary axis, $\lambda(\varepsilon)$ is a continuous path of locally rightmost points of $\sigma_\varepsilon(A, B, C, D)$, and $\lambda_{u_k v_k}(\varepsilon)$ is the rightmost eigenvalue of $M_{u_k v_k}(\varepsilon)$ defined in (4.1), so $g_{u_k v_k}(\varepsilon) = \alpha(M_{u_k v_k}(\varepsilon)) = \mathrm{Re}(\lambda_{u_k v_k}(\varepsilon))$. The labelled arrows C and E respectively represent the contraction of the locally rightmost point λ_k to the eigenvalue $\hat{\lambda}_k$ on the imaginary axis by reducing ε_k to $\hat{\varepsilon}_k$ with u_k and v_k fixed, and the subsequent expansion from $\hat{\lambda}_k$ to the locally rightmost point λ_{k+1} of $\sigma_{\hat{\varepsilon}_k}(A, B, C, D)$ via SVSAR updating the corresponding rank-1 perturbation to u_{k+1} and v_{k+1}.

contract $\varepsilon_{\mathrm{ub}}$ to $\hat{\varepsilon}$ such that $\varepsilon_\star \leqslant \hat{\varepsilon} < \varepsilon_{\mathrm{ub}}$ by finding a root of the function g_{uv} where

$$g_{uv}(\varepsilon) := \begin{cases} \alpha(M_{uv}(\varepsilon)) & \text{(continuous-time case)}, \\ \rho(M_{uv}(\varepsilon)) - 1 & \text{(discrete-time case)}. \end{cases} \tag{4.2}$$

Unlike the hybrid Newton-bisection outer iteration in GGO where the lower bounds cannot be trusted, we have *a priori* true lower and upper bounds of 0 and ε for finding a root of g_{uv} using a hybrid Newton-bisection routine. Furthermore, although the continuous function g_{uv} may be nonsmooth at a finite number of points, we rarely if ever encounter these in practice. Note that, for all ε, $g_{uv}(\varepsilon) \leqslant g(\varepsilon)$, where g is defined in (3.1). The derivatives of both g and g_{uv} are given in Section 5. The function g is monotonically increasing, but g_{uv} might not be. Hence, it is possible that g_{uv} has several roots, but this causes no difficulty.

Given an initial perturbation defined by $\varepsilon_{\mathrm{ub}}$ and vectors u and v such that (4.2) is non-negative indicating $\varepsilon_{\mathrm{ub}} \geqslant \varepsilon_\star$, a *hybrid expansion–contraction iteration* is the combined process of moving the rightmost or outermost eigenvalue of $M_{uv}(\varepsilon_{\mathrm{ub}})$ back to the boundary of the stability region by contracting $\varepsilon_{\mathrm{ub}}$ to $\hat{\varepsilon}$ *while keeping the perturbation vectors u and v fixed*, and then subsequently pushing the rightmost or outermost eigenvalue of $M(\hat{\varepsilon}uv^*)$ away from the boundary again, either rightward or outward from the origin, by now *keeping* $\hat{\varepsilon}$ *fixed* and modifying only the perturbation vectors u and v via the SVSAR iteration. The algorithm repeats this expansion–contraction process in a loop until SVSAR can no longer find a new perturbation that moves an eigenvalue off the stability boundary into the unstable region. A single iteration of the algorithm is depicted in Fig. 3.

We now define the algorithm formally. We discuss how to obtain ε_0, u_0 and v_0 in Section 4.4.

ALGORITHM HEC (hybrid expansion–contraction)

Input: $\varepsilon_0 \in \mathbb{R}^{++}$ with $\varepsilon_0\|D\| < 1$ and unit-norm vectors u_0 and v_0 such that $g_{u_0v_0}(\varepsilon_0) > 0$, along with λ_0, a rightmost eigenvalue of $M_{u_0v_0}(\varepsilon_0)$ in the right half-plane in the continuous-time case (an outermost eigenvalue of $M_{u_0v_0}(\varepsilon_0)$ outside the unit circle in the discrete-time case).

For $k = 0, 1, 2, \ldots,$

(1) *Contraction:* call a Newton-bisection zero-finding algorithm to compute $\hat{\varepsilon}_k \in (0, \varepsilon_k]$ so that $g_{u_kv_k}(\hat{\varepsilon}_k) = 0$, along with $\hat{\lambda}_k$, a rightmost eigenvalue of $M_{u_kv_k}(\hat{\varepsilon}_k)$ on the imaginary axis in the continuous-time case (an outermost eigenvalue of $M_{u_0v_0}(\varepsilon_0)$ on the unit circle in the discrete-time case).

(2) *Expansion:* call the SVSAR algorithm with input $\hat{\varepsilon}_k$, u_k, v_k and $\hat{\lambda}_k$ to compute u_{k+1}, v_{k+1}, λ_{k+1} satisfying Assumption 3.2 and $\mathrm{Re}(\lambda_{k+1}) \geqslant \mathrm{Re}(\hat{\lambda}_k) = 0$ in the continuous-time case ($|\lambda_{k+1}| \geqslant |\hat{\lambda}_k| = 1$ in the discrete-time case).

(3) Set $\varepsilon_{k+1} = \hat{\varepsilon}_k$.

In practice, we pass the right and left eigenvectors as well as eigenvalues computed by the contraction phase into the expansion phase and vice versa. See Mitchell (2014, Procedure 5, p. 41) for more details.

4.1 *Convergence of hybrid expansion–contraction*

We now present our main convergence theorem, which depends on Assumptions 3.2 and 3.3. The idea is that, in the continuous-time case, since $\alpha(M_{u_kv_k}(\varepsilon_k)) \geqslant 0$ for all k, and $\alpha(M_{u_kv_k}(\hat{\varepsilon}_k)) = 0$ for all k, and since $\varepsilon_{k+1} = \hat{\varepsilon}_k$ with ε_k monotonically decreasing, it must happen that $\alpha(M_{u_kv_k}(\varepsilon_k)) \to 0$.

THEOREM 4.3 Given valid initial data, Algorithm HEC generates a sequence $\{\varepsilon_k\}$ converging monotonically to a limit $\tilde{\varepsilon}$ and a sequence $\{\lambda_k\}$ having at least one cluster point $\tilde{\lambda}$, where $\|G(\tilde{\lambda})\| = \tilde{\varepsilon}^{-1}$, with $\mathrm{Re}(\tilde{\lambda}) = 0$ or $|\tilde{\lambda}| = 1$ for the continuous- and discrete-time cases, respectively. Assuming that the simplicity condition of Definition 2.3 holds at $\tilde{\lambda}$ with respect to $\tilde{\varepsilon}$, we also have the following.

(1) In the continuous-time case, $\tilde{\lambda}$ satisfies the first-order necessary condition to be a local maximizer of the optimization problem in (2.5) given in Lemma 2.11 for $\varepsilon = \tilde{\varepsilon}$, and, in addition, $\mathrm{Im}(\tilde{\lambda})$ is a stationary point of $\|G(\mathrm{i}\omega)\|$ with stationary value $\tilde{\varepsilon}^{-1}$. Furthermore, if $\tilde{\lambda}$ is a locally rightmost point of $\sigma_{\tilde{\varepsilon}}(A, B, C, D)$, then $\mathrm{Im}(\tilde{\lambda})$ is a local maximizer of $\|G(\mathrm{i}\omega)\|$ with locally maximal value $\tilde{\varepsilon}^{-1}$.

(2) In the discrete-time case, $\tilde{\lambda}$ satisfies the first-order necessary condition to be a local maximizer of the optimization problem in (2.7) given in Lemma 2.13 for $\varepsilon = \tilde{\varepsilon}$, and, in addition, $\angle\tilde{\lambda}$ is a stationary point of $\|G(e^{\mathrm{i}\theta})\|$ with stationary value $\tilde{\varepsilon}^{-1}$. Furthermore, if $\tilde{\lambda}$ is a locally outermost point of $\sigma_{\tilde{\varepsilon}}(A, B, C, D)$, then $\angle\tilde{\lambda}$ is a local maximizer of $\|G(e^{\mathrm{i}\theta})\|$ with locally maximal value $\tilde{\varepsilon}^{-1}$.

Proof. We give the proof only for the continuous-time case. The algorithm ensures that $\{\varepsilon_k\}$ is a monotonically decreasing non-negative sequence so it must converge to a limit $\tilde{\varepsilon}$, and so it follows that $\hat{\varepsilon}_k = \varepsilon_{k+1}$ converges to the same limit. The algorithm also ensures that $\mathrm{Re}(\lambda_k) \geqslant 0$ for all k. Suppose that $\mathrm{Re}(\lambda_k)$ does not converge to zero. Then there is a subsequence $\{\lambda_{k_i}\}$ for which $\mathrm{Re}(\lambda_{k_i})$ is bounded below by some $\gamma > 0$. By definition of the expansion step, using Assumption 3.2, λ_{k_i} is an eigenvalue of $M_{u_{k_i}v_{k_i}}(\varepsilon_{k_i})$, so it follows that $\alpha(M_{u_{k_i}v_{k_i}}(\varepsilon_{k_i})) \geqslant \gamma$. By taking a further subsequence if necessary,

we may assume without loss of generality that $u_{k_i} v_{k_i}^*$ converges to a limit $\tilde{u}\tilde{v}^*$. It follows that the matrix $M_{u_{k_i} v_{k_i}}(\varepsilon_{k_i})$ converges to a limit $M_{\tilde{u}\tilde{v}^*}(\tilde{\varepsilon})$, and therefore, since the spectral abscissa is continuous, that $\alpha(M_{u_{k_i} v_{k_i}}(\varepsilon_{k_i}))$ converges to $\alpha(M_{\tilde{u}\tilde{v}^*}(\tilde{\varepsilon}))$, which must be greater than or equal to γ. But since $\hat{\varepsilon}_{k_i}$ also converges to $\tilde{\varepsilon}$, $\alpha(M_{u_{k_i} v_{k_i}}(\hat{\varepsilon}_{k_i}))$ must converge to the same limit $\alpha(M_{\tilde{u}\tilde{v}^*}(\tilde{\varepsilon}))$—which is a contradiction since, by definition of the contraction step, $\alpha(M_{u_{k_i} v_{k_i}}(\hat{\varepsilon}_{k_i})) = 0$ for all i. So, $\mathrm{Re}(\lambda_k)$ must converge to zero.

