Robust and efficient methods for approximation and optimization of stability measures

by

Tim Mitchell

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

September 2014

Michael L. Overton

Dedication

To E.C.

ACKNOWLEDGMENTS

First and foremost, I am very grateful to Michael Overton, who has been a wonderful advisor over these past years and furthermore has become a good friend. Our mutual collaborators, Frank Curtis, Nicola Guglielmi, and Mert Gürbüzbalaban, have each provided me with interesting problems and insightful feedback and broadened my pool of knowledge. Indeed, they have been a pleasure to work with and know. David Bindel and Josef Sifuentes have also been quite supportive and engaging in my research endeavors as a graduate student, particularly in the earlier years. I am also appreciative to the members of my thesis committee for their experienced comments and suggestions. Rosemary Amico, Stephanie Tracy, Santiago Pizzini, and Andy Howell have all helped me though administrative and technical hurdles and in general, have just made graduate life very pleasant by allowing me to remain focused on my work without distractions. I am also similarly indebted to Pat Quillen, who has been a very valuable resource to have at The MathWorks, for being responsive and helpful when I have encountered the thorny kinds of computational issues that we all just wish would go away. Finally, I have been extremely fortunate to have had the particular support of Michael Overton, Margaret Wright, Denis Zorin, Olof Widlund, Rosemary Amico, Santiago Pizzini, and perhaps others unbeknownst to me, in helping to ensure that this thesis and my degree were not derailed by the kind of challenging events that can spontaneously arise at times in life. Thank you all for believing in me and allowing me a wide latitude of patience and understanding when it was most needed.

ABSTRACT

We consider two new algorithms with practical application to the problem of designing controllers for linear dynamical systems with input and output: a new spectral value set based algorithm called hybrid expansion-contraction intended for approximating the H_∞ norm, or equivalently, the complex stability radius, of large-scale systems, and a new BFGS SQP based optimization method for nonsmooth, nonconvex constrained optimization motivated by multi-objective controller design. In comprehensive numerical experiments, we show that both algorithms in their respect domains are significantly faster and more robust compared to other available alternatives. Moreover, we present convergence guarantees for hybrid expansion-contraction, proving that it converges at least superlinearly, and observe that it converges quadratically in practice, and typically to good approximations to the H_∞ norm, for problems which we can verify this. We also extend the hybrid expansion-contraction algorithm to the real stability radius, a measure which is known to be more difficult to compute than the complex stability radius. Finally, for the purposes of comparing multiple optimization methods, we present a new visualization tool called relative minimization profiles that allow for simultaneously assessing the relative performance of algorithms with respect to three important performance characteristics, highlighting how these measures interrelate to one another and compare to the other competing algorithms on heterogenous test sets. We employ relative minimization profiles to empirically validate our proposed BFGS SQP method in terms of quality of minimization, attaining feasibility, and speed of progress compared to other available methods on challenging test sets comprised of nonsmooth, nonconvex constrained optimization problems arising in controller design.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

List of Algorithms

List of Appendices

INTRODUCTION

An important problem occurring in many engineering applications is that of controller design, which usually involves stabilizing and optimizing a given dynamical system with inputs and outputs, according to some specified stability measure. However, in practice, the mathematical model given by the dynamical system is almost unquestionably an imperfect representation of reality and it may even not be apparent to what degree its accuracy holds. Thus, it is imperative to consider robust stability measures, that is, measures which can quantify the least amount of perturbation a linear dynamical system with input and output can incur such that asymptotically stabiltiy can no longer be guaranteed. As robust stability measures are typically nonsmooth, nonconvex, expensive to compute, and sometimes even difficult to approximate, improved algorithms for the controller design problem may be realized not only by improved optimization algorithms but also by improved methods for calculating or approximating the stability measures themselves.

Perhaps the most well known and important robust stability measure arising in applications is the H_∞ norm, or equivalently, the complex stability radius. Calculating it however is an expensive proposition of finding a global optimum of a nonconvex and nonsmooth optimization problem, and while the standard Boyd-Balakrishnan-Bruinsma-Steinbuch (BBBS) algorithm [BB90, BS90] can find the optimal solution, its cubic cost per iteration limits the algorithm's applicability to rather small-dimensional systems. For large-dimensional systems, one approach is to use model reduction techniques, where one approximates a large-scale system by a smaller surrogate model, sufficiently reduced in size so that the H_∞ norm of it can be tractably computed via the standard BBBS algorithm.

However, there has been a recent flurry of interest in approximating the H_∞ norm for large and sparse systems without reducing the dimensions of the original matrices, notably by [GGO13] and, for related descriptor systems, by [BV14, FSVD14]. Though the last of these algorithms relies on solving linear systems, and is thus unlikely to scale as well as the first two methods, all three methods are much faster than the BBBS algorithm for large problems and appear to be able to compute good approximations to the H_∞ norm.

On the other hand, the algorithm presented by Guglielmi, Gürbüzbalaban, and Overton [GGO13], which we call the GGO algorithm, assumes that the otherwise large and sparse systems have relatively few inputs and outputs, while its main computational subroutine only converges linearly. Also, we demonstrate that the GGO algorithm can sometimes break down, which can potentially lead to a poor approximation of the H_∞ norm. Fortunately, it is usually trivial to check whether the method has incurred breakdown after the fact and it appears that the algorithm often does converge properly without incident on most problems. However, in the case of breakdown, it is difficult to assess whether the approximation quality has suffered and running time of the algorithm is typically greatly increased. Though this appears to be a somewhat infrequent scenario, it nonetheless is a real consideration if the method is to be used for designing controllers in actual applications.

In practice, it is often desirable to not just stabilize and enhance a single linear dynamical system with respect to some stability measure, but to in fact design a single controller which can simultaneously stabilize and enhance the stability of multiple dynamical systems. Even the single-objective case typically amounts to a nonsmooth, nonconvex optimization problem, one for which the objective may not even be locally

Lipschitz. Designing a controller with multiple objectives amounts to nonsmooth, nonconvex constrained optimization, a difficult problem for which there are few general methods.

In this thesis, we focus on improving the current state of the art in fast and reliable algorithms with application to controller design, both for approximating robust stability measures in large-scale settings and for designing controllers in multi-objective settings. We make the following contributions:

- We analyze the GGO algorithm for approximating the H_∞ norm and demonstrate how it can sometimes critically break down in practice by failing to converge to even stationary points of the optimization problem, a behavior that had gone unnoticed until the author of this thesis discovered it. We further detail how the inherent unpredictable nature of the main computational subroutine strongly suggests that merely modifying the Newton-bisection method used as the algorithm's outer iteration is unlikely to result in a provably breakdown-free method.

- Motivated by this, we present an improved spectral value set based algorithm using a novel hybrid expansion-contraction scheme that, under no additional assumptions, guarantees convergence to a stationary point of the optimization problem without incurring breakdown. In practice, we find that our method almost always converges to local maximizers that provide good approximations to the H_∞ norm and often, to the exact value, for cases where it is tractable to check this. Furthermore, we prove that hybrid expansion-contraction must converge at least superlinearly, and observe that it often converges quadratically. In its simplest form, that is, without further optimizations, we demonstrate that our method completes a large-scale test set almost six times faster than the GGO algorithm while being up to 57 times faster for a single problem in the set.

- To address the slow convergence of the linearly converging subroutine used by both hybrid expanion-contraction and the GGO algorithm, we present several optimizations for accelerating our method and the shared subroutine, some of which could also apply to the GGO algorithm. We further improve the subroutine by extending it to efficiently handle systems where the number of inputs and outputs may be large. The resulting optimizations, when enabled simultaneously, enabled hybrid expansion-contraction to complete the large-scale test set over 26 times faster than the implementation of the GGO algorithm while being over 82 times faster on a single problem.

- We further extend the hybrid expansion-contraction algorithm to the real stability radius, a measure which is known to be more difficult to compute than the complex stability radius. We show that superlinear convergence also holds in the real case and adapt the optimizations developed for the complex rank-one case to the real rank-two case.

- For the problem of multi-objective controller design, we propose combining a Broyden-Fletcher-Goldfarb-Shanno method with sequential quadratic programming, resulting in a method we call BFGS SQP and which is applicable for the general setting of nonsmooth, nonconvex constrained optimization. Though we do not provide convergence results, we demonstrate that BFGS SQP is exceptionally fast and reliable on challenging problems arising in controller design, and generally, greatly outperforms a competing algorithm which does provide convergence results for this nonsmooth, nonconvex constrained setting.

- We propose a new visualization tool called relative minimization profiles that allow for simultaneously assessing the relative performance of algorithms with respect to three important performance characteristics, highlighting how these measures interrelate to one another and compare to the other competing algorithms on heterogenous test sets. We employ relative minimization profiles to empirically validate our proposed BFGS SQP method in terms of quality of minimization, attaining feasibility, and speed of progress compared to other available methods on test sets comprised of difficult nonsmooth, nonconvex constrained optimization problems. We also mention that relative minimization profiles may be useful in other contexts as well, such as comparing methods for unconstrained and/or convex optimization.

The thesis is organized as follows. In the first chapter we define spectral value sets and then establish their fundamental properties and their relationship to the H_∞ norm necessary to discuss the algorithms in question. In Chapter 2, we outline the algorithm proposed in [GGO13] and characterize how that algorithm may break down. Then in Chapter 3, we present our new algorithm, hybrid expansion-contraction, for approximating the H_∞ norm, along with its convergence results. We further present a new fast upper bound procedure to initialize the method and discuss a key optimization of "contracting early" in order to increase efficiency. In Chapter 4, we present several optimizations to improve the efficiency of the aforementioned slowly converging subroutine and to extend it to handle systems with many inputs and outputs. Chapter 5 discusses the details of implementing hybrid expansion-contraction and gives extensive numerical experiments comparing our method to the GGO algorithm as well as measuring the effects of the numerous optimizations we have developed here. In Chapter 6, we extend the hybrid expansion-contraction algorithm and optimizations developed for the complex case to the real stability radius and prove analogous convergence results. Finally, in Chapter 7, we switch gears and present BFGS SQP as an efficient and robust algorithm for nonsmooth, nonconvex optimization and propose relative minimization profiles as a powerful visualization tool for comparing algorithms.

Remark 0.1. *In this thesis, we use the following notation. The set of nonnegative real numbers is denoted by $\mathbb{R}^+$, the set of strictly positive real numbers is denoted by $\mathbb{R}^{++}$, and $\sigma(\cdot)$ is the spectrum of a matrix. In Chapters 1 through 5, $\|\cdot\|$ specifies the spectral norm, while in Chapter 6, the spectral and Frobenius are specifically notated via $\|\cdot\|_2$ and $\|\cdot\|_\mathrm{F}$ respectively when necessary and by $\|\cdot\|$ when the norm can be either. In Chapter 7, all norms are notated specifically and $\|\cdot\|$ is not used.*

Remark 0.2. *We make several notational changes from that used in* [GGO13]. *We denote the perturbation matrix as Δ instead of E to avoid confusion and allow future notational compatibility with the E matrices used in related descriptor systems [Dai89], which use E to represent a possibly singular matrix on the left hand side of* (1.1) *and* (1.2)*. As a consequence, the Δ and $\widetilde{\Delta}$ matrices appearing in [GGO13] are now denoted here as Φ_p and Φ_m. We re-appropriate $\widetilde{\Delta}$ to denote the scaled versions of the rank-1 perturbations Δ, which have D appearing in the denominator, instead of the previously used F, to make the similarity of Δ and $\widetilde{\Delta}$ more obvious. Also, we change the indexing relation between perturbation Δ_k and any associated eigenvalue and eigenvectors such that $\lambda_k \in \sigma(M(\Delta_k))$ with right and left eigenvectors x_k and y_k, which we believe provides a clearer algorithmic presentation than the mixed indexing previously used. Finally, following* [HP05, Section 5.1]*, we use w and z in* (1.1) *and* (1.2) *to denote the disturbance feedback and output respectively, in lieu of u and y, to help eliminate* *possible confusion with the u vectors appearing in the rank-1 perturbations constructed by the algorithms discussed in this paper and their associated eigenvectors y. However, we continue to use x for both the state space vector in* (1.1) *and* (1.2) *as well as the eigenvectors x that appear in the algorithms, due to lack of a satisfactory alternative.*

1
THE H_∞ NORM AND SPECTRAL VALUE SETS

1.1 Linear dynamical systems with input and output

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

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

and the discrete-time analogue

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

where $A \in \mathbb{C}^{n,n}$, $B \in \mathbb{C}^{n,p}$, $C \in \mathbb{C}^{m,n}$, and $D \in \mathbb{C}^{m,p}$ and w is a disturbance feedback depending linearly on the output z [HP05, p.538].

Before formally defining the H_∞ norm for systems (1.1) and (1.2), we first consider their related spectral value sets.

1.2 Spectral value sets

This section follows the development in [HP05, Section 5.1] and [GGO13, Section 2]; we only provide the proof for a new fundamental result (Lemma 1.15) presented here which doesn't appear in either of these references.

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

$$M(\Delta) = A + B\Delta(I - D\Delta)^{-1}C \quad \text{for } \Delta \in \mathbb{C}^{p,m}, \tag{1.3}$$

assuming $I - D\Delta$ is invertible, and the associated *transfer matrix* [HP05, p.549]

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

Definition 1.1. *Let $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon\|D\| < 1$, and define the* spectral value set

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

Remark 1.2. *The sets σ_ε are called spectral value sets in* [HP05, Kar03] *and are also sometimes known as structured pseudospectra. In the special case $B = I$, $C = I$, $D = 0$, the sets σ_ε are called pseudospectra* [TE05]. *In contrast to the references just mentioned, our use of the non-strict inequalities above implies that the set $\sigma_\varepsilon(A, B, C, D)$ is compact for fixed ε.*

Corollary 1.3. *Let $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon\|D\| < 1$. Then*

$$\begin{aligned} \sigma_\varepsilon(A, B, C, D)\backslash\sigma(A) &\equiv \bigcup \{\sigma(M(\Delta)) : \Delta \in \mathbb{C}^{p,m}, \|\Delta\| \le \varepsilon, \operatorname{rank}(\Delta) = 1\} \quad (1.4) \\ &\equiv \bigcup \left\{\lambda \in \mathbb{C}\backslash\sigma(A) : \|G(\lambda)\| \ge \varepsilon^{-1}\right\}. \end{aligned}$$

Remark 1.4. *The corollary is a restatement of* [GGO13, Corollary 2.4] *which is an immediate consequence of* [GGO13, Theorem 2.1 and Remark 2.2].

[1] To motivate this formula, write $w = \Delta z$ and observe that it then follows from (1.1) that $\dot{x} = M(\Delta)x$ or from (1.2) that $x_{k+1} = M(\Delta)x_k$ [HP05, p.538].

Definition 1.5. *The* spectral abscissa *of the matrix A is*

$$\alpha(A) = \max\{\mathrm{Re}\,(\lambda) : \lambda \in \sigma(A)\}$$

with A Hurwitz stable *if $\alpha(A) < 0$. For $\varepsilon \in \mathbb{R}^+$ with $\varepsilon\|D\| < 1$, the* spectral value set abscissa *is*

$$\begin{aligned} \alpha_\varepsilon(A,B,C,D) &\coloneqq \max\{\mathrm{Re}\,(\lambda) : \lambda \in \sigma_\varepsilon(A,B,C,D)\} && (1.5) \\ &\equiv \max\left\{\mathrm{Re}\,(\lambda) : \lambda \in \sigma(A) \text{ or } \|G(\lambda)\| \geq \varepsilon^{-1}\right\}, && (1.6) \end{aligned}$$

with the equivalence following by Corollary 1.3 and $\alpha_0(A,B,C,D) = \alpha(A)$.

Definition 1.6. *The* spectral radius *of the matrix A is*

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

with A Schur stable *if $\rho(A) < 1$. For $\varepsilon \in \mathbb{R}^+$ with $\varepsilon\|D\| < 1$, the* spectral value set radius *is*

$$\begin{aligned} \rho_\varepsilon(A,B,C,D) &\coloneqq \max\{|\lambda| : \lambda \in \sigma_\varepsilon(A,B,C,D)\} && (1.7) \\ &\equiv \max\left\{|\lambda| : \lambda \in \sigma(A) \text{ or } \|G(\lambda)\| \geq \varepsilon^{-1}\right\}, && (1.8) \end{aligned}$$

with the equivalence following by Corollary 1.3 and $\rho_0(A,B,C,D) = \rho(A)$.

1.3 The H_∞ norm

We now formally define the H_∞ norm in terms of spectral value sets which states that the H_∞ norm is the reciprocal of the largest ε such that $\sigma_\varepsilon(A, B, C, D)$ is contained in the left half-plane for the continuous-time case or is contained in the unit circle around the origin for the discrete-time case.

Definition 1.7. *The* H_∞ norm of the transfer matrix function G *for continuous and discrete-time systems respectively is:*

$$\|G\|_\infty^c = \left\{ \inf_{\varepsilon\|D\|<1} \{\varepsilon : \alpha_\varepsilon(A, B, C, D) \geq 0\} \right\}^{-1}, \tag{1.9}$$

$$\|G\|_\infty^d = \left\{ \inf_{\varepsilon\|D\|<1} \{\varepsilon : \rho_\varepsilon(A, B, C, D) \geq 1\} \right\}^{-1} \tag{1.10}$$

where by convention $\|G\|_\infty^c = \|D\|^{-1}$ *if the set is empty and similarly for* $\|G\|_\infty^d$.

The following lemma states an equivalent definition of the H_∞ norm, which is actually the standard one:

Lemma 1.8.

$$\|G\|_\infty^c = \begin{cases} \infty & \text{if } \alpha(A) \geq 0 \\ \sup_{\omega \in \mathbb{R}} \|G(\mathbf{i}\omega)\| & \text{otherwise} \end{cases}, \tag{1.11}$$

$$\|G\|_\infty^d = \begin{cases} \infty & \text{if } \rho(A) \geq 1 \\ \sup_{\theta \in [0, 2\pi]} \|G(\mathrm{e}^{\mathbf{i}\theta})\| & \text{otherwise} \end{cases}. \tag{1.12}$$

Remark 1.9. *We use* $\|G\|_\infty$ *without a superscript to generically refer to the* H_∞ *norm for continuous or discrete-time systems.*

Remark 1.10. *The reciprocal of the H_∞ norm, which we denote by $\varepsilon_\star := \|G\|_\infty^{-1}$, is called the* complex stability radius [HP05, Section 5.3] *(complex because complex perturbations are admitted even if the data are real, and radius in the sense of the perturbation space, not the complex plane). When $B = I, C = I$, and $D = 0$ this is also known as the* distance to instability [VL85] *for the matrix A.*

1.4 Locally rightmost and outermost points of spectral value sets

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

Definition 1.11. *A* rightmost *point of a set $S \subset \mathbb{C}$ is a point where the maximal value of the real part of the points in S is attained. A* locally rightmost *point of a set $S \subset \mathbb{C}$ is a point λ which is a rightmost point of $S \cap \mathcal{N}$ for some neighborhood $\mathcal{N}$ of λ.*

Definition 1.12. *An* outermost *point of a set $S \subset \mathbb{C}$ is a point where the maximal value of the modulus of the points in S is attained. A* locally outermost *point of a set $S \subset \mathbb{C}$ is a point λ which is an outermost point of $S \cap \mathcal{N}$ for some neighborhood $\mathcal{N}$ of λ.*

Remark 1.13. *Since $\sigma_\varepsilon(A, B, C, D)$ is compact, its locally rightmost or locally outermost points, that is the local maximizers of the optimization problems in* (1.5) *and* (1.7)*, lie on its boundary. There can only be a finite number of these; otherwise, the boundary would need to contain an infinite number of points with the same real part or modulus, which can be ruled out by an argument similar to* [GO11, Lemma 2.5]*, exploiting* [HP05, Lemma 5.3.30].

Assumption 1.14. *Let $\varepsilon \in \mathbb{R}^{++}$ such that $\varepsilon\|D\| < 1$, and let $\lambda \notin \sigma(A)$ be a locally rightmost or locally outermost point of $\sigma_\varepsilon(A, B, C, D)$. Then:*

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

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

Lemma 1.15. *Under Assumption 1.14, a necessary condition for $\lambda \notin \sigma(A)$ to be a local maximizer of the optimization problem in* (1.6) *is*

$$\|G(\lambda)\| = \varepsilon^{-1} \quad \text{and} \quad v^* C \left(\lambda I - A\right)^{-2} Bu \in \mathbb{R}^{++}, \tag{1.13}$$

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

Remark 1.16. *Note that right and left singular vectors u and v in Lemma 1.15 are reversed from standard SVD notation to maintain notational consistency with* [GO11, GGO13].

Lemma 1.17. *If $\lambda \in \sigma_\varepsilon(A, B, C, D) \backslash \sigma(A)$ satisfies the first-order necessary conditions of Lemma 1.15 and $\operatorname{Re}(\lambda) = 0$, then $\operatorname{Im}(\lambda)$ is a stationary point of $\|G(\mathbf{i}\omega)\|$ and furthermore for $\lambda \in \sigma_\varepsilon(A, B, C, D) \backslash \sigma(A)$ with $\operatorname{Re}(\lambda) = 0$, if λ is a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$, then $\operatorname{Im}(\lambda)$ is a local maximizer of $\|G(\mathbf{i}\omega)\|$.*

Proof. The standard first-order necessary condition for $\hat{\zeta} \in \mathbb{R}^2$ to be a local maximizer of an optimization problem $\max\{f(\zeta) : g(\zeta) \leq 0,\ \zeta \in \mathbb{R}^2\}$, when f, g are continuously differentiable and $g(\hat{\zeta}) = 0$, $\nabla g(\hat{\zeta}) \neq 0$, is the existence of a Lagrange multiplier $\mu \geq 0$ such that $\nabla f(\hat{\zeta}) = \mu \nabla g(\hat{\zeta})$. Identifying $\lambda \in \mathbb{C}$ with $\zeta = [\zeta_1, \zeta_2] = [\operatorname{Re}(\lambda), \operatorname{Im}(\lambda)] \in$

$\mathbb{R}^2$, we equivalently write the optimization problem in (1.6) as objective and constraint

$$\begin{aligned} f(\zeta) &= \zeta_1, \\ g(\zeta) &= \frac{1}{\varepsilon} - \|C((\zeta_1 + \mathbf{i}\zeta_2)I - A)^{-1}B + D\|. \end{aligned}$$

The constraint $g(\zeta)$ is differentiable with respect to ζ by Assumption 1.14 and using standard perturbation theory for singular values [HJ90, Theorem 7.3.7], [GO11, Lemma 2.3], we thus have that $\nabla f(\hat{\zeta}) = \mu \nabla g(\hat{\zeta})$ for (1.6) is equivalent to

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mu \begin{bmatrix} \operatorname{Re}(v^* C((\hat{\zeta}_1 + \mathbf{i}\hat{\zeta}_2)I - A)^{-2}Bu) \\ \operatorname{Im}(v^* C((\hat{\zeta}_1 + \mathbf{i}\hat{\zeta}_2)I - A)^{-2}Bu) \end{bmatrix}, \tag{1.14}$$

where u and v are respectively the right and left singular vectors corresponding to the singular value $\varepsilon^{-1} = \|G(\hat{\zeta}_1 + \mathbf{i}\hat{\zeta}_2)\|$. By Lemma 1.15, it is clear that

$$\operatorname{Im}(v^* C((\hat{\zeta}_1 + \mathbf{i}\hat{\zeta}_2)I - A)^{-2}Bu) = 0,$$

and since by assumption $\lambda = [0, \hat{\zeta}_2]$ must satisfy (1.14), we have that

$$\frac{d}{d\zeta_2} g(\zeta)\bigg|_{\zeta = [0, \hat{\zeta}_2]} = -\frac{d}{d\omega}\|G(\mathbf{i}\omega)\|\bigg|_{\omega = \hat{\zeta}_2} = 0.$$

Thus, $\hat{\zeta}_2 = \operatorname{Im}(\lambda)$ is a stationary point of $\|G(\mathbf{i}\omega)\|$.

For the second part of the Lemma, let $\lambda \in \sigma_\varepsilon(A, B, C, D) \backslash \sigma(A)$ be a locally rightmost point with $\operatorname{Re}(\lambda) = 0$ and let $\lambda_I := \operatorname{Im}(\lambda)$. Suppose λ_I is not a local maximizer of $\|G(\mathbf{i}\omega)\|$, that is there exists $\delta \in \mathbb{R}^{++}$ such that $\|G(\mathbf{i}\lambda_I)\| < \|G(\mathbf{i}(\lambda_I + t\delta))\|$ for all $t \in [0, 1]$. By Corollary 1.3, all points $\mathbf{i}(\lambda_I + t\delta) \in \sigma_\varepsilon(A, B, C, D)$ and since $\lambda = \mathbf{i}y$ is a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ by assumption, for some sufficiently small value $\hat{t} \in \mathbb{R}^{++}$ we have that the points $\mathbf{i}(\lambda_I + t\delta)$ for $0 \leq t < \hat{t}$ must also be locally rightmost. Thus, $\sigma_\varepsilon(A, B, C, D)$ must have an infinite number of locally rightmost points, contradicting Remark 1.13. $\square$

Remark 1.18. *We note that local maximizers of $\|G(\mathbf{i}\omega)\|$ can be associated with either locally rightmost or locally leftmost points of $\sigma_\varepsilon(A, B, C, D)$ that lie on the imaginary axis. However, even if A is Hurwitz stable, it is still not clear whether $\sigma_\varepsilon(A, B, C, D)$ can have locally leftmost points that lie on the imaginary axis. If not, then the second part of Lemma 1.17 would become an if and only if condition with the addition of a simple argument in the proof. However, as this property is not germane to the discussion of the algorithms presented in this thesis, we do not explore the matter further.*

Lemma 1.19. *Under Assumption 1.14, a necessary condition for $\lambda \notin \sigma(A)$ to be a local maximizer of the optimization problem in* (1.8) *is*

$$\|G(\lambda)\| = \varepsilon^{-1} \quad \text{and} \quad \lambda \left(v^* C \left(\lambda I - A\right)^{-2} B u\right) \in \mathbb{R}^{++}, \tag{1.15}$$

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

Lemma 1.20. *If $\lambda \in \sigma_\varepsilon(A, B, C, D) \backslash \sigma(A)$ satisfies the first-order necessary conditions of Lemma 1.19 and $|\lambda| = 1$, then $\angle\lambda$ is a stationary point of $\|G(\mathrm{e}^{\mathbf{i}\theta})\|$ and furthermore for $\lambda \in \sigma_\varepsilon(A, B, C, D) \backslash \sigma(A)$ with $|\lambda| = 1$, if λ is a locally outermost point of $\sigma_\varepsilon(A, B, C, D)$, then $\angle\lambda$ is a local maximizer of $\|G(\mathrm{e}^{\mathbf{i}\theta})\|$.*

Proof. The proof follows analogously to the proof of Lemma 1.17 except with $\zeta = [\zeta_1, \zeta_2] = [|\lambda|, \angle\lambda] \in \mathbb{R}^2$. $\square$

Remark 1.21. *For any λ satisfying the assumption of either Lemma 1.17 or Lemma 1.20, we denote* $\tilde{\varepsilon}_\star := \|G(\lambda)\|^{-1} \geq \varepsilon_\star = \|G\|_\infty^{-1}$.

