SIAM J. SCI. COMPUT.
Vol. 40, No. 5, pp. A3609–A3635

FASTER AND MORE ACCURATE COMPUTATION OF THE $\mathcal{H}_\infty$ NORM VIA OPTIMIZATION*

PETER BENNER† AND TIM MITCHELL†

Abstract. In this paper, we propose an improved method for computing the $\mathcal{H}_\infty$ norm of linear dynamical systems that results in a code that is often several times faster than existing methods. By using standard optimization tools to rebalance the work load of the standard algorithm due to Boyd, Balakrishnan, Bruinsma, and Steinbuch, we aim to minimize the number of expensive eigenvalue computations that must be performed. Unlike the standard algorithm, our modified approach can also calculate the $\mathcal{H}_\infty$ norm to full precision with little extra work and also offers more opportunity to further accelerate its performance via parallelization. Finally, we demonstrate that the local optimization we have employed to speed up the standard globally convergent algorithm can also be an effective strategy on its own for approximating the $\mathcal{H}_\infty$ norm of large-scale systems.

Key words. H-infinity norm, L-infinity norm, robust control, transfer functions, descriptor systems, Hamiltonian

AMS subject classifications. 93C05, 93C55, 93A15, 93B40, 93B36, 93D09

DOI. 10.1137/17M1137966

1. Introduction.

Consider the continuous-time linear dynamical system

$$E\dot{x} = Ax + Bu, \tag{1a}$$

$$y = Cx + Du, \tag{1b}$$

where $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $C \in \mathbb{C}^{p\times n}$, $D \in \mathbb{C}^{p\times m}$, and $E \in \mathbb{C}^{n\times n}$. The system defined by (1) arises in many engineering applications, and as a consequence, there has been a strong motivation for fast methods to compute properties that measure the sensitivity of the system or its robustness to noise. Specifically, a quantity of great interest is the $\mathcal{H}_\infty$ norm, which is defined as

$$\|G\|_{\mathcal{H}_\infty} \coloneqq \sup_{\omega\in\mathbb{R}} \|G(\mathbf{i}\omega)\|_2, \tag{2}$$

where

$$G(\lambda) = C(\lambda E - A)^{-1}B + D \tag{3}$$

is the associated *transfer function* of the system given by (1). The $\mathcal{H}_\infty$ norm measures the maximum sensitivity of the system; in other words, the higher the value of the $\mathcal{H}_\infty$ norm, the less robust the system is, an intuitive interpretation considering that the $\mathcal{H}_\infty$ norm is in fact the reciprocal of the *complex stability radius*, which itself is a generalization of the *distance to instability* [23, section 5.3]. The $\mathcal{H}_\infty$ norm is also a key metric for assessing the quality of reduced-order models that attempt to capture/mimic the dynamical behavior of large-scale systems; see, e.g., [2, 5]. Before continuing, as various matrix pencils of the form $\lambda B - A$ will feature frequently in this work, we use the notation (A, B) to abbreviate them.

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 10, 2017; accepted for publication (in revised form) July 17, 2018; published electronically October 23, 2018.

http://www.siam.org/journals/sisc/40-5/M113796.html

†Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, 39106 Germany (benner@mpi-magdeburg.mpg.de, mitchell@mpi-magdeburg.mpg.de).

When $E = I$, the $\mathcal{H}_\infty$ norm is finite as long as A is stable, whereas an unstable system would be considered infinitely sensitive. If $E \neq I$, then (1) is called a *descriptor system*. Assuming that E is singular but (A, E) is regular and at most index 1, then (2) still yields a finite value, provided that all the controllable and observable eigenvalues of (A, E) are finite and in the open left half plane, where for λ an eigenvalue of (A, E) with right and left eigenvectors x and y, λ is considered *uncontrollable* if $B^*y = 0$ and *unobservable* if $Cx = 0$. However, the focus of this paper is not about detecting when (2) is infinite or finite but to introduce an improved method for computing the $\mathcal{H}_\infty$ norm when it is finite. Thus, for conciseness in presenting our improved method, we will assume in this paper that any system provided to an algorithm has a finite $\mathcal{H}_\infty$ norm, as checking whether it is infinite can be considered a preprocessing step.[1]

While the first algorithms [12, 11, 13] for computing the $\mathcal{H}_\infty$ norm date back to nearly 30 years ago, there has been continued interest in improved methods, particularly as the state-of-art methods remain quite expensive with respect to their dimension n, meaning that computing the $\mathcal{H}_\infty$ norm is generally only possible for rather small-dimensional systems. In 1998, [17] proposed an interpolation refinement to the existing algorithm of [11, 13] to accelerate its rate of convergence. In the following year, for the special case of the distance to instability where $B = C = E = I$ and $D = 0$, [22] used an inverse iteration to successively obtain increasingly better locally optimal approximations as a way of reducing the number of expensive Hamiltonian eigenvalue decompositions; as we will discuss in section 4, this method shares some similarity with the approach we propose here. More recently, in 2011, [3] presented an entirely different approach to computing the $\mathcal{H}_\infty$ norm, by finding isolated common zeros of two certain bivariate polynomials. While they showed that their method was much faster than an implementation of [11, 13] on two SISO examples (single-input, single-output, that is, $m = p = 1$), more comprehensive benchmarking does not appear to have been done yet. Shortly thereafter, [7] extended the now standard algorithm of [11, 13] to descriptor systems. There also has been a very recent surge of interest in efficient $\mathcal{H}_\infty$ norm approximation methods for large-scale systems. These methods fall into two broad categories: those that are applicable for descriptor systems with possibly singular E matrices but require solving linear systems [10, 16, 1] and those that do not solve linear systems but require that $E = I$ or that E is at least cheaply inverted [20, 28].

Our contribution in this paper is twofold. First, we improve upon the exact algorithms of [11, 13, 17] to not only compute the $\mathcal{H}_\infty$ norm significantly faster but also obtain its value to machine precision with negligible additional expense (a notable difference compared to these earlier methods). This is accomplished by incorporating local optimization techniques within these algorithms, a change that also makes our new approach more amenable to additional acceleration via parallelization. Second, that standard local optimization can even be used on its own to efficiently obtain locally optimal approximations to the $\mathcal{H}_\infty$ norm of large-scale systems, a simple and direct approach that surprisingly has not yet been considered and is even embarrassingly parallelizable.

The paper is organized as follows. In sections 2 and 3, we describe the standard algorithms for computing the $\mathcal{H}_\infty$ norm and then give an overview of their computational costs. In section 4, we introduce our new approach to computing the $\mathcal{H}_\infty$ norm via leveraging local optimization techniques. Section 5 describes how the results and

[1] We note that while our proposed improvements are also directly applicable for computing the $\mathcal{L}_\infty$ norm, we will restrict the discussion here to just the $\mathcal{H}_\infty$ norm for brevity.

algorithms are adapted for discrete-time problems. We present numerical results in section 6 for both continuous- and discrete-time problems. Section 7 provides additional experiments demonstrating how local optimization can also be a viable strategy for approximating the $\mathcal{H}_\infty$ norm of large-scale systems. Finally, in section 8 we discuss how significant speedups can be obtained for some problems when using parallel processing with our new approach, in contrast to the standard algorithms, which benefit very little from multiple cores. Concluding remarks are given in section 9.

2. The standard algorithm for computing the $\mathcal{H}_\infty$ norm. We begin by presenting a key theorem relating the singular values of the transfer function to purely imaginary eigenvalues of an associated matrix pencil. For the case of simple ODEs, where $B = C = E = I$ and $D = 0$, the result goes back to [15, Theorem 1] and was first extended to linear dynamical systems with input and output with $E = I$ in [12, Theorem 1], and then most recently generalized to systems where $E \neq I$ in [7, Theorem 1]. We state the theorem without the proof, since it is readily available in [7].

THEOREM 2.1. *Let $\lambda E - A$ be regular with no finite eigenvalues on the imaginary axis, $\gamma > 0$ not a singular value of D, and $\omega \in \mathbb{R}$. Consider the matrix pencil $(\mathcal{M}_\gamma, \mathcal{N})$, where*

$$\mathcal{M}_\gamma := \begin{bmatrix} A - BR^{-1}D^*C & -\gamma BR^{-1}B^* \\ \gamma C^*S^{-1}C & -(A - BR^{-1}D^*C)^* \end{bmatrix} \text{ and } \mathcal{N} := \begin{bmatrix} E & 0 \\ 0 & E^* \end{bmatrix} \tag{4}$$

and $R = D^*D - \gamma^2 I$ and $S = DD^* - \gamma^2 I$. Then $\mathbf{i}\omega$ is an eigenvalue of matrix pencil $(\mathcal{M}_\gamma, \mathcal{N})$ if and only if γ is a singular value of $G(\mathbf{i}\omega)$.

Theorem 2.1 immediately leads to an algorithm for computing the $\mathcal{H}_\infty$ norm based on computing the imaginary eigenvalues, if any, of the associated matrix pencil (4). For brevity in this section, we assume that $\max \|G(\mathbf{i}\omega)\|_2$ is not attained at $\omega = \infty$, in which case the $\mathcal{H}_\infty$ norm would be $\|D\|_2$. Evaluating the norm of the transfer function for any finite frequency along the imaginary axis immediately gives a lower bound to the $\mathcal{H}_\infty$ norm while an upper bound can be obtained by successively increasing γ until the matrix pencil given by (4) no longer has any purely imaginary eigenvalues. Then, it is straightforward to compute the $\mathcal{H}_\infty$ norm using bisection, as first proposed in [12], which as the authors themselves noted, was inspired by the breakthrough result of [15] for computing the *distance to instability* (i.e., the reciprocal of the $\mathcal{H}_\infty$ norm for the special case of $B = C = E = I$ and $D = 0$).

As Theorem 2.1 provides a way to calculate all the frequencies where $\|G(\mathbf{i}\omega)\|_2 = \gamma$, it was shortly thereafter proposed in [11, 13] that instead of computing an upper bound and then using bisection, the initial lower bound could be successively increased in a monotonic fashion to the value of the $\mathcal{H}_\infty$ norm. For convenience, it will be helpful to establish the following notation for the transfer function and its largest singular value, both as parameters of frequency ω:

$$G_{\mathrm{c}}(\omega) := G(\mathbf{i}\omega), \tag{5}$$

$$g_{\mathrm{c}}(\omega) := \|G(\mathbf{i}\omega)\|_2 = \|G_{\mathrm{c}}(\omega)\|_2. \tag{6}$$

Let $\{\omega_1, \ldots, \omega_l\}$ be the set of imaginary parts of the purely imaginary eigenvalues of (4) for the initial value γ, sorted in increasing order. Considering the intervals $I_k = [\omega_k, \omega_{k+1}]$, [13] proposed increasing γ via

$$\gamma_{\mathrm{mp}} = \max g_{\mathrm{c}}(\hat{\omega}_k), \qquad \text{where} \qquad \hat{\omega}_k \text{ are the midpoints of the intervals } I_k. \tag{7}$$

Simultaneously and independently, a similar algorithm was proposed by [11], with the additional results that (a) it was possible to calculate which intervals I_k satisfied $g_c(\omega) \geq \gamma$ for all $\omega \in I_k$, thus reducing the number of evaluations of $g_c(\omega)$ needed at every iteration, and (b) this midpoint scheme actually had a local quadratic rate of convergence, greatly improving upon the linear rate of convergence of the earlier, bisection-based method. This midpoint-based method, which we refer to as the BBBS algorithm for its authors, Boyd, Balakrishnan, Bruinsma, and Steinbuch, is now considered the standard algorithm for computing the $\mathcal{H}_\infty$ norm and it is the algorithm implemented in the MATLAB Robust Control Toolbox, e.g., routine `hinfnorm`. Algorithm 1 provides a high-level pseudocode description for the standard BBBS algorithm, while Figure 1(a) provides a corresponding pictorial description of how the method works.

Algorithm 1 The standard BBBS algorithm.

Input: $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $C \in \mathbb{C}^{p\times n}$, $D \in \mathbb{C}^{p\times m}$, $E \in \mathbb{C}^{n\times n}$, and $\omega_0 \in \mathbb{R}$.
Output: $\gamma = \|G\|_{\mathcal{H}_\infty}$ and ω such that $\gamma = g_c(\omega)$.