Although the sequence $\{\lambda_k\}$ might not converge, it is bounded and hence has at least one cluster point $\tilde{\lambda}$. Clearly, $\mathrm{Re}(\tilde{\lambda}) = 0$. By Assumption 3.2, for all k, we have $\|G(\lambda_k)\| = \varepsilon_k^{-1}$, the simplicity condition holds at λ_k with respect to ε_k, and the first-order necessary condition

$$v_k^* C(\lambda_k I - A)^{-2} B u_k \in \mathbb{R}^{++}$$

holds. It follows that $\|G(\tilde{\lambda})\| = \tilde{\varepsilon}^{-1}$ and, provided that the simplicity condition holds at $\tilde{\lambda}$ with respect to $\tilde{\varepsilon}$, that the limiting first-order necessary condition

$$\tilde{v}^* C(\tilde{\lambda} I - A)^{-2} B \tilde{u} \in \mathbb{R}^{++}$$

also holds, where $\tilde{u}$ and $\tilde{v}$ are, respectively, right and left singular vectors corresponding to $\tilde{\varepsilon}^{-1}$, the largest singular value of $G(\tilde{\lambda})$. The possibility that the left-hand side is zero is excluded by the simplicity condition, because, by Guglielmi *et al.* (2013, Equation (2.18)), this would imply the orthogonality of the right and left eigenvectors of $M(\tilde{\varepsilon}\tilde{u}\tilde{v}^*)$ corresponding to the eigenvalue $\tilde{\lambda}$. The result then follows from applying Lemma 2.12 with $\lambda = \tilde{\lambda}$. □

4.2 *Quadratic convergence rate of hybrid expansion–contraction*

We now explain how the hybrid expansion–contraction algorithm is actually an adaptively positively or negatively damped Newton method, in contrast to the Newton-bisection outer iteration of the GGO algorithm. We focus for the moment on the continuous-time case.

By definition of Algorithm HEC, we always have $\tilde{\varepsilon} \leqslant \varepsilon_{k+1} = \hat{\varepsilon}_k \leqslant \varepsilon_k$. In the Newton-bisection contraction phase, the first updated value from ε_k attempting to make $g_{u_k v_k}(\cdot)$ zero is the Newton update

$$\varepsilon_k^{\mathrm{N}} = \varepsilon_k - \frac{g_{u_k v_k}(\varepsilon_k)}{g'_{u_k v_k}(\varepsilon_k)}$$

where $g'_{u_k v_k}$ denotes the derivative of $g_{u_k v_k}$ given in (5.5). The value $g_{u_k v_k}(\varepsilon_k)$ is positive and as we shall explain shortly, so is $g'_{u_k v_k}(\varepsilon_k)$, and thus $\varepsilon_k^{\mathrm{N}} < \varepsilon_k$ holds. Furthermore, for k large enough, the bisection safeguards will not prevent the Newton step from being attempted since $\varepsilon_k^{\mathrm{N}} > 0$ follows from $\tilde{\varepsilon} > 0$. If the rightmost eigenvalue, say λ_k^{N}, of $M_{u_k v_k}(\varepsilon_k^{\mathrm{N}})$ is in the left half-plane, as shown in the top left of Fig. 4, it follows that a locally rightmost point computed by SVSAR with input $\varepsilon_k^{\mathrm{N}}$, u_k and v_k could also be in the left half-plane, which might potentially cause the breakdown scenario observed in the GGO algorithm. Thus, the contraction phase instead finds $\hat{\varepsilon}_k > \varepsilon_k^{\mathrm{N}}$ such that $\alpha(M_{u_k v_k}(\hat{\varepsilon}_k)) = 0$ holds to ensure that the next locally rightmost point, found by SVSAR with input $\hat{\varepsilon}$, u_k and v_k resides in the right half-plane (recalling that SVSAR is a monotonic method). On the other hand, if λ_k^{N} is in the right half-plane, as shown in the top right of Fig. 4, it is guaranteed that the next locally rightmost point computed by SVSAR will also be in the right half-plane and thus the contraction phase can take an even larger reduction than that provided via $\varepsilon_k^{\mathrm{N}}$ by instead finding $\hat{\varepsilon}_k < \varepsilon_k^{\mathrm{N}}$ such that $\alpha(M_{u_k v_k}(\hat{\varepsilon}_k)) = 0$.

However, as explained in Section 5, $\varepsilon_k^{\mathrm{N}}$ is equivalent to the Newton step the GGO algorithm would have taken from ε_k; this observation also explains why $g'_{u_k v_k}(\varepsilon_k) > 0$. As a consequence, the

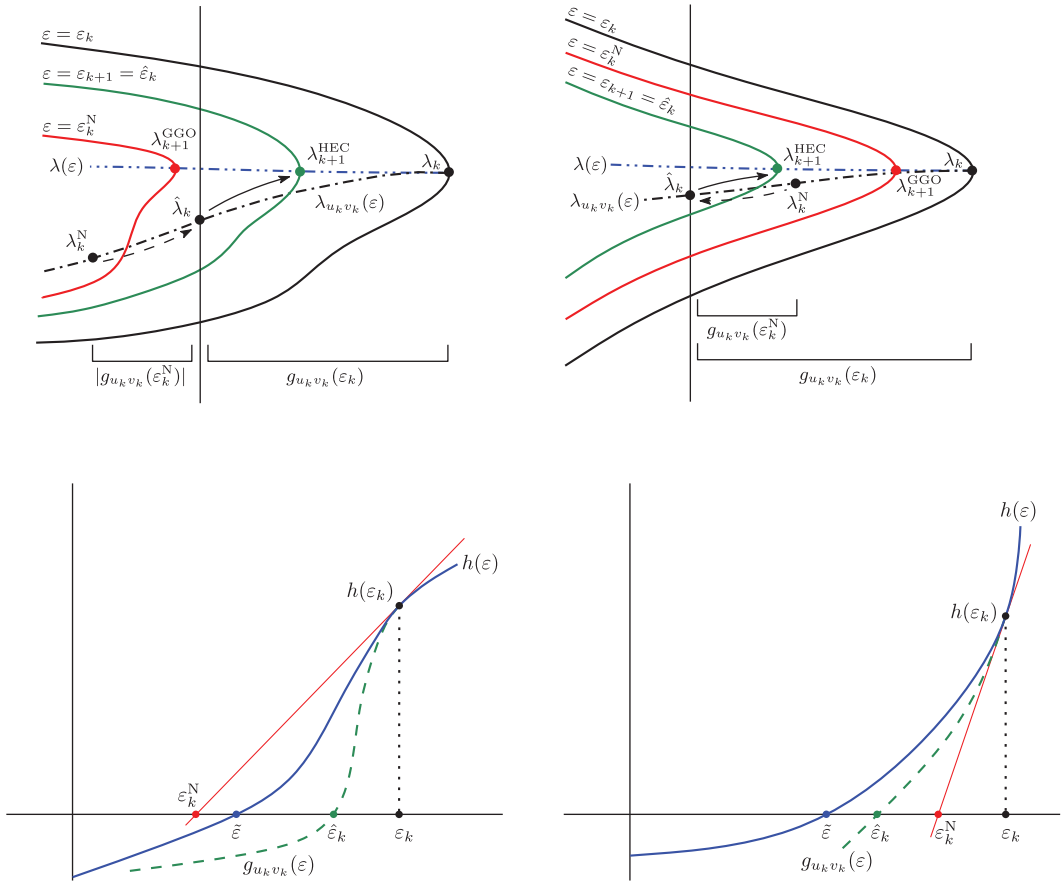

FIG. 4. Left: positive damping. Right: negative damping. As in Fig. 3, the top row depicts the boundaries of $\sigma_\varepsilon(A, B, C, D)$ but now $\varepsilon_{k+1} = \hat{\varepsilon}_k$ and $\varepsilon_k^{\mathrm{N}}$ are, respectively, the next steps that HEC and the GGO algorithm would take from ε_k. Setting $\lambda_k^{\mathrm{N}} := \lambda_{u_k v_k}(\varepsilon_k^{\mathrm{N}})$, the dashed arrows represent the change from eigenvalue λ_k^{N} to $\hat{\lambda}_k$ as the contraction phase converges after taking its first step to $\varepsilon_k^{\mathrm{N}}$, and the solid arrows represent the expansion phase 'pushing' $\hat{\lambda}_k$ off the imaginary axis and rightward to the locally rightmost point λ_{k+1}. In the top left panel, we see that the initial contraction attempt overshoots the imaginary axis and thus it must be damped in comparison with the Newton step of the GGO algorithm to ensure breakdown cannot occur. In the top right panel, we see that the initial contraction attempt falls short of reaching the imaginary axis and thus hybrid expansion-contraction can take an accelerated step compared with the Newton step taken by the GGO algorithm, that is, $\mathrm{Re}(\lambda_{k+1}^{\mathrm{HEC}}) < \mathrm{Re}(\lambda_{k+1}^{\mathrm{GGO}})$. In the bottom row, the plots depict $h(\varepsilon) := \mathrm{Re}(\lambda(\varepsilon))$, along with its root $\tilde{\varepsilon}$, corresponding to the two spectral value set plots in the complex plane immediately above. The solid curve is $h(\varepsilon)$, the root $\varepsilon_k^{\mathrm{N}}$ of the tangent line to $h(\varepsilon)$ at ε_k is the Newton step of the GGO algorithm while the root $\hat{\varepsilon}_k$ of the dashed curve representing $g_{u_k v_k}(\varepsilon) = \mathrm{Re}(\lambda_{u_k v_k}(\varepsilon))$, which is also tangent to $h(\varepsilon)$ at ε_k, is the HEC contraction step. While the positive damping example on the left depicts a scenario where $\varepsilon_k^{\mathrm{N}} < \tilde{\varepsilon}$ holds since $\mathrm{Re}(\lambda_{k+1}^{\mathrm{GGO}}) < 0$, note that this is not always the case. It can also happen that λ_k^{N} is in the left half-plane while $\lambda_{k+1}^{\mathrm{GGO}}$ is in the right half-plane; in this case, HEC will still positively damp the Newton step of the GGO algorithm but it will then follow that $\tilde{\varepsilon} < \varepsilon_k^{\mathrm{N}} < \hat{\varepsilon}_k$ holds.

contraction phase of hybrid expansion–contraction is actually scaling the Newton step of the GGO algorithm either by damping it sufficiently to avoid the potential breakdown that might occur if $\varepsilon_k^{\mathrm{N}}$ were accepted or, alternatively, by enlarging the step size when possible. In fact, by finding $\hat{\varepsilon}_k$ such that $\alpha(M_{u_k v_k}(\hat{\varepsilon}_k)) = 0$, we see that the contraction phase is either damping the step $\varepsilon_k^{\mathrm{N}}$ the *minimal* amount

when $\alpha(M_{uv}(\varepsilon_k^{\mathrm{N}})) < 0$ or increasing it the *maximal* amount when $\alpha(M_{uv}(\varepsilon_k^{\mathrm{N}})) > 0$, subject to guaranteeing that breakdown cannot occur. In the bottom row of Fig. 4, we depict the differences between the Newton step of the GGO algorithm and the damped solution used by HEC for the two examples in the top row.