2 An idealized large-scale algorithm for approximating the H_∞ norm

2.1 The algorithm by Guglielmi, Gürbüzbalaban, and Overton

Consider the continuous-time case and assume that A is Hurwitz stable, so that the H_∞ norm is finite. We start by observing that the spectral value set abscissa $\alpha_\varepsilon(A, B, C, D)$ is a monotonically increasing function of ε, and we may thus compute the H_∞ norm of the transfer function (1.9) by finding the single root of the following equation:

$$g_\alpha(\varepsilon) \coloneqq \alpha_\varepsilon(A, B, C, D). \tag{2.1}$$

For the discrete-time case and assuming that A is Schur stable, we similarly observe that $\rho_\varepsilon(A, B, C, D)$ is also a monotonically increasing function of ε, and we may thus correspondingly compute the H_∞ norm of the transfer function (1.10) by finding the single root of

$$g_\rho(\varepsilon) \coloneqq \rho_\varepsilon(A, B, C, D) - 1. \tag{2.2}$$

For convenience, we define the function

$$g_{\alpha\rho}(\varepsilon) \coloneqq \begin{cases} g_\alpha(\varepsilon) & \text{if } (A, B, C, D) \text{ is a continuous-time system} \\ g_\rho(\varepsilon) & \text{if } (A, B, C, D) \text{ is a discrete-time system} \end{cases} \tag{2.3}$$

and function $f_{\alpha\rho} : \mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$:

$$f_{\alpha\rho}(\varepsilon) \coloneqq \left(g_{\alpha\rho}(\varepsilon), g'_{\alpha\rho}(\varepsilon)\right). \tag{2.4}$$

In order to find the root $\varepsilon_\star$ of (2.3), the algorithm by Guglielmi, Gürbüzbalaban, and Overton [GGO13], henceforth called the GGO algorithm, attempts to first bracket and then converge to $\varepsilon_\star$ using a Newton-based scheme, thus computing $H_\infty = \varepsilon_\star^{-1}$. However, since a pure Newton method may only converge slowly or even fail to converge at all if it is not initialized sufficient close to $\varepsilon_\star$ and furthermore cannot handle encountering points of (2.3) where its derivative is not defined, the GGO algorithm instead employs a hybrid Newton-bisection iteration, which we've defined as a generic interface as Procedure 1. By providing (2.4) as input to Procedure 1 along with proper bounds, that is $\varepsilon^{\mathrm{lb}} < \varepsilon_\star < \varepsilon^{\mathrm{ub}}$, convergence to $\varepsilon_\star$ should be guaranteed.

```
Procedure 1 $[x_\star, \ldots]$ = newton_bisection($f(\cdot), x^{\mathrm{lb}}, x^{\mathrm{ub}}$)
Input:
    Function $f : \mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$ providing $g(x)$ and $g'(x)$ ($g(x)$ continuous & differentiable)
    Lower and upper bounds $x^{\mathrm{lb}}$ and $x^{\mathrm{ub}}$
Output:
    $x_\star$ such that $g(x_\star) = 0$ and $x^{\mathrm{lb}} \leq x_\star \leq x^{\mathrm{ub}}$
    May return optional output arguments related to root $x_\star$ and function $g$

    Find a root of $g(\cdot)$ between $x^{\mathrm{lb}}$ and $x^{\mathrm{lb}}$ using Newton and bisection steps.
```

In order to provide a characterization of the derivatives of (2.1) and (2.2) necessary to employ a Newton-based scheme for finding their roots, it will be useful to first have the following definitions pertaining to eigenvalues and their associated left and right eigenvectors.

Definition 2.1. *A pair of complex vectors $x, y \in \mathbb{C}^n$ is called* RP(z)-compatible *if $\|x\| = \|y\| = 1$ and y^*x is a positive real multiple of $z \in \mathbb{C}$. When the argument z is omitted, it is understood to be 1 and y^*x is simply real and positive.*

Definition 2.2. *A complex number λ and vectors $x, y \in \mathbb{C}^n$ comprise an* RP(z)-compatible eigentriple (λ, x, y) *if x and y are RP(z)-compatible right and left eigenvectors respectively for a simple eigenvalue λ of some matrix A. When the argument z is omitted, it is understood to be 1 so that y^*x is simply real and positive.*

For some $\hat{\varepsilon} \in \mathbb{R}^+$ with $\hat{\varepsilon}\|D\| < 1$, consider a point $\lambda \in \sigma_{\hat{\varepsilon}}(A, B, C, D)$ with corresponding perturbation vectors u and v, that is $\lambda \in \sigma(M(\hat{\varepsilon}uv^*))$. In [GGO13, Section 4], the authors show that if λ is the globally rightmost point of $\sigma_{\hat{\varepsilon}}(A, B, C, D)$ and (λ, x, y) is an RP-compatible eigentriple or alternatively, if λ is the globally outermost point of $\sigma_{\hat{\varepsilon}}(A, B, C, D)$ and (λ, x, y) is an RP($\overline{\lambda}$)-compatible eigentriple, then respectively

$$g'_\alpha(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{1}{\overline{\beta}\gamma y^*x} \quad \text{and} \quad g'_\rho(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{1}{\overline{\beta}\gamma |y^*x|} \tag{2.5}$$

where via [GGO13, Equations (3.9) and (3.12)]

$$\beta = \frac{1 - \hat{\varepsilon}u^*D^*v}{u^*B^*y} \quad \text{and} \quad \gamma = \frac{1 - \hat{\varepsilon}v^*Du}{v^*Cx} \tag{2.6}$$

though we omit the full set of conditions under which these equations hold for brevity.[1]

Remark 2.3. *Although not explicitly stated in* [GGO13], *if λ is only a* locally *rightmost or outermost point of $\sigma_{\hat{\varepsilon}}(A, B, C, D)$, then the right-hand sides of the two equations in*

[1]Actually, in [GGO13], β appears unconjugated, but this is because it is assumed that u and v have been normalized so that $\beta, \gamma \in \mathbb{R}^{++}$; see [GGO13, p.721-2]. We also note that in the proof of Theorem 3.2 and the statement of Corollary 4.2 of [GGO13], the scalings of x and y are reversed: x should be scaled by γ or $1/\gamma$ respectively, and y by β or $1/\beta$ respectively.

(2.5) *still provide the derivatives of* $\operatorname{Re}(\lambda'(\varepsilon))$ *and* $|\lambda'(\varepsilon)|$ *respectively at* $\varepsilon = \hat{\varepsilon}$.

Unfortunately, like the standard BBBS algorithm for computing the H_∞ norm, computing either $\alpha_\varepsilon(A, B, C, D)$ or $\rho_\varepsilon(A, B, C, D)$ also requires a cubic order of operations and thus the benefit of this alternative formulation of calculating the H_∞ norm is not immediately apparent. In order to scale to large-dimensional problems, the GGO algorithm makes use of a subroutine called SVSAR (Spectral Value Set Abscissa or Radius) with desirable scaling properties that instead approximates (2.4) by generalizing a recent method of [GO11] for approximating the pseudospectral abscissa and radius of a matrix. The benefit is that the GGO algorithm can forgo any need to directly work with the norm of the transfer function, which involves the expensive calculation of $(\lambda_k I - A)^{-1}B$ at any iterate λ_k, and instead only requires the computation of a rightmost (or outermost) eigenvalue of $M(\Delta_k)$ for a sequence of matrices $\{\Delta_k\}$ for which sparse eigenvalue solvers based on matrix-vector products such as [LS96] may be used.

As the subroutine SVSAR only provides approximations to the spectral value set abscissa and radius, the GGO algorithm cannot guarantee that Procedure 1 will converge to $\varepsilon_\star$. Yet in practice, SVSAR does in fact often return the true values of $\alpha_\varepsilon(A, B, C, D)$ and $\rho_\varepsilon(A, B, C, D)$ or at least values close to them, and hence a hybrid Newton-bisection outer iteration seems to be an effective strategy to approximating $\varepsilon_\star$. However, as we will describe in Section 2.3, the GGO method can sometimes critically break down.

2.2 Approximating the spectral value set abscissa and radius

We now describe the fast subroutine SVSAR developed and used by the GGO algorithm in detail, both because our improved algorithm will also make use of SVSAR, along with some of its components, but also to take the opportunity to present a corrected version of one of these SVSAR components that addresses a small corner-case bug in the original description in [GGO13]. We will generally refer to the abscissa case but will make notes of the necessary changes to adapt the method to instead approximate the spectral value set radius.

Remark 2.4. *By* (1.4)*,* $\sigma_\varepsilon(A,B,C,D)\backslash\sigma(A)$ *may be characterized solely by rank-1 perturbations and thus for* $\Delta = \varepsilon uv^*$*, by* [GGO13, Lemma 2.8]

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

and $\varepsilon \in \mathbb{R}^{++}$ *with* $\varepsilon\|D\| < 1$ *and vectors* $u \in \mathbb{C}^p$*,* $v \in \mathbb{C}^m$ *are normalized such that* $\|u\| = \|v\| = 1$.

For some fixed value $\varepsilon \in \mathbb{R}^{++}$ and vectors $u_{k-1} \in \mathbb{C}^p$ and $v_{k-1} \in \mathbb{C}^m$ of unit norm, consider the matrix $\Delta_{k-1} = \varepsilon u_{k-1}v_{k-1}^*$ and the corresponding rank-1 structured perturbation of A

$$M(\Delta_{k-1}) = A + B\widetilde{\Delta}_{k-1}C \quad \text{where} \quad \widetilde{\Delta}_{k-1} = \frac{\varepsilon u_{k-1}v_{k-1}^*}{1-\varepsilon v_{k-1}^* Du_{k-1}}.$$

The core of the SVSAR procedure is how to choose the next rank-1 matrix Δ_k such that a chosen eigenvalue $\lambda_{k-1} \in \sigma(M(\Delta_{k-1}))$, say the rightmost (or outermost), may be "pushed" farther rightward (or outward) to some $\lambda_k \in \sigma_\varepsilon(A,B,C,D)$. To that end, we consider the matrix-valued linear function

$$K(t) = A + B\widetilde{\Delta}_{k-1}C + tB\left(\widetilde{\Delta}_k - \widetilde{\Delta}_{k-1}\right)C$$

with $t \in \mathbb{R}$, noting that $K(0) = A + B\widetilde{\Delta}_{k-1}C$ and $K(1) = A + B\widetilde{\Delta}_k C$. Let $(\lambda_{k-1}, x_{k-1},$

y_{k-1}) be an RP-compatible eigentriple of $K(0) = A + B\widetilde{\Delta}_{k-1}C$ and define $\lambda_{k-1}(t)$ to be the eigenvalue of $K(t)$ that converges to λ_{k-1} as $t \to 0$. Using standard first-order perturbation theory for eigenvalues [HJ90, Theorem 6.3.12], [GO11, Lemma 2.1], we have

$$\begin{aligned} \lambda'_{k-1}(0) := \left.\frac{d\lambda_{k-1}(t)}{dt}\right|_{t=0} &= \frac{y_{k-1}^* B \left(\widetilde{\Delta}_k - \widetilde{\Delta}_{k-1}\right) C x_{k-1}}{y_{k-1}^* x_{k-1}} \\ &= \varepsilon \frac{y_{k-1}^* B \left(\frac{u_k v_k^*}{1-\varepsilon v_k^* D u_k}\right) C x_{k-1}}{y_{k-1}^* x_{k-1}} - \varepsilon \frac{y_{k-1}^* B \left(\frac{u_{k-1} v_{k-1}^*}{1-\varepsilon v_{k-1}^* D u_{k-1}}\right) C x_{k-1}}{y_{k-1}^* x_{k-1}}. \end{aligned} \tag{2.7}$$

If (A, B, C, D) is a continuous-time system, then it will be desirable to choose vectors u_k and v_k such that $\operatorname{Re}(\lambda'_{k-1}(0))$ is maximized to ensure that the next perturbation Δ_k moves λ_{k-1} towards the right to some λ_k. Noting that the second term on the right-hand side of (2.7) is fixed and $y_{k-1}^* x_{k-1} > 0$ by RP-compatibility, the next u_k and v_k are chosen by explicitly solving the following optimization problem:

$$\max_{\substack{u_k \in \mathbb{C}^p, \|u_k\|=1 \\ v_k \in \mathbb{C}^m, \|v_k\|=1}} \operatorname{Re}\left(y_{k-1}^* B \left(\frac{u_k v_k^*}{1 - \varepsilon v_k^* D u_k} \right) C x_{k-1} \right). \tag{2.8}$$

In the special case when $D = 0$, it is clear that the maximal solution is attained by setting $u_k := B^* y_{k-1}/\|B^* y_{k-1}\|$ and $v_k := C^* x_{k-1}/\|C^* x_{k-1}\|$. For the explicit solutions for the $D \neq 0$ case as well as the discrete-time analogue where we wish to maximize $|\lambda'_{k-1}(0)|$, we refer to the pseudocode in Procedure 2 and [GGO13] for justification. However, just choosing u_k and v_k such that $\operatorname{Re}(\lambda'_{k-1}(0)) > 0$ is not sufficient to guarantee that the $\operatorname{Re}(\lambda_k) > \operatorname{Re}(\lambda_{k-1})$.

Instead, consider the continuous matrix family

$$N(t) = A + B\widetilde{\Delta}(t)C \quad \text{where} \quad \widetilde{\Delta}(t) = \frac{\varepsilon u(t)v(t)^*}{1 - \varepsilon v(t)^* D u(t)} \tag{2.9}$$

with

$$u(t) = \frac{tu_k + (1-t)u_{k-1}}{\|tu_k + (1-t)u_{k-1}\|} \quad \text{and} \quad v(t) = \frac{tv_k + (1-t)v_{k-1}}{\|tv_k + (1-t)v_{k-1}\|}, \tag{2.10}$$

where u_k and u_{k-1} defining $u(t)$ are fixed and v_k and v_{k-1} defining $v(t)$ are fixed. In the case that $\operatorname{Re}(\lambda_k) < \operatorname{Re}(\lambda_{k-1})$, where λ_k is a rightmost eigenvalue of $N(1) \equiv K(1)$, and λ_{k-1} is a rightmost eigenvalue of $N(0) \equiv K(0)$, we can instead choose $t \in (0,1)$ such that the rightmost eigenvalue of $N(t)$ has real part greater than $\operatorname{Re}(\lambda_{k-1})$. Now let $\lambda_{k-1}(t)$ denote the eigenvalue of $N(t)$ converging to λ_{k-1} as $t \to 0$. Again from standard eigenvalue perturbation theory we have

$$\lambda'_{k-1}(0) := \left.\frac{d\lambda_{k-1}(t)}{dt}\right|_{t=0} = \frac{y_{k-1}^* N'(0) x_{k-1}}{y_{k-1}^* x_{k-1}} = \frac{\psi_k}{y_{k-1}^* x_{k-1}} \tag{2.11}$$

We refer to [GGO13] for an elaboration on the derivation of ψ_k, and instead present an equivalent more compact formulation:

$$\begin{aligned} \psi_k = {} & \frac{\varepsilon y_{k-1}^* B}{1 - \varepsilon v_{k-1}^* D u_{k-1}} \Big((v_{k-1}^* C x_{k-1})(u_k - \eta u_{k-1}) + (v_k^* C x_{k-1}) u_{k-1} \Big) \\ & + \frac{\varepsilon^2 (y_{k-1}^* B u_{k-1})(v_{k-1}^* C x_{k-1})}{(1 - \varepsilon v_{k-1}^* D u_{k-1})^2} \Big(v_{k-1}^* D(u_k - \eta u_{k-1}) + v_k^* D u_{k-1} \Big), \end{aligned} \tag{2.12}$$

where $\eta = \operatorname{Re}(u_k^* u_{k-1}) + \operatorname{Re}(v_k^* v_{k-1})$. Again recalling that $y_{k-1}^* x_{k-1} > 0$ by RP-compatibility, we see by (2.11) that $\operatorname{Re}(\lambda'_{k-1}(0)) > 0$ if and only if $\psi_k > 0$. Thus, if $\operatorname{Re}(\psi_k) < 0$, we may simply change the signs of both u_k and v_k to ensure that $\operatorname{Re}(\psi_k) > 0$ instead holds.

Procedure 2 $[u_k, v_k]$ = **compute_next_uv**($\varepsilon, u_{k-1}, v_{k-1}, \lambda_{k-1}, x_{k-1}, y_{k-1}$)

```
Input:
  Perturbation level $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$
  Vectors $u_{k-1} \in \mathbb{C}^p$ and $v_{k-1} \in \mathbb{C}^m$ defining $\Delta_{k-1}$ with $\|u_{k-1}\| = \|v_{k-1}\| = 1$
  RP-compatible eigentriple $(\lambda_{k-1}, x_{k-1}, y_{k-1})$ of $A + B\widetilde{\Delta}_{k-1}C$
  Constants: Matrices $B \in \mathbb{C}^{n,p}, C \in \mathbb{C}^{m,p}$, and $D \in \mathbb{C}^{p,m}$
Output:
  Vectors $u_k, v_k$ such that $\operatorname{Re}(\lambda'_{k-1}(0)) > 0$ and $\|u_k\| = \|v_k\| = 1$

  Solve $b = (I - \varepsilon^2 D^* D)^{-1} B^* y_{k-1}$ and $c = (I - \varepsilon^2 D D^*)^{-1} C x_{k-1}$
  Set $\rho = \sqrt{\frac{\|b\|^2 - \|\varepsilon D b\|^2}{\|c\|^2 - \|\varepsilon D^* c\|^2}}$ and subsequently $\beta = \frac{1}{\|b + \rho\varepsilon D^* c\|}$
  Set $u_k = \beta\,(b + \rho\varepsilon D^* c)$ and $v_k = \beta\,(\rho c + \varepsilon D b)$
  Set $\psi_k$ as in (2.12)
  if $\operatorname{Re}(\psi_k) < 0$ then
    Set $u_k = -u_k$ and $v_k = -v_k$
  end if
```

NOTE: *For the discrete-time case, Procedure 2,* compute_next_uv($\cdot$)*, will compute vectors* u_k, v_k *such that* $|\lambda'_{k-1}(0)| > 0$ *if it is provided with an input eigentriple that is RP($\overline{\lambda}$)-compatible rather than just RP-compatible. See* [GGO13] *for more details.*

Having computed u_k and v_k via Procedure 2, if $\operatorname{Re}(\lambda_k) < \operatorname{Re}(\lambda_{k-1})$, Procedure 3 can be used to perform a backtracking line search on t to find an intermediate perturbation $\widetilde{\Delta}(t)$ that produces an eigenvalue λ_k such that $\operatorname{Re}(\lambda_k) > \operatorname{Re}(\lambda_{k-1})$. We present Procedure 3 in detail because it contains a necessary modification to the simple line search originally stated in [GGO13], which was not entirely correct. The issue is that the GGO algorithm assumes that it is highly unlikely that neither

$$tu_k + (1-t)u_{k-1} \quad \text{nor} \quad tv_k + (1-t)v_{k-1} \tag{2.13}$$

will pass directly through the origin and thus (2.10) should both be well-defined. Ironically, as the GGO algorithm also assumes that $p, m \ll n$, the odds are greatly increased that one or both of (2.13) might pass through the origin and certainly this will occur if for example $p = 1$ and $u_k = -u_{k-1}$, which is exactly what has been observed in practice on occasion.

Procedure 3 $[u_t, v_t, \lambda_t, x_t, y_t]$ = **uv_line_search**($\varepsilon, u_k, v_k, u_{k-1}, v_{k-1}, \lambda_{k-1}$)

```
Input:
  Perturbation level $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$
  Vectors $u_k, u_{k-1} \in \mathbb{C}^p$ and $v_k, v_{k-1} \in \mathbb{C}^m$ defining $\Delta_k$ and $\Delta_{k-1}$ such that:
      $\|u_k\| = \|v_k\| = \|u_{k-1}\| = \|v_{k-1}\| = 1$ and
      $\psi_k$ of (2.12) is positive.
  $\lambda_{k-1}$ is a rightmost computed eigenvalue of $A + B\widetilde{\Delta}_{k-1}C$
  Constants: Matrices $B \in \mathbb{C}^{n,p}, C \in \mathbb{C}^{m,p}$, and $D \in \mathbb{C}^{p,m}$ and fixed $\delta$, $0 < \delta \ll 1$
Output:
  Vectors $u_t$, $v_t$ defining perturbation $\Delta_t$ with $\|u_t\| = \|v_t\| = 1$
  RP-compatible eigentriple $(\lambda_t, x_t, y_t)$ of $A + B\widetilde{\Delta}_tC$ such that $\mathrm{Re}\,(\lambda_t) > \mathrm{Re}\,(\lambda_{k-1})$

  Set $t = 1$
  repeat
    Set $t = \frac{1}{2}t$
    Set $u_t = tu_k + (1-t)\,u_{k-1}$ and $v_t = tv_k + (1-t)\,v_{k-1}$
    while $u_t = 0$ or $v_t = 0$ do
      Set $t = (1-\delta)\,t$
      Set $u_t = tu_k + (1-t)\,u_{k-1}$ and $v_t = tv_k + (1-t)\,v_{k-1}$
    end while
    $u_t = \frac{u_t}{\|u_t\|}$ and $v_t = \frac{v_t}{\|v_t\|}$
    $\widetilde{\Delta}_t = \frac{\varepsilon u_t v_t^*}{1-\varepsilon v_t^* D u_t}$
    Compute rightmost RP-compatible eigentriple $(\lambda_t, x_t, y_t)$ of $A + B\widetilde{\Delta}_tC$
  until $\mathrm{Re}\,(\lambda_t) > \mathrm{Re}\,(\lambda_{k-1})$
```

NOTE: *For the discrete-time case, Procedure 3,* uv_line_search($\cdot$)*, can compute* u_t *and* v_t *by changing occurrences of "rightmost" to "outermost", "RP-compatible" to "RP(*$\overline{\lambda}_t$*)-compatible" and* $\mathrm{Re}\,(\lambda_t) > \mathrm{Re}\,(\lambda_{k-1})$ *to* $|\lambda_t| > |\lambda_{k-1}|$.

The SVSAR method for approximating the spectral value set abscissa or radius, as described in pseudocode in Procedure 4, simply repeatedly calls Procedure 2 with the current u_{k-1} and v_{k-1} perturbation vectors to produce the next u_k and v_k along with its resulting rightmost eigenvalue λ_k of $A + B\widetilde{\Delta}_k C$, and only performs a line search using Procedure 3 if $\operatorname{Re}(\lambda_k) \leq \operatorname{Re}(\lambda_{k-1})$, which we note is usually not necessary. Thus, the SVSAR process continually pushes an eigenvalue rightward until it has hopefully reached a globally rightmost point of $\sigma_\varepsilon(A, B, C, D)$.

However, SVSAR is not guaranteed to converge to a globally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ and in fact, it is only known to converge to locally rightmost points if ε is sufficiently small. In practice though and for any valid value of ε, we observe that SVSAR does often converge to a globally rightmost point of $\sigma_\varepsilon(A, B, C, D)$, and in the cases it does not, it typically finds locally rightmost points which provide good approximations to the globally optimal value. If SVSAR does not converge to a locally rightmost point, then we observe that it is often that case that it instead converges to a point λ satisfying the first-order necessary conditions of Lemma 1.15, provided that $\|G(\lambda)\| = \varepsilon^{-1}$. Unfortunately, even though ε^{-1} is guaranteed to be a singular value of $G(\lambda)$ by [GGO13, Theorem 3.2], it is not necessarily the largest singular value and it is possible that SVSAR may terminate at a point λ strictly in the interior of the given spectral value set.