1: $\gamma = g_c(\omega_0)$
2: **while** not converged **do**
3: *// Compute the intervals that lie under $g_c(\omega)$ using eigenvalues of the pencil:*
4: Compute $\Lambda_I = \{\operatorname{Im}\lambda : \lambda \in \Lambda(\mathcal{M}_\gamma, \mathcal{N}) \text{ and } \operatorname{Re}\lambda = 0\}$.
5: Index and sort $\Lambda_I = \{\omega_1, \ldots, \omega_l\}$ s.t. $\omega_j \leq \omega_{j+1}$.
6: Form all intervals $I_k = [\omega_k, \omega_{k+1}]$ s.t. each interval at height γ is below $g_c(\omega)$.
7: *// Compute candidate frequencies of the level-set intervals I_k:*
8: Compute midpoints $\hat{\omega}_k = 0.5(\omega_k + \omega_{k+1})$ for each interval I_k.
9: *// Update to the highest gain evaluated at these candidate frequencies:*
10: $\omega = \arg\max g_c(\hat{\omega}_k)$.
11: $\gamma = g_c(\omega)$.
12: **end while**

NOTE. *The quartically converging variant proposed by* [17] *replaces the midpoints of I_k with the maximizing frequencies of Hermite cubic interpolants, which are uniquely determined by interpolating the values of $g_c(\omega)$ and $g_c'(\omega)$ at both endpoints of each interval I_k.*

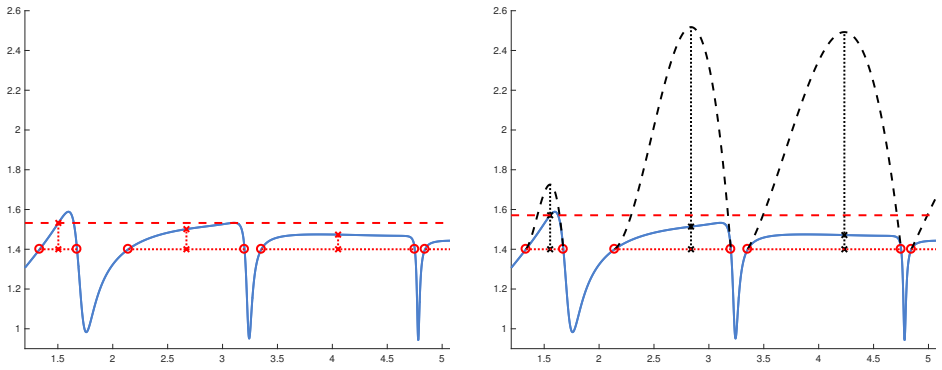
In [17], a refinement to the BBBS algorithm was proposed which increased its local quadratic rate of convergence to quartic. This was done by evaluating $g_c(\omega)$ at the maximizing frequencies of the unique cubic Hermite interpolants for each level-set interval, instead of at the midpoints. That is, for each interval $I_k = [\omega_k, \omega_{k+1}]$, the unique interpolant $c_k(\omega) = c_3\omega^3 + c_2\omega^2 + c_1\omega + c_0$ is constructed so that

$$\begin{aligned} c_k(\omega_k) &= g_c(\omega_k) \\ c_k(\omega_{k+1}) &= g_c(\omega_{k+1}) \end{aligned} \quad \text{and} \quad \begin{aligned} c_k'(\omega_k) &= g_c'(\omega_k) \\ c_k'(\omega_{k+1}) &= g_c'(\omega_{k+1}). \end{aligned} \tag{8}$$

Then, γ is updated via

$$\gamma_{\text{cubic}} = \max g_c(\hat{\omega}_k), \qquad \text{where} \qquad \hat{\omega}_k = \arg\max_{\omega\in I_k} c_k(\omega), \tag{9}$$

that is, $\hat{\omega}_k$ is now the maximizing value of interpolant $c_k(\omega)$ on its interval I_k, which of course can be cheaply and explicitly computed. In [17], the single numerical example



(a) The standard BBBS method (b) With cubic interpolation

FIG. 1. *The blue curves show the value of* $g_c(\omega) = \|G(\mathbf{i}\omega)\|_2$, *while the red circles mark the frequencies* $\{\omega_1, \ldots, \omega_l\}$, *where* $g_c(\omega) = \gamma = 1.4$, *computed by taking the imaginary parts of the purely imaginary eigenvalues of* (4) *on a sample problem. The left plot shows the midpoint scheme of the standard BBBS algorithm to obtain an increased value for* γ, *depicted by the dashed horizontal line. The right plot shows the cubic interpolation refinement of* [17] *for the same problem and initial value of* γ. *The dashed black curves depict the cubic Hermite interpolants for each interval, while the dotted vertical lines show their respective maximizing frequencies. As can be seen, the cubic interpolation scheme results in a larger increase in* γ *(again represented by the dashed horizontal line) compared to the standard BBBS method on this iterate for this problem.*

shows the concrete benefit of this interpolation scheme, where the standard BBBS algorithm required six eigenvalue decompositions of (4) to converge, while their new method required only four. As only the selection of the ω_k values is different, the pseudocode for this improved version of the BBBS algorithm remains largely the same, as mentioned in the note of Algorithm 1. Figure 1(b) provides a corresponding pictorial description of the cubic-interpolant-based refinement.

As it turns out, computing the derivatives in (8) for the Hermite interpolations of each interval can also be done with little extra work. Let $u(\omega)$ and $v(\omega)$ be the associated left and right singular vectors corresponding to $g_c(\omega)$, recalling that $g_c(\omega)$ is the largest singular value of $G_c(\omega)$, and assume that $g_c(\hat{\omega})$ is a simple singular value for some value $\hat{\omega} \in \mathbb{R}$. By standard perturbation theory (exploiting the equivalence of singular values of a matrix A and eigenvalues of $\left[\begin{smallmatrix} 0 & A \\ A^* & 0 \end{smallmatrix}\right]$ and applying [26, Theorem 5]), it then follows that

$$g_c'(\omega)\Big|_{\omega=\hat{\omega}} = \mathrm{Re}\,(u(\hat{\omega})^* G_c'(\hat{\omega}) v(\hat{\omega})), \tag{10}$$

where, by standard matrix differentiation rules with respect to parameter ω,

$$G_c'(\omega) = -\mathbf{i}C\,(\mathbf{i}\omega E - A)^{-1}\,E\,(\mathbf{i}\omega E - A)^{-1}\,B. \tag{11}$$

As shown in [32], it is actually fairly cheap to compute (10) if the eigenvectors corresponding to the purely imaginary eigenvalues of (4) have also been computed, as there is a correspondence between these eigenvectors and the associated singular vectors for γ. For $\gamma = g_c(\hat{\omega})$, if $\left[\begin{smallmatrix} q \\ s \end{smallmatrix}\right]$ is an eigenvector of (4) for imaginary eigenvalue $\mathbf{i}\hat{\omega}$, then the equivalences

$$q = (\mathbf{i}\hat{\omega}E - A)^{-1}\,Bv(\hat{\omega}) \quad \text{and} \quad s = (\mathbf{i}\hat{\omega}E - A)^{-*}\,C^*u(\hat{\omega}) \tag{12}$$

both hold, where $u(\hat{\omega})$ and $v(\hat{\omega})$ are left and right singular vectors associated with singular value $g_c(\hat{\omega})$. (To see why these equivalences hold, we refer the reader to the proof of Theorem 5.1 for the analogous discrete-time result.) Thus, (10) may be rewritten as follows:

$$g_c'(\omega)\Big|_{\omega=\hat{\omega}} = \operatorname{Re}\left(u(\hat{\omega})^* G_c'(\hat{\omega}) v(\hat{\omega})\right) \tag{13}$$

$$= -\operatorname{Re}\left(u(\hat{\omega})^* \mathbf{i} C \left(\mathbf{i}\hat{\omega}E - A\right)^{-1} E \left(\mathbf{i}\hat{\omega}E - A\right)^{-1} B v(\hat{\omega})\right) \tag{14}$$

$$= -\operatorname{Re}\left(\mathbf{i} s^* E q\right), \tag{15}$$

and it is thus clear that (10) is cheaply computable for all the endpoints of the intervals I_k, provided that the eigenvalue decomposition of (4) has already been computed.

3. The computational costs involved in the BBBS algorithm. The main drawback of the BBBS algorithm is its algorithmic complexity, which is $\mathcal{O}(n^3)$ work per iteration. This not only limits the tractability of computing the $\mathcal{H}_\infty$ norm to rather low-dimensional (in n) systems but can also make computing the $\mathcal{H}_\infty$ norm to full precision an expensive proposition for even moderately sized systems. In fact, the default tolerance for `hinfnorm` in MATLAB is set quite large, 0.01, presumably to keep its runtime as fast as possible, at the expense of sacrificing accuracy. In Table 1, we report the relative error of computing the $\mathcal{H}_\infty$ norm when using `hinfnorm`'s default tolerance of 0.01 compared to 10^{-14}, along with the respective runtimes for several test problems, observing that computing the $\mathcal{H}_\infty$ norm to near full precision can often take between two and four times longer, approximately. While computing only a handful of the most significant digits of the $\mathcal{H}_\infty$ norm may be sufficient for some applications, this is certainly not true in general. Indeed, the source code for HIFOO [14], which

TABLE 1

For various problems, the relative error of computing the $\mathcal{H}_\infty$ norm using `hinfnorm` with its quite loose default tolerance is shown. The first eight are continuous time problems, while the last six, ending in `d`, are discrete time. As can be seen, using `hinfnorm` to compute the $\mathcal{H}_\infty$ norm to near machine accuracy can often take between two and four times longer, and this penalty is not necessarily related to dimension: for example, the running times are increased by factors of 3.02 and 3.51 for `CM3` and `FOM`, respectively, even though `FOM` is nearly 10 times larger in dimension.

h = `hinfnorm(·,1e-14)` versus $\hat{h}$ = `hinfnorm(·,0.01)`							
	Dimensions				Relative error	Wall-clock time (sec.)	
Problem	n	m	p	$E = I$	$\frac{\hat{h}-h}{h}$	`tol=1e-14`	`tol=0.01`
`CSE2`	63	1	32	Y	-2.47×10^{-4}	0.137	0.022
`CM3`	123	1	3	Y	-2.75×10^{-3}	0.148	0.049
`CM4`	243	1	3	Y	-4.70×10^{-3}	1.645	0.695
`ISS`	270	3	3	Y	-1.04×10^{-6}	0.765	0.391
`CBM`	351	1	2	Y	-4.20×10^{-5}	3.165	1.532
`randn 1`	500	300	300	Y	0	21.084	30.049
`randn 2`	600	150	150	N	-6.10×10^{-8}	31.728	16.199
`FOM`	1006	1	1	Y	-1.83×10^{-5}	128.397	36.529
`LAHd`	58	1	3	Y	-7.36×10^{-3}	0.031	0.015
`BDT2d`	92	2	4	Y	-7.67×10^{-4}	0.070	0.031
`EB6d`	170	2	2	Y	-5.47×10^{-7}	0.192	0.122
`ISS1d`	280	1	273	Y	-1.53×10^{-3}	16.495	3.930
`CBMd`	358	1	2	Y	-2.45×10^{-6}	1.411	0.773
`CM5d`	490	1	3	Y	-7.38×10^{-3}	10.802	2.966

designs $\mathcal{H}_\infty$-norm fixed-order optimizing controllers for a given open-loop system via nonsmooth optimization, specifically contains the comment regarding `hinfnorm`: "default is .01, which is too crude." In HIFOO, the $\mathcal{H}_\infty$ norm is minimized by updating the controller variables at every iteration but the optimization method assumes that the objective function is continuous; if the $\mathcal{H}_\infty$ norm is not calculated sufficiently accurately, then it may appear to be discontinuous, which can cause the underlying optimization method to break down. Thus there is motivation to not only improve the overall runtime of computing the $\mathcal{H}_\infty$ norm for large tolerances, but also to make the computation as fast as possible when computing the $\mathcal{H}_\infty$ norm to full precision.