We now establish that the damping mechanism just described does not impede the quadratic convergence rate of the GGO algorithm, that is, the HEC method is also quadratically convergent, under assumptions that generally seem to hold in practice.

THEOREM 4.4 Using the notation established in Theorem 4.3, suppose that Algorithm HEC has a unique cluster point $\tilde{\lambda}$, with the simplicity condition holding at $\tilde{\lambda}$ with respect to $\tilde{\varepsilon}$. Let $\tilde{u}$ and $\tilde{v}$ respectively denote the right and left singular vectors corresponding to the largest singular value $\tilde{\varepsilon}^{-1}$ of $G(\tilde{\lambda})$, so that $\tilde{\lambda}$ is an eigenvalue of $M(\tilde{\varepsilon}\tilde{u}\tilde{v}^*)$, and assume that $\tilde{\lambda}$ is the rightmost *(*or outermost*)* eigenvalue of $M(\tilde{\varepsilon}\tilde{u}\tilde{v}^*)$, so that $g_{\tilde{u}\tilde{v}}(\tilde{\varepsilon}) = 0$. Finally assume that for all $k \geqslant K$, the points λ_k generated by Algorithm HEC lie on a twice continuously differentiable path of locally rightmost (or locally outermost) points $\lambda(\varepsilon)$ of $\sigma_\varepsilon(A, B, C, D)$, defined on $[\tilde{\varepsilon}, \varepsilon_K]$, with $\lambda_k = \lambda(\varepsilon_k)$ for all $k \geqslant K$, converging to $\lambda(\tilde{\varepsilon}) = \tilde{\lambda}$. Then the sequence ε_k converges to $\tilde{\varepsilon}$ quadratically.

Proof. Since the simplicity condition holds at $\tilde{\lambda}$ with respect to $\tilde{\varepsilon}$, by continuity it also holds at λ_k with respect to ε_k for all $k \geqslant K$ (increasing K if necessary). In what follows, assume that $k \geqslant K$.

Define the function $h : [\tilde{\varepsilon}, \varepsilon_K] \to \mathbb{R}$ by $h(\varepsilon) = \mathrm{Re}(\lambda(\varepsilon))$ in the continuous-time case or $h(\varepsilon) = |\lambda(\varepsilon)|$ in the discrete-time case. The function h is the same as g defined in (3.1) if $\lambda(\varepsilon)$ is a path of rightmost or outermost points of $\sigma_\varepsilon(A, B, C, D)$, not just locally rightmost or outermost. Note that $h(\tilde{\varepsilon}) = 0$ by Theorem 4.3 and that h is monotonically increasing with respect to ε.

The key to proving the result is that, by Lemma 5.2 combined with Remark 5.3, the functions h and $g_{u_k v_k}$ and their derivatives both coincide at ε_k, so the Newton steps for h and $g_{u_k v_k}$ are the same, namely

$$\varepsilon_k^{\mathrm{N}} := \varepsilon_k - \frac{h(\varepsilon_k)}{h'(\varepsilon_k)} = \varepsilon_k - \frac{g_{u_k v_k}(\varepsilon_k)}{g'_{u_k v_k}(\varepsilon_k)}. \tag{4.3}$$

The GGO algorithm would set $\varepsilon_{k+1} := \varepsilon_k^{\mathrm{N}}$, so that quadratic convergence follows immediately from the usual analysis of Newton's method. However, Algorithm HEC sets $\varepsilon_{k+1} := \hat{\varepsilon}_k$, where $g_{u_k v_k}(\hat{\varepsilon}_k) = 0$. Applying Taylor's theorem to h and to $g_{u_k v_k}$ separately (see Remark 5.4), we have

$$0 = h(\tilde{\varepsilon}) = h(\varepsilon_k) + h'(\varepsilon_k)(\tilde{\varepsilon} - \varepsilon_k) + \tfrac{1}{2} h''(\xi_k)(\tilde{\varepsilon} - \varepsilon_k)^2$$

for some $\xi_k \in [\tilde{\varepsilon}, \varepsilon_k]$ and

$$0 = g_{u_k v_k}(\hat{\varepsilon}_k) = g_{u_k v_k}(\varepsilon_k) + g'_{u_k v_k}(\varepsilon_k)(\hat{\varepsilon}_k - \varepsilon_k) + \tfrac{1}{2} g''_{u_k v_k}(\eta_k)(\hat{\varepsilon}_k - \varepsilon_k)^2$$

for some $\eta_k \in [\hat{\varepsilon}_k, \varepsilon_k]$. Dividing these equations by the derivative factors, subtracting the first from the second and using (4.3) along with $\varepsilon_{k+1} = \hat{\varepsilon}_k$, we obtain

$$\varepsilon_{k+1} - \tilde{\varepsilon} = c_k(\tilde{\varepsilon} - \varepsilon_k)^2 + d_k(\varepsilon_{k+1} - \varepsilon_k)^2 \tag{4.4}$$

where

$$c_k = \frac{h''(\xi_k)}{2\, h'(\varepsilon_k)}, \quad d_k = -\frac{g''_{u_k v_k}(\eta_k)}{2 g'_{u_k v_k}(\varepsilon_k)}.$$

So, to establish quadratic convergence, we need only to bound $\varepsilon_{k+1} - \varepsilon_k$ in terms of $\tilde{\varepsilon} - \varepsilon_k$. This is done by again writing out Taylor expansions for h and $g_{u_k v_k}$, but with one fewer term:

$$0 = h(\tilde{\varepsilon}) = h(\varepsilon_k) + h'(\zeta_k)(\tilde{\varepsilon} - \varepsilon_k)$$

for some $\zeta_k \in [\tilde{\varepsilon}, \varepsilon_k]$ and

$$0 = g_{u_k v_k}(\hat{\varepsilon}_k) = g_{u_k v_k}(\varepsilon_k) + g'_{u_k v_k}(\tau_k)(\hat{\varepsilon}_k - \varepsilon_k)$$

for some $\tau_k \in [\hat{\varepsilon}_k, \varepsilon_k]$. Since $h(\varepsilon_k) = g_{u_k v_k}(\varepsilon_k)$ and $\varepsilon_{k+1} = \hat{\varepsilon}_k$, it follows that

$$\frac{\varepsilon_{k+1} - \varepsilon_k}{\tilde{\varepsilon} - \varepsilon_k} = \frac{h'(\zeta_k)}{g'_{u_k v_k}(\tau_k)}$$

which converges to 1 as $k \to \infty$ since $h'(\zeta_k)$ and $g'_{u_k v_k}(\varepsilon_k)$ both converge to the positive number $h'(\tilde{\varepsilon})$. So, quadratic convergence follows from (4.4), as c_k and d_k both converge as $k \to \infty$. □

As we report in Section 8, we typically observe quadratic convergence in our experimental results.

4.3 *Contracting early*

Since SVSAR's slow convergence in the expansion phase is potentially expensive, we consider here whether we can terminate SVSAR early with the hope of reducing the overall number of eigentriples that need to be computed. With the Newton-bisection outer iteration of the GGO algorithm, this would be risky, as terminating SVSAR early when the current iterate is in the left half-plane (in the continuous-time case) could make bound mismatch errors more likely. Furthermore, terminating SVSAR early may degrade the quality of GGO's Newton steps towards the imaginary axis due to loss of accuracy in the corresponding derivative computation. However, under hybrid expansion–contraction, as long as the expansion phase makes *any* progress into the right half-plane, the procedure can always begin contracting again early, in lieu of incurring the full number of iterations for SVSAR to converge. We propose terminating the expansion phase when the current step $\mathrm{Re}(\lambda^{j+1}) - \mathrm{Re}(\lambda^j)$ falls below, say, 1% of the maximum of such differences generated so far for the current value of ε_k. A benefit of this scheme is that it is self-scaling, meaning that as hybrid expansion–contraction converges, the expansion phase will be permitted to take smaller steps before switching back to the contraction phase; in other words, this relative-step-size termination condition applied to the expansion phase has no effect in the limit and as a consequence, the theoretical convergence rate of HEC must still be at least superlinear when it is enabled (see Dembo *et al.*, 1982 and Mitchell, 2014, Sections 3.1 and 3.2). The potential cost of a handful of extra HEC iterations seems a small price to pay if it can dramatically reduce the number of iterations incurred by the linearly converging SVSAR subroutine; we report on the benefits of this optimization in Section 8.

4.4 *A fast new method to find an initial upper bound*

In order for the GGO algorithm to apply the outer Newton-bisection iteration, it must first find an upper bound on $\varepsilon_\star$ to initialize the bracket range necessary for bisection. One possibility would be to use a value just slightly less than $\|D\|^{-1}$, as this is the supremum of all values of ε for which $\sigma_\varepsilon(A, B, C, D)$ is well defined, but this would invite numerical difficulties. Another, used in Guglielmi *et al.* (2013), is

to first calculate an initial ε_0 along with vectors u_0 and v_0, and then call SVSAR to expand rightward or outward as far as possible using $\varepsilon = \varepsilon_0$. If the final eigenvalue returned by SVSAR is still in the stability region, then this method increases ε_0 via $\varepsilon_k = \min(2\varepsilon_{k-1}, 0.5(\varepsilon_{k-1} + \|D\|^{-1}))$ and again calls SVSAR. Unfortunately, this precomputation phase to initialize the bracket range that provides an upper bound to $\varepsilon_\star$ can be very expensive as SVSAR's mere linear convergence may be incurred multiple times before a large enough value of ε is found.