Assumption 2.5. *We assume that* SVSAR *at least always converges to a point λ satisfying the first-order necessary conditions of Lemma 1.15 or Lemma 1.19 corresponding to whether the spectral value set abscissa or radius is requested, respectively.*

Remark 2.6. *A justification of Assumption 2.5 is in order. The authors of* [GGO13] *conjecture that the only* attractive *fixed points for* SVSAR *correspond to points λ that*

```
**Procedure 4** $[u_k, v_k, \lambda_k, x_k, y_k]$ = **svsar**($\varepsilon, u_0, v_0, \lambda_0, x_0, y_0, k_{\max}$)
**Input:**
  Perturbation level $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$
  Vectors $u_0 \in \mathbb{C}^p$ and $v_0 \in \mathbb{C}^m$ defining $\Delta_0 = \varepsilon u_0 v_0^*$ with $\|u_0\| = \|v_0\| = 1$
  RP-compatible eigentriple $(\lambda_0, x_0, y_0)$ of $A + B\widetilde{\Delta}_0 C$
  Positive integer $k_{\max}$ specifying max number of iterations
  Constants: Matrices $A \in \mathbb{C}^{n,n}, B \in \mathbb{C}^{n,p}, C \in \mathbb{C}^{m,p}$, and $D \in \mathbb{C}^{p,m}$
**Output:**
  Vectors $u_k, v_k$ defining perturbation $\Delta_k$ with $\|u_k\| = \|v_k\| = 1$
  RP-compatible eigentriple $(\lambda_k, x_k, y_k)$ of $A + B\widetilde{\Delta}_k C$ with $\lambda_k$ satisfying Lemma 1.15
  // *Typically $\lambda_k$ will be a locally or globally rightmost point of $\sigma_\varepsilon(A, B, C, D)$*

  **for** $k = 1, 2, \ldots$ until $\mathrm{Re}\,(\lambda_k) \leq \mathrm{Re}\,(\lambda_{k-1})$ or $k > k_{\max}$ **do**
    $[u_k, v_k]$ = **compute_next_uv**($\varepsilon, u_{k-1}, v_{k-1}, \lambda_{k-1}, x_{k-1}, y_{k-1}$)
    $\widetilde{\Delta}_k = \frac{\varepsilon u_k v_k^*}{1 - \varepsilon v_k^* D u_k}$
    Compute rightmost RP-compatible eigentriple $(\lambda_k, x_k, y_k)$ of $A + B\widetilde{\Delta}_k C$
    **if** $\mathrm{Re}\,(\lambda_k) \leq \mathrm{Re}\,(\lambda_{k-1})$ **then**
      $[u_k, v_k, \lambda_k, x_k, y_k]$ = **uv_line_search**($\varepsilon, u_k, v_k, u_{k-1}, v_{k-1}, \lambda_{k-1}$)
    **end if**
  **end for**
```

NOTE: *For the discrete-time case, Procedure 4,* svsar($\cdot$)*, can instead approximate approximate the spectral value set radius by changing all occurrences of "RP-compatible" to "RP($\overline{\lambda}_k$)-compatible", "locally rightmost" to "locally outermost", "Lemma 1.15" to "Lemma 1.19", "Compute rightmost" to "Compute outermost",* $\mathrm{Re}\,(\lambda_k) \leq \mathrm{Re}\,(\lambda_{k-1})$ *to* $|\lambda_k| \leq |\lambda_{k-1}|$*, and making the corresponding changes to* uv_line_search($\cdot$) *as outlined in the note for Procedure 3.*

are locally rightmost or outermost points of $\sigma_\varepsilon(A, B, C, D)$. Consider the case that SVSAR *does not converge as intended, that is, it halts at some λ_k such that ε^{-1} is not the largest singular value of $G(\lambda_k)$ and thus λ_k is strictly in the interior of $\sigma_\varepsilon(A, B, C, D)$. It thus seems reasonable that* SVSAR *can be restarted by perturbing the final Δ_k matrix with the hope of pushing rightward or outward past the undesired interior fixed point encountered at λ_k. In practice, we have noted this is a viable strategy and furthermore,* *has been effective for restarting* SVSAR *even if it converges to a point on the boundary of the spectral value set that satisfies the first-order necessary conditions of Lemma 1.15 or Lemma 1.19 but is not actually a locally rightmost or outermost point. Note however that Assumption 2.5 does not make the stronger claim that* SVSAR *converges to a locally rightmost or outermost point.*

2.3 The breakdown case of the GGO algorithm

We have previously mentioned in Section 2.2 that SVSAR may not necessarily converge to a locally rightmost or outermost point $\lambda \in \sigma_\varepsilon(A, B, C, D)$. In these cases, we should not expect SVSAR to provide a fair approximation to the spectral value set abscissa or radius and thus, it is likely that any hybrid Newton-bisection iteration will either fail or converge to an inaccurate approximation of the H_∞ norm. However, even under the stronger assumption that SVSAR always guarantees convergence to either locally rightmost or outermost points of $\sigma_\varepsilon(A, B, C, D)$, the GGO algorithm can still sometimes critically break down. Though the circumstances leading to breakdown are briefly discussed in [GGO13], we describe the problem in significantly more detail here as it was in fact myself who discovered and analyzed the breakdown case [GGO13, Acknowledgements] and it has motivated the development of our new improved method.

Let us first examine the problematic breakdown cases of examples `CM3` and `CM4` reported in [GGO13] where notably only bisection steps were taken and every Newton step was rejected. In Figure 2.1, we plot $\|G(\mathbf{i}\omega)\|$ and the approximations produced by the GGO algorithm. Under the assumption that SVSAR always converges to locally rightmost points when the abscissa is requested, one would hope that the GGO algorithm would return local, possibly global, maximizers of the norm of the transfer func-

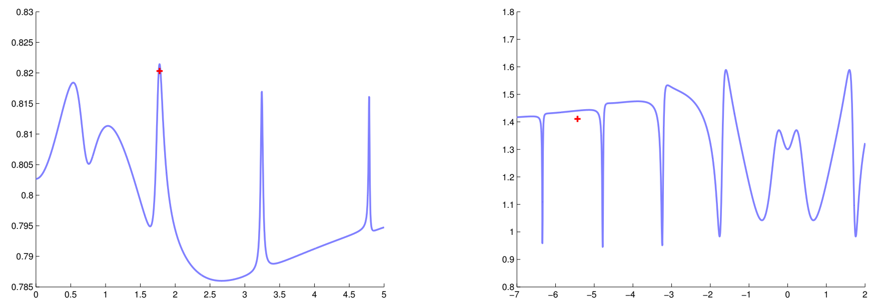

FIGURE 2.1: *Left: example* CM3. *Right: example* CM4. *Each panel plots the norm of the transfer function evaluated along the imaginary axis in blue with the resulting approximations $(\tilde{\omega}, \tilde{\varepsilon})$ returned by the* GGO *algorithm as red crosses. In* CM3*, the* GGO *algorithm appears to have returned an accurate value of $\tilde{\omega}$ to be a maximizer of $\|G(\mathbf{i}\omega)\|$ but its computed value of $\tilde{\varepsilon}$ is inaccurate as $(\tilde{\omega}, \tilde{\varepsilon}^{-1})$ does not lie on the graph of $\|G(\mathbf{i}\omega)\|$. In* CM4*, this situation is worse as it can clearly be seen that neither $\tilde{\omega}$ or $\tilde{\varepsilon}$ are accurate.*

tion evaluated along the imaginary axis. Here, we see that the approximations returned by GGO do not even lie on the graphs of the functions and while the approximation for CM3 seems to at least accurately approximate the vicinity of a local maximizer, even though it underestimates its function value, the approximation for CM4 seems particularly egregious as it clearly neither locates a maximizer nor approximates a maximal function value well.

As the spectral value sets of CM3 and CM4 have quite needle-like shapes that make visualization difficult and the boundaries are expensive to compute, we present an analogous synthetic example in Figure 2.2 where it is easier to see how the GGO algorithm breaks down, even when SVSAR always finds locally rightmost points. In the top left panel, for $\varepsilon_1 = 0.08$ we see that SVSAR initially finds a locally rightmost point λ_1 in the left half-plane and accordingly, its lower bound ε^{lb} is updated to 0.08. Recalling the definition of $\varepsilon_\star := \|G\|_\infty^{-1}$ from Remark 1.10, it can clearly be seen that $\varepsilon_\star < 0.08$. Yet, it appears that ε^{lb} is still a valid lower bound for some locally rightmost point of

$\alpha_{\tilde{\varepsilon}_\star}(A,B,C,D)$, with $\tilde{\varepsilon}_\star > \varepsilon_1$, on the imaginary axis immediately to the right of the current iterate λ_1. In the top-right panel however, with $\varepsilon_2 = 0.1$, we see that the structure of the boundary of the larger spectral value set has only two locally rightmost points instead of three as before and that the locally rightmost point λ_2 found by SVSAR is in quite a different region than λ_1. Nonetheless, since λ_2 is in the right half-plane, SVSAR updates its upper bound $\varepsilon^{\mathrm{ub}}$ to 0.1 and subsequently attempts a smaller value of ε than 0.1. However, what we observe in the bottom-left panel is that the Newton step ε^{N} which approximates $\varepsilon_\star$ is now rejected since $\varepsilon^{\mathrm{N}} \approx \varepsilon_\star < \varepsilon^{\mathrm{lb}} = 0.08$. As a result, the hybrid Newton-bisection outer iteration is forced to take a bisection step with $\varepsilon_3 = 0.09$. As we see in the bottom-right panel, this slow march of bisection steps towards $\varepsilon_1 = 0.08$ continues on every subsequent iteration until the termination tolerance is reached at a locally rightmost point strictly in the right half-plane, thus demonstrating the breakdown.

To make the discussion more concrete, for a fixed value of $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$, consider the set

$$\Lambda^{\mathrm{r}}_{\varepsilon}(A,B,C,D) \coloneqq \{\lambda : \lambda \text{ a locally rightmost point of } \sigma_\varepsilon(A,B,C,D)\}.$$

By continuity, for $\lambda \in \Lambda^{\mathrm{r}}_{\varepsilon}(A,B,C,D)$, at least generically, we can construct a continuous path $\lambda^{\mathrm{r}}(t)$ consisting only of locally rightmost points of $\sigma_t(A,B,C,D)$ for $t \in (\varepsilon-\delta,\varepsilon+\delta)$ for some sufficiently small value of $\delta \in \mathbb{R}^{++}$ and where $\lambda^{\mathrm{r}}(\varepsilon) = \lambda$. Let λ_1 be the upper rightmost point in $\Lambda^{\mathrm{r}}_{\varepsilon_1}(A,B,C,D)$ shown in the top left panel of Figure 2.2, and let λ_2 be the upper rightmost point in $\Lambda^{\mathrm{r}}_{\varepsilon_2}(A,B,C,D)$ shown in the top

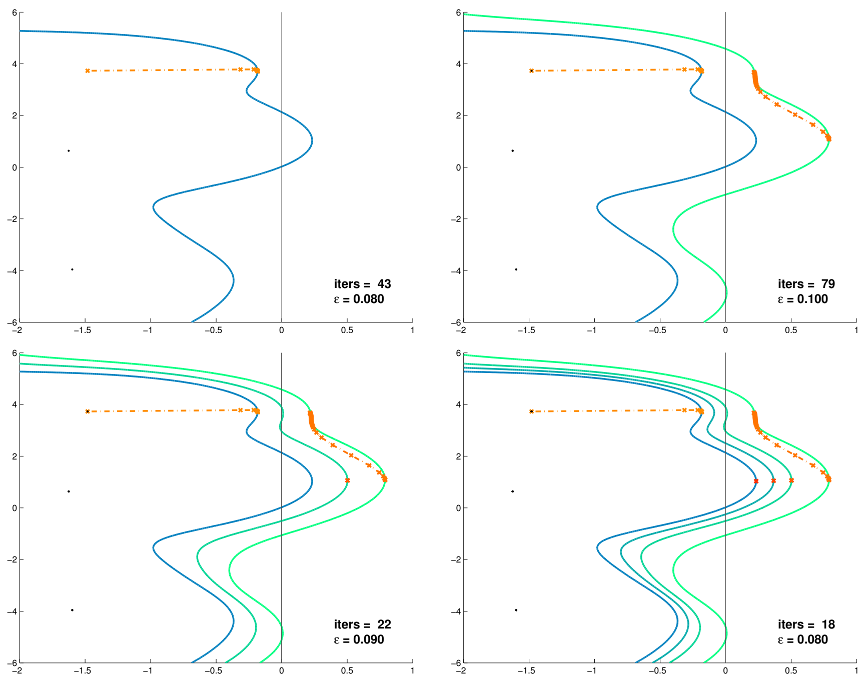

FIGURE 2.2: *Progression of breakdown of the* GGO *algorithm on a synthetic example. Iterates of* SVSAR *are in orange-red, spectral value set boundaries are in blue-green, and eigenvalues of A are black dots. Top left:* GGO *finds a locally rightmost point λ_1 in left half-plane for $\varepsilon_1 = 0.08$ and sets $\varepsilon^{\mathrm{lb}} = 0.08$. Top right:* GGO *attempts $\varepsilon_2 = 0.1$ and finds new locally rightmost point λ_2 in right half-plane and sets $\varepsilon^{\mathrm{ub}} = 0.1$. Bottom left:* GGO *rejects leftwards Newton step ε^{N} towards imaginary axis, since $\varepsilon^{\mathrm{N}} < \varepsilon^{\mathrm{lb}}$, and instead via bisection step, finds a locally rightmost point λ_3 in right half-plane and updates $\varepsilon^{\mathrm{ub}} = 0.09$. Bottom right:* GGO *continually rejects leftward Newton steps $\varepsilon_k^{\mathrm{N}}$ towards the imaginary axis, since $\varepsilon_k^{\mathrm{N}} < \varepsilon^{\mathrm{lb}}$ and erroneously $\mathrm{Re}\,(\lambda_k) \to \gamma > 0$ and $\varepsilon_k \to 0.08$ as $k \to \infty$, with slow convergence due to only bisection steps being taken, though we only depict two bisection steps and the last step for clarity. The value for ε_k and the number of* SVSAR *steps to find a locally rightmost for that level are listed in the bottom-corner of each panel.*

right panel, and consider their respective continuous paths of locally rightmost points

$$\begin{aligned}
&\lambda_1^{\mathrm{r}}(t) \quad \text{where} \quad t \in (\varepsilon_1 - \delta_1, \varepsilon_1 + \delta_1) \text{ and } \lambda_1^{\mathrm{r}}(\varepsilon_1) = \lambda_1 \\
&\lambda_2^{\mathrm{r}}(t) \quad \text{where} \quad t \in (\varepsilon_2 - \delta_2, \varepsilon_2 + \delta_2) \text{ and } \lambda_2^{\mathrm{r}}(\varepsilon_2) = \lambda_2,
\end{aligned}$$

for some fixed values $\{\delta_1, \delta_2\} \subset \mathbb{R}^{++}$. From the plots in Figure 2.2, we see that $\lambda_1^{\mathrm{r}}(t)$ does not extend to $t = 0.1$ while presumably $\lambda_2^{\mathrm{r}}(t)$ extends leftward to cross the imaginary axis. The breakdown arises because $\varepsilon^{\mathrm{lb}}$ corresponds to a lower bound for which $\lambda_1^{\mathrm{r}}(t)$ crosses the imaginary axis but all subsequent computed upper bounds actually correspond to upper bounds on where $\lambda_2^{\mathrm{r}}(t)$ crosses the imaginary axis.

In [GGO13], the authors state that the breakdown results from what they call an *"invalid update to the lower bound"* but we note that this is a bit of a misnomer. Actually, SVSAR is providing valid bounds on where specific paths of locally rightmost points, such as $\lambda_1^{\mathrm{r}}(t)$ and $\lambda_2^{\mathrm{r}}(t)$, might cross the imaginary axis. While bounds for such paths may be a good proxy for approximating (2.1), SVSAR neither gives concrete indication of which path it has bounded nor guarantees any consistency that it will bound the same path as ε_k changes. If circumstances are fortunate, then at some point, all remaining lower and upper bound updates computed in the GGO algorithm will correspond to one particular path of locally rightmost points that crosses the imaginary axis and the method will thus converge to a local maximizer of $\|G(\mathbf{i}\omega)\|$. If circumstances are not so fortunate, as in `CM3` and `CM4`, the lower and upper bound updates may at some point fail to correspond to the same path, potentially preventing the GGO algorithm from converging to a locally rightmost point on the imaginary axis. Thus, the breakdown is more aptly described as a *lower and upper bound mismatch error* and we say that bound mismatch error is *fatal* if it ultimately prevents convergence to the imaginary axis.

Remark 2.7. *Though in the examples presented here the breakdown has occurred by converging to a point in the right half-plane, we see no reason why the* GGO *algorithm cannot also break down by converging to a point in the left half-plane.*

Remark 2.8. *Note that while we have only specifically discussed bound mismatch errors for the continuous-time case, the discrete-time version of the* GGO *algorithm is similarly susceptible to such breakdowns in an analogous manner where a fatal bound mismatch error may induce the* GGO *algorithm to converge to a point strictly outside or inside the unit circle.*

The authors of [GGO13] suggest that the GGO algorithm might overcome a bound mismatch error by reconsidering a lower or upper bound, since breakdown is easily detected once the sequence of computed locally rightmost points converges to a point strictly in the left or right half-plane to some prescribed termination tolerance. However, there are a number of potential problems with this approach. The first of these is the aforementioned issue that under breakdown, the outer iteration of the GGO algorithm is essentially downgraded to a pure bisection method, and thus incurring slow convergence to detect breakdown is a doubly unappealing proposition. Unfortunately, it is not clear how to provide reliable early detection that a bound mismatch error has occurred and it is even less clear how to predict if such a bound mismatch error will be fatal or not. A heuristic approach might keep a history of the past few iterations, recording both the direction of the Newton steps and whether they were accepted. A consecutive run of rejected Newton steps all in the same direction could indicate and provide early warning that a potentially fatal bound mismatch error has occurred, allowing the algorithm to override the bisection step in favor of the Newton step, despite that it will move the next value of ε_k outside the current lower and upper bound bracket. Yet, even if this ad hoc procedure successfully prevents the bound mismatch error from being fatal, there is no guarantee the algorithm won't subsequently encounter another mismatch error. Worse still, if the lower and upper bounds are allowed to be rolled back, it seems conceivable that such an algorithm might be able to enter infinite loops.

A different approach for preventing fatal bound mismatch errors in the GGO algorithm is to instead attempt to coax the SVSAR subroutine to only return lower and upper bounds for a single path of locally rightmost points that crosses the imaginary axis. However, as we have seen from Figure 2.2, taming the unpredictable nature of where SVSAR might converge seems to hinge upon only making small changes to ε_k, which is squarely at odds with an algorithm that is designed to be fast by using Newton steps. Furthermore, one might assume that warm-starting SVSAR with the last perturbation computed for ε_{k-1} might help ensure SVSAR converges to a point on the same path of locally rightmost points for ε_k but, ironically for the synthetic example demonstrating breakdown, the induced consistency provided by warm-starting prevented SVSAR from possibly rediscovering the first path $\lambda_1^{\mathrm{r}}(t)$. If SVSAR is cold-started from the rightmost eigenvalue of A for every ε_k, then breakdown does not occur on the synthetic example. However, this is clearly not a solution and comes at a great increase in computational cost. Lastly, as the GGO algorithm must first attempt to bracket $\varepsilon_\star$ by lower and upper bounds to then be able to apply the Newton-bisection outer iteration, these initial and often quite large steps of the algorithm with respect to ε_k may return bounds for many different paths of locally rightmost points and thus a fatal bound mismatch error may be seeded at the outset of the algorithm.

3

A FAST AND BREAKDOWN-FREE ALGORITHM FOR APPROXIMATING THE H_∞ NORM

We now present our new H_∞ norm approximation algorithm that sheds the Newton-bisection outer iteration of the GGO algorithm and replaces it with a novel hybrid expansion-contraction scheme. Under the same assumptions as the GGO algorithm, we provably demonstrate that hybrid expansion-contraction converges to a stationary point of the norm of the transfer function evaluated along the imaginary axis or unit circle without incurring breakdown, and furthermore, converges to a local maximizer if the subroutine SVSAR finds locally rightmost or outermost points of spectral value sets. We also prove that in the worst case, our algorithm converges at least superlinearly, while in practice, the observed convergence rate is often quadratic or faster.

3.1 Hybrid expansion-contraction: a breakdown-free algorithm

Key Observation 3.1. *Lower bounds on $\varepsilon_\star$ reported by Procedure 4 cannot be trusted to be accurate, that is, they may only be valid locally, for some $\tilde{\varepsilon}_\star > \varepsilon_\star$ corresponding to where some continuous path of locally rightmost points of spectral value sets crosses the imaginary axis. However, upper bounds reported by Procedure 4 always bound $\varepsilon_\star$.*

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

Now consider the matrix family with respect to the single parameter ε for a fixed rank-1 perturbation uv^*:

$$M_{uv}(\varepsilon) \coloneqq A + B\widetilde{\Delta}_{uv}(\varepsilon)C \quad \text{where} \quad \widetilde{\Delta}_{uv}(\varepsilon) \coloneqq \frac{\varepsilon uv^*}{1 - \varepsilon v^* D u} \tag{3.1}$$

with $\varepsilon \in \mathbb{R}^+$, $\varepsilon\|D\| < 1$ along with a corresponding eigenvalue $\lambda_{uv}(\varepsilon)$ of $M_{uv}(\varepsilon)$.

Key Observation 3.2. *Let $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$, and fixed non-zero vectors $u \in \mathbb{C}^p$, $v \in \mathbb{C}^m$ be given. If $\operatorname{Re}(\lambda_{uv}(\varepsilon)) > 0$ and $M_{uv}(0) = A$ is Hurwitz stable, then by continuity of $\lambda_{uv}(\cdot)$ there exists $\hat{\varepsilon}$ such that $0 < \hat{\varepsilon} < \varepsilon$ and $\operatorname{Re}(\lambda_{uv}(\hat{\varepsilon})) = 0$. Similarly, if $|\lambda_{uv}(\varepsilon)| > 1$ and $M_{uv}(0)$ is Schur stable, then there exists $\hat{\varepsilon}$ such that $0 < \hat{\varepsilon} < \varepsilon$ and $|\lambda_{uv}(\hat{\varepsilon})| = 1$.*

Thus, given ε, u and v which demonstrate that $\varepsilon > \varepsilon_\star$, that is matrix $M_{uv}(\varepsilon)$ has at least one eigenvalue $\lambda_{uv}(\varepsilon)$ such that $\operatorname{Re}(\lambda_{uv}(\varepsilon)) > 0$ or $|\lambda_{uv}(\varepsilon)| > 1$ respectively, by Key Observation 3.2, it is clear that we may always contract ε to $\hat{\varepsilon}$ such that $\varepsilon_\star < \hat{\varepsilon} < \varepsilon$ by finding a root of the function

$$g^c_{uv}(\varepsilon) \coloneqq \operatorname{Re}(\lambda_{uv}(\varepsilon)) \quad \text{or} \quad g^d_{uv}(\varepsilon) \coloneqq |\lambda_{uv}(\varepsilon)| - 1. \tag{3.2}$$

For convenience, we define

$$g_{uv}(\varepsilon) \coloneqq \begin{cases} g^c_{uv}(\varepsilon) & \text{if } (A,B,C,D) \text{ is a continuous-time system} \\ g^d_{uv}(\varepsilon) & \text{if } (A,B,C,D) \text{ is a discrete-time system} \end{cases} \tag{3.3}$$

so that $\lambda_{uv}(\varepsilon)$ is on the boundary of the stability region if and only if $g_{uv}(\varepsilon) = 0$ for either the continuous or discrete-time cases depending on the context. Unlike the hybrid

Newton-bisection outer iteration in GGO where the lower bounds cannot be trusted, we have *a priori* true lower and upper bounds of 0 and ε for finding a root of (3.3) using a hybrid Newton-bisection routine. Furthermore, though the derivative of (3.3) may have points where it is nonsmooth, we typically find in practice that either this is not the case or we do not encounter them during the iteration. It is possible that $M_{uv}(\cdot)$ may have several eigenvalues in the right half-plane, or outside the unit circle for the discrete-time case, and thus (3.3) may correspondingly have several roots, but we note this is not problematic. However, we have yet to observe a case where (3.3) has more than a single root of multiplicity one (or a single conjugate pair in the real case).

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

Remark 3.4. *It is important to note that Assumption 3.3 is implicitly assumed in* [GGO13] *in order for their algorithm to work as well, so it is not an additional assumption we are making here.*

Key Observation 3.5. *Let $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$ and fixed non-zero vectors $u \in \mathbb{C}^p$, $v \in \mathbb{C}^m$ be given. Under Assumption 3.3, if $\alpha(M_{uv}(\varepsilon)) > 0$ and $M_{uv}(0) = A$ is Hurwitz stable, then a Newton-bisection method will converge to some positive $\hat{\varepsilon} < \varepsilon$ such that $\alpha(M_{uv}(\hat{\varepsilon})) = 0$. Similarly, if $\rho(M_{uv}(\varepsilon)) > 1$ and $M_{uv}(0)$ is Schur stable, then a Newton-bisection method will converge to some positive $\hat{\varepsilon} < \varepsilon$ such that $\rho(M_{uv}(\hat{\varepsilon})) = 1$.*

To obtain the derivatives of (3.2) with respect to ε for calculating the Newton step, we first find the derivative of the matrix family of (3.1) with respect to ε by the quotient rule:

$$M'_{uv}(\varepsilon) = B\Big(\frac{uv^*(1-\varepsilon v^*Du) + \varepsilon uv^*(v^*Du)}{(1-\varepsilon v^*Du)^2}\Big)C = \frac{Buv^*C}{(1-\varepsilon v^*Du)^2}. \tag{3.4}$$

For an RP-compatible eigentriple $(\lambda_{uv}(\varepsilon), x_{uv}(\varepsilon), y_{uv}(\varepsilon))$ of (3.1), again by standard first-order perturbation theory of simple eigenvalues [HJ90, Theorem 6.3.12], [GO11, Lemma 2.1], we have that

$$\lambda'_{uv}(\varepsilon) = \frac{y_{uv}(\varepsilon)^* M'_{uv}(\varepsilon) x_{uv}(\varepsilon)}{y_{uv}(\varepsilon)^* x_{uv}(\varepsilon)}, \tag{3.5}$$

and the derivatives with respect to ε of (3.2) for the continuous and discrete-time cases are:

$$g'^{c}_{uv}(\varepsilon) = \operatorname{Re}\left(\lambda'_{uv}(\varepsilon)\right) \quad \text{and} \quad g'^{d}_{uv}(\varepsilon) = \operatorname{Re}\left(\frac{\overline{\lambda_{uv}(\varepsilon)}\lambda'_{uv}(\varepsilon)}{|\lambda_{uv}(\varepsilon)|}\right). \tag{3.6}$$

We note that if $(\lambda_{uv}(\varepsilon), x_{uv}(\varepsilon), y_{uv}(\varepsilon))$ is instead an RP($\overline{\lambda_{uv}(\varepsilon)}$)-compatible eigentriple, then we may reformulate the discrete-time derivative in (3.6) as

$$g'^{d}_{uv}(\varepsilon) = \operatorname{Re}\left(\frac{y_{uv}(\varepsilon)^* M'_{uv}(\varepsilon) x_{uv}(\varepsilon)}{|y_{uv}(\varepsilon)^* x_{uv}(\varepsilon)|}\right). \tag{3.7}$$

We may thus find a root of (3.3) by providing the function $f_{uv} : \mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$ defined by

$$f_{uv}(\varepsilon) \coloneqq (g_{uv}(\varepsilon), g'_{uv}(\varepsilon)) \tag{3.8}$$

as an input to Procedure 1 along with lower bound 0 and upper bound ε.