The dominant cost of the BBBS algorithm is computing the eigenvalues of (4) at every iteration. Even though the method converges quadratically, and quartically when using the cubic interpolation refinement, the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ will still generally be computed for multiple values of γ before convergence, for either variant of the algorithm. Furthermore, pencil $(\mathcal{M}_\gamma, \mathcal{N})$ is $2n \times 2n$, meaning that the $\mathcal{O}(n^3)$ work per iteration also contains a significantly larger constant factor; computing the eigenvalues of a $2n \times 2n$ problem typically takes at least eight times longer than an $n \times n$ one. If cubic interpolation is used, computing the derivatives (10) via the eigenvectors of $(\mathcal{M}_\gamma, \mathcal{N})$, as proposed by [17] using the equivalences in (12) and (15), can sometimes be quite expensive as well. If on a particular iteration, the number of purely imaginary eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ is close to n, say, $\hat{n}$, then assuming 64-bit computation, an additional $4\hat{n}^2$ doubles of memory would be required to store these eigenvectors.[2] Finally, computing the purely imaginary eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ using the regular QZ algorithm can be ill advised; in practice, rounding error in the real parts of the eigenvalues can make it difficult to detect which of the computed eigenvalues are supposed to be the purely imaginary ones and which are merely just close to the imaginary axis. Indeed, purely imaginary eigenvalues can easily be perturbed off of the imaginary axis when using standard QZ; [8, Figure 4] illustrates this issue particularly well. Failure to properly identify the purely imaginary eigenvalues can cause the BBBS algorithm to return incorrect results. As such, it is instead recommended [7, section II.D] to use the specialized Hamiltonian-structure-preserving eigensolvers of [4, 8] to avoid this problem. However, doing so can be even more expensive as it requires computing the eigenvalues of a related matrix pencil that is even larger: $(2n + m + p) \times (2n + m + p)$.

On the other hand, computing (6), the norm of the transfer function, is typically rather inexpensive, at least relative to computing the imaginary eigenvalues of the matrix pencil (4); Table 2 presents data on how much faster computing the singular value decomposition of $G(\mathbf{i}\omega)$ can be compared to computing the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ (using regular QZ), using randomly generated systems composed of dense matrices of various dimensions. In the first row of Table 2, we see that computing the eigenvalues of (4) for tiny systems ($n = m = p = 20$) can take up to two and a half times longer than computing the SVD of $G(\mathbf{i}\omega)$ on modern hardware and this disparity quickly grows larger as the dimensions are all increased (up to 36.8 faster for $n = m = p = 400$). Furthermore, for moderately sized systems where $m, p \ll n$ (the typical case in practice), the performance gap dramatically widens to up to 119 times faster to compute the SVD of $G(\mathbf{i}\omega)$ versus the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ (the last row of Table 2). Of course, this disparity in runtime speeds is not surprising. Com-

[2] Although computing eigenvectors with `eig` in MATLAB is currently an all or none affair, LAPACK does provide the user the option to only compute certain eigenvectors, so that all $2n$ eigenvectors would not always need to be computed.

puting the eigenvalues of (4) involves working with a $2n \times 2n$ (or larger when using structure-preserving eigensolvers) matrix pencil, while the main costs to evaluate the norm of the transfer function at a particular frequency involve first solving a linear system of dimension n to compute either the $(\mathbf{i}\omega E - A)^{-1}B$ or the $C(\mathbf{i}\omega E - A)^{-1}$ term in $G(\mathbf{i}\omega)$ and then computing the maximum singular value of $G(\mathbf{i}\omega)$, which is $p \times m$. If $\max(m, p)$ is small, the cost to compute the largest singular value is negligible, and even if $\max(m, p)$ is not small, the largest singular value can still typically be computed easily and efficiently using sparse methods. Solving the n-dimensional linear system is typically going to be much cheaper than computing the eigenvalues of the $2n \times 2n$ pencil, and more so if A and E are not dense and $(\mathbf{i}\omega E - A)$ permits a fast (sparse) LU decomposition.

4. The improved algorithm. Recall that computing the $\mathcal{H}_\infty$ norm is done by maximizing $g_c(\omega)$ over $\omega \in \mathbb{R}$ but that the BBBS algorithm (and the cubic interpolation refinement) actually converges to a global maximum of $g_c(\omega)$ by iteratively computing the eigenvalues of the large matrix pencil $(\mathcal{M}_\gamma, \mathcal{N})$ for successively larger values of γ. However, we could alternatively consider a more direct approach of finding maximizers of $g_c(\omega)$, which, as discussed above, is a much cheaper function to evaluate numerically. Computing such maximizers could allow larger increases in γ to be obtained on each iteration, compared to just evaluating $g_c(\omega)$ at the midpoints or maximizers of the cubic interpolants. This in turn should reduce the number of times that the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ must be computed and thus speed up the overall running time of the algorithm; given the performance data in Table 2, the additional cost of any evaluations of $g_c(\omega)$ needed to find maximizers seems like it should be more than offset by fewer eigenvalue decompositions of $(\mathcal{M}_\gamma, \mathcal{N})$. Of course, computing the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ at each iteration cannot be eliminated completely, as it is still necessary for asserting whether any of the maximizers was a global maximizer (in which case, the $\mathcal{H}_\infty$ norm has been computed), or if not, to provide the remaining level set intervals where a global maximizer lies so the computation can continue.

As alluded to in the introduction, for the special case of the distance to instability, a similar cost-balancing strategy has been considered before in [22] but the authors themselves noted that the inverse iteration scheme they employed to find locally optimal solutions could sometimes have very slow convergence and expressed concern that other optimization methods could suffer similarly. Of course, in this paper we

TABLE 2

For each set of dimensions (given in the leftmost three columns), five different systems were randomly generated and the running times to compute $\|G(\mathbf{i}\omega)\|_2$ and `eig`$(\mathcal{M}_\gamma, \mathcal{N})$ were recorded to form the ratios of these five pairs of values. Ratios greater than one indicate that it is faster to compute $\|G(\mathbf{i}\omega)\|_2$ than `eig`$(\mathcal{M}_\gamma, \mathcal{N})$, and by how much, while ratios less than one indicate the opposite. The rightmost two columns of the table give the smallest and largest of the five ratios observed per set of dimensions.

Computing $\|G(\mathbf{i}\omega)\|_2$ versus `eig`$(\mathcal{M}_\gamma, \mathcal{N})$				
			Times faster	
n	m	p	min	max
20	20	20	0.71	2.47
100	100	100	6.34	10.2
400	400	400	19.2	36.8
400	10	10	78.5	119.0

are considering the more general case of computing the $\mathcal{H}_\infty$ norm and, as we will observe in our later experimental evaluation, the first- and second-order optimization techniques we now propose do in fact seem to work well in practice.

Though $g_c(\omega)$ is typically nonconvex, standard optimization methods should generally still be able to find local maximizers, if not always global maximizers, provided that $g_c(\omega)$ is sufficiently smooth. Since $g_c(c)$ is the maximum singular value of $G(\mathbf{i}\omega)$, it is locally Lipschitz (e.g., [18, Corollary 8.6.2]). Furthermore, in proving the quadratic convergence of the midpoint-based BBBS algorithm, it was shown that at local maximizers, the second derivative of $g_c(\omega)$ not only exists but is even locally Lipschitz [11, Theorem 2.3]. The direct consequence is that Newton's method for optimization can be expected to converge quadratically when it is used to find a local maximizer of $g_c(\omega)$. Since there is only one optimization variable, namely, ω, there is also the benefit that we need only work with first and second derivatives, instead of gradients and Hessians, respectively. Furthermore, if $g_c''(\omega)$ is expensive to compute, one can instead resort to the secant method (which is a quasi-Newton method in one variable) and still obtain superlinear convergence. Given the large disparity in costs to compute the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ and $g_c(\omega)$, it seems likely that even just superlinear convergence could still be sufficient to significantly accelerate the computation of the $\mathcal{H}_\infty$ norm. Of course, when $g_c''(\omega)$ is relatively cheap to compute, additional acceleration is likely to be obtained when using Newton's method. Note that many alternative optimization strategies could also be employed here, potentially with additional efficiencies. But, for the sake of simplicity, we will just restrict the discussion in this paper to the secant method and Newton's method, particularly since conceptually there is no difference.

Since $g_c(\omega)$ will now need to be evaluated at any point requested by an optimization method, we will need to compute its first and possibly second derivatives directly; recall that using the eigenvectors of the purely imaginary eigenvalues of (4) with the equivalences in (12) and (15) only allows us to obtain the first derivatives at the endpoints of the level-set intervals. However, as long as we also compute the associated left and right singular vectors $u(\omega)$ and $v(\omega)$ when computing $g_c(\omega)$, the value of the first derivative $g_c'(\omega)$ can be computed via the direct formulation given in (14) and without much additional cost over computing $g_c(\omega)$ itself. For each frequency ω of interest, an LU factorization of $(\mathbf{i}\omega E - A)$ can be done once and reused to solve the linear systems due to the presence of $(\mathbf{i}\omega E - A)^{-1}$, which appears once in $g_c(\omega)$ and twice in $g_c'(\omega)$.

To compute $g_c''(\omega)$, we will need the following result for second derivatives of eigenvalues, which can be found in various forms in [26], [29], and [25].

THEOREM 4.1. *For $t \in \mathbb{R}$, let $H(t)$ be a twice-differentiable $n \times n$ Hermitian matrix family with distinct eigenvalues at $t = 0$ with (λ_k, x_k) denoting the kth such eigenpair and where each eigenvector x_k has unit norm and the eigenvalues are ordered $\lambda_1 > \cdots > \lambda_n$. Then,*

$$\lambda_1''(t)\Big|_{t=0} = x_1^* H''(0) x_1 + 2\sum_{k=2}^{n} \frac{|x_1^* H'(0) x_k|^2}{\lambda_1 - \lambda_k}.$$

Since $g_c(\omega)$ is the largest singular value of $\|G(\mathbf{i}\omega)\|_2$, it is also the largest eigenvalue of the matrix

$$H(\omega) = \begin{bmatrix} 0 & G_c(\omega) \\ G_c(\omega)^* & 0 \end{bmatrix}, \tag{16}$$

which has first and second derivatives

$$H'(\omega) = \begin{bmatrix} 0 & G_{\mathrm{c}}'(\omega) \\ G_{\mathrm{c}}'(\omega)^* & 0 \end{bmatrix} \quad \text{and} \quad H''(\omega) = \begin{bmatrix} 0 & G_{\mathrm{c}}''(\omega) \\ G_{\mathrm{c}}''(\omega)^* & 0 \end{bmatrix}. \tag{17}$$

The formula for $G_{\mathrm{c}}'(\omega)$ is given by (14), while the corresponding second derivative is obtained by straightforward application of matrix differentiation rules:

$$G_{\mathrm{c}}''(\omega) = -2C(\mathbf{i}\omega E - A)^{-1}E(\mathbf{i}\omega E - A)^{-1}E(\mathbf{i}\omega E - A)^{-1}B. \tag{18}$$

Furthermore, the eigenvalues and eigenvectors of (16) needed to apply Theorem 4.1 are essentially directly available from just the full SVD of $G_{\mathrm{c}}(\omega)$. Let σ_k be the kth singular value of $G_{\mathrm{c}}(\omega)$, along with associated right and left singular vectors u_k and v_k, respectively. Then $\pm\sigma_k$ is an eigenvalue of (16) with eigenvector $\left[\begin{smallmatrix} u_k \\ v_k \end{smallmatrix}\right]$ for σ_k and eigenvector $\left[\begin{smallmatrix} u_k \\ -v_k \end{smallmatrix}\right]$ for $-\sigma_k$. When $\sigma_k = 0$, the corresponding eigenvector is either $\left[\begin{smallmatrix} u_k \\ \mathbf{0} \end{smallmatrix}\right]$ if $p > m$ or $\left[\begin{smallmatrix} \mathbf{0} \\ v_k \end{smallmatrix}\right]$ if $p < m$, where $\mathbf{0}$ denotes a column of m or p zeros, respectively. Given the full SVD of $G_{\mathrm{c}}(\omega)$, computing $g_{\mathrm{c}}''(\omega)$ can also be done with relatively little additional cost. The stored LU factorization of $(\mathbf{i}\omega E - A)$ used to obtain $G_{\mathrm{c}}(\omega)$ can again be reused to quickly compute the $G_{\mathrm{c}}'(\omega)$ and $G_{\mathrm{c}}''(\omega)$ terms in (17). If obtaining the full SVD is particularly expensive, i.e., for systems with many inputs/outputs, as mentioned above, sparse methods can still be used to efficiently obtain the largest singular value and its associated right/left singular vectors, in order to at least calculate $g_{\mathrm{c}}'(\omega)$, if not $g_{\mathrm{c}}''(\omega)$ as well.