On the other hand, given $\varepsilon_{\text{ub}} \geqslant \varepsilon_\star$ and unit vectors $u_0 \in \mathbb{C}^p$ and $v_0 \in \mathbb{C}^m$ as input, hybrid expansion–contraction can be initialized from the corresponding point $\lambda_0 \in \sigma_{\varepsilon_{\text{ub}}}(A, B, C, D)$, provided that λ_0 is outside the stability region and λ_0 is either a rightmost or outermost eigenvalue of $M(\varepsilon_{\text{ub}} u_0 v_0^*)$ for the continuous- or discrete-time cases, respectively; there is no requirement that the initial point λ_0 additionally be a locally rightmost or outermost point of $\sigma_{\varepsilon_{\text{ub}}}(A, B, C, D)$. To that end, a more efficient procedure for finding a suitable upper bound to initialize Algorithm HEC is as follows.

ALGORITHM FAST UPPER BOUND Input: $\varepsilon \in \mathbb{R}^{++}$ with $\varepsilon\|D\| < 1$ and unit-norm vectors u and v. While $g_{uv}(\varepsilon) < 0$,

(1) set

$$\varepsilon_{\text{new}} := \min\left(\max\left(\varepsilon, \varepsilon - 2\frac{g_{uv}(\varepsilon)}{g'_{uv}(\varepsilon)}\right), \frac{1}{2}(\varepsilon + \|D\|^{-1})\right),$$

that is, double the Newton step for $g_{uv}(\varepsilon)$, subject to ensuring that ε_{new} is at least ε and is not too close to the maximal value allowed for ε;

(2) if $g_{uv}(\varepsilon_{\text{new}}) > g_{uv}(\varepsilon)$, set $\varepsilon := \varepsilon_{\text{new}}$;

(3) else if $g'_{uv}(\varepsilon) > 0$ holds, do a line search to find $\hat{\varepsilon}_{\text{new}} \in (\varepsilon, \varepsilon_{\text{new}})$ such that $g_{uv}(\hat{\varepsilon}_{\text{new}}) > g_{uv}(\varepsilon)$ holds, and set $\varepsilon := \hat{\varepsilon}_{\text{new}}$;

(4) if $g_{uv}(\varepsilon) \geqslant 0$, break;

(5) update vectors u and v via a single step of SVSAR.

For more details, see Mitchell (2014, Procedure 6, p. 55).

The benefit of this new strategy is that at no point do we incur SVSAR's linear convergence and by attempting to increase the perturbation level ε before each update to the perturbation vectors, we can expect that every SVSAR update step will typically be quite large as it will generally be updating from a point in the interior of the spectral value set for that perturbation level. However, one possible concern is whether this compromises the likelihood of HEC converging to a globally optimal value compared with the bound bracketing procedure used in the GGO algorithm. As a compromise, it may be beneficial to make a single call to SVSAR once an eigenvalue has been found outside the stability region before commencing hybrid expansion–contraction, as discussed further in Section 8.1.

5. Derivatives of the eigenvalue functions

In order to obtain the derivatives of the functions $g(\varepsilon)$ and $g_{uv}(\varepsilon)$ defined in (3.1) and (4.2), we first need to define appropriate normalization for the relevant eigenvectors.

DEFINITION 5.1 A complex number λ and vectors $x, y \in \mathbb{C}^n$ comprise an *RP(z)-compatible eigentriple* (λ, x, y) of a matrix $W \in \mathbb{C}^{n\times n}$ if

(1) λ is a simple eigenvalue of W and

(2) x and y are, respectively, corresponding right and left eigenvectors that satisfy $\|x\| = \|y\| = 1$ and y^*x is a positive real multiple of $z \in \mathbb{C}$.

When the argument z is omitted, it is understood to be 1, so that y^*x is real and positive.

Let $\lambda(\varepsilon)$ denote a rightmost (continuous-time case) or outermost (discrete-time case) point of $\sigma_\varepsilon(A, B, C, D)$, with $\|G(\lambda)\| = \varepsilon^{-1}$ and corresponding singular vectors $u(\varepsilon)$ and $v(\varepsilon)$, so that $\lambda(\varepsilon)$ is an eigenvalue of $M(\varepsilon u(\varepsilon)v(\varepsilon)^*)$ by Lemma 2.2. Define $x(\varepsilon)$, $y(\varepsilon)$ so that $(\lambda(\varepsilon), x(\varepsilon), y(\varepsilon))$ form an RP-compatible eigentriple (continuous-time case) or RP($\bar{\lambda}(\varepsilon)$)-compatible eigentriple (discrete-time case). The derivative of $g(\varepsilon)$ defined in (3.1), or equivalently of $\mathrm{Re}(\lambda(\varepsilon))$ (continous-time case) or $|\lambda(\varepsilon)|$ (discrete-time case), was derived in Guglielmi *et al.* (2013, Section 4) under assumptions that ensure that $\lambda(\varepsilon)$ is well defined and continuously differentiable in a neighborhood of some $\check{\varepsilon}$, namely, that the simplicity condition of Definition 2.3 holds at $\lambda(\varepsilon)$ with respect to ε and that $\lambda(\check{\varepsilon})$ is the unique rightmost or outermost point of $\sigma_{\check{\varepsilon}}(A, B, C, D)$ (the unique one in the closed upper half-plane if A, B, C, D are all real). Letting $\check{x} = x(\check{\varepsilon})$, $\check{y} = y(\check{\varepsilon})$, $\check{u} = u(\check{\varepsilon})$, $\check{v} = v(\check{\varepsilon})$, the derivative of g is given by[3]

$$g'(\varepsilon)\big|_{\varepsilon=\check{\varepsilon}} = \begin{cases} (\bar{\beta}\gamma\check{y}^*\check{x})^{-1} & \text{(continuous-time case)}, \\ (\bar{\beta}\gamma|\check{y}^*\check{x}|)^{-1} & \text{(discrete-time case)}, \end{cases} \tag{5.1}$$

where, via Guglielmi *et al.* (2013, Equations (3.9) and (3.12)),

$$\beta = \frac{1 - \check{\varepsilon}\check{u}^*D^*\check{v}}{\check{u}^*B^*\check{y}} \quad \text{and} \quad \gamma = \frac{1 - \check{\varepsilon}\check{v}^*D\check{u}}{\check{v}^*C\check{x}}. \tag{5.2}$$

Furthermore, the product $\bar{\beta}\gamma$ is real and positive by Guglielmi *et al.* (2013, Equation (3.8)), so the derivative is real and positive.

To obtain the derivative of $g_{uv}(\varepsilon)$ defined in (4.2), *where the vectors u and v defining g_{uv} are arbitrary fixed vectors*, we first find the derivative of the matrix family $M_{uv}(\varepsilon)$ defined in (4.1) with respect to ε by the quotient rule:

$$M'_{uv} := M'_{uv}(\varepsilon)\big|_{\varepsilon=\check{\varepsilon}} = B\left(\frac{uv^*(1 - \check{\varepsilon}v^*Du) + \check{\varepsilon}uv^*(v^*Du)}{(1 - \check{\varepsilon}v^*Du)^2}\right)C = \frac{Buv^*C}{(1 - \check{\varepsilon}v^*Du)^2}. \tag{5.3}$$

Let $(\lambda_{uv}(\varepsilon), x_{uv}(\varepsilon), y_{uv}(\varepsilon))$ be an RP-compatible eigentriple (continuous-time case) or RP($\bar{\lambda}_{uv}(\varepsilon)$)-compatible eigentriple (discrete-time case)[4] of (4.1), where $\lambda_{uv}(\varepsilon)$ is a rightmost or outermost eigenvalue of $M_{uv}(\varepsilon)$ which is assumed to be simple. We have from standard first-order perturbation theory

[3] Actually, in Guglielmi *et al.* (2013), β appears unconjugated, but this is because it is assumed that u and v have been normalized so that $\beta, \gamma \in \mathbb{R}^{++}$; see Guglielmi *et al.* (2013, pp. 721–2). We also note that in the proof of Theorem 3.2 and the statement of Guglielmi *et al.* (2013, Corollary 4.2), the scalings of x and y were incorrect: in the proof of Theorem 3.2, x and y should be scaled by $1/\beta$ and $1/\gamma$ respectively (instead of β and γ), and in the statement of Corollary 4.2, x and y should be scaled by β and γ respectively (instead of $1/\beta$ and $1/\gamma$).

[4] The next equation does not depend on RP-compatibility, but we will exploit this property in Lemma 5.2.

(Horn & Johnson, 1990, Theorem 6.3.12), writing $\check{x}_{uv} = x_{uv}(\check{\varepsilon})$, $\check{y}_{uv} = y_{uv}(\check{\varepsilon})$, that

$$\lambda'_{uv} := \lambda'_{uv}(\varepsilon)\big|_{\varepsilon=\check{\varepsilon}} = \frac{\check{y}^*_{uv} M'_{uv} \check{x}_{uv}}{\check{y}^*_{uv} \check{x}_{uv}}, \tag{5.4}$$

so the derivative of $g_{uv}(\varepsilon)$ is given by

$$g'_{uv}(\varepsilon)\big|_{\varepsilon=\check{\varepsilon}} = \begin{cases} \mathrm{Re}(\lambda'_{uv}) & \text{(continuous-time case)}, \\ \mathrm{Re}(\tau\lambda'_{uv}) & \text{(discrete-time case)}, \end{cases} \tag{5.5}$$

where $\tau = \bar{\lambda}_{uv}(\check{\varepsilon})/|\lambda_{uv}(\check{\varepsilon})|$.

In fact, as the next lemma shows, *as long as* g_{uv} *is defined with* $u = \check{u} \equiv u(\check{\varepsilon})$, $v = \check{v} \equiv v(\check{\varepsilon})$, so that $\lambda_{uv}(\check{\varepsilon})$, the rightmost (or outermost) eigenvalue of $M(\check{\varepsilon}uv^*)$, and $\lambda(\check{\varepsilon})$, the rightmost (or outermost) point of $\sigma_{\check{\varepsilon}}(A, B, C, D)$, coincide, then their derivatives at $\check{\varepsilon}$ also coincide.