Remark 3.6. *In fact, at a locally rightmost or outermost point λ of a spectral value set, the derivatives specified in* (3.6) *are equivalent to the derivatives specified on the* *right-hand sides of* (2.5) *which we state formally as Lemma 3.7.*

Lemma 3.7. *Let $\hat{\varepsilon} \in \mathbb{R}^{++}$, $\hat{\varepsilon}\|D\| < 1$ and a point $\lambda_{uv} \in \sigma_{\hat{\varepsilon}}(A, B, C, D)$ with corresponding perturbation vectors u and v, that is $\lambda_{uv} \in \sigma(M_{uv}(\hat{\varepsilon}))$, with corresponding right and left eigenvectors x and y be given. If λ_{uv} is a locally rightmost point of $\sigma_{\hat{\varepsilon}}(A, B, C, D)$ and (λ_{uv}, x, y) is an RP-compatible eigentriple or alternatively, if λ_{uv} is a locally outermost point of $\sigma_{\hat{\varepsilon}}(A, B, C, D)$ and (λ_{uv}, x, y) is an RP($\overline{\lambda_{uv}}$)-compatible eigentriple, then respectively*

$$g'^{c}_{uv}(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{1}{\overline{\beta}\gamma y^* x} \quad \text{and} \quad g'^{d}_{uv}(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{1}{\overline{\beta}\gamma |y^* x|} \tag{3.9}$$

where β and γ are as defined in (2.6).

Proof. Using (3.4) and (3.5) and then substituting in (2.6), we see that

$$\lambda'_{uv}(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{(y^* B u)}{(1 - \hat{\varepsilon} v^* D u)} \frac{(v^* C x)}{(1 - \hat{\varepsilon} v^* D u)} \frac{1}{y^* x} = \frac{1}{\overline{\beta}\gamma y^* x}.$$

We note that $\overline{\beta}\gamma = \mu^{-1} \in \mathbb{R}^{++}$ where μ is given in [GGO13, Equation (3.8)], and thus, in the locally rightmost case where the eigenvectors are RP-compatible, we have that

$$g'^{c}_{uv}(\hat{\varepsilon}) = \operatorname{Re}\left(\lambda'_{uv}(\hat{\varepsilon})\right) = \lambda'_{uv}(\hat{\varepsilon}).$$

In the locally outermost case, where the eigenvectors are RP($\overline{\lambda_{uv}}$)-compatible, using (3.7) we see that

$$g'^{d}_{uv}(\hat{\varepsilon}) = \operatorname{Re}\left(\frac{1}{\overline{\beta}\gamma |y^* x|}\right) = \frac{1}{\overline{\beta}\gamma |y^* x|}.$$

□

Remark 3.8. *As a consequence of Lemma 3.7, we see that at a locally rightmost or* *outermost point λ_{uv} of a spectral value set with corresponding perturbation vectors u and v, that taking the real part of either derivative in* (3.6) *respectively is superfluous but only when λ_{uv} is locally rightmost or outermost.*

Remark 3.9. *It is worth stressing that the equivalences in Lemma 3.7 are only applicable at a point $\lambda_{uv} \in \sigma_{\hat{\varepsilon}}(A, B, C, D)$ that is either truly locally rightmost* (*or outermost*)*. Since Procedure 4 won't typically converge exactly to λ_{uv}, the derivative computed by the* GGO *algorithm may suffer some loss of precision with respect to the true value. On the other hand, the derivatives* (3.6) *used in hybrid expansion-contraction are valid at any $\lambda_{uv} \in \sigma_{\hat{\varepsilon}}(A, B, C, D)$ such that (λ_{uv}, x, y) is an RP-compatible eigentriple of $M_{uv}(\hat{\varepsilon})$ for some nonzero perturbation vectors u and v.*

Given a perturbation defined by ε and vectors u and v such that (3.3) is nonnegative, indicating $\varepsilon \geq \varepsilon_\star$, *hybrid expansion-contraction* is the alternating process of moving the eigenvalue $\lambda_{uv}(\varepsilon)$ back to the boundary of the stability region (if it's not there already) by contracting ε to $\hat{\varepsilon}$ while keeping the perturbation vectors u and v fixed and then subsequently pushing eigenvalue $\lambda_{uv}(\hat{\varepsilon})$ away from the boundary again, either rightward or outward from the origin, by now instead keeping $\hat{\varepsilon}$ fixed and only modifying the perturbation vectors u and v via the SVSAR iteration, as shown in pseudocode Procedure 5. The algorithm repeats this expansion-contraction process in a loop until SVSAR can no longer find a new perturbation that moves $\lambda_{uv}(\hat{\varepsilon})$ outward from the stability boundary, thus signaling that a stationary point of the norm of the transfer function has been found, provided that SVSAR has returned a point satisfying the first-order necessary conditions of Lemma 1.15 or Lemma 1.19.

Theorem 3.10. *Given an initial perturbation $\Delta = \varepsilon_0 uv^*$ with $\varepsilon_0 \in \mathbb{R}^{++}$, $\varepsilon_0 \|D\| < 1$ and vectors $u \in \mathbb{C}^p$ and $v \in \mathbb{C}^m$ such that $\|u\| = \|v\| = 1$, if Assumption 3.3 holds,*

Procedure 5 $[\lambda_k, \varepsilon_k] =$ **hybrid_expansion_contraction**$(\varepsilon_0, u_0, v_0, \lambda_0, x_0, y_0, k_{\mathrm{svsar}})$

```
Input:
  Initial perturbation level $\varepsilon_0 \in \mathbb{R}^{++}$, $\varepsilon_0 \|D\| < 1$
  Vectors $u \coloneqq u_0 \in \mathbb{C}^p$ and $v \coloneqq v_0 \in \mathbb{C}^m$ defining $\Delta = \varepsilon_0 uv^*$ with $\|u\| = \|v\| = 1$
  RP-compatible eigentriple $(\lambda_0, x_0, y_0)$ of $A + B\widetilde{\Delta}_{uv}(\varepsilon_0)C$ such that $\mathrm{Re}(\lambda_0) \geq 0$
  Positive integer $k_{\mathrm{svsar}}$ specifying max number of iterations for svsar$(\cdot)$ subroutine
  Constants: Matrices $A \in \mathbb{C}^{n,n}, B \in \mathbb{C}^{n,p}, C \in \mathbb{C}^{m,p}$, and $D \in \mathbb{C}^{p,m}$
Output:
  Eigenvalue $\lambda_k$ of $A + B\widetilde{\Delta}_{uv}(\varepsilon_k)C$ s.t. $\mathrm{Im}\,(\lambda_k)$ is a stationary point of $\|G(\mathbf{i}\omega)\|$
  $\varepsilon_k \leq \varepsilon_0$ such that $\|G(\mathbf{i}\,\mathrm{Im}\,(\lambda_k))\| = \varepsilon_k^{-1}$

  for $k = 0, 1, 2, \ldots$ until $\mathrm{Re}\,(\lambda_k) = 0$ do
    Set $u \coloneqq u_k$ and $v \coloneqq v_k$
    Set function $f_{uv}(\cdot)$ to (3.8) via (3.2), (3.3) and (3.6)
    // $(\lambda_{uv}(\varepsilon_k), x_{uv}(\varepsilon_k), y_{uv}(\varepsilon_k)) \equiv (\lambda_k, x_k, y_k)$
    $[\hat{\varepsilon}_k, \hat{\lambda}_k, \hat{x}_k, \hat{y}_k] \coloneqq$ newton_bisection$(f_{uv}(\cdot), 0, \varepsilon_k)$
    // $(\hat{\lambda}_k, \hat{x}_k, \hat{y}_k)$ is an RP-compatible eigentriple of $A + B\widetilde{\Delta}_{uv}(\hat{\varepsilon})C$ s.t. $\mathrm{Re}(\hat{\lambda}_k) = 0$
    $[u_{k+1}, v_{k+1}, \lambda_{k+1}, x_{k+1}, y_{k+1}] \coloneqq$ svsar$(\hat{\varepsilon}_k, u_k, v_k, \hat{\lambda}_k, \hat{x}_k, \hat{y}_k, k_{\mathrm{svsar}})$
    $\varepsilon_{k+1} \coloneqq \hat{\varepsilon}_k$
  end for
```

NOTE: *For the discrete-time case, the hybrid expansion-contraction process described in Procedure 5 is easily modified by changing* $\mathrm{Re}(\lambda_0) \geq 0$ *to* $|\lambda_0| \geq 1$, *"locally rightmost" to "locally of largest modulus",* $\mathrm{Re}(\lambda_k) = 0$ *to* $|\lambda_k| = 1$, $\mathrm{Re}(\hat{\lambda}_k) = 0$ *to* $|\hat{\lambda}_k| = 1$, *and correspondingly setting* $\mathrm{svsar}(\cdot)$ *to approximate the radius instead of the abscissa of spectral value sets. The output conditions are correspondingly changed such that* $\angle\lambda_k$ *is a stationary point of* $\|G(\mathrm{e}^{\mathbf{i}\theta})\|$ *and* $\|G(\mathrm{e}^{\mathbf{i}\angle\lambda_k})\| = \varepsilon_k^{-1}$.

then:

1. *For the continuous-time case, under Assumption 2.5 that SVSAR always returns a point satisfying the first-order necessary conditions of Lemma 1.15 and additionally* $\alpha(M_{uv}(\varepsilon_0)) \geq 0$, *hybrid expansion-contraction converges to* $\lambda \in \mathbb{C}$ *and* $\varepsilon \in \mathbb{R}^{++}$ *such that* $\mathrm{Im}\,(\lambda)$ *is a stationary point of* $\|G(\mathbf{i}\omega)\|$ *with stationary value* ε^{-1} *and furthermore, if* λ *is a locally rightmost point of* $\sigma_\varepsilon(A, B, C, D)$, *then*

$\operatorname{Im}(\lambda)$ *is a local maximizer of* $\|G(\mathbf{i}\omega)\|$ *with locally maximal value* ε^{-1}.

2. *For the discrete-time case, under Assumption 2.5 that SVSAR always returns a point satisfying the first-order necessary conditions of Lemma 1.19 and additionally* $\rho(M_{uv}(\varepsilon_0)) \geq 1$, *hybrid expansion-contraction converges to* $\lambda \in \mathbb{C}$ *and* $\varepsilon \in \mathbb{R}^{++}$ *such that* $\angle\lambda$ *is a stationary point of* $\|G(\mathrm{e}^{\mathbf{i}\theta})\|$ *with stationary value* ε^{-1} *and furthermore, if* λ *is a locally outermost point of* $\sigma_\varepsilon(A, B, C, D)$, *then* $\angle\lambda$ *is a local maximizer of* $\|G(\mathrm{e}^{\mathbf{i}\theta})\|$ *with locally maximal value* ε^{-1}.

Proof. For the continuous-time case, assume that hybrid expansion-contraction stagnates and thus $\operatorname{Re}(\lambda_k) \neq 0$ for all $k \in \{1, 2, \ldots\}$ and the sequence $\{\operatorname{Re}(\lambda_k)\} \not\to 0$. By Key Observation 3.5, the contraction phase of hybrid expansion-contraction ensures that from any iterate with ε_k and $\operatorname{Re}(\lambda_k) > 0$, the algorithm will find $\hat{\lambda}_k$ such that $\operatorname{Re}(\hat{\lambda}_k) = 0$ corresponding to root $\hat{\varepsilon}_k \equiv \varepsilon_{k+1} < \varepsilon_k$ of (3.3) for the current perturbation vectors $u := u_k$ and $v := v_k$. As the SVSAR method is monotonic and $\operatorname{Re}(\lambda_0) \geq 0$, it then follows that $\operatorname{Re}(\lambda_{k+1}) > \operatorname{Re}(\hat{\lambda}_k) = 0$ must hold for all k in the expansion phases of the algorithm. While the sequence $\{\operatorname{Re}(\lambda_k)\}$ is not necessarily monotonic, $\operatorname{Re}(\lambda_k) \leq \alpha_{\varepsilon_0}(A, B, C, D)$ must hold for all k since $\{\varepsilon_k\}$ is a monotically decreasing sequence. Thus, there exists a monotone convergent subsequence $\{\operatorname{Re}(\lambda_{k_j})\} \to \beta > 0$. Since $\{\operatorname{Re}(\lambda_{k_j})\}$ may be either decreasing or increasing, by setting $\gamma = \min\{\beta, \operatorname{Re}(\lambda_{k_0})\}$ we have that $0 < \gamma \leq \operatorname{Re}(\lambda_{k_j})$ for all $j \in \{0, 1, 2, \ldots\}$.

Letting $u := u_{k_j}$ and $v := u_{k_j}$ and recalling that $\lambda_{k_j} \equiv \lambda_{uv}(\varepsilon_{k_j})$, $\operatorname{Re}(\lambda_{uv}(\hat{\varepsilon}_{k_j})) = 0$

by Key Observation 3.2, and $\hat{\varepsilon}_{k_j} \equiv \varepsilon_{k_j+1}$, it then follows that:

$$\begin{aligned} 0 < \gamma \leq \operatorname{Re}\left(\lambda_{k_j}\right) &= \operatorname{Re}\left(\lambda_{uv}(\varepsilon_{k_j})\right) \\ &= \operatorname{Re}\left(\lambda_{uv}(\varepsilon_{k_j})\right) - \operatorname{Re}\left(\lambda_{uv}(\hat{\varepsilon}_{k_j})\right) \\ &= \operatorname{Re}\left(\lambda_{uv}(\varepsilon_{k_j})\right) - \operatorname{Re}\left(\lambda_{uv}(\varepsilon_{k_j+1})\right). \end{aligned} \tag{3.10}$$

By continuity of the eigenvalue $\lambda_{uv}(\varepsilon)$, (3.10) implies that there exists a fixed real value $\delta > 0$ such that

$$\delta \leq \varepsilon_{k_j} - \varepsilon_{k_j+1}$$

for all j which in turn implies that

$$\lim_{j \to \infty} \varepsilon_{k_j} = -\infty.$$

As $\varepsilon_k \geq 0$ for all k, we thus have a contradiction and therefore hybrid expansion-contraction guarantees convergence to λ such that $\operatorname{Re}(\lambda) = 0$. Since λ is computed by SVSAR, the rest of the statement follows directly from Assumption 2.5 and Lemma 1.17. The proof is analogous for the discrete-time case, using Lemma 1.20. $\square$

3.2 The convergence rate of hybrid expansion-contraction

We now describe how the hybrid expansion-contraction process of Procedure 5 is actually an adaptively positively or negatively damped Newton method, in contrast to the Newton-bisection outer iteration of the GGO algorithm. Though we limit the discussion here to the continuous-time case, the analysis holds for the discrete-time case as well.

Given some $\varepsilon_k \in \mathbb{R}^{++}$, $\varepsilon_k \|D\| < 1$, let λ_{uv} be a locally or globally rightmost point of $\sigma_{\varepsilon_k}(A,B,C,D)$ such that $\operatorname{Re}(\lambda_{uv}) > 0$ and let u and v denote its corresponding optimal vectors, that is (λ_{uv}, x, y) is a globally rightmost eigentriple of $M_{uv}(\varepsilon_k)$, and furthermore let $g_{uv}^c(\cdot)$ be defined for λ_{uv} as in (3.2) along with its derivative given in (3.6). In the contraction phase, the first updated value from ε_k attempting to make $g_{uv}^c(\cdot) = \operatorname{Re}(\lambda_{uv}(\cdot))$ zero is

$$\varepsilon^{\mathrm{C}} = \varepsilon_k - \frac{g_{uv}^c(\varepsilon_k)}{g_{uv}^{\prime c}(\varepsilon_k)}.$$

If $\alpha(M_{uv}(\varepsilon^{\mathrm{C}})) < 0$, it then follows that a locally rightmost point computed by SVSAR for perturbation level ε^{C} and starting vectors u and v could be in the left half-plane, which might potentially cause the breakdown scenario observed in the GGO algorithm. Thus, the contraction phase instead finds $\hat{\varepsilon}_k > \varepsilon^{\mathrm{C}}$ such that $\alpha(M_{uv}(\hat{\varepsilon}_k)) \geq 0$ holds to ensure that the next locally rightmost point found by SVSAR resides in the right half-plane. Similarly, if $\alpha(M_{uv}(\varepsilon^{\mathrm{C}})) > 0$, it is guaranteed that the next locally rightmost point computed by SVSAR will be in the right half-plane (as SVSAR is a monotonic method) and thus the contraction phase can take an even larger reduction than provided via ε^{C} by instead finding $\hat{\varepsilon}_k < \varepsilon^{\mathrm{C}}$ such that $\alpha(M_{uv}(\hat{\varepsilon}_k)) = 0$.

However, as λ_{uv} is a locally rightmost point of $\sigma_{\varepsilon_k}(A,B,C,D)$, by Lemma 3.7, ε^{C} is actually equivalent to the Newton step ε^{N} the GGO algorithm would have taken from ε_k and λ_{uv}, provided that the conditions for a bisection step were not triggered in either routine. As a consequence, the contraction phase of hybrid expansion-contraction is actually scaling the Newton step of the GGO algorithm by either damping it sufficiently to avoid the potential breakdown that may have occurred if $\varepsilon^{\mathrm{C}} \equiv \varepsilon^{\mathrm{N}}$ had been accepted or alternatively, by enlarging the step size when possible. In fact, by finding $\hat{\varepsilon}_k$ such that $\alpha(M_{uv}(\hat{\varepsilon}_k)) = 0$, we see that the contraction phase is either damping the step ε^{N} the *minimal* amount when $\alpha(M_{uv}(\varepsilon^{\mathrm{N}})) < 0$ or increasing it the *maximal* amount when

$\alpha(M_{uv}(\varepsilon^{\mathrm{N}})) > 0$ such that breakdown can still be guaranteed to not occur via evaluating $\alpha(M_{uv}(\cdot))$.

We now present two assumptions in order to formally state and prove a worst-case convergence rate for hybrid expansion-contraction.

Assumption 3.11. *Given a sequence of rightmost points computed by hybrid expansion-contraction, that is the sequence $\{\lambda_k\}$ such that each λ_k is a locally rightmost point of $\sigma_{\varepsilon_k}(A, B, C, D)$ for all k, we assume that the points λ_k all lie along a single continuously differentiable branch of locally rightmost points of $\sigma_\varepsilon(A, B, C, D)$ defined for at least $\varepsilon \in [\min \varepsilon_k, \max \varepsilon_k]$. Analogously, if $\{\lambda_k\}$ is such that each λ_k is a locally outermost point of $\sigma_{\varepsilon_k}(A, B, C, D)$ for all k, we then assume that the points λ_k all lie along a single continuously differentiable branch of locally outermost points of $\sigma_\varepsilon(A, B, C, D)$.*

Remark 3.12. *Note that we make Assumption 3.11 only for the sake of simplicity of the convergence rate analysis. In fact, hybrid expansion-contraction always converges regardless of whether, for example in the continuous-time case, it stays upon one path of locally rightmost points or jumps amongst several. In contrast, such jumps are precisely why the* GGO *algorithm can potentially incur a fatal bound mismatch error. For analyzing the converge rate however, Assumption 3.11 allows us to preclude the possibility that hybrid expansion-contraction oscillates between different branches as it converges, that is, the case where hybrid expansion-contraction is converging to a root of a function that is actually changing over the course of the algorithm.*

Assumption 3.13. *Given $\varepsilon \in \mathbb{R}^{++}$, $\varepsilon\|D\| < 1$ and λ_{uv} a locally rightmost point of $\sigma_\varepsilon(A, B, C, D)$ such that $\operatorname{Re}(\lambda_{uv}) > 0$ and where u and v denote its corresponding optimal vectors, that is λ_{uv} is a rightmost eigenvalue of $M_{uv}(\varepsilon)$, we assume that $g_{uv}^{\mathrm{c}}(\cdot)$ as defined in* (3.2) *is a monotonically increasing function. Analogously, if λ_{uv} is instead* *a locally outermost point of $\sigma_\varepsilon(A, B, C, D)$ such that $|\lambda_{uv}| > 1$ and λ_{uv} is an outermost eigenvalue of $M_{uv}(\varepsilon)$, we then assume that $g_{uv}^d(\cdot)$ as defined in* (3.2) *is a monotonically increasing function.*

Remark 3.14. *We note that we make Assumption 3.13 for simplicity of presentation here, and furthermore, have yet to observe a situation in practice where the assumption doesn't hold. Nonetheless, we conjecture that the assumption is likely not necessary for the convergence rate result of Theorem 3.15 to hold. In lieu of assuming the functions $g_{uv}^{(k)}(\cdot)$ are monotonic, it seems that we can allow for these functions to change direction and to even have several roots. We then might proceed to instead define a monotonic weighting function that assigns an appropriate value of the damping factor parameter γ_k based on the relative position of $\varepsilon_k^{\mathrm{C}}$ along the length of the path of the eigenvalue and whether $\lambda_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$ is in the left or right half-plane. In the case of several roots, we would instead need to separate the path $\lambda_{uv}^{(k)}(\cdot)$ into separate pieces and define individual monotone damping factor weighting functions separately for each piece. As this complicates the essential argument of the proof without providing much additional insight, or even utility in the sense that doing so addresses a problem we haven't observed, we present the proof using Assumption 3.13.*

Theorem 3.15. *Let $\{\lambda_k\}$ and $\{\varepsilon_k\}$ respectively be the sequences produced by hybrid expansion-contraction where $\{\lambda_k\}$ is the sequence of either locally rightmost points of $\sigma_{\varepsilon_k}(A, B, C, D)$ or locally outermost points of $\sigma_{\varepsilon_k}(A, B, C, D)$, for the continuous and discrete-time cases respectively, while $\{\varepsilon_k\}$ is the corresponding strictly decreasing perturbation levels ε_k converging to some $\varepsilon_\star \in \mathbb{R}^{++}$. Under Assumption 3.11 and assuming Assumption 3.13 holds for each pair λ_k and ε_k, then hybrid expansion-contraction converges to $\varepsilon_\star$ at least superlinearly.*

Proof. We first consider the continuous-time case where $\{\lambda_k\}$ is a sequence of locally rightmost points of $\sigma_{\varepsilon_k}(A,B,C,D)$. Let $\lambda_{uv}^{(k)} = \lambda_k$, where u and v denote the specific corresponding optimal vectors for $\lambda_{uv}^{(k)}$, that is $\lambda_{uv}^{(k)}$ is an eigenvalue of $M_{uv}(\varepsilon_k)$. Furthermore, let $g_{uv}^{(k)}(\cdot)$ be defined for $\lambda_{uv}^{(k)}$ for the continuous-time case, that is as defined by $g_{uv}^{c}(\cdot)$ in (3.2), and let $\lambda^{\mathrm{r}}(\varepsilon)$ be the continuous path of locally rightmost points in Assumption 3.11.

At a locally rightmost point $\lambda_{uv}^{(k)}$ computed by hybrid expansion-contraction, the first value considered so to make $g_{uv}^{(k)}(\cdot) = \mathrm{Re}\,(\lambda_{uv}^{(k)}(\cdot))$ zero is

$$\varepsilon_k^{\mathrm{C}} \coloneqq \varepsilon_k - \frac{g_{uv}^{(k)}(\varepsilon_k)}{g_{uv}^{\prime(k)}(\varepsilon_k)}, \tag{3.11}$$

assuming $\varepsilon_k^{\mathrm{C}} > 0$, since the contraction phase is implemented by a Newton-bisection method initialized with a lower bound of zero. Since $\lambda_{uv}^{(k)}$ is a locally rightmost point, by Lemma 4.7 it follows that the update to ε_k given by a Newton step to find the root of $\mathrm{Re}\,(\lambda^{\mathrm{r}}(\cdot))$ has the following equivalence

$$\varepsilon_{k+1}^{\mathrm{N}} \coloneqq \varepsilon_k - \frac{\mathrm{Re}\,(\lambda^{\mathrm{r}}(\varepsilon_k))}{\mathrm{Re}\,(\lambda^{\mathrm{r}}(\varepsilon_k))'} = \varepsilon_k - \frac{g_{uv}^{(k)}(\varepsilon_k)}{g_{uv}^{\prime(k)}(\varepsilon_k)} \tag{3.12}$$

and thus by (3.11), we see that $\varepsilon_{k+1}^{\mathrm{N}} = \varepsilon_k^{\mathrm{C}}$. However, since at every iteration hybrid expansion-contraction instead chooses $\varepsilon_{k+1} = \hat{\varepsilon}_k$ such that $g_{uv}^{(k)}(\hat{\varepsilon}_k) = 0$, we may instead consider hybrid expansion-contraction as an adaptive positively or negatively damped Newton method given by the update function

$$\varepsilon_{k+1} \coloneqq \varepsilon_k - \gamma_k \frac{g_{uv}^{(k)}(\varepsilon_k)}{g_{uv}^{\prime(k)}(\varepsilon_k)} = \varepsilon_k - \gamma_k \frac{\mathrm{Re}\,(\lambda^{\mathrm{r}}(\varepsilon_k))}{\mathrm{Re}\,(\lambda^{\mathrm{r}}(\varepsilon_k))'},$$

where $\gamma_k > 1$ if $g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) > 0$ or $0 < \gamma_k < 1$ if $g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) < 0$. Furthermore, we may also rewrite the damped Newton update step as an *inexact* Newton step as follows:

$$\begin{aligned} \varepsilon_{k+1} = \varepsilon_k - \gamma_k \frac{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))}{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))'} &= \varepsilon_k - \frac{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))}{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))'} + (1-\gamma_k)\frac{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))}{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))'} \\ &= \varepsilon_k - \frac{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))}{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))'} + \frac{r_k}{\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))'}, \end{aligned} \tag{3.13}$$

where $r_k \coloneqq (1-\gamma_k)\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))$ is defined as the residual from not having solved the regular Newton step exactly. Consider the *forcing* sequence $\{\eta_k\}$ defined by

$$\eta_k \coloneqq \frac{|r_k|}{|\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon_k))|} = |1-\gamma_k|.$$

As $\varepsilon_k \to \varepsilon_{\tilde{\star}}$, by [DES82, Corollary 3.5(a)], it follows that if $\eta_k \to 0$, then the inexact Newton method converges at least superlinearly. For hybrid expansion-contraction, it thus suffices to show that $\gamma_k \to 1$ as $k \to \infty$.