Remark 4.2. On a more theoretical point, by invoking Theorem 4.1 to compute $g_{\mathrm{c}}''(\omega)$, we are also assuming that the singular values of $G_{\mathrm{c}}(\omega)$ are unique as well. In practice, this will almost certainly hold numerically, and it would have to frequently fail to hold in order to adversely impact the convergence rate of Newton's method, which seems an exceptionally unlikely scenario. As such, we feel that this additional assumption is not of practical concern.

Thus, our new proposed improvement to the BBBS algorithm is to not settle for the increase in γ provided by the standard midpoint or cubic interpolation schemes but to increase γ *as far as possible* on every iteration using standard optimization techniques applied to $g_{\mathrm{c}}(\omega)$. Assume that γ is still less than the value of the $\mathcal{H}_\infty$ norm and let

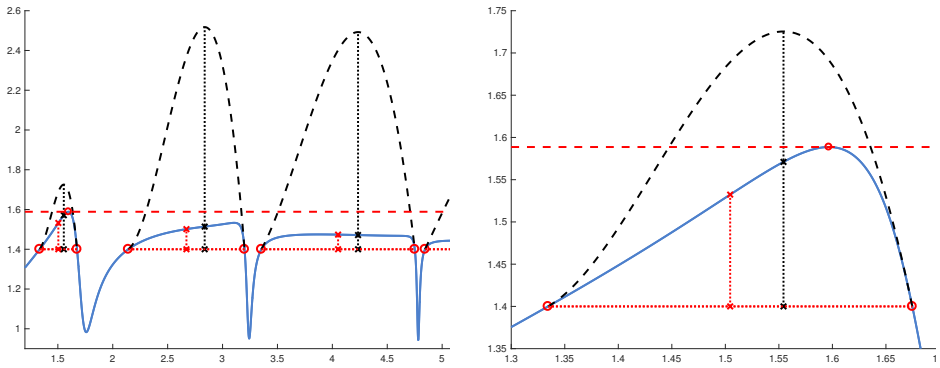
$$\hat{\omega}_j = \arg\max g_{\mathrm{c}}(\hat{\omega}_k),$$

where the finite set of $\hat{\omega}_k$ values are the midpoints of the level-set intervals I_k or the maximizers of the cubic interpolants on these intervals, respectively defined in (7) or (9). Thus the solution $\hat{\omega}_j \in I_j$ is the frequency that provides the updated value γ_{mp} or γ_{cubic} in the standard algorithms. Now consider applying either Newton's method or the secant method (the choice of which one will be more efficient can be made more or less automatically depending on how m, p compares to n) to the following optimization problem with a simple box constraint:

$$\max_{\omega \in I_j} g_{\mathrm{c}}(\omega). \tag{19}$$

If the optimization method is initialized at $\hat{\omega}_j$, then even if ω_{opt}, a *computed* solution to (19), is actually just a *local* maximizer (a possibility since (19) could be nonconvex), it is still guaranteed that

$$g_{\mathrm{c}}(\omega_{\mathrm{opt}}) > \begin{cases} \gamma_{\mathrm{mp}}, & \text{initial point } \hat{\omega}_j \text{ is a midpoint of } I_j, \\ \gamma_{\mathrm{cubic}}, & \text{initial point } \hat{\omega}_j \text{ is a maximizer of interpolant } c_j(\omega) \end{cases}$$



(a) Optimization-based approach (b) Close up of leftmost peak in Figure 2(a)

FIG. 2. *For the same example as Figure* 1*, the larger increase in* γ *attained by optimizing* (19) *is shown by the red dashed line going through the red circle at the top of the leftmost peak of* $\|G(\mathbf{i}\omega)\|_2$*. By comparison, the BBBS midpoint (red dotted vertical lines and x's) and the cubic-interpolant-based schemes (black dotted vertical lines and x's) only provide suboptimal increases in* γ*.*

holds, provided that $\hat{\omega}_j$ does not happen to be a stationary point of $g_c(\omega)$. Furthermore, (19) can only have more than one maximizer when the current estimate γ of the $\mathcal{H}_\infty$ norm is so low that there are multiple peaks above level-set interval I_j. Consequently, as the algorithm converges, computed maximizers of (19) will be assured to be globally optimal over I_j and in the limit, over all frequencies along the entire imaginary axis. By setting tight tolerances for the optimization code, maximizers of (19) can also be computed to full precision with little to no penalty, due to the superlinear or quadratic rate of convergence we can expect from the secant method or Newton's method, respectively. If the computed optimizer of (19) also happens to be a global maximizer of $g_c(\omega)$, for all $\omega \in \mathbb{R}$, then the $\mathcal{H}_\infty$ norm has indeed been computed to full precision, but the algorithm must still verify this by computing the imaginary eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ just one more time. However, if a global optimizer has not yet been found, then the algorithm must compute the imaginary eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ at least two times more: one or more times as the algorithm increases γ to the globally optimal value, and then a final evaluation to verify that the computed value is indeed globally optimal. Figure 2 shows a pictorial comparison of optimizing $g_c(\omega)$ compared to the midpoint and cubic-interpolant-based updating methods.

In the above discussion, we have so far only considered applying optimization over the single level-set interval I_j but we certainly could attempt to solve (19) for other level-set intervals as well. Let $\phi = 1, 2, \ldots$ be the max number of level-set intervals to optimize over per iteration and q be the number of level-set intervals for the current value of γ. Compared to just optimizing over I_j, optimizing over all q of the level-set intervals could yield an even larger increase in the estimate γ but would be the most expensive option computationally. If we do not optimize over all the level-set intervals, i.e., $\phi < q$, there is the question of which intervals should be prioritized for optimization. In our experiments, we found that prioritizing by first evaluating $g_c(\omega)$ at all the $\hat{\omega}_k$ values and then choosing the intervals I_k to optimize where $g_c(\hat{\omega}_k)$ takes on the largest values seems to be a good strategy. However, with a serial MATLAB code, we have observed that just optimizing over the most promising interval, i.e., $\phi = 1$ so just I_j, is generally more efficient in terms of running time than trying to

optimize over more intervals. For discussion about optimizing over multiple intervals in parallel, see section 8.

Finally, if a global maximizer of $g_c(\omega)$ can potentially be found before ever computing eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ even once, then only one expensive eigenvalue decomposition $(\mathcal{M}_\gamma, \mathcal{N})$ will be incurred, just to verify that the initial maximizer is indeed a global one. Thus, we also propose initializing the algorithm at a maximizer of $g_c(\omega)$, obtained via applying standard optimization techniques to

$$\max g_c(\omega), \tag{20}$$

which is just (19) without the box constraint. In the best case, the computed maximizer will be a global one, but even a local maximizer will still provide a higher initial estimate of the $\mathcal{H}_\infty$ norm compared to initializing at a guess that may not even be locally optimal. Of course, finding maximizers of (20) by starting from multiple initial guesses can also be done in parallel; we again refer to section 8 for more details.

Algorithm 2 provides a high-level pseudocode description of our improved method. As a final step in the algorithm, it is necessary to check whether the value of the $\mathcal{H}_\infty$ norm is indeed attained at $\omega = \infty$. We check this case after the convergent phase has computed a global maximizer over the union of all intervals it has considered. The only possibility that the $\mathcal{H}_\infty$ norm may be attained at $\omega = \infty$ in Algorithm 2 is when the initial value of γ computed in line 5 is less than $\|D\|_2$. As the assumptions of Theorem 2.1 require that γ not be a singular value of D, it is not valid to use $\gamma = \|D\|_2$ in $(\mathcal{M}_\gamma, \mathcal{N})$ to check if this pencil has any imaginary eigenvalues. However, if the optimizer computed by the convergent phase of Algorithm 2 yields a γ value less than $\|D\|_2$, then it is clear that the optimizing frequency is at $\omega = \infty$.

5. Handling discrete-time systems. Now consider the discrete-time linear dynamical system

$$Ex_{k+1} = Ax_k + Bu_k, \tag{21a}$$

$$y_k = Cx_k + Du_k, \tag{21b}$$

where the matrices are defined as before in (1). In this case, the $\mathcal{H}_\infty$ norm is defined as

$$\|G\|_{\mathcal{H}_\infty} := \max_{\theta \in [0, 2\pi)} \|G(e^{\mathbf{i}\theta})\|_2, \tag{22}$$

again assuming that pencil (A, E) is at most index one. If all finite eigenvalues either are strictly inside the unit disk centered at the origin or are uncontrollable or unobservable, then (22) is finite and the $\mathcal{H}_\infty$ norm is attained at some $\theta \in [0, 2\pi)$. Otherwise, it is infinite.

We now show the analogous version of Theorem 2.1 for discrete-time systems. The $D = 0$ and $E = I$ case was considered in [24, section 3], while the more specific $B = C = E = I$ and $D = 0$ case was given in [15, Theorem 4].[3] These results relate singular values of the transfer function for discrete-time systems to eigenvalues with modulus one of associated matrix pencils. Although the following more general result is already known, the proof, to the best of our knowledge, is not in the literature so we include it in full here. The proof follows a similar argumentation as the proof of Theorem 2.1.

[3]Note that equation (10) in [15] has a typo: $A^H - e^{\mathbf{i}\theta} I$ in the lower left block of $K(\theta)$ should actually be $A^H - e^{-\mathbf{i}\theta} I$.

Algorithm 2 The improved algorithm using local optimization.

Input: Matrices $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $C \in \mathbb{C}^{p\times n}$, $D \in \mathbb{C}^{p\times m}$, and $E \in \mathbb{C}^{n\times n}$, initial frequency guesses $\{\omega_1, \ldots, \omega_q\} \in \mathbb{R}$ and ϕ, a positive integer indicating the number of intervals/frequencies to optimize per iteration.

Output: $\gamma = \|G\|_{\mathcal{H}_\infty}$ and ω such that $\gamma = g_{\mathrm{c}}(\omega)$.

// *Initialization:*
Compute $[\gamma_1, \ldots, \gamma_q] = [g_{\mathrm{c}}(\omega_1), \ldots, g_{\mathrm{c}}(\omega_q)]$.
Reorder $\{\omega_1, \ldots, \omega_q\}$ s.t. $\gamma_j \geq \gamma_{j+1}$.
Find maximal values $[\gamma_1, \ldots, \gamma_\phi]$ of (20), using initial points ω_j, $j = 1, \ldots, \phi$.
$\gamma = \max([\gamma_1, \ldots, \gamma_\phi])$.
$\omega = \omega_j$ for j s.t. $\gamma = \gamma_j$.
// *Convergent phase:*
while not converged **do**
  // *Compute the intervals that lie under $g_{\mathrm{c}}(\omega)$ using eigenvalues of the pencil:*
  Compute $\Lambda_{\mathrm{I}} = \{\operatorname{Im}\lambda : \lambda \in \Lambda(\mathcal{M}_\gamma, \mathcal{N}) \text{ and } \operatorname{Re}\lambda = 0\}$.
  Index and sort $\Lambda_{\mathrm{I}} = \{\omega_1, \ldots, \omega_l\}$ s.t. $\omega_j \leq \omega_{j+1}$.
  Form all intervals $I_k = [\omega_k, \omega_{k+1}]$ s.t. each interval at height γ is below $g_{\mathrm{c}}(\omega)$.
  // *Compute candidate frequencies of the level-set intervals I_k (q of them):*
  Compute all $\hat{\omega}_k$ using either (7) or (9).
  // *Run box-constrained optimization on the ϕ most promising frequencies:*
  Compute $[\gamma_1, \ldots, \gamma_q] = [g_{\mathrm{c}}(\hat{\omega}_1), \ldots, g_{\mathrm{c}}(\hat{\omega}_q)]$.
  Reorder $\{\hat{\omega}_1, \ldots, \hat{\omega}_q\}$ and intervals I_k s.t. $\gamma_j \geq \gamma_{j+1}$.
  Find maximal values $[\gamma_1, \ldots, \gamma_\phi]$ of (19) using initial points $\hat{\omega}_j$, $j = 1, \ldots, \phi$.
  // *Update to the highest gain computed:*
  $\gamma = \max([\gamma_1, \ldots, \gamma_\phi])$.
  $\omega = \omega_j$ for j s.t. $\gamma = \gamma_j$.
end while
// *Check that the maximizing frequency is not at infinity (continuous-time only)*
if $\gamma < \|D\|_2$ **then**
  $\gamma = \|D\|_2$.
  $\omega = \infty$.
end if

NOTE. *For details on how embarrassingly parallel processing can be used to further improve the algorithm, see section* 8.