LEMMA 5.2 Using the notation established above, if $u = \check{u}$ and $v = \check{v}$, then the derivatives given in (5.1) and (5.5) are equivalent.

Proof. Since the RP-compatibility conditions define right and left eigenvectors uniquely up to the same unimodular scalar, we can write $x = \check{x} = e^{\mathrm{i}\theta}\check{x}_{uv}$, $y = \check{y} = e^{\mathrm{i}\theta}\check{y}_{uv}$. Using (5.3) and (5.4) and then substituting in (5.2), we see that

$$\lambda'_{uv} = \frac{(y^*Bu)}{(1 - \check{\varepsilon}v^*Du)} \frac{(v^*Cx)}{(1 - \check{\varepsilon}v^*Du)} \frac{1}{y^*x} = \frac{1}{\bar{\beta}\gamma y^*x}.$$

The product $\bar{\beta}\gamma$ is real and positive by Guglielmi *et al.* (2013, Equation (3.8)). The result follows because y^*x is real and positive in the continuous-time case and a positive real multiple of τ in the discrete-time case. □

REMARK 5.3 Although not explicitly stated in Guglielmi *et al.* (2013), if $\lambda(\varepsilon)$ is only a *locally* rightmost or outermost point of $\sigma_\varepsilon(A, B, C, D)$, and we define $h(\varepsilon)$ to be $\mathrm{Re}(\lambda(\varepsilon))$ or $|\lambda(\varepsilon)|$, respectively, then, under assumptions ensuring that $\lambda(\varepsilon)$ is well defined and continuously differentiable near $\check{\varepsilon}$, the right-hand side of (5.1) still provides the derivative of $h(\varepsilon)$ at $\varepsilon = \check{\varepsilon}$. Furthermore, when setting $u = \check{u}$, $v = \check{v}$, as long as $\lambda_{uv}(\check{\varepsilon})$ coincides with $\lambda(\check{\varepsilon})$, Lemma 5.2 still holds when h is substituted for g.

It is worth stressing that the equivalence in Lemma 5.2 is applicable only at a point $\lambda(\check{\varepsilon}) \in \sigma_{\check{\varepsilon}}(A, B, C, D)$ that satisfies the first-order optimality condition of Lemma 2.11 or 2.13. Since the SVSAR algorithm will not typically converge exactly to such a point, the derivative (5.1) computed by the GGO algorithm may suffer some loss of precision with respect to the true value. On the other hand, the derivatives (5.5) used in hybrid expansion–contraction are valid at any $\check{\varepsilon} \in [0, \varepsilon_k]$, where ε_k is the kth iterate of Algorithm HEC, regardless of where $\lambda_{uv}(\check{\varepsilon})$ lies in $\sigma_{\varepsilon_k}(A, B, C, D)$. Note also that the property that λ'_{uv} or $\tau\lambda'_{uv}$ is real, which was exploited in the proof of Lemma 5.2, generally holds only at points satisfying the first-order optimality condition.

REMARK 5.4 Because of the simplicity condition, both g (or h) and g_{uv} are twice continuously differentiable at $\varepsilon = \check{\varepsilon}$, although the second derivatives do not generally coincide. The second derivative of g can be obtained by extending Guglielmi *et al.* (2013, Theorem 4.1), while the second derivative of g_{uv} can be determined from well-known results for second derivatives of eigenvalues (Lancaster & Tismenetsky, 1985, Theorem 1, p. 396).

6. Improving the SVSAR subroutine

Two potential downsides of using the SVSAR subroutine, either in the original GGO algorithm or in hybrid expansion–contraction, are an assumption stated in Guglielmi *et al.* (2013) that $m, p \ll n$ must hold for SVSAR to be efficient, and its slow linear rate of convergence. We begin with the former.

At every iteration of SVSAR, the following two linear systems must be solved:

$$\Phi_p b^j = B^* y^j \quad \text{and} \quad \Phi_m c^j = C x^j, \tag{6.1}$$

where x^j and y^j are the right and left computed eigenvectors, respectively, at the jth iteration and

$$\Phi_p = (I_p - \varepsilon^2 D^* D) \quad \text{and} \quad \Phi_m = (I_m - \varepsilon^2 D D^*).$$

Here, for clarity, we have included subscripts on the identity matrices to indicate their dimensions. Hence, the assumption that $m, p \ll n$ to ensure that these two linear systems can be solved quickly. However, by the Sherman–Morrison–Woodbury formula (Golub & Van Loan, 1983), we observe the following equivalence:

$$\Phi_p^{-1} = (I_p - \varepsilon^2 D^* D)^{-1} = I_p + \varepsilon^2 D^* (I_m - \varepsilon^2 D D^*)^{-1} D = I_p + \varepsilon^2 D^* \Phi_m^{-1} D$$

and similarly

$$\Phi_m^{-1} = (I_m - \varepsilon^2 D D^*)^{-1} = I_m + \varepsilon^2 D (I_p - \varepsilon^2 D^* D)^{-1} D^* = I_m + \varepsilon^2 D \Phi_p^{-1} D^*.$$

Thus, solving both linear systems of (6.1) may be done by creating only a single Cholesky factorization of whichever has smaller dimension, say $\Phi_p = LL^*$, and then, using MATLAB notation, performing the two pairs of backsolves as follows:

$$\Phi_p b = B^* y^j \Longrightarrow b = L^* \backslash (L \backslash (B^* y^j)),$$
$$\Phi_m c = C x^j \Longrightarrow c = C x^j + \varepsilon^2 D (L^* \backslash (L \backslash (D^* (C x^j)))).$$

We may thus relax the condition on SVSAR's viability to require only that $\min(p, m)$ is small, while $\max(p, m)$ is free to be on the order of n. If D is given as a rank-k outer product, we may perform a similar procedure using the Sherman–Morrison–Woodbury formula to solve both systems of (6.1) where the dominating cost is performing only two LU factorizations of two small $k \times k$ matrices. On the other hand, if D is not given explicitly (since if it is large, an explicit form may be dense), we can instead use a warm-starting strategy to at least provide the conjugate gradient method with a good starting vector for each solve. See Mitchell (2014, Section 4.1) for a more detailed discussion.

Regarding the issue of SVSAR's slow convergence, hybrid expansion–contraction's ability to safely contract early, as discussed in Section 4.3, often significantly reduces the total number of SVSAR iterations incurred. However, this is not a panacea, so we also consider accelerating SVSAR directly.

For the special case of $B = I$, $C = I$ and $D = 0$, a superlinearly converging subspace acceleration method for computing the ε-pseudospectral abscissa was presented in Kressner & Vandereycken (2014) and, in practice, the authors observe dramatically faster performance compared with the algorithm of Guglielmi & Overton (2011). However, the subspace-accelerated algorithm involves computing the smallest singular value of $A - \mu_k I$ for each $\mu_k \in \mathbb{C}$ that it produces per iteration, which may sometimes be costly, and furthermore, the technique seems difficult to extend to the structured perturbations

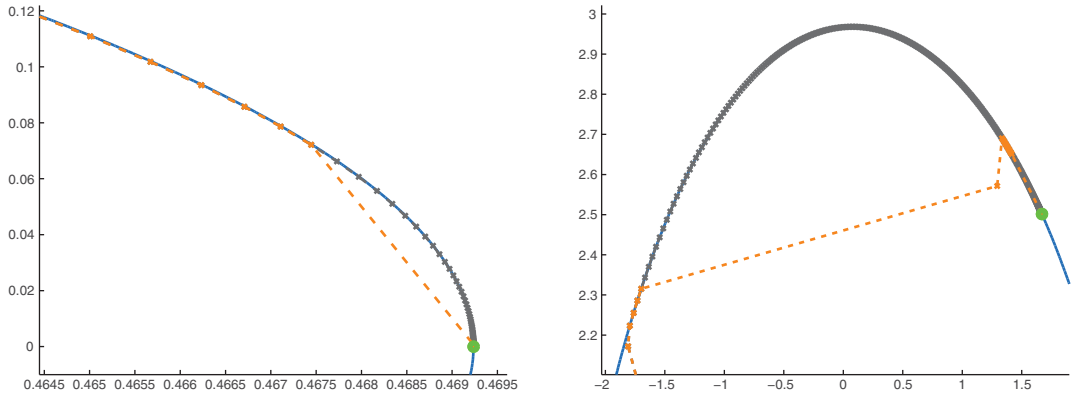

FIG. 5. Left: example ROC3 (continuous time). Right: example ROC5 (discrete time). Each panel depicts a region of the spectral value set boundary of each problem for perturbation level $\varepsilon = 10^{-2}$. The iterates of SVSAR without vector extrapolation enabled are plotted as closely spaced $\times$'s while the iterates of SVSAR with vector extrapolation enabled are plotted on top connected by dashed lines. For each panel, the final point that both variants converge to is indicated by the solid circle.

required for spectral value sets. So, we instead consider an alternative vector extrapolation-based technique which circumvents both problems.

The idea is to exploit vector extrapolation methods such as Graves-Morris (1994), Jbilou & Sadok (2000) and Smith *et al.* (1987) to the SVSAR iterates, but we cannot apply them directly to the vectors u^j and v^j, because these are well defined only up to a scalar and hence generally do not converge. However, the outer product $u^j(v^j)^*$ defining each rank-1 perturbation is unique and is generally convergent. Explicitly extrapolating the entire rank-1 matrix would be far too costly, so instead we extrapolate it *implicitly* by selecting a row and column of the rank-1 matrix to extrapolate. From the vector sequences of u^j and v^j given by SVSAR, we may form two sequences corresponding to how a single row and column of the rank-1 perturbation matrices evolve and then apply vector extrapolation to those two sequences. Using the extrapolated row and column, we then recover two vectors that have an outer product which recovers the extrapolated row and column for the rank-1 matrix. As these two recovered vectors will have arbitrary normalization, we must normalize these two vectors to have unit norm as a final step. Provided that the extrapolated perturbation indeed does provide progress, we then continue SVSAR from that perturbation and attempt a subsequent extrapolation after some prescribed number of iterations. Otherwise, the extrapolation perturbation is discarded and SVSAR continues normally, until another extrapolation opportunity is attempted. We note that it is often better to discard the initial few iterates of SVSAR before attempting extrapolation since those steps are typically quite large and can thus yield poor extrapolations.