Let $\{\gamma_k\}$ be the sequence of damping factors in hybrid expansion-contraction. We first consider the case where $\gamma_k > 1$ holds for all k and thus $g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) > 0$ holds as well. Hence, for all k, we have that

$$0 = g_{uv}^{(k)}(\hat{\varepsilon}_k) < g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) < g_{uv}^{(k)}(\varepsilon_k),$$

as shown in the left panel of Figure 3.1. By Theorem 4.1, $g_{uv}^{(k)}(\varepsilon_k) \to 0$ as $k \to \infty$ and thus $g_{uv}^{(k)}(\hat{\varepsilon}_k) = g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$ holds in the limit. Since $g_{uv}^{(k)}(\cdot)$ is always a monotonically increasing function by Assumption 3.13, it follows that $\hat{\varepsilon}_k = \varepsilon_k^{\mathrm{C}}$ holds in the limit as well and thus $\gamma_k \to 1$.

Now let us consider the case when $\gamma_k < 1$ holds for all k and thus $g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) < 0$

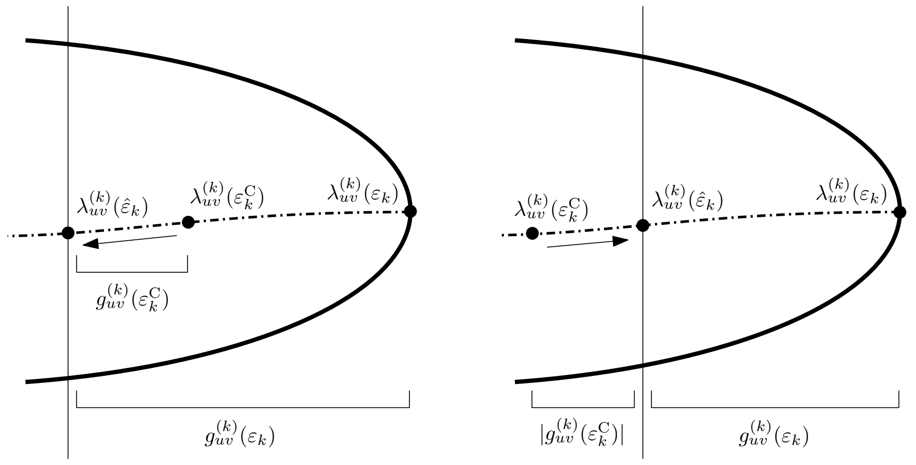
$\lambda_{uv}^{(k)}(\hat{\varepsilon}_k)$
$\lambda_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$
$\lambda_{uv}^{(k)}(\varepsilon_k)$
$g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$
$g_{uv}^{(k)}(\varepsilon_k)$
$\lambda_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$
$\lambda_{uv}^{(k)}(\hat{\varepsilon}_k)$
$\lambda_{uv}^{(k)}(\varepsilon_k)$
$|g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})|$
$g_{uv}^{(k)}(\varepsilon_k)$

FIGURE 3.1: *Left: negative damping. Right: positive damping. In the left panel, we see that the initial contraction attempt falls short of reaching the imaginary axis and thus hybrid expansion contraction can take an accelerated step compared to the Newton step of the* GGO *algorithm, that is with $\gamma_k > 1$. In the right panel, we see that the initial contraction attempt overshoots the imaginary axis and thus it must be damped in comparison to the Newton step of the* GGO *algorithm, that is with $\gamma_k \in (0,1)$, to ensure breakdown cannot occur.*

holds. It then follows for all k, that

$$g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}}) < 0 < g_{uv}^{(k)}(\varepsilon_k), \tag{3.14}$$

as shown in the right panel of Figure 3.1. Consider the ratio

$$\frac{g_{uv}^{(k)}(\varepsilon_k)}{g_{uv}^{(k)}(\varepsilon_k) - g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})} = \frac{1}{1 - \frac{g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})}{g_{uv}^{(k)}(\varepsilon_k)}}, \tag{3.15}$$

noting that it is always in the interval $(0,1)$. Since $\varepsilon_k^{\mathrm{C}} = \varepsilon_{k+1}^{\mathrm{N}}$, it follows that

$$\frac{g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})}{g_{uv}^{(k)}(\varepsilon_k)} = \frac{g_{uv}^{(k)}(\varepsilon_{k+1}^{\mathrm{N}})}{g_{uv}^{(k)}(\varepsilon_k)}$$

and thus since Newton's method converges quadratically, it additionally follows that

$$\lim_{k\to\infty} \frac{g_{uv}^{(k)}(\varepsilon_{k+1}^{\mathrm{N}})}{g_{uv}^{(k)}(\varepsilon_k)} = 0.$$

Thus, the ratio in (3.15) must go to one in the limit as well, which then implies that $g_{uv}^{(k)}(\hat{\varepsilon}_k) = g_{uv}^{(k)}(\varepsilon_k^{\mathrm{C}})$ also holds in the limit. Again by Assumption 3.13, $g_{uv}^{(k)}(\cdot)$ is always a monotonically increasing function and hence it again follows that $\hat{\varepsilon}_k = \varepsilon_k^{\mathrm{C}}$ must hold in the limit and thus $\gamma_k \to 1$ as $k \to \infty$.

Finally, we consider the case when $\{\gamma_k\}$ is mixed in the sense that it contains some damping factors that are less than one and some damping factors that are greater than one. Consider the subsequence $\{\gamma_{k_j}\}$ of $\{\gamma_k\}$ consisting of all the damping factors strictly greater than one. Thus, it follows that $\gamma_{k_j} > 1$ holds for all k_j. If $\varepsilon_{k_j} \to \varepsilon_{\tilde{\star}}$ holds, then we have already shown that it must follow that $\gamma_{k_j} \to 1$. Otherwise, $\varepsilon_{k_j} \not\to \varepsilon_{\tilde{\star}}$ must hold and consequently, it must be that $\{\varepsilon_{k_j}\}$ and $\{\gamma_{k_j}\}$ are finite sequences, since $\varepsilon_k \to \varepsilon_{\tilde{\star}}$ by assumption. Thus, there exists some positive integer l for $\{\gamma_k\}$ such that $0 < \gamma_k \leq 1$ holds for all $k > l$. However, by the earlier part of the proof, this again implies that $\gamma_k \to 1$. The argument follows analogously if we had instead chosen the subsequence $\{\gamma_{k_j}\}$ of $\{\gamma_k\}$ consisting of all the damping factors strictly between zero and one, and hence $\gamma_k \to 1$ holds generically.

The proof also holds for the discrete-time by taking $g_{uv}^{(k)}(\cdot)$ to be defined for the discrete-time case, that is as defined by $g_{uv}^{d}(\cdot)$ in (3.2) and by replacing all occurrences of $\operatorname{Re}(\lambda^{\mathrm{r}}(\cdot))$ by $|\lambda^{\mathrm{o}}(\cdot)| - 1$, where $\lambda^{\mathrm{o}}(\varepsilon)$ is a continuous path of locally outermost points as in Assumption 3.11.

□

As a consequence of Theorem 3.15, even in the pessimistic case where the steps of hybrid expansion-contraction are positively damped compared to the Newton steps of the GGO algorithm, hybrid expansion-contraction still converges at least superlinearly and possibly quadratically if it is started sufficiently near a root of $g_{uv}^{c}(\cdot)$. On the other hand, when hybrid expansion-contraction is negatively damped in comparison to the GGO algorithm, since the sequence of ε_k values is monotonically decreasing and the method is actually taking accelerated steps (that is, larger steps than the GGO algorithm), we can expect hybrid expansion-contraction to be an exceptionally fast method and to demonstrate *at least* quadratic convergence in some cases. In fact, as we report in Section 5.3, we observe quadratic convergence on most and perhaps all the problems tested.

3.3 Contracting early

Since SVSAR's slow convergence in the expansion phase is potentially expensive, an open question is whether we can terminate SVSAR early, thus reducing the number of eigentriples that need to be computed while still converging to a locally or globally optimal value. With the Newton-bisection outer iteration of the GGO, this is potentially quite dangerous, as terminating SVSAR early while it is in the left half-plane could result in making fatal bound mismatch errors more likely. Furthermore, terminating SVSAR early may degrade the quality of GGO's Newton steps towards the imaginary axis due to loss of accuracy in the corresponding derivation computation. However, under hybrid expansion-contraction, as long as the expansion phase makes significant progress into the right half-plane, the procedure can always begin contracting again early, in lieu of incurring the full number of iterations for SVSAR to converge to a locally rightmost point. A natural choice then is to not just terminate the expansion phase if the step size becomes minuscule but to also considering terminating it if the current step length falls below, say for example, one percent of the max step size taken so far for that particular value of $\hat{\varepsilon}_k$. A benefit to this scheme is that it is self-scaling, meaning that as hybrid expansion-extraction converges, the expansion phase will be permitted to take smaller and smaller steps before switching back to the contraction phase. In other words, in the limit, such a relative step size termination condition has no effect and thus does not alter the theoretical rate of convergence guarantees shown in Section 3.2.

Remark 3.16. *Alternatively, one might consider a relative step termination condition trigged by when the step size falls below some specify percentage of the sum of the all previous step sizes taken so far for a given value of ε. This could allow for even earlier termination of* SVSAR *but potentially at the expense of increased iterations for hybrid expansion-contraction. We however do not explore this strategy or alternate variants when evaluating enabling relative step size termination in the experiments in Chapter 5.*

3.4 A fast new method to find an initial upper bound

In order for the GGO algorithm to apply the outer Newton-bisection iteration, it first must find an upper bound on $\varepsilon_\star$ to initialize the bracket range necessary for bisection. The implementation of the GGO algorithm, version 1.02 of `hinfnorm` [Hin], does this by first calculating an initial ε_0 along with vectors u_0 and v_0. From that perturbation, it calls SVSAR to expand rightward as far as possible for ε_0. If the final eigenvalue returned by SVSAR is still in the left half-plane, then `hinfnorm` increases ε_0 via $\varepsilon_k = \min(2\varepsilon_{k-1}, 0.5(\varepsilon_{k-1} + \|D\|^{-1}))$ and again calls SVSAR. Unfortunately, this precomputation phase to initialize the bracket range that provides an upper bound to $\varepsilon_\star$

can be very expensive as SVSAR's mere linear convergence may be incurred multiple times before a large enough value of ε is found. Increasing ε via taking, say double, the Newton step with respect to the spectral value set abscissa is a potentially better strategy but it still might require several calls to SVSAR being made before an upper bound is found.

On the other hand, for a given value of ε, hybrid expansion-contraction can be initialized from any point $\lambda_0 \in \sigma_\varepsilon(A, B, C, D)$ if $\operatorname{Re}(\lambda_0) \geq 0$. Thus, a more efficient procedure for finding a suitable upper bound to start hybrid expansion contraction is to take some initial ε_0 and perturbation vectors u and v and consider $\alpha(M_{uv}(\varepsilon_0))$. If $\alpha(M_{uv}(\varepsilon_0)) \geq 0$, then hybrid expansion-contraction can begin. If not, we first attempt to set $\varepsilon_1 > \varepsilon_0$ via taking double the Newton step with respect to $\alpha(M_{uv}(\varepsilon_0))$ using (3.5) and (3.6). Of course, attempting such a step is only sensible if (3.6) is positive, a condition which is not guaranteed but often holds in practice. However, even if (3.6) is indeed positive, it may still be that $\alpha(M_{uv}(\varepsilon_1)) < \alpha(M_{uv}(\varepsilon_0))$, in which case a line search may be performed to find $\hat{\varepsilon}_1 \in [\varepsilon_0, \varepsilon_1]$ such that $\alpha(M_{uv}(\hat{\varepsilon}_1)) > \alpha(M_{uv}(\varepsilon_0))$ is attained. After attempting to increase the perturbation level from ε_0, successfully or not, we then update the perturbation vectors u and v by taking a single step of the SVSAR subroutine. Until we find some perturbation level ε with perturbation vectors u and v such that $\alpha(M_{uv}(\varepsilon)) \geq 0$, we continue alternating between attempting to increase the perturbation level ε and doing a single SVSAR update step to perturbation vectors u and v. The benefit is that at no point do we incur SVSAR's linear convergence and by attempting to increase the perturbation level ε before updating the perturbation vectors, we can expect that the SVSAR update step will typically be quite large as it will be updating from a point in the interior of the spectral value set for that perturbation level. However, a concern with Procedure 6's efficient strategy is whether it compromises the likelihood of hybrid expansion-contraction converging to a globally optimal value compared to the bound bracketing procedure used in `hinfnorm`. As a compromise, it may be beneficial to make a single call to SVSAR once Procedure 6 has found an eigenvalue in the right half-plane before commencing hybrid expansion-contraction.

Procedure 6 $[\varepsilon, u, v]$ = **find_upper_bound**$(\varepsilon_0, u_0, v_0)$

```
Input:
  Initial perturbation level $\varepsilon_0 \in \mathbb{R}^{++}$, $\varepsilon_0\|D\| < 1$
  Vectors $u := u_0 \in \mathbb{C}^p$ and $v := v_0 \in \mathbb{C}^m$ defining $\Delta = \varepsilon_0 u_0 v_0^*$ with $\|u\| = \|v\| = 1$
Output:
  $\varepsilon$, $u$ and $v$ such that $\alpha(M_{uv}(\varepsilon)) \geq 0$

  while $\alpha(M_{uv}(\varepsilon_0)) < 0$ do
    Set function $g_{uv}(\cdot)$ to (3.3) for a rightmost eigenvalue $\lambda_{uv}$ of $M_{uv}(\varepsilon_0)$.
    if $g'_{uv}(\varepsilon_0) > 0$ then
      Set $\varepsilon := \min(\varepsilon_0 - 2\frac{g_{uv}(\varepsilon_0)}{g'_{uv}(\varepsilon_0)}, \frac{1}{2}(\varepsilon_0 + \|D\|^{-1}))$
      while $\alpha(M_{uv}(\varepsilon)) \leq \alpha(M_{uv}(\varepsilon_0))$ do
        Set $\varepsilon := \frac{1}{2}(\varepsilon_0 + \varepsilon)$
      end while
      if $\alpha(M_{uv}(\varepsilon)) \geq 0$ then
        break
      end if
    else
      Set $\varepsilon := \varepsilon_0$
    end if
    // $(\lambda_{uv}, x_{uv}, y_{uv})$ is a rightmost RP-compatible eigentriple of $M_{uv}(\varepsilon)$
    // Update vectors $u$ and $v$ by a single iteration of svsar$(\cdot)$
    $[u, v, \lambda_{uv}, x_{uv}, y_{uv}]$ := svsar$(\varepsilon, u, v, \lambda_{uv}, x_{uv}, y_{uv}, 1)$
    if $\alpha(M_{uv}(\varepsilon)) \geq 0$ then
      break
    end if
    Set $\varepsilon_0 := \varepsilon$
  end while
```

NOTE: *For the discrete-time case, Procedure 6,* find_upper_bound$(\cdot)$*, is easily modified by changing all occurrences of* $\alpha(\cdot)$ *to* $\rho(\cdot)$ *and "rightmost" to "outermost". Furthermore, the procedure can be modified as the user desires to instead first call* svsar$(\cdot)$*, to expand* ε *using the derivative of the spectral value set abscissa* (2.3) (*described in* [GGO13]) *which would increase* ε *more slowly and to optionally call* svsar$(\cdot)$ *with its normal limit on its max iteration count before beginning hybrid expansion-contraction of Procedure 5.*

4
AN IMPROVED SVSAR SUBROUTINE

4.1 Extending SVSAR to large-scale D matrices

A potential downside of using the default SVSAR subroutine, in either the original GGO algorithm or in our improved algorithm, is that at every iteration of SVSAR, the following two linear systems must be solved:

$$\Phi_p b_k = B^* y_k \quad \text{and} \quad \Phi_m c_k = C x_k \tag{4.1}$$

where

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

and x_k and y_k are the right and left computed eigenvectors at the k-th iteration, a cost that is compounded by the fact that SVSAR's linear convergence may lead to many iterations and thus solves being incurred. Indeed, this not only motivated the strong assumption that $p, m \ll n$ in [GGO13] but also that the version 1.02 implementation of GGO simply used MATLAB's backslash operator for every solve involving Φ_p and Φ_m in a given SVSAR call. Noting that both Φ_p and Φ_m are Hermitian, a more efficient approach is to precompute Cholesky factorizations of each for a given value of ε and then use the factorized forms for efficient backsolves at each iterate within a call to SVSAR. Of course, such an optimization can only reduce SVSAR's runtime by a constant factor but this can be quite a significant savings in runtime performance if p and/or m are more moderately sized. However, even this is still an overly pessimistic assessment of using SVSAR on large-scale problems.

By the Sherman-Morrison-Woodbury formula [GV83], we observe the following equivalence:

$$\mathit{\Phi}_p^{-1} = (I_p - \varepsilon^2 D^* D)^{-1} = I_p + \varepsilon^2 D^*(I_m - \varepsilon^2 DD^*)^{-1} D = I_p + \varepsilon^2 D^* \mathit{\Phi}_m^{-1} D$$

and similarly

$$\mathit{\Phi}_m^{-1} = (I_m - \varepsilon^2 DD^*)^{-1} = I_m + \varepsilon^2 D(I_p - \varepsilon^2 D^* D)^{-1} D^* = I_m + \varepsilon^2 D \mathit{\Phi}_p^{-1} D^*.$$

Hence, solving both linear systems of (4.1) may instead be done by creating a only single Cholesky factorization of say $\mathit{\Phi}_p = LL^*$ and then, using MATLAB notation, performing the two pairs of backsolves as follows:

$$\begin{aligned} \mathit{\Phi}_p b = B^* y_k \quad &\Longrightarrow \quad b = L^* \backslash (L \backslash (B^* y_k)) \\ \mathit{\Phi}_m c = C x_k \quad &\Longrightarrow \quad c = C x_k + \varepsilon^2 D(L^* \backslash (L \backslash (D^*(C x_k)))). \end{aligned}$$

Furthermore, we can opportunistically choose whether to factorize $\mathit{\Phi}_p$ or $\mathit{\Phi}_m$ by opting for the smaller dimensional one of the two, thus relaxing the condition on SVSAR's viability to only requiring that $\min(p, m)$ is small, while $\max(p, m)$ is free to be on the order of n.

For the case when both p and m may be large, it is helpful to first examine the typical structure that the matrix D may have. In many applications, the linear time invariant systems (1.1) and (1.2) whose H_∞ norm is desired arises from the following standard state-space open loop system [BHLO06]:

$$\left[\begin{array}{c} \dot{x} \\ \hline z \\ y \end{array}\right] = \left[\begin{array}{c|cc} A_1 & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \left[\begin{array}{c} x \\ \hline w \\ u \end{array}\right] \tag{4.2}$$

where x contains the states, u and w are the physical (control) and performance inputs respectively, and y and z are the physical (measured) and performance outputs respectively. However, to improve the stability and robustness of the open loop system (4.2), customarily a controller K is designed

$$\begin{bmatrix} \dot{\hat{x}} \\ u \end{bmatrix} = K \begin{bmatrix} \hat{x} \\ y \end{bmatrix} = \begin{bmatrix} \widehat{A} & \widehat{B} \\ \widehat{C} & \widehat{D} \end{bmatrix} \begin{bmatrix} \hat{x} \\ y \end{bmatrix}$$

where $\hat{x} \in \mathbb{R}^{\hat{n}}$ is the controller state and $\hat{n}$ is the order of the controller, to minimize the H_∞ norm of the resulting closed loop system:

$$\left[\begin{array}{c} \dot{x} \\ \dot{\hat{x}} \\ \hline z \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \left[\begin{array}{c} x \\ \hat{x} \\ \hline w \end{array}\right]$$

with, assuming $D_{22} = 0$ purely for conciseness of notation,

$$\begin{aligned} A &= \begin{bmatrix} A_1 + B_2\widehat{D}C_2 & B_2\widehat{C} \\ \widehat{B}C_2 & \widehat{A} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 + B_2\widehat{D}D_{21} \\ \widehat{B}D_{21} \end{bmatrix}, \\ C &= \begin{bmatrix} C_1 + D_{12}\widehat{D}C_2 & D_{12}\widehat{C} \end{bmatrix}, \quad D = \begin{bmatrix} D_{11} + D_{12}\widehat{D}D_{21} \end{bmatrix}. \end{aligned} \tag{4.3}$$

We focus on the design of *low-order controllers* for which the matrices of the controller, $\widehat{A}$, $\widehat{B}$, $\widehat{C}$, and $\widehat{D}$, are relatively small, because this makes it much easier to physically implement the controller in hardware. However, the best way to design such controllers seems to be using nonsmooth, nonconvex optimization, for which we cannot guarantee finding global minimizers. Perhaps the most promising method is this domain has been full BFGS [BHLO06, Hif], which requires work per iteration that is quadratic in the number of variables.

With this in mind, we now consider the structure of the matrix D for the generic case of $D_{22} \neq 0$

$$D = D_{11} + D_{12}(I - \widehat{D}D_{22})^{-1}\widehat{D}D_{21}.$$

When p and m are both large, we can reasonably assume that D_{11} is sparse and that $(I - \widehat{D}D_{22})^{-1}$ is cheaply computable, as otherwise merely forming D for input to SVSAR or doing a matrix-vector product with D would be an expensive operation. Given that the $\widehat{D}$ matrix of controller variables is typically small, we also note that

$$D_{12}(I - \widehat{D}D_{22})^{-1}\widehat{D}D_{21} = \widetilde{D}_{12}D_{21} = D_{12}\widetilde{D}_{21}$$

will often be a low-rank outer product, with $\operatorname{rank}(\widetilde{D}_{12}D_{21}) \ll \min(p, m)$ if both p and m are large. If $D_{11} = 0$, then we have an *a priori* low-rank factorization of D. Even if $D_{11} \neq 0$, it may still be known that $\operatorname{rank}(D_{11})$ is small and thus a low-rank factorization of D can be quickly computed or provided as input.

Assuming D is given as a rank-k outer product

$$D = D_L D_R^*,$$

with $k \ll n$, one potential option for quickly solving the linear systems of (4.1) is to use CHOLMOD [DH99, DH09] to create a sparse Cholesky factorization of a low-rank update to the identity matrix. Furthermore, as only ε changes for every call to SVSAR, the nonzero structure of $\varPhi_p$ and $\varPhi_m$ remains constant and thus the symbolic analysis required for computing the factorizations done by CHOLMOD need only be computed once and can be subsequently reused to alleviate some of the cost of updating the factorizations for new values of ε. Unfortunately, such specialized routines of CHOLMOD are not currently exposed to the user in MATLAB. As an alternative, once again by the Sherman-Morrison-Woodbury formula, we see that

$$\begin{aligned}(I_p - \varepsilon^2 D^* D)^{-1} &= (I_p - \varepsilon^2 (D_L D_R^*)^* (D_L D_R^*))^{-1} \\ &= (I_p - \varepsilon^2 (D_R D_L^* D_L) D_R^*)^{-1} \\ &= I_p + \varepsilon^2 (D_R D_L^* D_L)(I_k - \varepsilon^2 (D_R^* D_R)(D_L^* D_L))^{-1} D_R^*\end{aligned}$$

and similarly

$$(I_m - \varepsilon^2 D D^*)^{-1} = I_m + \varepsilon^2 (D_L D_R^* D_R)(I_k - \varepsilon^2 (D_L^* D_L)(D_R^* D_R))^{-1} D_L^*,$$

thus providing efficient forms for solving the linear systems of (4.1) where we only need to do LU factorizations on two small k-by-k matrices.

In the case when $\operatorname{rank}(D_{11})$ is large, it will generally be best to provide D as a function handle, unless D_{12} or D_{21} happen to be zero. If $D = D_{11}$, then CHOLMOD may be used to create a sparse factorization. However, if $D_{12} \neq 0$ and $D_{21} \neq 0$, forming D explicitly could cause dense fill-in due to the low-rank $\widetilde{D}_{12} D_{21}$ term, hence the need to provide D as a function handle for applying D to an input vector. As an aside, likewise to avoid fill-in from the low-rank terms in (4.3), function handles should also be provided for applying matrices A, B, and C.

Without an explicit form for D, solving (4.1) must be done via an iterative method such as conjugate gradient, which can unfortunately be slow without a good preconditioner. Furthermore, even constructing the preconditioner must also be done iteratively in this case. It is thus conceivable that the cost of building preconditioners for when D is not given explicitly may outweigh any performance enhancement they provide, unless they are particularly effective, cheap to compute, and/or reusable across many problems before an updated preconditioner is required. On the other hand, within SVSAR, we may be able to use information from the previous iterate to effectively warm-start conjugate gradient for the current one, despite the fact that the right-hand sides of (4.1), $B^* y_k$ and Cx_k, may not necessarily form convergent sequences.