THEOREM 5.1. *Let $\lambda E - A$ be regular with no finite eigenvalues on the unit circle, $\gamma > 0$ not a singular value of D, and $\theta \in [0, 2\pi)$. Consider the matrix pencil $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$, where*

$$\mathcal{S}_\gamma := \begin{bmatrix} A - BR^{-1}D^*C & -\gamma BR^{-1}B^* \\ 0 & E^* \end{bmatrix}, \qquad \mathcal{T}_\gamma := \begin{bmatrix} E & 0 \\ -\gamma C^* S^{-1} C & (A - BR^{-1}D^*C)^* \end{bmatrix}, \tag{23}$$

$R = D^*D - \gamma^2 I$, and $S = DD^* - \gamma^2 I$. Then $e^{\mathbf{i}\theta}$ is an eigenvalue of matrix pencil $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ if and only if γ is a singular value of $G(e^{\mathbf{i}\theta})$.

Proof. Let γ be a singular value of $G(e^{\mathbf{i}\theta})$ with left and right singular vectors u and v, that is, so that $G(e^{\mathbf{i}\theta})v = \gamma u$ and $G(e^{\mathbf{i}\theta})^* u = \gamma v$. Using the expanded versions

of these two equivalences

$$\left(C\left(e^{\mathbf{i}\theta}E-A\right)^{-1}B+D\right)v=\gamma u \quad \text{and} \quad \left(C\left(e^{\mathbf{i}\theta}E-A\right)^{-1}B+D\right)^{*}u=\gamma v, \tag{24}$$

we define

$$q=\left(e^{\mathbf{i}\theta}E-A\right)^{-1}Bv \quad \text{and} \quad s=\left(e^{-\mathbf{i}\theta}E^{*}-A^{*}\right)^{-1}C^{*}u. \tag{25}$$

Rewriting (24) using (25) yields the following matrix equation:

$$\begin{bmatrix}C & 0\\ 0 & B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}-D & \gamma I\\ \gamma I & -D^{*}\end{bmatrix}\begin{bmatrix}v\\ u\end{bmatrix} \implies \begin{bmatrix}v\\ u\end{bmatrix}=\begin{bmatrix}-D & \gamma I\\ \gamma I & -D^{*}\end{bmatrix}^{-1}\begin{bmatrix}C & 0\\ 0 & B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}, \tag{26}$$

where

$$\begin{bmatrix}-D & \gamma I\\ \gamma I & -D^{*}\end{bmatrix}^{-1}=\begin{bmatrix}-R^{-1}D^{*} & -\gamma R^{-1}\\ -\gamma S^{-1} & -DR^{-1}\end{bmatrix} \quad \text{and} \quad \begin{bmatrix}q\\ s\end{bmatrix}\neq 0. \tag{27}$$

Rewriting (25) as

$$\left(\begin{bmatrix}e^{\mathbf{i}\theta}E & 0\\ 0 & e^{-\mathbf{i}\theta}E^{*}\end{bmatrix}-\begin{bmatrix}A & 0\\ 0 & A^{*}\end{bmatrix}\right)\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}B & 0\\ 0 & C^{*}\end{bmatrix}\begin{bmatrix}v\\ u\end{bmatrix}, \tag{28}$$

and then substituting in (26) for the rightmost term of (28) yields

$$\left(\begin{bmatrix}e^{\mathbf{i}\theta}E & 0\\ 0 & e^{-\mathbf{i}\theta}E^{*}\end{bmatrix}-\begin{bmatrix}A & 0\\ 0 & A^{*}\end{bmatrix}\right)\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}B & 0\\ 0 & C^{*}\end{bmatrix}\begin{bmatrix}-D & \gamma I\\ \gamma I & -D^{*}\end{bmatrix}^{-1}\begin{bmatrix}C & 0\\ 0 & B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}. \tag{29}$$

Multiplying the above on the left by

$$\begin{bmatrix}I & 0\\ 0 & -e^{\mathbf{i}\theta}I\end{bmatrix}$$

and then rearranging terms, we have

$$e^{\mathbf{i}\theta}\begin{bmatrix}E & 0\\ 0 & A^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}A & 0\\ 0 & E^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}+\begin{bmatrix}B & 0\\ 0 & -e^{\mathbf{i}\theta}C^{*}\end{bmatrix}\begin{bmatrix}-D & \gamma I\\ \gamma I & -D^{*}\end{bmatrix}^{-1}\begin{bmatrix}C & 0\\ 0 & B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}.$$

Substituting the inverse in (27) for its explicit form and multiplying terms yields

$$e^{\mathbf{i}\theta}\begin{bmatrix}E & 0\\ 0 & A^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}A & 0\\ 0 & E^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}+\begin{bmatrix}B & 0\\ 0 & -e^{\mathbf{i}\theta}C^{*}\end{bmatrix}\begin{bmatrix}-R^{-1}D^{*}C & -\gamma R^{-1}B^{*}\\ -\gamma S^{-1}C & -DR^{-1}B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}.$$

Finally, multiplying terms further, separating out the $-e^{\mathbf{i}\theta}$ terms to bring them over to the left-hand side, and then recombining, we have that

$$e^{\mathbf{i}\theta}\begin{bmatrix}E & 0\\ -\gamma C^{*}S^{-1}C & A^{*}-C^{*}DR^{-1}B^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}=\begin{bmatrix}A-BR^{-1}D^{*}C & -\gamma BR^{-1}B^{*}\\ 0 & E^{*}\end{bmatrix}\begin{bmatrix}q\\ s\end{bmatrix}.$$

It is now clear that $e^{\mathbf{i}\theta}$ is an eigenvalue of pencil $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$.

Now suppose that $e^{\mathbf{i}\theta}$ is an eigenvalue of pencil $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ with eigenvector given by q and s as above. Then it follows that (29) holds, which can be rewritten as (28) by defining u and v using the right-hand-side equation of (26), noting that neither can be identically zero. It is then clear that the two equivalences in (25) both hold. Finally, substituting (25) into the left-hand-side equation of (26), it is clear that γ is a singular value of $G(e^{\mathbf{i}\theta})$, with left and right singular vectors u and v. ∎

Adapting Algorithm 2 to the discrete-time case is straightforward. First, all instances of $g_{\mathrm{c}}(\omega)$ must be replaced with

$$g_{\mathrm{d}}(\theta) \coloneqq \|G(e^{\mathbf{i}\theta})\|_2.$$

To calculate its first and second derivatives, we will need the first and second derivatives of $G_{\mathrm{d}}(\theta) \coloneqq G(e^{\mathbf{i}\theta})$, and for notational brevity, it will be convenient to define $Z(\theta) \coloneqq (e^{\mathbf{i}\theta}E - A)$. Then

$$G'_{\mathrm{d}}(\theta) = -\mathbf{i}e^{\mathbf{i}\theta}CZ(\theta)^{-1}EZ(\theta)^{-1}B \tag{30}$$

and

$$G''_{\mathrm{d}}(\theta) = e^{\mathbf{i}\theta}CZ(\theta)^{-1}EZ(\theta)^{-1}B - 2e^{2\mathbf{i}\theta}CZ(\theta)^{-1}EZ(\theta)^{-1}EZ(\theta)^{-1}B. \tag{31}$$

The first derivative of $g_{\mathrm{d}}(\theta)$ can thus be calculated using (13), where ω, $g_{\mathrm{c}}(\omega)$, and $G'_{\mathrm{c}}(\omega)$ are replaced by θ, $g_{\mathrm{d}}(\theta)$, and $G'_{\mathrm{d}}(\theta)$ using (30). The second derivative of $g_{\mathrm{d}}(\theta)$ can be calculated using Theorem 4.1 using (30) and (31) to define $H(\theta)$, the analogue of (16). Line 10 must be changed to instead compute the eigenvalues of unit modulus of (23). Line 11 must instead index and sort the angles $\{\theta_1, \ldots, \theta_l\}$ of these unit modulus eigenvalues in ascending order. Due to the periodic nature of (22), line 12 must additionally consider the "wrap-around" interval $[\theta_l, \theta_0 + 2\pi]$.

6. Numerical experiments. We implemented Algorithm 2 in MATLAB, for both continuous-time and discrete-time cases. Since we can only get timing information from `hinfnorm` and we wished to verify that our new method does indeed reduce the number of times the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ and $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ are computed, we also designed our code so that it can run just using the standard BBBS algorithm or the cubic-interpolant scheme. For our new optimization-based approach, we used `fmincon` for both the unconstrained optimization calls needed for the initialization phase and for the box-constrained optimization calls needed in the convergent phase; `fmincon`'s optimality and constraint tolerances were set to 10^{-14} in order to find maximizers to near machine precision. Our code supports starting the latter optimization calls from either the midpoints of the BBBS algorithm (7) or the maximizing frequencies calculated from the cubic-interpolant method (9). Furthermore, the optimizations may be done using either the secant method (first-order information only) or with Newton's method using second derivatives, thus leading to four variants of our proposed method to test. Our code has a user-settable parameter that determines when m, p should be considered too large relative to n, and thus when it is likely that using the secant method will actually be faster than Newton's method, due to the additional expense of computing the second derivative of the norm of the transfer function.

For initial frequency guesses, our code simply tests zero and the imaginary part of the rightmost eigenvalue of (A, E), excluding eigenvalues that are either infinite, uncontrollable, or unobservable. Eigenvalues are deemed uncontrollable or unobservable if $\|B^*y\|_2$ or $\|Cx\|_2$ are respectively below a user-set tolerance, where x and y are respectively the right and left eigenvectors for a given eigenvalue of (A, E). In the discrete-time case, the default initial guesses are zero, π, and the angle for the largest modulus eigenvalue.[4]

[4]For producing a production-quality implementation, see [13] for more sophisticated initial guesses that can be used, [9, section III] for dealing with testing properness of the transfer function, and [33] for filtering out uncontrollable/unobservable eigenvalues of (A, E) when it has index higher than one.

For efficiency of implementing our method and conducting these experiments, our code does not yet take advantage of structure-preserving eigensolvers. Instead, it uses the regular QZ algorithm (`eig` in MATLAB) to compute the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ and $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$. To help mitigate issues due to rounding errors, we consider any eigenvalue λ imaginary or of unit modulus if it lies within a margin of width 10^{-8} on either side of the imaginary axis or unit circle. Taking the imaginary parts of these nearly imaginary eigenvalues forms the initial set of candidate frequencies, or the angles of these nearly unit modulus eigenvalues for the discrete-time case. Then we simply form all the consecutive intervals, including the wrap-around interval for the discrete-time case, even though not all of them will be level-set intervals, and some intervals may only be a portion of a level-set interval (e.g., if the use of QZ causes spurious candidate frequencies). The reason we do this is because we can easily sort which of the intervals at height γ are below $g_{\mathrm{c}}(\omega)$ or $g_{\mathrm{d}}(\omega)$ just by evaluating these functions at the midpoint or the maximizer of the cubic interpolant for each interval. This is less expensive because we need to evaluate these interior points regardless, so also evaluating the norm of the transfer function at all these endpoints just adds additional cost. However, for the cubic interpolant refinement, we nonetheless still evaluate $g_{\mathrm{c}}(\omega)$ or $g_{\mathrm{d}}(\omega)$ at the endpoints since we need the corresponding derivatives there to construct the cubic interpolants; we do not use the eigenvectors of $(\mathcal{M}_\gamma, \mathcal{N})$ or $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ to bypass this additional cost as `eig` in MATLAB does not currently provide a way to only compute selected eigenvectors, i.e., those corresponding to the imaginary (unit-modulus) eigenvalues. Note that while this strategy is sufficient for our experimental comparisons here, it certainly does not negate the need for structure-preserving eigenvalue solvers.