For computing the vector extrapolations, we use the minimum polynomial extrapolation method of Cabay & Jackson (1976), whose main cost is solving an over-determined linear least squares problem. Extrapolating a row and column from k implicitly formed rank-1 matrices requires $\mathcal{O}((p+m)k^2)$ flops, and as k will typically be small, such as 5, the method can scale to problems where D has large dimensions as well as the special pseudospectral case when $B = I$, $C = I$ and $D = 0$ and $u^j, v^j \in \mathbb{C}^n$. In Fig. 5, we see that the vector extrapolation technique on sets of five vectors at a time substantially accelerates SVSAR, reducing the number of iterations from 136 to 18 on the continuous-time problem ROC3 and from 548 to 28 on the discrete-time problem ROC5. For more details, see Mitchell (2014, Section 4.2).

7. Implementation

We have implemented our hybrid expansion–contraction algorithm and initial upper bounding procedure from scratch in a MATLAB code called `getStabRadBound` (get stability radius bound).[5] Though we could have directly modified the SVSAR subroutine from the implementation of the GGO algorithm, `hinfnorm` version 1.02,[6] we have instead implemented an entirely new version of SVSAR designed around supporting numerous improvements and efficient integration into the hybrid expansion–contraction iteration. Our implementation of SVSAR not only efficiently scales to problems with large-dimensional D matrices and supports acceleration via our vector extrapolation technique but also includes the ability to store the state of all relevant computations so that they can be reloaded at will, without needing to either recompute them or manually manage saving and reloading them all. This checkpointing feature is particular useful for efficient hand-off between the expansion and contraction phases.

Our implementation of HEC minimizes the number of matrix–vector products required, most notably by precomputing Bu^j and $(v^j)^*C$ at each step and then utilizing the resulting vectors in the function that multiplies a vector by $M(\varepsilon u^j(v^j)^*)$ (since forming it explicitly would cause fill-in). Thus, to compute an eigenvalue of $M(\varepsilon u^j(v^j)^*)$ with an iterative sparse eigensolver such as the MATLAB function `eigs`, multiple matrix–vector products are needed only with A, approximately one per iteration of the implicitly restarted Arnoldi method. In this way, the number of matrix–vector products involving B, C and D is limited to only about double the total number of eigentriples computed by the algorithm, rather than being the same order as the number of matrix–vector products done with A. This reduction in matrix–vector products should provide an additional tangible benefit for high-dimensional problems where p, m are large and accordingly, our hybrid expansion–contraction code supports A, B, C and D to each be given as function handles.

Finally, as HEC usually generates a converging sequence of matrices, our implementation also supports the option of recycling the previous right and left eigenvectors corresponding to the rightmost (or outermost) eigenvalue to potentially warm-start `eigs` for the next computed eigentriple. Alternatively, one may also choose to use the average of all the eigenvectors returned by `eigs` as the initial vector for the next eigenproblem to solve (done for the right and left eigenvectors separately). Either option can potentially help reduce the number of iterations required in the implicitly restarted Arnoldi method used by `eigs`.

As a result of the additional complexity of all these configurable features, we have elected to implement the new algorithm using an object-oriented architecture. However, we have facilitated this not by actually using true classes but instead via liberal use of the nested function feature in MATLAB, with the hope that the MATLAB code interpretation overhead would be lower with this approach than using full-blown objects, though we have not done a comparison. In any case, we thus expect the interpretation speed of `getStabRadBound` to be slower than that of `hinfnorm`, but this difference should have a negligible effect on large-scale problems.

7.1 *Tolerances*

We now describe the main tolerance criteria necessary to realize a practical implementation of hybrid expansion–contraction, starting with the continuous-time case. For the SVSAR expansion phase, we set

[5] The `getStabRadBound` code is publicly available from http://www.cims.nyu.edu/~tmitchell.

[6] `hinfnorm` (H-infinity norm approximation). Available at http://www.cims.nyu.edu/~mert/software/hinfinity.html. Accessed 23 October 2014.

it to terminate if $\mathrm{Re}(\lambda^j) - \mathrm{Re}(\lambda^{j-1}) < \tau_{uv}|\mathrm{Re}(\lambda_j)|$, for some small user-provided tolerance $\tau_{uv} \in \mathbb{R}^{++}$. For the Newton-bisection contraction phase, since hybrid expansion–contraction requires all iterates to remain strictly in the right half-plane, finding the root of $\alpha(M_{uv}(\cdot))$ to some tolerance is potentially problematic since it could result in a solution just slightly in the left half-plane. To address this, we instead have the Newton-bisection routine find the root $\hat{\varepsilon}_k$ of $\alpha(M_{uv}(\cdot)) - 0.5\tau_\varepsilon$ for some small user-provided tolerance $\tau_\varepsilon \in \mathbb{R}^{++}$ and have it terminate when $|\alpha(M_{uv}(\hat{\varepsilon}_k)) - 0.5\tau_\varepsilon| < 0.5\tau_\varepsilon$. By doing so, we ensure a contracted value $\hat{\varepsilon}_k$ such that $0 < \alpha(M_{uv}(\hat{\varepsilon}_k)) < \tau_\varepsilon$. Furthermore, we also set the Newton-bisection iteration to terminate the contraction process if $\hat{\varepsilon}_k < \tau_\varepsilon \varepsilon_k$ and $\alpha(M_{uv}(\hat{\varepsilon}_k)) > 0$ are both satisfied. Finally, the contraction routine keeps track of the most contracted value of $\hat{\varepsilon}_k$ such that $\alpha(M_{uv}(\hat{\varepsilon}_k)) > 0$ is satisfied and will return that best encountered value of $\hat{\varepsilon}_k$. In the case that all the subsequent iterates in the Newton-bisection iteration are in the left half-plane though possibly closer to the imaginary axis, hybrid expansion–contraction can at least continue with some amount of contraction while remaining in the right half-plane, even if its termination tolerances were not satisfied. These requirements (in addition to the optimizations discussed above) necessitated that we implement our own custom Newton-bisection code for `getStabRadBound`.

Algorithm HEC must be set to terminate once it can no longer make progress contracting and expanding as measured by the tolerances τ_ε and τ_{uv}, respectively. We terminate it when either both phases fail to make any progress consecutively, in either order, or if for λ_{k+1} produced by SVSAR, $\mathrm{Re}(\lambda_{k+1}) < \tau_\varepsilon + \tau_{uv}$ holds. The latter condition is necessary since SVSAR will typically always at least take a single step, assuming its line search does not fail. If we were to set the tolerance condition any tighter, HEC might take a long sequence of alternating expansion–contraction steps where SVSAR is able to take only a single step of exceedingly small step size less than τ_ε while the contraction phases fail to reduce ε_k any further.

The tolerance criteria for the discrete-time case are described analogously by replacing $\mathrm{Re}(\lambda^j) - \mathrm{Re}(\lambda^{j-1}) < \tau_{uv}|\mathrm{Re}(\lambda_j)|$ by $|\lambda^j| - |\lambda^{j-1}| < \tau_{uv}|\lambda_j|$, changing the condition $\alpha(\cdot) > 0$ to $\rho(\cdot) > 1$, and finally, changing $\mathrm{Re}(\lambda_{k+1}) < \tau_\varepsilon + \tau_{uv}$ to $|\lambda_{k+1}| < 1 + \tau_\varepsilon + \tau_{uv}$.

8. Numerical results

In order to evaluate HEC and its new initial upper bound procedure, we ran experiments on 33 small and 14 large-scale test problems used by Guglielmi *et al.* (2013) and present the results in the aggregate. We used version 1.0 of `getStabRadBound` and version 1.02 of `hinfnorm`, the implementation of the GGO algorithm. All experiments were done using MATLAB R2014a running on a single-user desktop with Ubuntu 14.04 (64-bit) and an Intel i7-3770K CPU with 8 GB of RAM.[7]

For `getStabRadBound`, we used $\tau_\varepsilon = 10^{-10}$ and $\tau_{uv} = 10^{-12}$ for the termination tolerances. Correspondingly for `hinfnorm`, we reused τ_ε for the tolerance for `rtsafe` (the Newton-bisection code used by `hinfnorm`) and τ_{uv} for `hinfnorm`'s implementation of SVSAR. We allowed `getStabRadBound` to take a maximum of 100 iterations to find an upper bound using the method of Section 4.4 and similarly allowed up to a maximum of 100 HEC iterations. As `hinfnorm` does not provide a user option to individually set maximum iteration limits for the upper bound and Newton-bisection phases separately, we set its maximum iteration limit to 200. Both `getStabRadBound` and

[7] We discarded the results from a 34th small-scale problem (`ROC2`) as `hinfnorm` abnormally terminated mid-computation due to `eig` crashing, which is a known but not commonly encountered platform-specific bug. Though we verified that `hinfnorm` ran successfully on `ROC2` using a Mac, the timings would not have been comparable with the data collected using the Linux machine.

hinfnorm were set such that their respective versions of SVSAR could take up to 1000 iterations per call while the Newton-bisection contraction phase of HEC was allowed up to 10 iterations per call.

REMARK 8.1 The rtsafe code used by hinfnorm to perform the Newton-bisection iteration makes no check upon how close the function value is to zero. Instead, it merely terminates once progress in ε_k has slowed but this is no guarantee that hinfnorm has converged to a point on or acceptably near the imaginary axis. As a consequence, hinfnorm may terminate at a point significantly farther away from the imaginary axis and we observe this in practice on some problems. For evaluative purposes, it is not so problematic if hinfnorm terminates while at a locally rightmost point in the right half-plane, since doing so will simply lower the reported approximation to $\|G\|_\infty$ and thus its results reflect that it has not converged in these cases. If it instead terminates at a locally rightmost point that is in the left-plane to a significant degree, hinfnorm will incorrectly report a value of $\|G\|_\infty$ that may in fact be too large. Thus, we consider hinfnorm to have failed if its last locally rightmost point λ_k found satisfies $\mathrm{Re}(\lambda_k) < -100\tau_\varepsilon$ (or $|\lambda_k| < 1 - 100\tau_\varepsilon$ for the discrete-time problems).