Consider the following two solutions to the linear systems from both the current and previous iterates within the SVSAR subroutine:

$$c_k := \Phi_m^{-1} C x_k \tag{4.4}$$

$$c_{k-1} := \Phi_m^{-1} C x_{k-1}. \tag{4.5}$$

If we are solving (4.4) iteratively, we expect that for an initial vector $\tilde{c}_k$, the smaller the value of $\|c_k - \tilde{c}_k\|$ is, the less number of iterations will be necessary to find c_k to acceptable tolerance. As the eigenvectors x_k and y_k are scaled at each iteration to be RP-compatible, we cannot expect that $\|Cx_k - Cx_{k-1}\|$ is necessarily a diminishing quantity as $k \to \infty$. However, each eigenvector is only unique up to a unimodular scalar, and thus any computed solution to (4.5) trivially provides all the solutions to

$$e^{\mathbf{i}\theta} c_{k-1} = \mathit{\Phi}_m^{-1} C \left(e^{\mathbf{i}\theta} x_{k-1}\right) \tag{4.6}$$

for all $\theta \in [0, 2\pi)$. Subtracting (4.6) from (4.4) and using the Cauchy-Schwartz inequality, we see that

$$\left\| c_k - e^{\mathbf{i}\theta} c_{k-1} \right\| \leq \left\| \mathit{\Phi}_m^{-1} C \right\| \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\| . \tag{4.7}$$

Thus, we can minimize the worst-case of how well $e^{\mathbf{i}\theta} c_{k-1}$ approximates c_k by choosing θ such that the right-hand side of (4.7) is minimized, that is we choose

$$\theta_{\min} = \underset{\theta \in [0,2\pi)}{\arg\min} \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\| . \tag{4.8}$$

We first derive the derivative of the objective of (4.8) with respect to θ:

$$\begin{aligned} \frac{d}{d\theta} \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\| &= \frac{\frac{d}{d\theta} \left(x_k - e^{\mathbf{i}\theta} x_{k-1}\right)^* \left(x_k - e^{\mathbf{i}\theta} x_{k-1}\right)}{2 \left\| x_k - e^{i\theta} x_{k-1} \right\|} \\ &= \frac{\frac{d}{d\theta} \left(2 - e^{-\mathbf{i}\theta} x_{k-1}^* x_k - e^{\mathbf{i}\theta} x_k^* x_{k-1}\right)}{2 \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\|} \\ &= \frac{\mathbf{i} e^{-\mathbf{i}\theta} x_{k-1}^* x_k - \mathbf{i} e^{\mathbf{i}\theta} x_k^* x_{k-1}}{2 \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\|} . \end{aligned} \tag{4.9}$$

Setting $\frac{d}{d\theta} \left\| x_k - e^{\mathbf{i}\theta} x_{k-1} \right\| = 0$ to find the stationary points of the minimization problem

(4.8), we see that the numerator of (4.9) must be 0:

$$\begin{aligned}
\mathbf{i}\mathrm{e}^{-\mathbf{i}\hat{\theta}} x_{k-1}^* x_k - \mathbf{i}\mathrm{e}^{\mathbf{i}\hat{\theta}} x_k^* x_{k-1} = 0 \quad &\implies \quad \mathrm{e}^{2\mathbf{i}\hat{\theta}} = \frac{x_{k-1}^* x_k}{x_k^* x_{k-1}} = \frac{\left(x_{k-1}^* x_k\right)^2}{\left|x_{k-1}^* x_k\right|^2} \\
&\implies \quad \hat{\theta} = \frac{1}{2\mathbf{i}} \log\left(\frac{\left(x_{k-1}^* x_k\right)^2}{\left|x_{k-1}^* x_k\right|^2}\right).
\end{aligned}$$

Since (4.8) is sinusoidal with a period of 2π, computing $\hat{\theta}$ will either yield a minimizer or a maximizer and thus it suffices to set

$$\theta_{\min} = \begin{cases} \hat{\theta} & \text{if } \|x_k - \mathrm{e}^{\mathbf{i}\hat{\theta}} x_{k-1}\| < \|x_k - \mathrm{e}^{\mathbf{i}(\hat{\theta}+\pi)} x_{k-1}\| \\ \hat{\theta} + \pi & \text{otherwise} \end{cases}$$

and to then correspondingly set the initial vector for conjugate gradient to solve (4.4) as

$$\tilde{c}_k := \mathrm{e}^{\mathbf{i}\theta_{\min}} c_{k-1}. \tag{4.10}$$

Analogously, we apply the same procedure to provide the initial vector $\tilde{b}_k$ for iteratively solving $b_k = \Phi_p^{-1} B^* y_k$.

4.2 Accelerating SVSAR's linear convergence

As mentioned previously, the linear convergence of SVSAR can be quite slow in practice, a cost which is amplified by the fact that it is used as a subroutine in both the original GGO algorithm and our new hybrid expansion-contraction scheme. Warm-starting SVSAR can help alleviate these costs, and is done in both methods, but if ε is changing significantly, the warm starts may not be particularly close to a local optimum and thus not helpful for side-stepping the slow convergence.

For the special case of $B = I$, $C = I$, and $D = 0$, a superlinearly converging subspace acceleration method for computing the ε-pseudospectral abscissa was recently presented [KV14], but the technique seems quite difficult to extend to the structured perturbations involved with spectral value sets. Furthermore, this algorithm involves computing the smallest singular value of $A - \mu_k I$ for each $\mu_k \in \mathbb{C}$ that it produces per iteration. To that end, the inverse of $(A - \mu_k I)^{-1}$ is applied to a vector using an LU factorization of $A - \mu_k I$ but the authors note that for some examples, the factorization cost can offset the acceleration gains resulting in an overall slower running time, presumably due to fill-in in the sparse factorizations on those problems. In such cases, they recommend an iterative method to be used instead of LU factorization but such a strategy can present its own problems with respect to reliability and efficiency.

In our own experiments, we have had much success using Aitken's Δ^2 process [Ait26] to produce useful extrapolates $\lambda_{\tilde{\star}}$ from short sequences of consecutive iterates of the λ_k eigenvalues produced by SVSAR that are converging to some locally optimal eigenvalue $\lambda_\star$. However, while extrapolating the eigenvalue itself is an extraordinarily cheap procedure, computing the right and left singular vectors of

$$\|C(\lambda_{\tilde{\star}} I - A)^{-1} B + D\|$$

to recover the corresponding perturbation $u_{\tilde{\star}} v_{\tilde{\star}}^*$ necessary to continue to the next iteration is prohibitively expensive for large-scale problems. Like the authors of [KV14], we too have observed numerous example problems where computing a sparse factorization of $(\lambda_{\tilde{\star}} I - A)$ is costly due to a large amount of fill-in. Indeed, for precisely such reasons, GGO and our algorithm are specifically designed to avoid doing any solves involving

A yet alone solving $(\lambda_{\tilde{\star}} I - A)^{-1}B$ where the number of columns of B may be on the order of n. We thus turn our focus to vector extrapolation methods.

4.2.1 A vector extrapolation based acceleration technique

Given a converging sequence of vectors $\{a_k\}$ with $a_k \in \mathbb{C}^n$ and

$$\lim_{k \to \infty} a_k = a_\star,$$

vector extrapolation methods [GM94, JS00, SFS87] attempt to approximate $a_\star$ by estimating it from a small subset of vectors from the sequence. This can be particularly useful for accelerating many methods where some final $a_\star$ is desired but may only be attained by the expensive and successive computations of each vector a_k in a slowly convergence sequence. Though the sequence of eigenvalues $\{\lambda_k\}$ computed by SVSAR almost always converges, the corresponding eigenvectors x_k and y_k will not necessarily do so, as they are only unique up to a scalar and may be scaled differently at every iteration to satisfy RP-compatibility. Similarly, the associated perturbation vectors u_k and v_k may not form converging sequences either and thus cannot be reliably used for extrapolation.

The perturbation matrices Δ_k, on the other hand, do constitute a convergent sequence with

$$\lim_{k \to \infty} \Delta_k = \Delta_\star = \varepsilon u_\star v_\star^*$$

such that the computed rightmost or largest-modulus eigenvalue $\lambda_\star$ of $A + B\widetilde{\Delta}_\star C$ is locally optimal, for continuous and discrete-time cases respectively, assuming SVSAR converges to a locally rightmost or outermost point. Since ε is constant, we can consider the matrices $\Delta_k = u_k v_k^*$ with ε dropped and the sequence

$$\lim_{k \to \infty} u_k v_k^* = u_\star v_\star^* = \widehat{\Delta}_\star. \tag{4.11}$$

While we could conceivably use (4.11) directly to produce some $\widehat{\Delta}_{\tilde{\star}} \approx \widehat{\Delta}_\star$ from which to recover vectors $u_{\tilde{\star}} \approx u_\star$ and $v_{\tilde{\star}} \approx u_\star$, we do not wish to explicitly form any of the $\widehat{\Delta}_k$ matrices since we desire an algorithm that can scale to cases where both p and m are large. However, since each $\widehat{\Delta}_k$ is rank-1, we can instead extrapolate the matrix sequence implicitly by only extrapolating over a single row and column pair of $\widehat{\Delta}_\star$ and from which we can then recover our extrapolated perturbation vectors $u_{\tilde{\star}}$ and $v_{\tilde{\star}}$.

For ease of notation, let us consider the first extrapolation attempt using the first k iterates produced by SVSAR. Using MATLAB notation, let

$$\begin{aligned} r_\star &= \widehat{\Delta}_\star(i,:) \qquad && \text{for some } i \in \{1,\ldots,p\} \\ c_\star &= \widehat{\Delta}_\star(:,j) \qquad && \text{for some } j \in \{1,\ldots,m\} \end{aligned}$$

be a row and column respectively of the matrix $\widehat{\Delta}_\star$ that we wish to implicitly extrapolate. For reasons that will become clear later, we'd like to choose (i,j) such that $|\widehat{\Delta}_\star(i,j)|$ is large but since we don't want to fully extrapolate $\widehat{\Delta}_\star$, we must instead guess a candidate (i,j) pair determined from the vector sequences that we do have, namely $\{u_1,\ldots,u_k\}$ and $\{v_1,\ldots,v_k\}$. One possible simple heuristic is to assume that u_k and v_k are good approximations of $u_\star$ and $v_\star$ and thus choose

$$i = \operatorname*{arg\,max}_{l \in \{1,\ldots,p\}} |u_k(l)| \quad \text{and} \quad j = \operatorname*{arg\,max}_{l \in \{1,\ldots,m\}} |v_k(l)|.$$

To extrapolate $r_\star$ and $c_\star$, we construct row and column vectors

$$r_k = \widehat{\Delta}_k(i,:) = u_k(i)v_k^*$$
$$c_k = \widehat{\Delta}_k(:,j) = \overline{v_k(j)}u_k$$

and similarly construct vectors $r_1, \dots, r_{k-1}$ and $c_1, \dots, c_{k-1}$. We then apply vector extrapolation to both sequences $\{r_1, \dots, r_k\}$ and $\{c_1, \dots, c_k\}$ to give us $r_{\tilde{\star}}$ and $c_{\tilde{\star}}$. As

$$r_\star = u_\star(i)v_\star^*$$
$$c_\star = \overline{v_\star(j)}u_\star,$$

recovering $u_\star$ and $v_\star$ constitutes finding $p+m$ unknowns while we only have $p+m-1$ equations (since row vector $r_\star$ and column vector $c_\star$ intersect at $\widehat{\Delta}_\star(i,j)$). Thus, we have the freedom to assume $v_\star(j) = 1$. By doing so, and instead using our $r_{\tilde{\star}}$ and $c_{\tilde{\star}}$ extrapolated vectors, we have that

$$u_{\tilde{\star}} = c_{\tilde{\star}}$$

and thus $u_{\tilde{\star}}(i) = c_{\tilde{\star}}(i)$ and so we may also recover:

$$v_{\tilde{\star}} = \frac{r_{\tilde{\star}}}{c_{\tilde{\star}}(i)}.$$

As we must divide by $c_{\tilde{\star}}(i) = \widehat{\Delta}_{\tilde{\star}}(i,j)$, it is now clear why we wish to choose (i,j), such that $|\widehat{\Delta}_{\tilde{\star}}(i,j)|$ is large.

Though we have now recovered a pair of vectors $u_{\tilde{\star}}$ and $v_{\tilde{\star}}$ such that $u_{\tilde{\star}}v_{\tilde{\star}}^* = \widehat{\Delta}_{\tilde{\star}}$, we have no guarantee that $\|\widehat{\Delta}_{\tilde{\star}}(i,j)\| = 1$. Therefore, we must always normalize $u_{\tilde{\star}}$ and $v_{\tilde{\star}}$

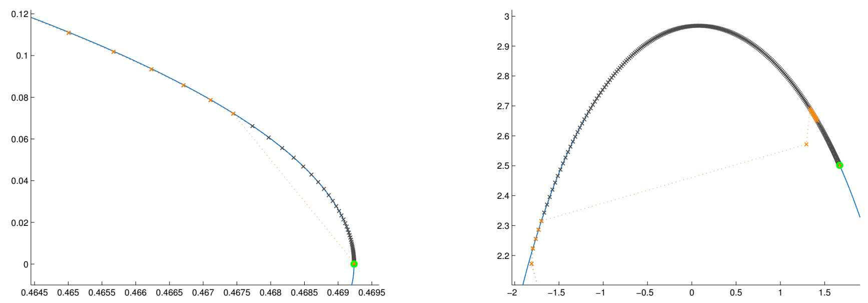

FIGURE 4.1: *Left: example* ROC3 (*continuous-time*). *Right: example* ROC5 (*discrete-time*). *Each panel depicts a region of the spectral value set boundary of each problem for perturbation level* $\varepsilon = 10^{-2}$ *in blue. The iterates of* SVSAR *without vector extrapolation enabled are plotted as dark grey* $\times$*'s while the iterates of* SVSAR *with vector extrapolation enabled are plotted on top in orange* $\times$*'s. For each panel, the final point that both variants converge to is indicated by the green circle.*

to each be unit norm, that is

$$u_{\star} := \frac{u_{\tilde{\star}}}{\|u_{\tilde{\star}}\|} \quad \text{and} \quad v_{\tilde{\star}} := \frac{v_{\tilde{\star}}}{\|v_{\tilde{\star}}\|}. \tag{4.12}$$

Provided that the extrapolated perturbation indeed does provide progress, we then warm start SVSAR with the vectors of (4.12) until SVSAR completes another k more iterations. If the extrapolated perturbation does not yield an improvement, it is discarded and SVSAR is allowed to complete another k iterations continuing from u_k and v_k. In either case, an extrapolation can be attempted every k iterations of SVSAR. We note that it is often better to discard the initial few iterates of SVSAR before attempting extrapolation since those steps are typically quite large and yield poor extrapolations.

For computing the vector extrapolations, we use the minimum polynomial extrapolation (MPE) method of [CJ76], whose main cost is solving an over-determined linear least squares problem. Extrapolating a row and column from k implicitly formed rank-1 matrices requires $\mathcal{O}((p+m)k^2)$ flops, and as k will typically be small, such as 10, the method can scale to problems where D has large dimensions as well as the special pseudospectral case when $B = I$, $C = I$, and $D = 0$ and $u_k, v_k \in \mathbb{C}^n$. In Figure 4.1, we see that the vector extrapolation technique on sets of five vectors at a time substantially accelerates SVSAR, reducing the number of iterations from 136 to 18 on the continuous-time problem `ROC3` and from 548 to 28 on the discrete-time problem `ROC5`.

4.2.2 A comparison with subspace acceleration for pseudospectra

For the special case of pseudospectra, that is problems where $B = C = I$ and $D = 0$, the vector extrapolation technique for accelerating SVSAR can still be computed efficiently, namely with only $\mathcal{O}(nk^2)$ operations where k is the number of iterates used to per extrapolation. As such, we now empirically investigate its performance compared to `subspace_pspa`[1], the implementation of the subspace acceleration method presented by [KV14] to accelerate the pseudospectral abscissa or radius approximation algorithm of [GO11], which itself is a special case and a precursor to the SVSAR method.

While the subspace acceleration method provides a provable superlinear convergence rate, in [KV14] the authors actually observe local quadratic convergence rates in practice. However, it is unclear if the technique can be extended beyond the special case of pseudospectra to generic spectral value sets and indeed this appears to be a quite difficult problem. Furthermore, as previously mentioned, the subspace acceleration method relies on being being able to quickly solve linear systems at every iteration and depending on the problem, this can sometimes be so prohibitively expensive that it negates any benefit of the subspace acceleration. While the vector extrapolation tech-

[1]Due to an apparent bug in both the 0.1 and 0.2 versions of `subspace_pspa` [Sps], the routine appears to erroneously return an incorrect value for its computed pseudospectral abscissa approximation. However, this bug appears to only be limited to the return value while the correct approximation is obtainable via the history of iterates that is also optionally passed back to the user, which is the value we use for the experiments here. See Appendix A for more details.

nique we propose does not provide any guarantees, in practice we find that it is typically highly effective at mitigating the slow linear convergence of SVSAR.

For the comparison, we use the same problems used in [KV14] for the test suite, with the exception of the particular dimension used for `sparserandom`. Many of the problems tested in [KV14] were chosen as to replicate the test set from [GO11] but with the discrepancy of 1000 being used in lieu of 10000 for `sparserandom`, as was done in [GO11]. Since the larger dimensional problem is more difficult and to be consistent with the earlier source, we have chosen to use 10000 for `sparserandom`.

In an attempt to improve the test suite for the purposes of evaluating the effectiveness of vector extrapolation, an adversarial initialization strategy was chosen to try to increase the likelihood that SVSAR would experience slow convergence on many of the problems, in contrast to the typical initialization procedure used for the experiments in [KV14] and discussed in [GO11]. To that end, each problem was warm-started from a randomly generated complex-valued pair of vectors u_0 and v_0 using `randn` twice to generate the real and imaginary parts, normalized so that $\|u\| = \|v\| = 1$. Thus, the algorithms were initialized at some $\lambda_0 \in \mathbb{C}$ such that λ_0 is a rightmost eigenvalue of $A + \varepsilon u_0 v_0^*$ for the specified perturbation level ε, with the idea being that SVSAR might be more likely to first encounter the boundary of the pseudospectrum far from a locally rightmost point, and thereby incur an increased number of iterations until convergence. For the purposes of a fair comparison, `subspace_pspa` was also warm-started from the same point, that is `opts.start_z =` λ_0. This adversarial initialization strategy proved effective in increasing the SVSAR's iteration count on many of the problems tested while in constrast, it probably had a little effect on the convergence behavior of `subspace_pspa`. Furthermore, it likely also disproportionally increased the FLOP count for the SVSAR algorithm compared to `subspace_pspa`, due to the increased operations necessary to handle complex matrices over real ones at every iteration.

The 0.2 version of `subspace_pspa` was used for the experiments with a max iteration count set to 1000 iterations, a termination tolerance of 10^{-12} and the following default parameters specified: `opts.exact = false`, `opts.restart = 30`, and `opts.sparse = issparse(A)`. SVSAR was also allowed up to 1000 iterations with an absolute step size termination condition of 10^{-12} and correspondingly only invoked `eigs` in lieu of `eig` when `issparse(A)` held. For the sparse problems, `eigs` was set with `k = 1`, that is to request only one single eigenvalue, to be consistent with the experiments of [KV14].

Remark 4.1. *In contrast to the experiments performed here, we note that it is typically safer when using* SVSAR *to instead request a handful of eigenvalues from* `eigs`*, though this does increase the running time of the algorithm. If only one eigenvalue is requested, it is more likely that the rightmost eigenvalue will not be returned, since it typically won't be the first to converge in the Krylov subspace used by* `eigs`*. Furthermore, requesting more than one eigenvalue to converge from* `eigs` *also increases the probability that a proper match will be found when computing the right and left eigenvectors at every iteration of* SVSAR*, which involves two separate calls to* `eigs`*.*

Unlike the results in [KV14] where `psapsr` was used, the results here neither use the `psapsr` code implementing the algorithm of [GO11] nor use the `svsar` routine included with `hinfnorm` [Hin]. Instead, the routines from the new hybrid expansion-contraction (HEC) codebase were employed for these experiments, which is a complete rewrite of `psapsr` and `svsar` (see Chapter 5.1 for details). As the implementation of HEC uses an object-oriented design intended for large-scale use, the interpretation speed of the code may be noticeably slower than either `psapsr` and `svsar` on small-scale test problems.

As a result of the differences in both initialization and the different codes used, the relative timing results reported here for SVSAR compared to the running times for subspace_pspa are not necessarily comparable to the results reported in [KV14]. Additionally, the timings and iteration counts for all the algorithms reported here do not include the cost of computing λ_0 for initialization.

Regarding the new vector extrapolation technique for accelerating SVSAR, experiments were run with it completely disabled as a baseline and then with it enabled where vector extrapolation was attempted every five iterates, using only the previous five iterates. Attempting extrapolation every five iterates, using the previous 10 iterates, as well as attempting extrapolation every 10 iterates, using the previous 20 iterates, were also explored. While these variants were also successful in providing acceleration, in general they were not as effective as extrapolating every five iterates using just the previous five iterates (probably due to their decreased opportunity to extrapolate) and we thus do not report the detailed results.

Besides reporting the running times and iteration counts for each algorithm, we address the accuracy of their solutions. For the small-scale problems in Tables 4.1 and 4.2, where it is computationally tractable to compute the true value of the pseudospectral abscissa via the criss-cross algorithm [BLO03], we use pspa [Psp] to assess the relative differences in the computed solutions by each algorithm from the actual value given by pspa. For the large-scale problems in Table 4.3 where it is not feasible to use pspa, we instead report per problem relative differences for each algorithm's solution compared to the largest value reported for the pseudospectral abscissa amongst the three algorithms.

Fom the results contained in Tables 4.1 for the special case of pseudospectra, we

SVSAR Acceleration Comparison: Small-scale $\|\Delta\| = 10^{-2}$									
	SVSAR			SVSAR + E			SPSA		
Problem ($n = 100$)	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.
`airy(n)`	-2×10^{-12}	12	0.91	-3×10^{-13}	12	1.06	-3×10^{-13}	4	0.16
`basor(n)`	-6×10^{-15}	5	0.44	-6×10^{-15}	5	0.43	-8×10^{-15}	3	0.15
`boeing('O')`	-6×10^{-11}	285	7.11	-2×10^{-12}	125	3.68	5×10^{-12}	17	0.54
`boeing('S')`	-1×10^{-11}	102	3.18	-1×10^{-11}	102	3.57	6×10^{-12}	15	0.44
`chebspec(n)`	-1×10^{-12}	64	5.64	4×10^{-12}	17	2.46	1×10^{-11}	9	0.31
`convdiff(n)`	-1×10^{-11}	204	15.07	-4×10^{-12}	27	3.16	-6×10^{-13}	7	0.24
`davies(n)`	-1×10^{-15}	2	0.14	-1×10^{-15}	2	0.15	-1×10^{-15}	2	0.11
`frank(n)`	-2×10^{-12}	59	6.55	-2×10^{-16}	11	1.57	6×10^{-14}	9	0.30
`gaussseidel({n,'C'})`	-2×10^{-12}	32	4.81	-1×10^{-12}	19	3.31	2×10^{-15}	3	0.18
`gaussseidel({n,'D'})`	-3×10^{-12}	55	8.26	3×10^{-15}	22	4.35	0	7	0.35
`gaussseidel({n,'U'})`	-3×10^{-11}	487	76.35	-1×10^{-11}	33	5.92	4×10^{-15}	6	0.25
`grcar(n)`	-2×10^{-11}	424	44.14	-8×10^{-12}	28	3.45	-5×10^{-15}	7	0.34
`hatano(n)`	-4×10^{-13}	15	1.70	-7×10^{-15}	12	1.60	4×10^{-15}	5	0.18
`kahan(n)`	-2×10^{-12}	51	5.07	-1×10^{-13}	18	2.06	-3×10^{-15}	6	0.25
`landau(n)`	-2×10^{-14}	5	0.27	-2×10^{-14}	5	0.28	-7×10^{-16}	3	0.11
`orrsommerfeld(n)`	-3×10^{-11}	58	6.26	-2×10^{-11}	28	4.20	-2×10^{-11}	8	0.54
`random(n)`	-3×10^{-15}	7	1.19	-3×10^{-15}	7	1.38	-4×10^{-15}	4	0.29
`randomtri(n)`	-3×10^{-12}	62	8.41	-4×10^{-12}	56	8.81	1×10^{-15}	9	0.49
`riffle(n)`	-1×10^{-12}	37	2.75	-2×10^{-12}	25	2.15	-6×10^{-15}	6	0.25
`transient(n)`	-5×10^{-11}	168	28.01	-9×10^{-12}	14	2.66	-2×10^{-15}	8	0.54
`twisted(n)`	-2×10^{-15}	5	0.71	-2×10^{-15}	5	0.72	2×10^{-15}	4	0.26

TABLE 4.1: SVSAR *denotes the implementation from the* HEC *code while* E *denotes that vector extrapolation is additionally enabled.* SPSA *denotes the 0.2 version of* `subspace_pspa`. Rel. Diff. *denotes the relative difference of each algorithm's computed value calculated compared to the true value calculated by* `pspa.m`.

generally conclude that the SVSAR vector extrapolation method is indeed highly effective at mitigating the potential cost of the slow linear convergence rate of the subroutine and we note that we typically observe such beneficial effects across most of the problems. However, vector extrapolation still does not quite meet the stellar convergence rate of the subspace acceleration method suggesting that extension of the subspace acceleration technique to the case of generic spectral value sets, if possible, could have substantial impact on improving SVSAR, even compared to when vector extrapolation is enabled.

SVSAR Acceleration Comparison: Small-scale $\|\Delta\| = 10^{-4}$									
	SVSAR			SVSAR + E			SPSA		
Problem	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.
`airy(100)`	-1×10^{-13}	3	0.32	-1×10^{-13}	3	0.33	-3×10^{-13}	4	0.28
`airy(200)`	-2×10^{-13}	3	1.35	-2×10^{-13}	3	1.35	1×10^{-11}	5	1.46
`airy(300)`	-3×10^{-13}	3	3.41	-3×10^{-13}	3	3.41	1×10^{-11}	4	2.74
`grcar(100)`	-3×10^{-11}	429	45.28	-1×10^{-12}	22	2.75	2×10^{-14}	10	0.37
`grcar(200)`	-5×10^{-11}	855	430.07	-2×10^{-14}	33	20.10	-2×10^{-14}	8	1.07
`grcar(300)`	-6×10^{-10}	1000	1218.89	-8×10^{-14}	33	54.77	-4×10^{-14}	8	2.60
`landau(100)`	-3×10^{-15}	2	0.10	-3×10^{-15}	2	0.11	-4×10^{-16}	2	0.08
`landau(200)`	-3×10^{-15}	2	0.63	-3×10^{-15}	2	0.65	9×10^{-16}	2	0.40
`landau(300)`	8×10^{-16}	3	2.20	8×10^{-16}	3	2.20	2×10^{-15}	3	1.11

TABLE 4.2: SVSAR *denotes the implementation from the* HEC *code while* E *denotes that vector extrapolation is additionally enabled.* SPSA *denotes the 0.2 version of* `subspace_pspa`. Rel. Diff. *denotes the relative difference of each algorithm's computed value calculated compared to the true value calculated by* `pspa.m`.