We evaluated our code on several continuous- and discrete-time problems up to moderate dimensions, all listed with dimensions in Table 1. For the continuous-time problems, we chose four problems from the test set used in [20] (`CBM`, `CSE2`, `CM3`, `CM4`), two from the SLICOT benchmark examples[5] (`ISS` and `FOM`), and two new randomly generated examples using `randn()` with a relatively large number of inputs and outputs. Notably, the four problems from [20] were generated via taking open-loop systems from $COMPL_eib$ [27] and then designing controllers to minimize the $\mathcal{H}_\infty$ norm of the corresponding closed-loop systems via HIFOO [14]. Such systems can be interesting benchmark examples because $g_{\mathrm{c}}(\omega)$ will often have several peaks, and multiple peaks may attain the value of the $\mathcal{H}_\infty$ norm, or at least be similar in height. Since the discrete-time problems from [20] were all very small scale (the largest order in that test set is only 16) and SLICOT only offers a single discrete-time benchmark example, we instead elected to take additional open-loop systems from $COMPL_eib$ and obtain usable test examples by minimizing the discrete-time $\mathcal{H}_\infty$ norm of their respective closed-loop systems, via optimizing controllers using HIFOOd [30], a fork of HIFOO for discrete-time systems. On all examples, the $\mathcal{H}_\infty$ norm values computed by our local-optimization-enhanced code (in all its variants) agreed on average to 13 digits with the results provided by `hinfnorm`, when used with the tight tolerance of 10^{-14}, with the worst discrepancy being only 11 digits of agreement. However, our improved method often found slightly larger values, i.e., more accurate values, since it optimizes $g_{\mathrm{c}}(\omega)$ and $g_{\mathrm{d}}(\omega)$ directly.

All experiments were performed using MATLAB R2016b running on a MacBook Pro with an Intel i7-6567U dual-core CPU, 16 GB of RAM, and Mac OS X v10.12.

[5] Available at http://slicot.org/20-site/126-benchmark-examples-for-model-reduction.

TABLE 3

The top half of the table reports the number of times the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ were computed in order to compute the $\mathcal{H}_\infty$ norm to near machine precision. From left to right, the methods are our hybrid optimization approach using Newton's method and the secant method, the cubic-interpolant scheme (column "Interp."), and the standard BBBS method, all implemented by our single configurable code. The subcolumns "Interp." and "MP" of our methods respectively indicate that the optimization routines were initialized at the points from the cubic-interpolant scheme and the BBBS midpoint scheme. The bottom half of the table reports the number of times it was necessary to evaluate the norm of the transfer function (with or without its derivatives). The problems are listed in increasing order of their state-space sizes n; for their exact dimensions, see Table 1.

Small-scale examples (continuous time)						
	Hybrid optimization				Earlier methods	
	Newton		Secant			
Problem	Interp.	MP	Interp.	MP	Interp.	BBBS
Number of eigenvalue computations of $(\mathcal{M}_\gamma, \mathcal{N})$						
`CSE2`	2	3	1	1	2	3
`CM3`	2	3	2	2	3	5
`CM4`	2	2	2	2	4	6
`ISS`	1	1	1	1	3	4
`CBM`	2	2	2	2	5	7
`randn 1`	1	1	1	1	1	1
`randn 2`	1	1	1	1	2	2
`FOM`	1	1	1	1	2	2
Number of evaluations of $g_c(\omega)$						
`CSE2`	10	7	10	10	9	8
`CM3`	31	26	53	45	31	24
`CM4`	19	17	44	43	46	36
`ISS`	12	12	22	22	39	27
`CBM`	34	28	59	55	46	36
`randn 1`	1	1	1	1	1	1
`randn 2`	4	4	17	17	6	4
`FOM`	4	4	16	16	7	5

6.1. Continuous-time examples. In Table 3, we list the number of times the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ were computed and the number of evaluations of $g_c(\omega)$ for our new method compared to our implementations of the existing BBBS algorithm and its interpolation-based refinement. As can be seen, our new method typically limited the number of required eigenvalue computations of $(\mathcal{M}_\gamma, \mathcal{N})$ to just two, and often it required only one (in the cases where our method found a global optimizer of $g_c(\omega)$ in the initialization phase). In contrast, the standard BBBS algorithm and its interpolation-based refinement had to evaluate the eigenvalues $(\mathcal{M}_\gamma, \mathcal{N})$ more times; for example, on problem `CBM`, the BBBS algorithm needed seven evaluations, while its interpolation-based refinement still needed five. Though our new method sometimes required more evaluations of $g_c(\omega)$ than the standard algorithms, often the number of evaluations of $g_c(\omega)$ was actually less with our new method, presumably due to its fewer iterations and particularly when using the Newton's method variants. Even when our method required more evaluations of $g_c(\omega)$ than the standard methods, the increases were not too significant (e.g., the secant method variants of our method on problems `CM4`, `CBM`, `randn 2`, and `FOM`). Indeed, the larger number of evaluations of $g_c(\omega)$ when employing the secant method in lieu of Newton's method was still generally quite low.

In Table 4, we compare the corresponding wall-clock times, and for convenience, we replicate the timing results of `hinfnorm` from Table 1 on the same problems. We

TABLE 4

In the top half of the table, the running times (fastest in bold) are reported in seconds for the same methods and configurations as in Table 3, with the running times of `hinfnorm` *additionally listed in the rightmost two columns, for a tolerance of* 10^{-14} *(as used by the other methods) and its default value of* 0.01*. The bottom half of the table normalizes all the times relative to the running times for our hybrid optimization method (Newton and "Interp."), along with the overall averages relative to this variant.*

Small-scale examples (continuous time)								
	Hybrid optimization				Earlier methods		`hinfnorm(·,tol)`	
	Newton		Secant				`tol`	
Problem	Interp.	MP	Interp.	MP	Interp.	BBBS	`1e-14`	`0.01`
Wall-clock running times in seconds								
`CSE2`	0.042	0.060	0.036	0.032	0.043	**0.031**	0.137	0.022
`CM3`	**0.125**	0.190	0.198	0.167	0.170	0.167	0.148	0.049
`CM4`	**0.318**	0.415	0.540	0.550	0.712	0.811	1.645	0.695
`ISS`	0.316	0.328	0.379	**0.303**	0.709	0.757	0.765	0.391
`CBM`	0.744	**0.671**	1.102	1.071	1.649	1.757	3.165	1.532
`randn 1`	0.771	0.868	1.006	0.871	**0.700**	0.756	21.084	30.049
`randn 2`	**9.551**	9.746	9.953	11.275	14.645	15.939	31.728	16.199
`FOM`	**3.039**	3.426	4.418	4.176	5.509	5.182	128.397	36.529
Running times relative to hybrid optimization (Newton with "Interp.")								
`CSE2`	1	1.42	0.86	0.75	1.02	0.75	3.24	0.53
`CM3`	1	1.52	1.59	1.34	1.36	1.34	1.19	0.39
`CM4`	1	1.31	1.70	1.73	2.24	2.55	5.18	2.19
`ISS`	1	1.04	1.20	0.96	2.24	2.39	2.42	1.24
`CBM`	1	0.90	1.48	1.44	2.22	2.36	4.26	2.06
`randn 1`	1	1.13	1.31	1.13	0.91	0.98	27.36	38.99
`randn 2`	1	1.02	1.04	1.18	1.53	1.67	3.32	1.70
`FOM`	1	1.13	1.45	1.37	1.81	1.71	42.25	12.02
Average	1	1.18	1.33	1.24	1.67	1.72	11.15	7.39

observe that our new method was fastest on six of the eight test problems, often significantly so. Compared to our own implementation of the BBBS algorithm, our new method was on average 1.72 times as fast, and on three problems, 2.36 to 2.55 times faster. We see similar speedups compared to the cubic-interpolation refinement method as well. Our method was even faster when compared to `hinfnorm`, which had the advantage of being a compiled code rather than interpreted like our code. Our new method was over 11 times faster than `hinfnorm` overall, but this was largely due to the two problems (`FOM` and `randn 1`) where our code was 27 to 42 times faster. We suspect that this large performance gap on these problems was not necessarily due to a correspondingly dramatic reduction in the number of times that the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ were computed but rather that the structure-preserving eigensolver `hinfnorm` employed sometimes has a steep performance penalty compared to standard QZ. However, it is difficult to verify this as `hinfnorm` is not open source. We also see that for the variants of our method, there was about a 24% to 33% penalty on average in the runtime when resorting to the secant method instead of Newton's method. Nonetheless, even the slower secant-method-based version of our hybrid optimization approach was still typically much faster than BBBS or the cubic-interpolation scheme. The only exception to this was problem `CSE2`, where our secant method variants were actually faster than our Newton's method variants; the reason for this was because during initialization, the Newton's method optimization just happened

to find worse initial local maximizers than the secant method approach, which led to more eigenvalue computations of $(\mathcal{M}_\gamma, \mathcal{N})$.

The two problems where the variants of our new method were not fastest were `CSE2` and `randn 1`. However, for `CSE2`, our secant method variant using midpoints was essentially as fast as the standard algorithm. As mentioned above, the Newton's method variants ended up being slower since they found worse initial local maximizers. For `randn 1`, all methods only required a single evaluation of $g_{\mathrm{c}}(\omega)$ and computing the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$; in other words, their respective initial guesses were all actually a global maximizer. As such, the differences in running times for `randn 1` seem likely attributed to the variability of interpreted MATLAB code.

6.2. Discrete-time examples. We now present corresponding experiments for the six discrete-time examples listed in Table 1. In Table 5, we see that our new approach on discrete-time problems also reduces the number of expensive eigenvalue computations of $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ compared to the standard methods and that in the worst cases, there is only a moderate increase in the number of evaluations of $g_{\mathrm{d}}(\omega)$ and often even a reduction, similarly as we saw in Table 3 for the continuous-time problems.

Wall-clock running times are reported in Table 6 and show similar, if not identical, results to those in Table 4 for the continuous-time comparison. We see that our Newton's method variants are, on average, 1.66 and 1.41 times faster, respectively, than the BBBS and cubic-interpolation refinement algorithms. Our algorithms are often up to two times faster than these two standard methods and were even up to 25.2 times faster on `ISS1d` compared to `hinfnorm` using `tol=1e-14`. For three of the six problems, our approach was not fastest but these three problems (`LAHd`, `BDT2d`, `EB6d`) also had the smallest orders among the discrete-time examples ($n = 58, 92, 170$, respectively). This underscores that our approach is likely most beneficial for all but rather small-scale problems, where there is generally an insufficient cost gap between computing $g_{\mathrm{d}}(\omega)$ and the eigenvalues of $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$. However, for `LAHd` and `EB6d`, it

TABLE 5
The column headers remain as described for Table 3.

Small-scale examples (discrete time)						
	Hybrid optimization				Earlier methods	
	Newton		Secant			
Problem	Interp.	MP	Interp.	MP	Interp.	BBBS
Number of eigenvalue computations of $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$						
`LAHd`	2	2	1	1	3	4
`BDT2d`	2	3	2	2	3	4
`EB6d`	1	1	2	1	3	5
`ISS1d`	1	1	1	1	2	2
`CBMd`	1	1	1	1	3	2
`CM5d`	2	2	2	2	3	4
Number of evaluations of $g_{\mathrm{d}}(\omega)$						
`LAHd`	13	11	24	24	17	15
`BDT2d`	17	18	43	40	18	17
`EB6d`	22	22	37	34	32	32
`ISS1d`	5	5	24	24	7	6
`CBMd`	5	5	26	26	12	6
`CM5d`	20	16	27	27	22	18

TABLE 6
The column headers remain as described for Table 4.

Small-scale examples (discrete time)								
	Hybrid optimization				Earlier methods		`hinfnorm(·,tol)`	
	Newton		Secant				`tol`	
Problem	Interp.	MP	Interp.	MP	Interp.	BBBS	`1e-14`	`0.01`
Wall-clock running times in seconds								
`LAHd`	0.051	0.038	0.056	0.056	0.034	0.040	**0.031**	0.015
`BDT2d`	0.075	0.123	0.146	0.191	**0.057**	0.076	0.070	0.031
`EB6d`	0.271	0.312	0.409	0.296	0.469	0.732	**0.192**	0.122
`ISS1d`	0.654	**0.636**	0.828	0.880	1.168	1.291	16.495	3.930
`CBMd`	0.898	**0.795**	0.999	1.640	2.015	1.420	1.411	0.773
`CM5d`	**7.502**	8.022	9.887	8.391	9.458	14.207	10.802	2.966
Running times relative to hybrid optimization (Newton with "Interp.")								
`LAHd`	1	0.74	1.09	1.11	0.67	0.78	0.60	0.29
`BDT2d`	1	1.65	1.96	2.55	0.76	1.01	0.93	0.41
`EB6d`	1	1.15	1.51	1.09	1.73	2.70	0.71	0.45
`ISS1d`	1	0.97	1.27	1.34	1.79	1.97	25.21	6.01
`CBMd`	1	0.89	1.11	1.83	2.24	1.58	1.57	0.86
`CM5d`	1	1.07	1.32	1.12	1.26	1.89	1.44	0.40
Avg.	1	1.08	1.38	1.51	1.41	1.66	5.08	1.40

was actually `hinfnorm` that was fastest, where we are comparing a compiled code to our own pure MATLAB interpreted code. Furthermore, on these two problems, our approach was nevertheless not dramatically slower than `hinfnorm` and for `EB6d` was actually faster than our own implementation of the standard algorithms. Finally, on `BDT2`, the fastest version of our approach essentially matched the performance of our BBBS implementation, if not the cubic-interpolation refinement.