Finally for SVSAR, after some experimentation with various-sized sets of the immediately preceding vector iterates to use for vector extrapolation, as well as different relative-step-size termination tolerances for allowing early contraction as discussed in Section 4.3, we settled on using extrapolation with the five previous vectors and a relative-step-size tolerance of 10^{-2}, as these parameters seemed to provide the best performance.

8.1 *Upper bound method variants*

In order to measure how efficient the method of Section 4.4 is for finding upper bounds compared with the strategy employed by hinfnorm, we ran both methods on the small-scale problems and counted the total number of computed eigentriples that each method required to compute their upper bounds for all the problems in the test set. As we expected, our new upper bound method is indeed fast, requiring a total of just 121 computed eigentriples to find all 33 upper bounds and furthermore, that number also includes calculating the rightmost or outermost eigenvalue of A for each of those problems, which is not necessary if the user initializes the routine with any acceptable nonzero perturbation $\Delta_0 = \varepsilon_0 u_0 v_0^*$. In contrast, hinfnorm 1.02's bounding phase described above required computing a total of 5606 eigentriples. However, we noted that the new upper bound initialization occasionally resulted in HEC converging to worse (lower) local maximizers compared with the maximizers HEC converged to when initialized by hinfnorm's bounding procedure. Consequently, before beginning HEC in getStabRadBound, we first call the new upper bound method, which efficiently finds an upper bound $\varepsilon_{\mathrm{ub}} \geqslant \varepsilon_\star$, and follow it by a single SVSAR call to then expand rightward (outward for the discrete-time case) as much as possible in the $\varepsilon_{\mathrm{ub}}$-spectral value set via updating the perturbation vectors u and v. The benefits in doing so are that we still avoid calling SVSAR more than once to find an upper bound to initialize HEC with, and this strategy seems to retain the approximation quality of HEC when initialized with hinfnorm's bounding code. While it is not nearly as fast as the new upper bound method alone, the 2507 computed eigentriples to compute the 33 upper bounds is still a significant reduction compared with hinfnorm's 5606.

A possible insight as to why initializing HEC using only the result of the fast upper bound procedure, without the subsequent single call to SVSAR, seems to cause convergence to an overall greater number of worse (lower) local maximizers on the test set may be gleaned by examining the panels of Fig. 2. Consider the top right panel where we see that the loss of a locally rightmost point of the spectral value set for $\varepsilon = 0.1$ causes the GGO algorithm to break down. Now, suppose the initial upper bound method has found a point in the upper right region of the spectral value set in the right

half-plane for $\varepsilon = 0.1$, near where SVSAR begins its iteration. If SVSAR is called, then it is likely that HEC will converge to the globally rightmost point on the imaginary axis for this problem, as shown in the top right panel for the GGO algorithm. However, if the initial contraction phase is begun immediately after the upper bound has been found then it seems very likely that HEC will instead converge to the topmost, only locally rightmost, point on the imaginary axis of the spectral value for some value $\tilde{\varepsilon}$ slightly less than 0.09, as we can see from the bottom left panel.

8.2 *Small-scale evaluation*

In order to numerically validate the approximations found by `getStabRadBound`, we consider the number of these approximations computed over the entire test set which agree to specific levels of precision with respect to the true value of the H_∞ norm for each problem, as computed by the MATLAB function `getPeakGain`. We consider these counts at multiple levels of precision since `getStabRadBound` and `hinfnorm` may find only local maximizers in some cases. We compute the relative differences of the approximations compared with the value computed by `getPeakGain` given a tolerance of 10^{-10}. For a precision level of 10^{-8}, we count the number of approximations that are either strictly greater than the value computed by `getPeakGain` or have a relative difference of at most 10^{-8}. We also tabulate counts for precision levels of 10^{-6} and 10^{-4}. However, for the counts for `hinfnorm`, we do not include any result that satisfies the failure-to-converge condition described in Remark 8.1 (there are three such problems in the small-scale test set: `ROC3`, `AC17` and `AC6`). We note that `getStabRadBound` cannot fail in this way as long as it first finds an upper bound, which it did successfully for all the problems. For the small-scale test set, all eigentriples were computed by the MATLAB function `eig` (not `eigs`).

In the small-scale section of Table 1, we see that `getStabRadBound` is nearly 2.5 times faster than `hinfnorm` on the entire test set and furthermore, `getStabRadBound` mostly finds good approximations to $\|G\|_\infty$. Enabling vector extrapolation in `getStabRadBound` shows a dramatic reduction in the number of eigentriples calculated over the test set but because these are small-scale problems, it does not fully translate to a correspondingly large reduction in total CPU time. Enabling extrapolation also resulted in convergence to different local maximizers for a couple of problems. Using the relative-step-size termination condition in SVSAR for the HEC phase also dramatically reduces the number of computed eigentriples, allowing `getStabRadBound` to complete the test set over 4.5 times faster than `hinfnorm`, while additionally enabling extrapolation increases the speed-up factor to over 5. As there is a large amount of variability in the test set, we also provide average, median and range data on the relative speed-up *per problem* with respect to the number of eigentriples computed and with respect to the CPU running time in the small-scale section of Table 2. We see that `getStabRadBound` with the relative-step-size termination option enabled is up to nearly 47 times faster (on `CSE2`) in terms of CPU time and when vector extrapolation is also enabled, we see that speed-up per problem ranges from nearly twice as fast to almost 5 times as fast compared with `hinfnorm` as reported by the median and average, respectively.

8.3 *Large-scale matrices*

We present analogous performance data in the large-scale sections of Tables 1 and 2 for the large-scale test set, with the following notable exceptions. First, as we cannot tractably compute the true value of $\|G\|_\infty$ for these large-dimensional problems,[8] we instead use the largest approximation computed

[8] We tested `getPeakGain` and `getStabRadBound` using a limited set of medium-scale sparse problems to assert the scaleability of our proposed method over `getPeakGain`. On the smallest of these problems, `getPeakGain` was at least an

TABLE 1 *NB denotes the* `hinfnorm` *code while HEC denotes the* `getStabRadBound` *code, #ET is the number of computed eigentriples, the variable E denotes that vector extrapolation is enabled and RS denotes that the relative-step-size termination tolerance is enabled for SVSAR with a value of* 10^{-2}. *For the large-scale problems, V denotes that eigenvector recycling is enabled such that the initial vector for* `eigs` *is set to the average of the eight requested eigenvectors computed by* `eigs` *for the previous computed eigentriple. The columns '# Rel diff to best' show the number of problems a given algorithm successfully converged on, to the three varying degrees of precision, with respect to the best of the computed values for each problem; for the small-scale problems, the best computed value for each problem is the true value obtained by* `getPeakGain`. *The variable S is the number of problems that a particular code successfully converged on, solely according to the convergence failure criteria of Remark 8.1*

`getStabRadBound` overall performance

		Totals		# Rel diff to best			
Scale	Alg + opts	#ET	sec	10^{-8}	10^{-6}	10^{-4}	S
Small	NB	32112	465.49	18	22	25	30
	HEC	16665	199.99	21	25	29	33
	HEC + E	9708	168.71	19	23	28	33
	HEC + RS	10565	99.70	21	25	28	33
	HEC + E, RS	6767	89.64	21	25	28	33
Large	NB	4196	20920	11	11	11	13
	HEC	2338	3756	9	10	12	14
	HEC + V	2336	2362	10	11	13	14
	HEC + E	636	1504	10	12	13	14
	HEC + E, V	690	1110	10	11	13	14
	HEC + RS	861	1046	9	10	11	14
	HEC + RS, V	849	919	10	11	12	14
	HEC + E, RS	700	960	9	10	11	14
	HEC + E, RS, V	794	841	11	12	13	14

from all the methods for each problem as a surrogate to compute the relative differences used in tabulating how many approximations for a given method agreed to a given level of precision with the best of the results calculated. Second, we use `eigs` with its default options to compute the rightmost or outermost eigentriples but with eight eigenvalues requested per call to `eigs` (done for the right and left eigenvectors separately), to be consistent with the experiments in Guglielmi *et al.* (2013). We also additionally evaluated both eigenvector recycling types that are optionally available in our code to attempt to reduce the number of iterations `eigs` requires. Both seemed to have a beneficial effect with respect to run-times on the large-scale test set but we report only the results for recycling the average of the eight computed eigenvectors from the previous computed eigentriple, as it outperformed recycling only the previously selected eigenvector. We note that `hinfnorm` fails on one problem in this test set (`skewlap3d`), according to the criterion in Remark 8.1.

order of magnitude slower than `getStabRadBound` and this performance gap widened to over three orders of magnitude slower in favour of `getStabRadBound` as the dimension of A increased. It is also worth noting that merely calling `ss(A,B,C,D)` to form the state-space model for input to `getPeakGain` automatically converts sparse matrices to dense versions.