Still, in Table 4.3, we see that the potential expense of solving a large linear system can be problematic. For `sparserandom(10000)`, the cost of the sparse LU is so significant that `subspace_pspa` ends up being over 558 times slower than SVSAR. Even in the experiments of [KV14], where a dimension of 1000 was used for this problem, `subspace_pspa` was still reported to be about 10 times slower than the SVSAR algorithm. Of course, `sparserandom` is an extremely poor candidate for efficient sparse LU factorization, as its structure is generated randomly and thus fill-in is high. However, the authors of [KV14] also report faster running times for SVSAR compared to `subspace_pspa` on problem `skewlap3d(30)`. Though the results in Table 4.3 do not immediately reflect that conclusion, that appears to only be a consequence of the adversarial initialization strategy used to increase the number of SVSAR iterations for the test set. In Table 4.3, we indeed see that `subspace_pspa` actually incurs a much higher cost with respect to CPU time per iteration compared to SVSAR, indicating that a sparse LU factorization is also somewhat expensive for `skewlap3d(30)`.

SVSAR Acceleration Comparison: Large-scale $\|\Delta\| = 10^{-2}$									
	SVSAR			SVSAR + E			SPSA		
Problem	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.	Rel. Diff.	Iters.	Sec.
`dwave(2048)`	-4×10^{-12}	41	12.69	0	30	11.29	-3×10^{-15}	3	1.90
`convdiff_fd(10)`	-3×10^{-11}	304	78.21	-3×10^{-12}	60	30.27	0	9	0.82
`markov(100)`	-2×10^{-7}	1000	1297.27	-2×10^{-10}	46	74.05	0	6	6.02
`olmstead(500)`	0	3	3.51	-5×10^{-14}	3	3.52	-9×10^{-14}	2	0.44
`pde(2961)`	-9×10^{-14}	11	3.18	-9×10^{-14}	11	3.81	0	4	2.79
`rdbrusselator(3200)`	-1×10^{-13}	4	3.95	0	4	3.18	-8×10^{-14}	2	2.37
`sparserandom(10000)`	-2×10^{-15}	5	1.27	0	5	1.02	-1×10^{-14}	4	569.54
`skewlap3d(30)`	-4×10^{-10}	200	859.38	0	27	193.42	-3×10^{-11}	10	128.80
`supg(20)`	-5×10^{-11}	119	5.41	-6×10^{-12}	69	3.79	0	6	0.12
`HF2D9`	-2×10^{-13}	3	1.41	0	3	1.63	-2×10^{-13}	2	1.02
`HF2D_IS2_M529`	0	3	0.28	-3×10^{-15}	3	0.27	-2×10^{-14}	3	0.15
`HF2D_IS4_M484`	-8×10^{-15}	3	0.15	0	3	0.26	-9×10^{-15}	2	0.07
`HF2D_CD3_M576`	-1×10^{-15}	3	0.25	-2×10^{-15}	3	0.25	0	3	0.14
`HF2D_IS4`	-1×10^{-14}	2	1.05	-1×10^{-14}	2	1.11	0	2	1.09
`HF2D_CD3`	-2×10^{-15}	3	2.63	0	3	2.57	-1×10^{-14}	3	2.29

TABLE 4.3: SVSAR *denotes the implementation from the* HEC *code while* E *denotes that vector extrapolation is additionally enabled.* SPSA *denotes the 0.2 version of* `subspace_pspa`. Rel. Diff. *denotes the relative difference of each algorithm's computed value calculated compared to the largest value computed by the three variants.*

In Table 4.1, we observe that `boeing('O')` and `boeing('S')` seem to be particularly challenging problems. For `boeing('O')`, enabling vector extrapolation is only able to reduce the 285 iterations normally needed for SVSAR to converge to 125, which is a considerable improvement but nonetheless still a quite large number of iterations. Even worse is the performance on `boeing('S')`, where vector extrapolation fails to eliminate even one of the 102 iterations of SVSAR. By comparison, `subspace_pspa` is able to converge to the same level of accuracy in just 17 and 15 iterations respectively for these two examples. As we observe in Figure 4.2, the inability of vector extrapolation to be very effective for these problems seems to stem from the fact the sequence of iterates produced by SVSAR is both oscillating and quite irregular. Unfortunately simple smoothing measures, such as averaging consecutive pairs of vectors or only using the odd or even numbered vectors for extrapolation, were completely

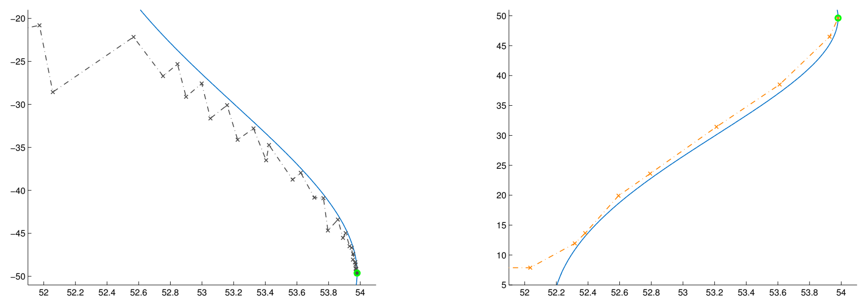

FIGURE 4.2: *Left: Iterates of* SVSAR *on* `boeing('0')` *in grey. Right: Iterates of* `subspace_pspa` *on* `boeing('0')` *in orange. Each panel depicts the region of the pseudospectral boundary for perturbation level $\varepsilon = 10^{-2}$ in blue while the final point that each method converges to is indicated by a green circle. Note that the methods converged to conjugates of each other and thus returned the same value for the pseudospectral abscissa.*

ineffective in improving the extrapolation results for either Boeing problem.

4.3 Improvements to the line search for monotonicity

Recall the continuous matrix family

$$N(t) = A + B\widetilde{\Delta}(t)C \quad \text{where} \quad \widetilde{\Delta}(t) = \frac{\varepsilon u(t)v(t)^*}{1 - \varepsilon v(t)^* D u(t)}$$

with

$$u(t) = \frac{tu_k + (1-t)u_{k-1}}{\|tu_k + (1-t)u_{k-1}\|} \quad \text{and} \quad v(t) = \frac{tv_k + (1-t)v_{k-1}}{\|tv_k + (1-t)v_{k-1}\|},$$

and $u_k, u_{k-1} \in \mathbb{C}^p$ and $v_k, v_{k-1} \in \mathbb{C}^m$ are all fixed. Let $(\lambda_{k-1}, x_{k-1}, y_{k-1})$ be an RP-compatible eigentriple where λ_{k-1} is rightmost eigenvalue of $N(0)$ and let $\lambda_{k-1}(t)$ defined on the interval $[0,1]$ be the continuous path of eigenvalues of $N(t)$ emanating from $\lambda_{k-1}(0) = \lambda_{k-1}$ and ending at some point $\lambda_{k-1}(1)$. By standard eigenvalue perturbation theory we have

$$\lambda_{k-1}'(0) := \left.\frac{d\lambda_{k-1}(t)}{dt}\right|_{t=0} = \frac{y_{k-1}^* N'(0) x_{k-1}}{y_{k-1}^* x_{k-1}} = \frac{\psi_k}{y_{k-1}^* x_{k-1}},$$

where ψ_k is defined in (2.12). Recalling that the sign of ψ_k can be flipped by flipping both the signs of u_k and v_k, assuming $\psi_k \neq 0$, it is ensured that one can always find a value $t_{\mathrm{m}} \in (0,1)$ such that $\operatorname{Re}(\lambda_{k-1}(t_{\mathrm{m}})) > \operatorname{Re}(\lambda_{k-1}(0))$ holds, thereby ensuring the monotonicity of the iterates of SVSAR.

One practical concern not addressed in [GGO13] is that if $\operatorname{Re}(\psi_k)$ defined is small in magnitude, its sign may not be accurate and thus Procedure 3 may subsequently fail no matter how much it reduces t from one to zero. If Procedure 3 fails or perhaps even just fails in the first few reductions of t, it seems prudent to just reattempt the line search with $u_k = -u_k$ and $v_k = -v_k$ rather than rely on some arbitrarily prescribed tolerance on $\operatorname{Re}(\psi_k)$ when it is too small for its sign to be trusted. In practice, it is more efficient to perform both line searches in parallel and to terminate as soon as one of the searches yields a rightward step for some reduced value of t.

A second practical concern is that the backtracking bisection strategy of Procedure 3 can be slow. It is first worth noting that the line search is typically not invoked for most of the iterates of SVSAR, if at all, depending on the problem. Most often, the line search is only called on the last few iterates or even just the last iterate, where the failure of the line search indicates that SVSAR has indeed converged to a locally rightmost point. As a consequence, the cost of the backtracking line search is usually not a dominant factor in the running time of SVSAR. Nonetheless, it is still a cost that can most likely be mitigated by using a quadratic or cubic line search model [NW06, Chapter 3.5] to dynamically determine the amount of reduction for t at each iteration in Procedure 3 in lieu of the bisection strategy. That is, let

$$f(t) \coloneqq \operatorname{Re}(\lambda_{k-1}(t))$$

be defined for $t \in [0,1]$ and let $\{t_k\}$ denote the sequence of t values computed by Procedure 3. At the start of the routine, $t_1 = 1$ and we correspondingly have the values of $f(0)$, $f'(0)$ and $f(1)$. In lieu of a bisection step, we can alternatively construct an interpolating quadratic function and set t_2 to its maximizer. In practice, we must check that the maximizer resides in $(0,1)$ and resort to a bisection otherwise, which can happen if ψ_k is very small in magnitude and possibly has an incorrect sign. If $f(t_2) > 0$ does not hold then, at every subsequent iteration, the values of $f(0)$, $f'(0)$, $f(t_{k-1})$ and $f(t_{k-2})$ are available and we can either construct an interpolating quadratic using $f(0)$, $f'(0)$ and $f(t_{k-1})$ or alternatively, an interpolating cubic using $f(0)$, $f'(0)$, $f(t_{k-1})$ and $f(t_{k-2})$, and subsequently take the best maximizer of either that resides in $(0,1)$ for t_k. We present brief results in Section 5.3.3 comparing the effect of using cubic models or standard bisection steps in the line search.

Remark 4.2. *Note the goal of the line search is to merely find any acceptable* $t_\mathrm{m} \in (0,1)$ *such that* $\operatorname{Re}(\lambda_{k-1}(t_\mathrm{m})) > \operatorname{Re}(\lambda_{k-1}(0))$ *holds, as opposed to trying to maximize* $\operatorname{Re}(\lambda_{k-1}(t))$. *It is unclear whether there would be a net benefit to maximizing* $\operatorname{Re}(\lambda_{k-1}(t))$ *since the next step of* SVSAR *may produce a more rightward point, possibly without requiring the line search.*

Remark 4.3. *It is also possible to construct a cubic line search search model for the initial reduction step but it requires that we derive an analogous value to* ψ_k *for* $N'(1)$ *to provide the value of* $f'(1)$, *which is a is a straightforward calculation since it is symmetric in the sense that derivative is taken at* λ_k *instead of* λ_{k-1}. *Furthermore, we* *may alternatively construct cubic models using $f(t_{k-1})$ and $f'(t_{k-1})$ at any iteration in the line search, as opposed to using $f(t_{k-1})$ and $f(t_{k-2})$ and it is possible that these would be better models. However, obtaining $f'(t)$ for $t \in (0,1)$ is somewhat more complicated algebraically. As we feel that using already quadratic or cubic models as stated in lieu of bisection in the line search is already a rather minor optimization, we do not investigate either of these additional options here.*

4.4 A randomized variant of SVSAR

Let $\varepsilon \in \mathbb{R}^{++}$ with $\varepsilon\|D\| < 1$ and $u_{k-1} \in \mathbb{C}^p$ and $v_{k-1} \in \mathbb{C}^m$ with $\|u_{k-1}\| = \|v_{k-1}\| = 1$ and let $(\lambda, x_{k-1}, y_{k-1})$ be an RP-compatible eigentriple of $M(\varepsilon u_{k-1}v_{k-1}^*)$. Given another fixed pair of vectors $u_k \in \mathbb{C}^p$ and $v_k \in \mathbb{C}^m$ defining $M(\varepsilon u_k v_k^*)$ with $\|u_{ks}\| = \|v_k\| = 1$, we note that ψ_k as defined in (2.12) allows for an exceptionally cheap computation of the derivative of real part of λ as it is "pushed" by smoothly changing its corresponding matrix $M(\varepsilon u_{k-1}v_{k-1}^*)$ to $M(\varepsilon u_k v_k)$ (see Section 2.2 for details). As a consequence, a natural question to ask is whether Procedure 2 is actually necessary for a viable SVSAR method. That is, given u_{k-1} and v_{k-1}, we may instead consider randomly sampling rank-1 updates (normalized so that each random vector has norm one) and take the one which most maximizes $|\psi_k|$. It seems likely that doing so won't necessarily produce an as large of step as compared to what is produced by Procedure 2 but as long as ψ_k is not zero for at least one of the randomly generated samples, then progress can still be made. Though we expect this might be a slow method, it may however be quite robust. Furthermore, we can typically evaluate ψ_k many many times and all more cheaply than doing a single eigenvalue solve.

We briefly consider this approach, presented here for two selected small randomly generated examples, as shown in Figure 4.3, though we have successfully used a randomized approach on many more examples. The only difference between the regular SVSAR method and the randomized version is that Procedure 2 is replaced by a loop that randomly generates rank-1 perturbations of unit norm and evaluates ψ_k, and then returns the best perturbation found, that is, the one which maximizes $|\psi_k|$. We attempted various number of samples to be generated and found that even as few as 20 samples was often enough to achieve high accuracy for converging to a locally rightmost point. However, the randomized method can be excruciatingly slow and often requires that the step size tolerance be set extremely low (e.g. to machine precision) for the randomized method to achieve full accuracy (though occasionally the randomized method is more accurate than the regular method). As we can visually gather from Figure 4.3, the randomized method is indeed taking much smaller and more numerous steps before terminating. Interestingly though, in both examples, we observe that the randomized method is actually able to push out of stationary points, even a locally rightmost point, though this latter case is quite exceptional. Though not shown, we have also observed cases where the regular SVSAR method can stagnate at a point in the interior of the spectral value set while the randomized version can reliably push beyond that same point. It thus suggests that although the randomized variant is slow, it might perhaps be used *sparingly* in concert with the regular SVSAR method to increase robustness and encourage convergence to globally rightmost points.

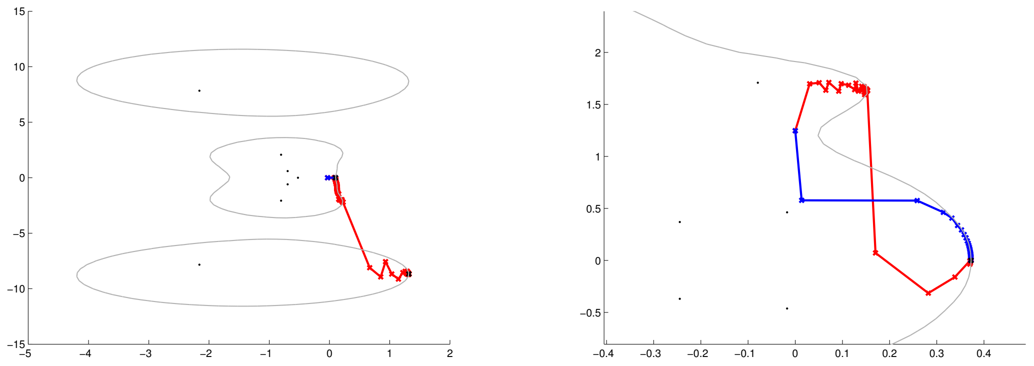

FIGURE 4.3: *Each panel shows a small randomly generated continuous-time spectral value set example, with the left panel showing a spectral value set for real-valued system and the right panel showing a portion of the spectral value set for a complex-valued system. The spectral value set boundaries are shown in grey and the eigenvalues of A are shown as black dots. The blue lines depict the progress of the regular SVSAR method while the red lines depict the progress of the randomized SVSAR routine. The crosses for each depict the actual iterates of both methods.*

5 HYBRID EXPANSION-CONTRACTION IN PRACTICE

5.1 Implementation

Our new hybrid expansion-contraction algorithm, along with the vector extrapolation acceleration technique and the ability to scale up to large-dimensional D matrices, has been implemented from scratch in MATLAB. There are also several additional optimizations realized in the new code as well. The first improvement is that the code is now written such that it stores the state of all relevant computations so that none of them need to be redone, for instance when switching between contraction and expansion. Furthermore, our new code minimizes the number of matrix-vector products, most notably by precomputing Bu_k and v_k^*C and then utilizing the resulting vectors in the function that multiplies a vector by $M(\varepsilon u_k v_k^*)$ to be used with an iterative eigenvolver such as `eigs` so that we don't form the matrix explicitly, which would cause fill-in. Thus, the number of matrix-vector products involving B, C and D is typically about double the number of eigentriples calculated during the course of the algorithm. By contrast, for `hinfnorm`, the number of multiplies involving matrices B, C and D is the same as the number of multiplies with A, which is approximately the sum of the iterations each call to `eigs` requires to converge for every eigentriple calculated. As the optimizations in Section 4.1 allow the algorithm to scale to cases where p, m are both large, the reduction in matrix-vector products should provide an additional tangible benefit for high-dimensional problems and accordingly, our new code now supports

A, B, C and D to each be given as function handles. Lastly, as the algorithm is generating a converging sequence of matrices, our new code now optionally supports recycling the previous eigenvectors corresponding to the rightmost (or outermost) eigenvalue, or if $k > 1$ eigenvalues have been requested, one may also choose to use the average of all the eigenvectors returned by `eigs` as the initial vector for the next eigenproblem to solve. Either option can potentially help reduce the number of iterations required in the implicitly restarted Arnoldi method used by `eigs`.

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

Another departure from `hinfnorm` is that we use our implementation of a hybrid Newton-bisection routine for the contraction procedure instead of `rtsafe`, though we use the same heuristic from `rtsafe` to choose when to switch to a bisection step.

5.2 Tolerances

We now describe the main tolerance criteria necessary to realize a practical implementation of hybrid expansion-contraction. For Procedure 4, we set the expansion to terminate if $\operatorname{Re}(\lambda_k) - \operatorname{Re}(\lambda_{k-1}) < \tau_{uv} |\operatorname{Re}(\lambda_k)|$, for some small user-provided tolerance $\tau_{uv} \in \mathbb{R}^{++}$, where λ_k are iterates of Procedure 4 in this context. Thus, once the step size for updating vectors u_k, v_k shrinks to a negligible amount, contraction will begin again. For the contraction-phase, since hybrid expansion-contraction requires all iterates to remain strictly in the right half-plane, finding the root of $\alpha(M_{uv}(\cdot))$ to some tolerance is potentially problematic since it could result in a solution just slightly in the left half-plane. To address this, we instead have the Newton-bisection routine find the root $\hat{\varepsilon}_k$ of $\alpha(M_{uv}(\cdot)) - 0.5\tau_\varepsilon$ for some small user-provided tolerance $\tau_\varepsilon \in \mathbb{R}^{++}$ and have it terminate when $|\alpha(M_{uv}(\hat{\varepsilon}_k)) - 0.5\tau_\varepsilon| < 0.5\tau_\varepsilon$. By doing so, we ensure a contracted value $\hat{\varepsilon}_k$ such that $0 < \alpha(M_{uv}(\hat{\varepsilon}_k)) < \tau_\varepsilon$. Furthermore, we also set the Newton-bisection iteration to terminate the contraction process if $\hat{\varepsilon}_k < \tau_\varepsilon \varepsilon_k$ and $\alpha(M_{uv}(\hat{\varepsilon}_k)) > 0$ is satisfied. Lastly, the contraction routine keeps track of the most contracted value of $\hat{\varepsilon}_k$ such that $\alpha(M_{uv}(\hat{\varepsilon}_k)) > 0$ is satisfied and will return that best encountered value of $\hat{\varepsilon}_k$. In the case that all the subsequent iterates in the Newton-bisection iteration are in the left half-plane though possibly closer to the imaginary axis, hybrid expansion-contraction can at least continue with some amount of contraction while remaining in the right half-plane, even if its termination tolerances were not satisfied.

The hybrid expansion-contraction algorithm described in Procedure 5 is set to terminate once it can no longer make progress contracting and expanding via tolerances τ_ε and τ_{uv} respectively. We thus terminate hybrid expansion-contraction when either both methods fail to make any progress consecutively, in either order, or if for λ_{uv} produced by SVSAR, $\operatorname{Re}(\lambda_{uv}) < \tau_\varepsilon + \tau_{uv}$ holds. The latter condition is necessary since SVSAR will typically always at least take a single step, assuming the line search of Procedure 3 doesn't fail. If we were to set the tolerance condition any tighter, hybrid expansion-contraction might take a long sequence of alternating expansion-contraction steps where SVSAR only is able to take a single step of exceedingly small step size less than τ_ε while the contraction phases fail to reduce ε_k any further.

Remark 5.1. *The tolerance criteria for the discrete-time case is described analogously by replacing* $\operatorname{Re}(\lambda_k) - \operatorname{Re}(\lambda_{k-1}) < \tau_{uv}|\operatorname{Re}(\lambda_k)|$ *by* $|\lambda_k| - |\lambda_{k-1}| < \tau_{uv}|\lambda_k|$*, changing the condition* $\alpha(\cdot) > 0$ *to* $\rho(\cdot) > 1$*, and finally changing* $\operatorname{Re}(\lambda_{uv}) < \tau_\varepsilon + \tau_{uv}$ *to* $|\lambda_{uv}| < 1 + \tau_\varepsilon + \tau_{uv}$.

5.3 Numerical results

In order to evaluate hybrid expansion-contraction relative to version 1.02 of `hinfnorm`, we ran experiments using the small and large-scale test sets contained in [GGO13] and present the data in the aggregate. Of these 34 small-scale continuous and discrete-time problems, we discarded the results from `ROC2` due to `hinfnorm` encountering a platform-specific bug with `eig` (not `eigs`) when running on this particular problem. Though we were able to run `hinfnorm` successfully on `ROC2` using a Mac, the timings would not have been comparable to the rest of the data computed on our Linux machine. As a consequence, the test sets we use here comprise a set of 33 small-scale problems and a second set of 14 large-scale problems. All experiments were done using MATLAB R2014a running on a single-user desktop with Ubuntu 14.04 (64-bit) and an Intel i7-3770K CPU with 8 GB of RAM.

For our hybrid expansion-contraction code, which we abbreviate HEC, we used $\tau_\varepsilon = 10^{-10}$ and $\tau_{uv} = 10^{-12}$ for its termination tolerances. Correspondingly for `hinfnorm`, we reused τ_ε for `rtsafe`'s tolerance and τ_{uv} for SVSAR. We allowed HEC to take a maximum of 100 iterations to find an upper bound using Procedure 6 and similarly let the hybrid expansion-contraction convergent phase also have up to 100 iterations. As `hinfnorm` doesn't provide a user option to individually set max iteration limits for the upper bound and Newton-bisection phases separately, we set its max iteration limit to 200. Both HEC and `hinfnorm` were set such that SVSAR could take up to 1000 iterations per call.

Remark 5.2. *The* `rtsafe` *code used by* `hinfnorm` *to perform the Newton-bisection iteration makes no check upon how close the function value is to zero. Instead, it merely terminates once progress in ε_k has slowed but this is no guarantee that* `hinfnorm` *has converged to a point on or acceptably near the imaginary axis. As a consequence,* `hinfnorm` *may terminate at a point significantly farther away from the imaginary axis and we observe this in practice on some problems. For evaluative purposes, it isn't so problematic if* `hinfnorm` *terminates while at a locally rightmost point in the right half-plane, since doing so will simply lower the reported approximation to $\|G\|_\infty$ and thus its results reflect that is hasn't converged in these cases. If it instead terminates at a locally rightmost point that is in the left-plane to a significant degree,* `hinfnorm` *will incorrectly report a value of $\|G\|_\infty$ that may in fact be too large. Thus, we consider* `hinfnorm` *to have failed if its last locally rightmost point λ_k found satisfies $\operatorname{Re}(\lambda_k) < -100\tau_\varepsilon$ (or $|\lambda_k| < 1 - 100\tau_\varepsilon$ for the discrete-time problems). We could have modified* `hinfnorm` *using analogous conditions to the ones described in Section 5.1 for* HEC *but that would have been at odds with our goal of making a baseline comparison with* HEC.

For HEC, we employ the new dual line-search technique discussed in Section 4.3, though we set it to only be adaptively invoked when the absolute value of (2.12) falls below 10^{-10}. Likewise for HEC only, we evaluated vector extrapolation using sets of 5, 10 and 20 vectors and similarly evaluated the effect of enabling early contraction via enabling the relative step size termination tolerance discussed in Section 3.3 using relative tolerances of 10^{-2}, 10^{-4}, 10^{-6} and 10^{-8}. However, we only report the results from extrapolation with 5 vectors and a relative step size tolerance of 10^{-2} since they provided the best performance on the test sets used here.