7. Local optimization for $\mathcal{H}_\infty$ norm approximation. Unfortunately, the $\mathcal{O}(n^3)$ work necessary to compute all the imaginary eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ restricts the usage of the level-set ideas from [15, 12] to rather small-dimensional problems. The same computational limitation of course also holds for obtaining all of the unit-modulus eigenvalues of $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$ in the discrete-time case. Currently there is no known alternative technique that would guarantee convergence to a global maximizer of $g_{\mathrm{c}}(\omega)$ or $g_{\mathrm{d}}(\theta)$, to thus ensure exact computation of the $\mathcal{H}_\infty$ norm, while also having more favorable scaling properties. Indeed, the aforementioned scalable methods of [20, 10, 16, 28, 1] for approximating the $\mathcal{H}_\infty$ norm of large-scale systems all forgo the expensive operation of computing all the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ and $(\mathcal{S}_\gamma, \mathcal{T}_\gamma)$, and consequently, the most that any of them can guarantee in terms of accuracy is that they converge to a local maximizer of $g_{\mathrm{c}}(\omega)$ or $g_{\mathrm{d}}(\theta)$. However, a direct consequence of our work here to accelerate the exact computation of the $\mathcal{H}_\infty$ norm is that the straightforward application of optimization techniques to compute local maximizers of either $g_{\mathrm{c}}(\omega)$ of $g_{\mathrm{d}}(\theta)$ can itself be considered an efficient and scalable approach for approximating the $\mathcal{H}_\infty$ norm of large-scale systems. It is perhaps a bit staggering that such a simple and direct approach seems to have been until now overlooked, particularly given the sophistication of the existing $\mathcal{H}_\infty$ norm approximation methods.

In more detail, recall that the initialization phase of Algorithm 2, lines 2–6, is simply just applying unconstrained optimization to find one or more maximizers of $g_{\mathrm{c}}(\omega)$.

Provided that $(\mathbf{i}\omega E - A)$ permits fast linear solves, e.g., a sparse LU decomposition, there is no reason why this cannot also be done for large-scale systems. In fact, the methods of [10, 16, 1] for approximating the $\mathcal{H}_\infty$ norm all require that such fast solves are possible (while the methods of [20, 28] only require fast matrix-vector products with the system matrices). When $m, p \ll n$, it is still efficient to calculate second derivatives of $g_c(\omega)$ to obtain a quadratic rate of convergence via Newton's method. Even if $m, p \ll n$ does not hold, first derivatives of $g_c(\omega)$ can still be computed using sparse methods for computing the largest singular value (and its singular vectors) and thus the secant method can be employed to at least get superlinear convergence. As such, the local convergence and superlinear/quadratic convergence rate guarantees of the existing methods are at least matched by the guarantees of direct optimization. For example, while the superlinearly convergent method of [1] requires that $m, p \ll n$, our direct optimization approach remains efficient even if $m, p \approx n$, when it also has superlinear convergence, and it has quadratic convergence in the more usual case of $m, p \ll n$.

Of course, there is also the question of whether there are differences in approximation quality between the methods. This is a difficult question to address since beyond local optimality guarantees, there are no other theoretical results concerning the quality of the computed solutions. Actual errors can be measured when running the methods on small-scale systems, where the exact value of the $\mathcal{H}_\infty$ norm can be computed, but for large-scale problems, only relative differences between the methods' respective approximations can be observed. Furthermore, any of these observations may not be predictive of performance on other problems. For nonconvex optimization, the quality of a locally optimal computed solution is often dependent on the starting point, which will be a strong consideration for the direct optimization approach. On the other hand, it is certainly plausible that the sophistication of the existing $\mathcal{H}_\infty$ norm algorithms may favorably bias them to converge to better (higher) maximizers more frequently than direct optimization would, particularly if only randomly selected starting points were used. With such complexities, in this paper we do not attempt to do a comprehensive benchmark with respect to existing $\mathcal{H}_\infty$ norm approximation methods but only attempt to demonstrate that direct optimization is a potentially viable alternative.

We implemented lines 2–6 of Algorithm 2 in a second, standalone routine, with the necessary addition for the continuous-time case that the value of $\|D\|_2$ is returned if the computed local maximizers of $g_c(\omega)$ only yield lower function values than $\|D\|_2$. Since we assume that the choice of starting points will be critical, we initialized our sparse routine using starting frequencies computed by `samdp`, a MATLAB code that implements the subspace-accelerated dominant pole algorithm of [31]. Correspondingly, we compared our approach to the MATLAB code `hinorm`, which implements the spectral-value-set-based method using dominant poles of [10] and also uses `samdp` (to compute dominant poles at each iteration). We tested `hinorm` using its default settings, and since it initially computes 20 dominant poles to find a good starting point, we also chose to compute 20 dominant poles via `samdp` to obtain 20 initial frequency guesses for optimization.[6] Like our small-scale experiments, we also ensured zero was always included as an initial guess and reused the same choices for `fmincon` parameter values. We tested our optimization approach by optimizing $\phi = 1, 5, 10$ of the most promising frequencies, again using a serial MATLAB code. Since we used

[6] Note that these are not necessarily the same 20 dominant poles, since [10] must first transform a system if the original system has nonzero D matrix.

TABLE 7

The list of test problems for the large-scale $\mathcal{H}_\infty$-norm approximation comparing direct local optimization against `hinorm`*, along with the corresponding problem dimensions and whether they are standard state-space systems ($E = I$) or descriptor systems ($E \neq I$).*

Large-scale examples (continuous time)				
Problem	n	p	m	$E = I$
`dwave`	2048	4	6	Y
`markov`	5050	4	6	Y
`bips98_1450`	11305	4	4	N
`bips07_1693`	13275	4	4	N
`bips07_1998`	15066	4	4	N
`bips07_2476`	16861	4	4	N
`descriptor_xingo6u`	20738	1	6	N
`mimo8x8_system`	13309	8	8	N
`mimo28x28_system`	13251	28	28	N
`ww_vref_6405`	13251	1	1	N
`xingo_afonso_itaipu`	13250	1	1	N

LU decompositions to solve the linear systems, we tested our code in two configurations: with and without permutations, i.e., for some matrix given by variable `A`, `[L,U,p,q] = lu(A,'vector')` and `[L,U] = lu(A)`, respectively.

Table 7 shows our selection or large-scale test problems, all continuous-time since `hinorm` does not support discrete-time problems (in contrast to our optimization-based approach which supports both). Problems `dwave` and `markov` are from the large-scale test set used in [20] while the remaining problems are freely available from the website of Joost Rommes.[7] As $m, p \ll n$ holds in all of these examples, we just present results for our code when using Newton's method. For all problems, our code produced $\mathcal{H}_\infty$ norm approximations that agreed to at least 12 digits with `hinorm`, meaning that the additional optimization calls done with $\phi = 5$ and $\phi = 10$ did not produce better maximizers than what was found with $\phi = 1$ and thus only added to the serial computation running time.

In Table 8, we present the running times of the codes and configurations. First, we observe that for our direct optimization code, using `lu` with permutations is two to eight times faster than without permutations; on average, using `lu` with permutations is typically 2.5 times faster. Interestingly, on the last five problems, using `lu` without permutations was actually best, but using permutations was typically only about 25% slower and at worse, about 1.7 times slower (`descriptor_xingo6u`). We found that our direct optimization approach, using just one starting frequency ($\phi = 1$), was typically 3.7 times faster than `hinorm` on average and almost up to 10 times faster on problem `bips07_1693`. Only on problem `dwave` was direct optimization actually slower than `hinorm` and only by a negligible amount. Interestingly, optimizing just one initial frequency versus running optimization for 10 frequencies ($\phi = 10$) typically only increased the total running time of our code by 20% to 30%. This strongly suggested that the dominant cost of running our code is actually just calling `samdp` to compute the 20 initial dominant poles to obtain starting guesses. As such, in Table 9, we report the percentage of the overall running time for each variant/method that was due to their initial calls to `samdp`. Indeed, our optimization code's single call to `samdp` accounted for 81.5% to 99.3% of its running time (`lu` with permutations and $\phi = 1$). In contrast, `hinorm`'s initial call to `samdp` usually accounted for about only a quarter

[7] Available at https://sites.google.com/site/rommes/software.

TABLE 8

In the top half of the table, the running times (fastest in bold) are reported in seconds for our direct Newton-method-based optimization approach in two configurations (`lu` with and without permutations) and `hinorm`. Each configuration of our approach optimizes the norm of the transfer function for up to ϕ different starting frequencies ($\phi = 1, 5, 10$), done sequentially. The bottom half of the table normalizes all the times relative to the running times for our optimization method using `lu` with permutations and $\phi = 1$, along with the overall averages relative to this variant.

Large-scale examples (continuous time)							
	Direct optimization: $\phi = 1, 5, 10$						`hinorm`
	`lu` with permutations			`lu` without permutations			
Problem	1	5	10	1	5	10	
Wall-clock running times in seconds (initialized via `samdp`)							
`dwave`	1.979	1.981	1.997	5.536	5.154	5.543	**1.861**
`markov`	**3.499**	3.615	3.593	26.734	26.898	27.219	3.703
`bips98_1450`	**6.914**	8.333	10.005	14.559	17.157	19.876	31.087
`bips07_1693`	**8.051**	9.155	11.594	18.351	21.367	24.322	75.413
`bips07_1998`	**10.344**	11.669	13.881	50.059	56.097	59.972	51.497
`bips07_2476`	**14.944**	16.717	18.942	65.227	70.920	73.206	76.697
`descriptor_xingo6u`	13.716	15.328	16.997	**7.907**	9.225	11.133	36.775
`mimo8x8_system`	7.566	8.934	11.211	**6.162**	7.562	9.321	30.110
`mimo28x28_system`	12.606	17.767	20.488	**10.815**	16.591	21.645	33.107
`ww_vref_6405`	7.353	6.785	7.552	**4.542**	5.076	5.437	18.553
`xingo_afonso_itaipu`	5.780	6.048	7.772	**4.676**	4.975	5.573	16.928
Running times relative to direct optimization (`lu` with permutations and $\phi = 1$)							
`dwave`	1	1.00	1.01	2.80	2.60	2.80	**0.94**
`markov`	**1**	1.03	1.03	7.64	7.69	7.78	1.06
`bips98_1450`	**1**	1.21	1.45	2.11	2.48	2.87	4.50
`bips07_1693`	**1**	1.14	1.44	2.28	2.65	3.02	9.37
`bips07_1998`	**1**	1.13	1.34	4.84	5.42	5.80	4.98
`bips07_2476`	**1**	1.12	1.27	4.36	4.75	4.90	5.13
`descriptor_xingo6u`	1	1.12	1.24	**0.58**	0.67	0.81	2.68
`mimo8x8_system`	1	1.18	1.48	**0.81**	1.00	1.23	3.98
`mimo28x28_system`	1	1.41	1.63	**0.86**	1.32	1.72	2.63
`ww_vref_6405`	1	0.92	1.03	**0.62**	0.69	0.74	2.52
`xingo_afonso_itaipu`	1	1.05	1.34	**0.81**	0.86	0.96	2.93
Average	1	1.12	1.30	2.52	2.74	2.97	3.70

of its running time on average, excluding `dwave` and `markov` as exceptional cases. In other words, the convergent phase of direct optimization is actually even faster than the convergent phase of `hinorm` than what Table 8 appears to indicate. On problem `bips07_1693`, we see that our proposal to use Newton's method to optimize $g_c(\omega)$ directly is actually over 53 times faster than `hinorm`'s convergent phase.