TABLE 2 *Average, median and range of per-problem speed-ups with respect to number of eigentriples computed and the CPU time required for each problem. See Table 1 caption for definitions of HEC options E, RS and V*

Per-problem speed-ups of `getStabRadBound` relative to `hinfnorm`							
		No. of computed ET			CPU time		
Scale	Opts	Avg	Med	Range	Avg	Med	Range
Small	—	2.25	1.32	[0.16, 16.22]	1.20	0.72	[0.19, 5.47]
	E	4.47	2.64	[0.21, 21.89]	2.27	1.13	[0.22, 11.71]
	RS	9.19	3.12	[0.26, 169.93]	3.55	1.20	[0.30, 46.87]
	E,RS	12.38	3.45	[0.26, 169.93]	4.71	1.71	[0.18, 45.41]
Large	—	3.99	1.33	[0.54, 21.50]	8.80	1.62	[0.41, 57.28]
	V	3.56	1.46	[0.51, 18.63]	11.48	2.09	[0.64, 64.86]
	E	5.48	1.68	[0.65, 23.29]	9.30	1.72	[0.23, 64.80]
	E,V	4.07	1.52	[0.50, 14.81]	13.92	2.20	[0.57, 78.51]
	RS	4.94	1.89	[0.50, 25.41]	13.79	2.40	[0.53, 66.41]
	RS,V	4.77	2.22	[0.50, 21.50]	15.24	2.55	[0.71, 82.43]
	E,RS	5.42	2.21	[0.80, 25.41]	14.51	1.88	[0.52, 66.19]
	E,RS,V	5.16	2.05	[0.50, 18.63]	15.77	2.56	[0.75, 75.31]

In the large-scale section of Table 1, as for the small-scale test set, we find overall that `getStabRadBound` appears to be successfully converging to good local or possibly global maximizers. Furthermore, even without any options enabled, `getStabRadBound` completes the entire test set 5.6 times faster than `hinfnorm`. Enabling eigenvector recycling results in a 14–59% speed boost, depending on what other options are simultaneously enabled. In contrast to the small-scale results, the substantial reduction in the number of computed eigentriples over the large-scale test set achieved by enabling extrapolation is actually accompanied by an even larger reduction in total running time, resulting in an overall speed-up factor of 13.9 when compared with the total CPU time of `hinfnorm`. Enabling eigenvector recycling on top of extrapolation further increases that CPU time speed-up factor to 18.8, even though it actually also increased the number of computed eigentriples by 8.5%. Enabling just the relative-step-size termination feature provides an even greater speed-up factor of 20.0 times faster, again despite an increase in the number of eigentriples computed compared with the extrapolation variants. Simultaneously enabling extrapolation, relative-step-size termination and eigenvector recycling allows `getStabRadBound` to complete the large-scale test set 24.9 times faster than `hinfnorm`. Though we do not report details and results here for replacing the backtracking bisection strategy in SVSAR's line search with cubic models, `getStabRadBound` also supports this option and enabling it does in fact provide a modest performance boost. Compared with `hinfnorm` on the sparse test set, enabling the cubic line search option results in an overall speed-up factor of 5.8 times faster for `getStabRadBound` with no options enabled and 26.2 times faster for `getStabRadBound` with extrapolation, relative-step-size termination and eigenvector recycling also all enabled.

Referring to the large-scale per-problem speed-up results in Table 2, we see that depending on the problem, `getStabRadBound` with relative-step-size termination and eigenvector recycling enabled can be over 82 times faster than `hinfnorm` (on `skewlap3d` in the discrete-time problem set). Judging from the per-problem speed-up median values, `getStabRadBound` appears to generally be 2.5 times

TABLE 3 *Average, median and range of the number of iterations in the NB (Newton-bisection) converging phase of* `hinfnorm` *and the HEC phase of* `getStabRadBound` *across the small-scale test set and the large-scale test set*

Per-problem convergence of HEC						
	Small scale			Large scale		
	Avg	Med	Range	Avg	Med	Range
NB iters	8.21	6.00	[4, 28]	5.79	5.00	[4, 12]
HEC iters	4.21	4.00	[2, 10]	2.50	2.00	[1, 4]
HEC + RS iters	5.36	5.00	[3, 14]	2.86	2.00	[1, 5]

faster than `hinfnorm` when all acceleration options are enabled and an order of magnitude faster as reported by the average time per problem. On the problems where `getStabRadBound` is slower than `hinfnorm`, even the most unfavourable of these typically demonstrates that the worst-case CPU time difference between the two methods is less than a factor of 2 in favour of `hinfnorm`. Comparing the individual approximations computed by `getStabRadBound` to the ones provided by `hinfnorm`, we found that in the worst case, on the discrete-time version of `skewlap3d`, `getStabRadBound` only agreed with `hinfnorm`'s better result to two digits. However, this was far from typical as the median of the computed relative differences compared with the best approximation reported demonstrated that `getStabRadBound` usually agreed to 11 digits with `hinfnorm` on the large-scale test set and furthermore, for `markov` in the discrete-time case, `getStabRadBound` found an approximation three times larger than the one found by `hinfnorm`.

8.4 *Convergence rate of hybrid expansion–contraction*

We now turn to empirically validating whether or not the quadratic convergence of hybrid expansion–contraction claimed in Section 4.2 is realized in practice.

In Table 3, we report the average, median and range of per-problem HEC iterations taken until convergence over the small- and large-scale test sets. On the small-scale problems, at worst HEC took nine and ten iterations to converge on problems `AC11` and `AC6` but upon closer analysis, we observed that `getStabRadBound`'s HEC phase in fact resolved the first eight digits of each approximation within four iterations and the remaining iterations of HEC were due to τ_ε and τ_{uv} being set too small. We note that this is a limitation of double-precision hardware for problems that are exceptionally sensitive to changes in ε. We also show the corresponding statistics for the Newton-bisection iteration of `hinfnorm` for comparison and find that the HEC phase of `getStabRadBound` typically requires even fewer iterations than `hinfnorm`'s Newton-bisection iteration, most likely due to HEC's ability to sometimes take even larger steps than that offered by the GGO algorithm in the negative dampening case. One might argue that this is an unfair comparison because a single iteration of HEC comprises both the contraction and the expansion phases while in contrast, the only work done in a single iteration of GGO's Newton-bisection method is the expansion. However, we have found that the contraction phase is exceedingly efficient and reliable and that on average, it computes only three eigentriples before converging to the imaginary axis or unit circle. Finally, we see from the third line of Table 3 that enabling `getStabRadBound`'s relative-step-size termination tolerance, which reduces our theoretical convergence rate analysis of HEC from quadratic to superlinear, actually increases the average

number of HEC iterations only ever so slightly; this small extra cost is far outweighed by the resulting overall large performance increase due to dramatically reducing of the number of SVSAR iterations incurred.

9. Conclusion

In this paper, we have presented a fast new algorithm for approximating the H_∞ norm that, under reasonable assumptions, guarantees convergence to stationary points of the underlying optimization problem, while in practice, it typically converges to local maximizers. Since the new algorithm is scaleable, an interesting subject for future work will be to exploit it to develop methods for designing and optimizing fixed-order controllers for large-scale dynamical systems, as can currently be done for small-scale systems using the open-source toolbox.[9]

Acknowledgement

We gratefully acknowledge the helpful comments of two anonymous referees.

Funding

National Science Foundation (DMS-1317205, in part).

References

Benner, P. & Voigt, M. (2014) A structured pseudospectral method for H_∞-norm computation of large-scale descriptor systems. *Math. Control Signals Syst.*, **26**, 303–338.

Boyd, S. & Balakrishnan, V. (1990) A regularity result for the singular values of a transfer matrix and a quadratically convergent algorithm for computing its L_∞-norm. *Syst. Control Lett.*, **15**, 1–7.

Bruinsma, N. A. & Steinbuch, M. (1990) A fast algorithm to compute the H^∞-norm of a transfer function matrix. *Syst. Control Lett.*, **14**, 287–293.

Burke, J. V., Lewis, A. S. & Overton, M. L. (2003) Robust stability and a criss-cross algorithm for pseudospectra. *IMA J. Numer. Anal.*, **23**, 359–375.

Cabay, S. & Jackson, L. W. (1976) A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM J. Numer. Anal.*, **13**, 734–752.

Dembo, R. S., Eisenstat, S. C. & Steihaug, T. (1982) Inexact Newton methods. *SIAM J. Numer. Anal.*, **19**, 400–408.

Freitag, M. A., Spence, A. & Van Dooren, P. (2014) Calculating the H_∞-norm using the implicit determinant method. *SIAM J. Matrix Anal. Appl.*, **35**, 619–635.

Golub, G. H. & Van Loan, C. (1983) *Matrix Computations*. Baltimore: Johns Hopkins University Press.

Graves-Morris, P. R. (1994) A review of Padé methods for the acceleration of convergence of a sequence of vectors. *Appl. Numer. Math.*, **15**, 153–174.

Guglielmi, N., Gürbüzbalaban, M. & Overton, M. L. (2013) Fast approximation of the H_∞ norm via optimization over spectral value sets. *SIAM J. Matrix Anal. Applic.*, **34**, 709–737.

Guglielmi, N. & Overton, M. L. (2011) Fast algorithms for the approximation of the pseudospectral abscissa and pseudospectral radius of a matrix. *SIAM J. Matrix Anal. Applic.*, **32**, 1166–1192.

Hinrichsen, D. & Pritchard, A. J. (2005) *Mathematical Systems Theory I: Modelling, State Space Analysis, Stability and Robustness*. Berlin, Heidelberg and New York: Springer.

[9] HIFOO (H_∞ fixed-order optimization). Available at http://www.cs.nyu.edu/overton/software/hifoo/. Accessed 23 October 2014.

HORN, R. A. & JOHNSON, C. R. (1990) *Matrix Analysis*. Cambridge: Cambridge University Press. Corrected reprint of the 1985 original.

JBILOU, K. & SADOK, H. (2000) Vector extrapolation methods: applications and numerical comparison. *J. Comput. Appl. Math.*, **122**, 149–165.

KAROW, M. (2003) Geometry of spectral value sets. *Ph.D. Thesis*, Universität Bremen.

KRESSNER, D. & VANDEREYCKEN, B. (2014) Subspace methods for computing the pseudospectral abscissa and the stability radius. *SIAM J. Matrix Anal. Appl.*, **35**, 292–313.

LANCASTER, P. & TISMENETSKY, M. (1985) *The Theory of Matrices*. New York and London: Academic Press.

LEHOUCQ, R. B. & SORENSEN, D. C. (1996) Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl.*, **17**, 789–821.

MENGI, E. & OVERTON, M. L. (2005) Algorithms for the computation of the pseudospectral radius and the numerical radius of a matrix. *IMA J. Numer. Anal.*, **25**, 648–669.

MITCHELL, T. (2014) Robust and efficient methods for approximation and optimization of stability measures. *Ph.D. Thesis*, New York University. Available at http://cs.nyu.edu/web/Research/Theses/mitchell_tim.pdf. Accessed 23 October 2014.

SMITH, D. A., FORD, W. F. & SIDI, A. (1987) Extrapolation methods for vector sequences. *SIAM Rev.*, **29**, 199–233.

TREFETHEN, L. N. & EMBREE, M. (2005) *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton, New Jersey: Princeton University Press.

VAN LOAN, C. (1985) How near is a stable matrix to an unstable matrix? *Linear Algebra and Its Role in Systems Theory (Brunswick, Maine, 1984)*. Contemporary Mathematics, Vol. 47. Providence, RI: American Mathematical Society, pp. 465–478.