8. Parallelizing the algorithms. The original BBBS algorithm, and the cubic-interpolation refinement, provide little opportunity for parallelization at the *algorithmic* level, i.e., when not considering that the underlying basic linear algebra operations may be parallelized themselves when running on a single shared-memory multicore machine. Once the imaginary eigenvalues[8] have been computed, constructing the level-set intervals (line 6 of Algorithm 1) and calculating $g_c(\omega)$ at their midpoints or cubic-interpolant-derived maximizers (line 8 of Algorithm 1) can both be done in an

[8]For conciseness, our discussion in section 8 will be with respect to the continuous-time case but note that it applies equally to the discrete-time case as well.

TABLE 9
The column headings are the same as in Table 8.

Large-scale examples (continuous time)							
Percentage of time just to compute 20 initial dominant poles (first call to `samdp`)							
	Direct optimization: $\phi = 1, 5, 10$						`hinorm`
	`lu` with permutations			`lu` without permutations			
Problem	1	5	10	1	5	10	
`dwave`	99.3	99.3	99.2	99.3	99.6	99.6	98.0
`markov`	99.2	99.1	99.1	99.6	99.6	99.6	98.1
`bips98_1450`	84.1	72.7	60.4	84.9	73.0	61.8	21.7
`bips07_1693`	84.2	75.1	61.6	85.6	77.1	64.4	10.1
`bips07_1998`	85.2	77.1	66.0	88.5	81.5	72.8	18.3
`bips07_2476`	88.0	80.2	71.4	89.2	83.6	73.0	18.6
`descriptor_xingo6u`	90.2	82.8	75.0	89.3	79.9	66.8	38.4
`mimo8x8_system`	84.6	74.4	59.1	84.1	67.7	55.0	24.2
`mimo28x28_system`	81.5	59.4	51.2	75.5	46.9	39.1	40.9
`ww_vref_6405`	95.1	79.6	63.9	91.9	78.3	75.1	33.7
`xingo_afonso_itaipu`	90.2	84.4	76.8	89.3	83.7	74.7	33.6

embarrassingly parallel manner, e.g., across nodes on a cluster. However, as we have discussed to motivate our improved algorithm, evaluating $g_c(\omega)$ is a rather cheap operation compared to computing the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$. Crucially, parallelizing these two steps does not result in an improved (higher) value of γ found per iteration and so the number of expensive eigenvalue computations of $(\mathcal{M}_\gamma, \mathcal{N})$ remains the same.

For our new method, we certainly can (and should) also parallelize the construction of the level-set intervals and the evaluations of their midpoints or cubic-interpolants-derived maximizers (lines 12 and 14 in Algorithm 2), even though we do not expect large gains to be had here. However, optimizing over the intervals (line 18 in Algorithm 2) is also an embarrassingly parallel task and here significant speedups can be obtained. As mentioned earlier, with serial computation (at the algorithmic level), we typically recommend only optimizing over a single level-set interval ($\phi = 1$) out of the q candidates (the most promising one, as determined by line 17 in Algorithm 2); otherwise, the increased number of evaluations of $g_c(\omega)$ can start to outweigh the benefits of performing the local optimization. By optimizing over more intervals in parallel, e.g., again across nodes on a cluster, we increase the chances on every iteration of finding even higher peaks of $g_c(\omega)$, and possibly a global maximum, *without any increased time penalty* (besides communication latency).[9] In turn, larger steps in γ can be taken, potentially reducing the number of expensive eigenvalue computations of $(\mathcal{M}_\gamma, \mathcal{N})$ incurred. Furthermore, parallelization can also be applied to the initialization stage to optimize from as many starting points as possible without time penalty (lines 2 and 4 in Algorithm 2), a standard technique for nonconvex optimization problems. Finding a global maximum of $g_c(\omega)$ during initialization means that the algorithm will only need to compute the eigenvalues of $(\mathcal{M}_\gamma, \mathcal{N})$ just once, to assert that maximum found is indeed a global one.

When using direct local optimization techniques for $\mathcal{H}_\infty$ approximation, as discussed in section 8, optimizing from as many starting points as possible of course also

[9] Note that distributing the starting points for optimization and taking the max of the resulting optimizers involves very little data being communicated on any iteration.

increases the chances of finding the true value of the $\mathcal{H}_\infty$ norm, or at least better approximations than just starting from one point. With parallelization, these additional starting points can also be tried without any time penalty (also lines 2 and 4 in Algorithm 2), unlike the experiments we reported in section 7 where we optimized using $\phi = 1, 5, 10$ starting guesses with a serial MATLAB code and therefore incurred longer running times as ϕ was increased.

For final remarks on parallelization, first note that there will generally be less benefit when using more than n parallel optimization calls, since there are at most n peaks of $g_c(\omega)$. However, for initialization, one could simply try as many starting guesses as there are parallel nodes available (even if the number of nodes is greater than n) to maximize the chances of finding a high peak of $g_c(\omega)$ or a global maximizer. Second, the number of level-set intervals encountered by the algorithm at each iteration may be significantly less than n, particularly if good starting guesses are used. Indeed, it is not entirely uncommon for the algorithm to only encounter one or two level-set intervals on each iteration. On the other hand, for applications where $g_c(\omega)$ has many similarly high peaks, such as controller design where the $\mathcal{H}_\infty$ norm is minimized, our new algorithm may consistently benefit from parallelization with a higher number of parallel optimization calls.

9. Conclusion and outlook. We have presented an improved algorithm that significantly reduces the time necessary to compute the $\mathcal{H}_\infty$ norm of linear control systems compared to existing algorithms. Furthermore, our proposed hybrid optimization approach also allows the $\mathcal{H}_\infty$ norm to be computed to machine precision with relatively little extra work, unlike earlier methods. We have also demonstrated that approximating the $\mathcal{H}_\infty$ norm of large-scale problems via directly optimizing the norm of the transfer function is not only viable but it can be quite efficient too. In contrast to the standard BBBS and cubic-interpolation refinement algorithms, our new approaches for $\mathcal{H}_\infty$ norm computation and approximation also can benefit significantly more from parallelization. Work is ongoing to add implementations of our new algorithms to a future release of the open-source library ROSTAPACK: RObust STAbility PACKage.[10] This is being done in coordination with our efforts to also add implementations of our new methods for computing the spectral value set abscissa and radius, proposed in [6] and which use ideas related to those in this paper. The current v1.0 release of ROSTAPACK contains implementations of scalable algorithms for approximating all of these aforementioned measures [21, 20, 28], as well as variants where the uncertainties are restricted to be real valued [19].

Regarding our experimental observations, the sometimes excessively longer compute times for `hinfnorm` (compared to all other methods we evaluated) possibly indicate that the structure-preserving eigensolver that it uses can sometimes be much slower than QZ. This certainly warrants further investigation and, if confirmed, suggests that optimizing the code/algorithm of the structure-preserving eigensolver could be a worthwhile pursuit. In the large-scale setting, we have observed that the dominant cost for our direct optimization approach is actually due to obtaining the starting frequency guesses via computing dominant poles. If the process of obtaining good initial guesses can be accelerated, then approximating the $\mathcal{H}_\infty$ norm via direct optimization could be significantly sped up even more.

Acknowledgment. The authors are extremely grateful to both referees for their many useful comments toward improving the paper.

[10] Available at http://www.timmitchell.com/software/ROSTAPACK.

REFERENCES

[1] N. Aliyev, P. Benner, E. Mengi, P. Schwerdtner, and M. Voigt, *Large-scale computation of $\mathcal{L}_\infty$-norms by a greedy subspace method*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1496–1516.

[2] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, Adv. Des. Control 6, SIAM, Philadelphia, 2005.

[3] M. N. Belur and C. Praagman, *An efficient algorithm for computing the H_∞ norm*, IEEE Trans. Automat. Control, 56 (2011), pp. 1656–1660.

[4] P. Benner, R. Byers, V. Mehrmann, and H. Xu, *Numerical computation of deflating subspaces of skew-Hamiltonian/Hamiltonian pencils*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 165–190.

[5] P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, *Model Reduction and Approximation: Theory and Algorithms*, Comput. Sci. Eng. 15, SIAM, Philadelphia, 2017.

[6] P. Benner and T. Mitchell, *Extended and Improved Criss-Cross Algorithms for Computing the Spectral Value Set Abscissa and Radius*, arXiv:1712.10067, 2017.

[7] P. Benner, V. Sima, and M. Voigt, *$\mathcal{L}_\infty$-norm computation for continuous-time descriptor systems using structured matrix pencils*, IEEE Trans. Automat. Control, 57 (2012), pp. 233–238.

[8] P. Benner, V. Sima, and M. Voigt, *Algorithm* 961: *Fortran* 77 *subroutines for the solution of skew-Hamiltonian/Hamiltonian eigenproblems*, ACM Trans. Math. Software, 42 (2016).

[9] P. Benner and M. Voigt, *On the computation of particular eigenvectors of Hamiltonian matrix pencils*, Proc. Appl. Math. Mech., 11 (2011), pp. 753–754.

[10] P. Benner and M. Voigt, *A structured pseudospectral method for $\mathcal{H}_\infty$-norm computation of large-scale descriptor systems*, Math. Control Signals Systems, 26 (2014), pp. 303–338.

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

[12] S. Boyd, V. Balakrishnan, and P. Kabamba, *A bisection method for computing the $\mathcal{H}_\infty$ norm of a transfer matrix and related problems*, Math. Control Signals Systems, 2 (1989), pp. 207–219.

[13] N. A. Bruinsma and M. Steinbuch, *A fast algorithm to compute the H_∞-norm of a transfer function matrix*, Systems Control Lett., 14 (1990), pp. 287–293.

[14] J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton, *HIFOO—A MATLAB package for fixed-order controller design and H_∞ optimization*, in 5th IFAC Symposium on Robust Control Design ROCOND 2006, IFAC Proc. Vol. 39, 2006, pp. 339–344.

[15] R. Byers, *A bisection method for measuring the distance of a stable to unstable matrices*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 875–881.

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

[17] Y. Genin, P. Van Dooren, and V. Vermaut, *Convergence of the calculation of $\mathcal{H}_\infty$-norms and related questions*, in Proceedings of MTNS-98, 1998, pp. 429–432.

[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins Stud. Math. Sci., Johns Hopkins University Press, Baltimore, 2013.

[19] N. Guglielmi, M. Gürbüzbalaban, T. Mitchell, and M. L. Overton, *Approximating the real structured stability radius with Frobenius-norm bounded perturbations*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1323–1353.

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

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

[22] C. He and G. A. Watson, *An algorithm for computing the distance to instability*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 101–116.

[23] D. Hinrichsen and A. J. Pritchard, *Mathematical Systems Theory* I: *Modelling, State Space Analysis, Stability and Robustness*, Springer-Verlag, Berlin, 2005.

[24] D. Hinrichsen and N. K. Son, *Stability radii of linear discrete-time systems and symplectic pencils*, Internat. J. Robust Nonlinear Control, 1 (1991), pp. 79–97.

[25] T. Kato, *A Short Introduction to Perturbation Theory for Linear Operators*, Springer-Verlag, New York, 1982.

[26] P. Lancaster, *On eigenvalues of matrices dependent on a parameter*, Numer. Math., 6 (1964), pp. 377–387.

[27] F. Leibfritz, *COMPl$_e$ib: COnstrained Matrix-optimization Problem library—a collection of test examples for nonlinear semidefinite programs, control system design and related problems*, http://www.compleib.de (2004).

[28] T. Mitchell and M. L. Overton, *Hybrid expansion-contraction: a robust scaleable method for approximating the H_∞ norm*, IMA J. Numer. Anal., 36 (2016), pp. 985–1014.

[29] M. L. Overton and R. S. Womersley, *Second derivatives for optimizing eigenvalues of symmetric matrices*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 697–718.

[30] A. P. Popov, H. Werner, and M. Millstone, *Fixed-structure discrete-time $\mathcal{H}_\infty$ controller synthesis with HIFOO*, in Proceedings of the 49th IEEE Conference on Decision and Control, 2010, pp. 3152–3155.

[31] J. Rommes and N. Martins, *Computing transfer function dominant poles of large-scale second-order dynamical systems*, IEEE Trans. Power Syst., 21 (2006), pp. 1471–1483.

[32] J. Sreedhar, P. Van Dooren, and A. Tits, *A fast algorithm to compute the real structured stability radius*, in Proceedings of the Conference Centennial Hurwitz on Stability Theory, Ticino, 1995.

[33] A. Varga, *Computation of irreducible generalized state-space realizations*, Kybernetika (Prague), 26 (1990), pp. 89–106.