5.3.1 Upper bound method variants

In order to measure how efficient Procedure 6 is for finding upper bounds compared to the strategy employed by `hinfnorm`, we ran both methods on the small-scale problems and counted the total number of computed eigentriples that each method required to compute their upper bounds for all the problems in the test set. As we expected, we see in Table 5.1 that indeed Procedure 6 is exceptionally fast, requiring a total of just 121 computed eigentriples to find all 33 upper bounds and furthermore, that number also includes calculating the rightmost or outermost eigenvalue of A for each of those problems, which is not necessary if the user initializes the routine with any acceptable nonzero perturbation $\Delta_0 = \varepsilon_0 u_0 v_0^*$. In stark contrast, `hinfnorm`'s bounding phase required computing a total of 5606 eigentriples.

Despite its efficiency however, Procedure 6's lazy upper bound approach appears to come at a steep penalty in terms of the number of worse (lower) local maximizers HEC ultimately converges to and in fact, we observe this consistently for enabling the lazy upper bound option across all the upper bound variants. As a consequence, for HEC, we thus always use Procedure 6 to first quickly find an upper bound on $\varepsilon_\star$, followed by single SVSAR call to subsequently expand out as much as possible via updating the perturbation vectors u_k and v_k before commencing hybrid expansion-contraction. While it is not nearly as fast as Procedure 6 alone, the total of 2507 computed eigentriples to produce upper bounds for the 33 problems in the data set is still a significant reduction compared to all the other non-lazy upper bound approaches, since we avoid calling

SVSAR more than once per problem as compared to the other methods. Furthermore, Procedure 6 coupled with SVSAR to find an upper bound appears to also be nearly the best in terms of the number of good quality approximations HEC ultimately converges to on the data set.

Remark 5.3. *A possible explanation for why initializing* HEC *from a lazy upper bound seems to cause convergence to an overall greater number of worse* (*lower*) *local maximizers is that, at a general level, the number of locally rightmost points of* $\sigma_\varepsilon(A, B, C, D)$ *tends to decrease as* ε *increases, and sometimes it decreases to just one. Consider the panels of Figure 2.2, where we see that the loss of a locally rightmost point of the spectral value set for* $\varepsilon = 0.1$ *in the top-right panel causes the* GGO *algorithm to break down. Now, suppose Procedure 6 has found a point in the upper right region of the spectral value set in the right half-plane for* $\varepsilon = 0.1$, *near where* SVSAR *begins its iteration. If* SVSAR *is called immediately after Procedure 6, then it is likely that* HEC *will converge to the globally rightmost point on the imaginery axis for this problem, as is shown in top-right panel for the* GGO *algorithm. However, if the initial contraction phase is begun immediately after Procedure 6, then it seems very likely that* HEC *will instead converge to the topmost only locally rightmost point on the imaginary axis of the spectral value for some value* $\varepsilon_\star$ *slightly less then* 0.09, *as we can see from the bottom-left panel.*

Comparison of Upper Bound Methods for HEC: Small-scale							
	Totals for U.B.			# Rel. Diff. to $\|G\|_\infty$			
U.B. Alg.	ε_k	u_k, v_k	λ_{RP}	10^{-8}	10^{-6}	10^{-4}	S
NB	124	n/a	5606	18	22	25	30
HEC + D	105	3063	3809	22	26	29	33
HEC + D,W	105	3036	3775	21	26	29	33
HEC + D,L	105	216	670	17	22	27	33
HEC + S	95	2972	3416	21	26	29	33
HEC + S,W	95	2953	3361	21	25	29	33
HEC + S,L	95	180	417	18	23	27	33
HEC + A	99	2160	2507	21	25	29	33
HEC + A,L	99	22	121	16	22	26	33

TABLE 5.1: NB *denotes the* `hinfnorm` *code while* HEC *is the implementation of hybrid-expansion contraction coupled with the following upper bound procedures (described for the continuous-time case; replace all occurrences of "rightmost" with "outermost" for the corresponding discrete-time versions):* D *is a new implementation of the "doubling of ε_k" strategy employed by* `hinfnorm` *based off successively computing locally rightmost points,* S *is a similar strategy but instead increases ε_k by taking double the Newton step with respect to each locally rightmost point computed, while* A *specifies using Procedure 6 to warm-start a single call of* SVSAR *so that the first contraction phase not only commences outside the stability region but also typically at a locally rightmost point. For variants* D *and* S*, option* W *specifies warm-starting the next* SVSAR *call via the previously computed locally rightmost point. Option* L *specifies enabling a lazy upper bound, whereby any of the procedures immediately terminate as soon as they encounter an iterate outside the stability region. The case of* A,L *implies that just Procedure 6 is called to find an upper bound without the followup* SVSAR *call, meaning the first contraction phase typically contracts a point already in the interior of the initial spectral value set but still outside of the stability region.* Totals for U.B. *lists the number of updates to the perturbation level ε_k, the number of updates to the perturbation vectors u_k, v_k, and λ_{RP} is the number of eigentriples computed in the course of finding upper bounds for all the problems. The columns* # Rel Diff to $\|G\|_\infty$ *show the number of problems a given algorithm successfully converged on to these varying degrees of precision with respect to the true value.* S *is the number of problems that a particular code successfully converged on, solely according to the convergence failure criteria of Remark 5.2.*

5.3.2 Small-scale evaluation

In order to numerically validate our new hybrid expansion-contraction method, we consider the number of approximations computed by HEC over the test set that agree to varying levels of precision with respect to the true value of the H_∞-norm for each problem, as computed by MATLAB's `getPeakGain`. We consider these counts at multiple levels of precision since HEC and `hinfnorm` may only find local maximizers in some cases. We compute the relative differences of the approximations compared to the value computed by `getPeakGain` given a tolerance of 10^{-10}. For a precision level of 10^{-8}, we count the number of approximations that are either strictly greater than the value computed by `getPeakGain` or have a relative difference of at most 10^{-8}. We also tabulate counts for precision levels of 10^{-6} and 10^{-4}. However, for the counts for `hinfnorm`, we do not include any result that satisfies the failure-to-converge condition described in Remark 5.2 (there are three such problems in the small-scale test set: `ROC3`, `AC17` and `AC6`). We note that as long as HEC finds an upper bound, which it did successfully for all the problems, HEC cannot fail in this way.

In Table 5.2, we see that HEC is nearly two and half times faster than `hinfnorm` at going through the entire test set and furthermore, HEC mostly finds find good approximations to $\|G\|_\infty$. Enabling vector extrapolation on HEC shows a dramatic reduction in the number of eigentriples calculated over the test set but because these are small scale problems, it does not fully translate to a correspondingly large reduction in total CPU time. It appears that enabling extrapolation also pushed the convergence of a couple problems to different local maximizers but that is not to be unexpected as different options are enabled in the code. Enabling the relative step size termination condition in SVSAR for HEC also dramatically reduces the number of computed eigentriples, allow-

HEC Overall Performance: Small-scale						
	Totals		# Rel Diff to $\|G\|_\infty$			
Alg + Opts	λ_{RP}	sec	10^{-8}	10^{-6}	10^{-4}	S
NB	32112	465.49	18	22	25	30
HEC	16665	199.99	21	25	29	33
HEC + E	9708	168.71	19	23	28	33
HEC + RS	10565	99.70	21	25	28	33
HEC + E,RS	6767	89.64	21	25	28	33

TABLE 5.2: NB *denotes the* `hinfnorm` *code while* HEC *is the implementation of hybrid-expansion contraction with upper bound Procedure 6 followed by a single call to* SVSAR *to expand out before the first contraction phase.* E *denotes that vector extrapolation is enabled and* RS *denotes that the relative step size termination tolerance is enabled for* SVSAR *with a value of* 10^{-2}. λ_{RP} *is the number of computed eigentriples. The columns* # Rel Diff to $\|G\|_\infty$ *show the number of problems a given algorithm successfully converged on to these varying degrees of precision with respect to the true value.* S *is the number of problems that a particular code successfully converged on, solely according to the convergence failure criteria of Remark 5.2.*

ing HEC to complete the test set over four and half times faster than `hinfnorm`, while additionally enabling extrapolating increases the speedup factor to over five. As there is a high amount of variability in the test set, the totals shown in Table 5.2 are only so informative and thus we provide average, median, and range data on the relative speedup *per problem* with respect to the number of eigentriples computed and with respect to the CPU running time in Table 5.3. We see that HEC with the relative step size termination option enabled is up to nearly 47 times faster (on `CSE2`) in terms of CPU time and when vector extrapolation is also enabled, we see that speedup per problem ranges from approaching twice as fast to almost five times as fast compared to `hinfnorm` as reported by the median and average respectively.

Per-Problem Speedups of HEC Relative to `hinfnorm`: Small-scale						
	w.r.t. # of Computed λ_{RP}			w.r.t. CPU Time		
Opts	Avg	Med	Range	Avg	Med	Range
-	2.25	1.32	[0.16, 16.22]	1.20	0.72	[0.19, 5.47]
E	4.47	2.64	[0.21, 21.89]	2.27	1.13	[0.22, 11.71]
RS	9.19	3.12	[0.26, 169.93]	3.55	1.20	[0.30, 46.87]
E,RS	12.38	3.45	[0.26, 169.93]	4.71	1.71	[0.18, 45.41]

TABLE 5.3: *Average, median and range of per-problem speedups with respect to number of eigentriples λ_{RP} computed and the CPU time required for each problem.*

5.3.3 Large-scale matrices

We present analogous data to the small-scale results in Tables 5.4 and 5.5 for the large-scale test set, with the following notable exceptions. First, as we cannot tractably compute the true value of $\|G\|_\infty$ for these large-dimensional problems, we instead use the largest approximation computed from all the methods for each problem as a surrogate to compute the relative differences used in tabulating how many approximations for a given method agreed to a given level of precision with the best of the results calculated. Second, as these matrices are all large, we use `eigs` with its default options to compute the rightmost or outermost eigentriples but with 8 eigenvalues requested per call to `eigs` (done for the right and left eigenvectors separately), to be consistent with the experiments in [GGO13]. We also additionally evaluated both eigenvector recycling types that are optionally available in our code to attempt to lessen the number of iterations `eigs` requires. Both seemed to have a beneficial effect with respect to run-times on the large-scale test set but we only report the results for recycling the average of the k computed eigenvectors from the previous computed eigentriple, as it outperformed recycling just the previously selected eigenvector. We also note that `hinfnorm` again fails to converge on a problem in the test-set (`skewlap3d`), according to the criterion in Remark 5.2.

HEC Overall Performance: Large-scale						
	Totals		# Rel Diff to Best			
Alg + Opts	λ_{RP}	sec	10^{-8}	10^{-6}	10^{-4}	S
NB	4196	20920	11	11	11	13
HEC	2338	3756	9	10	12	14
HEC + V	2336	2362	10	11	13	14
HEC + E	636	1504	10	12	13	14
HEC + E,V	690	1110	10	11	13	14
HEC + RS	861	1046	9	10	11	14
HEC + RS,V	849	919	10	11	12	14
HEC + E,RS	700	960	9	10	11	14
HEC + E,RS,V	794	841	11	12	13	14

TABLE 5.4: NB *denotes the* `hinfnorm` *code while* HEC *is the implementation of hybrid-expansion contraction with upper bound Procedure 6 followed by a single call to* SVSAR *to expand out before the first contraction phase.* E *denotes that vector extrapolation is enabled and* RS *denotes that the relative step size termination tolerance is enabled for* SVSAR *with a value of* 10^{-2}. V *denotes that eigenvector recycling is enabled such that the initial vector for* `eigs` *is set to the average of the* $k = 8$ *requested eigenvectors computed by* `eigs` *for the previous computed eigentriple.* λ_{RP} *is the number of computed eigentriples. The columns* # Rel Diff to Best *show the number of problems a given algorithm successfully converged on to those varying degrees of precision with respect to the best of the computed values for each problem.* S *is the number of problems that a particular code successfully converged on, solely according to the convergence failure criteria of Remark 5.2.*

In Table 5.4, as in the small-scale test set, we find overall that HEC appears to be successfully converging to good local or possibly global maximizers. Furthermore, even without any options enabled, HEC completes the entire test set 5.6 times faster than `hinfnorm`. Enabling eigenvector recycling results in a $14 - 59\%$ speed boost, depending on what other options are simultaneously enabled. In contrast to the small-scale results, the substantial reduction in the number of computed eigentriples over the large-scale test set by enabling extrapolation is actually overshadowed by an even larger reduction in total running time, resulting in an overall speedup factor of 13.9 when compared to the total CPU time of `hinfnorm`. Enabling eigenvector recycling on top of extrapolation substantially increases that CPU time speedup factor to 18.8, even though it actually also increased the number of computed eigentriples by 8.5%. Enabling just the relative step size termination feature provides an even greater speedup factor of 20.0 times faster, again despite an increase in the number of eigentriples computed compared to the extrapolation variants. Simultaneously enabling extrapolation, relative step size termination, and eigenvector recycling allows HEC to complete the large-scale test set 24.9 times faster than `hinfnorm`. Though we don't report full results here for additionally enabling cubic models for the SVSAR line search, as discussed in Section 4.3, we have found that it does in fact provide a modest performance boost across all the variants, ranging from a speedup factor of 5.8 times faster for HEC with no options enabled to 26.2 times faster for HEC with extrapolation, relative step size termination, and eigenvector recycling all enabled.

Referring to the per-problem speedup results in Table 5.5, we see that depending on the problem, HEC with relative step size termination and eigenvector recycling enabled can be over 82 times faster than `hinfnorm` (on `skewlap3d` in the discrete-time problem set). Judging from the per-problem speedup median values, HEC appears to generally be two and half times faster than `hinfnorm` when all acceleration options are enabled and an order of magnitude faster as reported by the average time per problem. It is also worth noting that in the worst-case, for problems where HEC is slower than `hinfnorm`, the running time is usually only worse by, at most, just under a factor of two. Comparing the individual approximations computed by HEC to the ones provided by `hinfnorm`, at worst we found that HEC only agreed to 2 digits (on the discrete-time version of skewlap3d). However, this was far from typical as the median of the computed relative differences compared to the best approximation reported demonstrated that HEC usually agreed to 11 digits with hinfnorm on the large-scale test set and furthermore, for markov in the discrete-time test set, HEC found an approximation *three times larger* than the one found by hinfnorm.

Per-Problem Speedups of HEC Relative to hinfnorm: Large-scale						
	# of Computed λ_{RP}			CPU Time		
Opts	Avg	Med	Range	Avg	Med	Range
-	3.99	1.33	[0.54, 21.50]	8.80	1.62	[0.41, 57.28]
V	3.56	1.46	[0.51, 18.63]	11.48	2.09	[0.64, 64.86]
E	5.48	1.68	[0.65, 23.29]	9.30	1.72	[0.23, 64.80]
E,V	4.07	1.52	[0.50, 14.81]	13.92	2.20	[0.57, 78.51]
RS	4.94	1.89	[0.50, 25.41]	13.79	2.40	[0.53, 66.41]
RS,V	4.77	2.22	[0.50, 21.50]	15.24	2.55	[0.71, 82.43]
E,RS	5.42	2.21	[0.80, 25.41]	14.51	1.88	[0.52, 66.19]
E,RS,V	5.16	2.05	[0.50, 18.63]	15.77	2.56	[0.75, 75.31]

TABLE 5.5: *Average, median and range of per-problem speedups with respect to number of eigentriples λ_{RP} computed and the CPU time required for each problem. See the caption of Table 5.4 for definitions of* HEC *options* E, RS, *and* V.

5.3.4 Convergence rates of hybrid expansion-contraction

We now turn to empirically evaluating how fast hybrid expansion-contraction actually converges in practice, given the worst-case superlinear rate of convergence result proven in Section 3.2 and the claim that quadratic convergence can also be expected. In Table 5.6, we see that on the continuous-time problems pde and rdbrusselator from the large-scale test set, the sequence of ε_k values produced by the hybrid expansion-contraction process indeed appears to be converging quadratically. In reviewing the corresponding convergence data for all the other problems tested, we see a similar and consistent picture of fast convergence.

HEC Rate of Convergence				
	`pde`		`rdbrusselator`	
k	ε_k	$\varepsilon_k - \varepsilon_5$	ε_k	$\varepsilon_k - \varepsilon_5$
1	0.156550945115623397	1.54×10^{-1}	0.0012451852810304153	7.10×10^{-4}
2	0.002744700134015856	3.28×10^{-5}	0.0005428221115940368	7.58×10^{-6}
3	0.002711903931832482	3.91×10^{-8}	0.0005352544476893113	8.11×10^{-9}
4	0.002711864788294241	1.25×10^{-13}	0.0005352463336213466	2.77×10^{-14}
5	0.002711864788169188	0	0.0005352463335936135	0

TABLE 5.6: *Observed convergence rate of* HEC *to its final values of* ε_k *for two continuous-time problems from the large-scale test set.*

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

TABLE 5.7: *Average, median and range of the number of iterations in the* NB *(Newton-bisection) converging phase of* `hinfnorm` *and* HEC *across the small-scale test set and the large-scale test set.*

In Table 5.7, we report the average, median, and range of iterations that hybrid expansion-contraction incurred to converge over the small and large-scale test sets. On the small-scale problems, at worst HEC took nine and ten iterations to converge on problems `AC11` and `AC6` but upon closer analysis, we observed that HEC in fact resolved the first eight digits of each approximation within four iterations and the remaining iterations of HEC were due to τ_ε and τ_{uv} being set too tight. We note that this is a limitation of double-precision hardware for problems that are exceptionally sensitive to changes in ε. We also show the corresponding statistics for the Newton-bisection iteration of `hinfnorm` for comparison and find that hybrid expansion-contraction typically requires even fewer iterations than the Newton-bisection iteration, most likely due to hybrid expansion-contraction's ability to safely take an even larger step than offered by the GGO algorithm. One might argue that this is an unfair comparison because a single iteration of hybrid expansion-contraction comprises calling both the contraction method and SVSAR to expand while in contrast, the only work done in a single iteration of Newton-bisection is a single call to SVSAR. However, we have found that the employment of Newton-bisection for the contraction phase is exceedingly efficient and reliable and that on average, it only computes three eigentriples before converging to the imaginary axis or unit circle. Finally, we see that enabling HEC's relative step size termination tolerance does in fact increase the number of hybrid expansion-contraction iterations but only slightly and it is certainly not a cause for concern given the large overall performance increase it provides.

6 Hybrid expansion-contraction for the real structured stability radius

6.1 A brief introduction to the real structured stability radius

Remark 6.1. *This introduction section is merely meant as a primer on the real structured stability radius to provide sufficient yet minimal context for detailing the author's contributions in adapting the hybrid expansion-contraction algorithm to approximate the real structured stability radius with Frobenius norm bounded perturbations, presented in* [GGMO14]*, as well as to present some initial steps towards a viable variant of hybrid expansion-contraction for the case of spectral norm bounded perturbations. For a more complete discourse of the background and theory, we refer the reader to* [GGMO14]

Given matrices A, B, C, D defining the linear dynamical system (1.1), recall the *perturbed system matrix*

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

assuming $I - D\Delta$ is invertible, and the associated *transfer matrix*

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

Definition 6.2. *Let $A \in \mathbb{K}^{n,n}$, $B \in \mathbb{K}^{n,p}$, $C \in \mathbb{K}^{m,n}$ and $D \in \mathbb{K}^{m,p}$. Let $\varepsilon \in \mathbb{R}^+$ be such that $\varepsilon \|D\|_2 < 1$ and define the* spectral value set *with respect to the norm $\|\cdot\|$ and the* *field* $\mathbb{K}$ *as*

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

Note that $\sigma_\varepsilon^{\mathbb{K},\|\cdot\|}(A,B,C,D) \supset \sigma_0^{\mathbb{K},\|\cdot\|}(A,B,C,D) = \sigma(A)$ and furthermore, that Definition 6.2 is a generalization of the ε-spectral value sets defined in Definition 1.1, that is for the specific case when $\mathbb{K} \coloneqq \mathbb{C}$ and $\|\cdot\| \coloneqq \|\cdot\|_2$. However, in contrast to when complex perturbations are allowed, spectral value sets defined for $\mathbb{K} \coloneqq \mathbb{R}$ cannot be solely be defined in terms of rank-1 perturbations (as in Corollary 1.3) and instead must consider the possibility of rank-2 perturbations as well [QBR$^+$95, GGMO14]. More concretely, for $\mathbb{K} \coloneqq \mathbb{R}$ and when $\|\cdot\|$ is either the spectral or Frobenius norm, then

$$\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D) = \bigcup \left\{ \sigma(M(\Delta)) : \Delta \in \mathbb{R}^{p,m}, \|\Delta\| \leq \varepsilon, \operatorname{rank}(\Delta) \leq 2 \right\}. \quad (6.1)$$

Definition 6.3. *For* $\varepsilon \in \mathbb{R}^+$ *and* $\varepsilon \|D\|_2 < 1$*, the* real spectral value set abscissa *and* real spectral value set radius *are defined respectively as*

$$\begin{aligned} \alpha_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D) &= \max\{\operatorname{Re}(\lambda) : \lambda \in \sigma_\varepsilon^{\mathbb{K},\|\cdot\|}(A,B,C,D)\} \\ \rho_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D) &= \max\{|\lambda| \qquad : \lambda \in \sigma_\varepsilon^{\mathbb{K},\|\cdot\|}(A,B,C,D)\} \end{aligned}$$

where $\alpha_0^{\mathbb{R},\|\cdot\|}(A,B,C,D) = \alpha(A)$ *and* $\rho_0^{\mathbb{R},\|\cdot\|}(A,B,C,D) = \rho(A)$*.*

Definition 6.4. *The* μ-value *of any matrix* $A \in \mathbb{K}^{m,p}$ *with respect to the norm* $\|\cdot\|$ *is defined by*

$$\mu_{\mathbb{K}}^{\|\cdot\|}(A) = \left[\inf\left\{\|\Delta\| : \Delta \in \mathbb{K}^{p,m}, \det(I - \Delta A) = 0\right\}\right]^{-1}. \quad (6.2)$$

with the convention that $\mu_{\mathbb{K}}^{\|\cdot\|}(A) = 0$ *when* $A = 0$.

Definition 6.5. *The* real structured stability radius *for continuous and discrete-time systems are defined respectively as*

$$r_{\mathbb{R}}^{\|\cdot\|,c}(A, B, C, D) \coloneqq \inf\{\|\Delta\| : \alpha(M(\Delta)) \geq 0 \text{ or } \det(I - D\Delta) = 0\} \tag{6.3}$$

$$r_{\mathbb{R}}^{\|\cdot\|,d}(A, B, C, D) \coloneqq \inf\{\|\Delta\| : \rho(M(\Delta)) \geq 1 \text{ or } \det(I - D\Delta) = 0\}. \tag{6.4}$$

For $\|\cdot\| = \|\cdot\|_2$ or $\|\cdot\| = \|\cdot\|_{\mathrm{F}}$, by [HP05, Section 5.3.3] and [GGMO14] respectively, (6.3) and (6.4) may be equivalently written as

$$\begin{aligned} r_{\mathbb{R}}^{\|\cdot\|,c}(A, B, C, D) &= \min\left(\left[\mu_{\mathbb{R}}^{\|\cdot\|}(D)\right]^{-1}, \inf_{\omega \in \mathbb{R}} \left[\mu_{\mathbb{R}}^{\|\cdot\|}\left(G(\mathbf{i}\omega)\right)\right]^{-1}\right) && (6.5)\\ &= \min\left(\|D\|_2^{-1}, \inf_{\varepsilon\|D\|_2 < 1} \left\{\varepsilon : \alpha_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A, B, C, D) \geq 0\right\}\right) && (6.6)\end{aligned}$$

and

$$\begin{aligned} r_{\mathbb{R}}^{\|\cdot\|,d}(A, B, C, D) &= \min\left(\left[\mu_{\mathbb{R}}^{\|\cdot\|}(D)\right]^{-1}, \inf_{\theta \in [0,2\pi)} \left[\mu_{\mathbb{R}}^{\|\cdot\|}\left(G(\mathrm{e}^{\mathbf{i}\theta})\right)\right]^{-1}\right) && (6.7)\\ &= \min\left(\|D\|_2^{-1}, \inf_{\varepsilon\|D\|_2 < 1} \left\{\varepsilon : \rho_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A, B, C, D) \geq 1\right\}\right), && (6.8)\end{aligned}$$

with the conventions that $\|D\|_2^{-1} = \infty$ if $D = 0$ and $r_{\mathbb{R}}^{\|\cdot\|,c}(A, B, C, D) = \|D\|_2^{-1}$ if the set over which the infimum is taken is empty and similarly for $r_{\mathbb{R}}^{\|\cdot\|,d}(A, B, C, D)$.

Definition 6.6. *For* $\varepsilon \in \mathbb{R}^{++}$ *with* $\varepsilon\|D\|_2 < 1$*, a point* $\lambda \in \mathbb{C}$ *is said to be a* static boundary point *of* $\sigma_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A, B, C, D)$ *if* λ *is on the boundary of* $\sigma_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A, B, C, D)$ *and there exists nonnegative real numbers* δ_1 *and* δ_2 *such that* $\delta_1 + \delta_2 > 0$ *and* λ *is also a boundary point of* $\sigma_{\tilde{\varepsilon}}^{\mathbb{R},\|\cdot\|}(A, B, C, D)$ *for all* $\tilde{\varepsilon}$ *such that* $\varepsilon - \delta_1 < \tilde{\varepsilon} < \varepsilon + \delta_2$.

Remark 6.7. *For the usual case of spectral value sets where the perturbations are complex-valued and rank-1, static boundary points cannot occur. We refer to* [GGMO14] *as to how such points can arise in the case of real spectral value sets.*

6.2 A high-level overview for adapting hybrid expansion-contraction

We now consider the modifications that must be made in order for the hybrid expansion-contraction algorithm to approximate the real structured stability radius for some specified norm $\|\cdot\|$.

- As per (6.1), the contraction phase must be modified to handle the possibility of rank-2 perturbations in addition to rank-1 perturbations. We discuss this modification in Section 6.3.1.

- In order for Procedure 4 (SVSAR) to converge to a locally rightmost or outermost point of a real spectral value set for some prescribed value of $\varepsilon \in \mathbb{R}^{++}$ and $\varepsilon \|D\|_2 < 1$, the following modifications must be made:

  – Given $U_{k-1} \in \mathbb{R}^{p,2}$ and $V_{k-1} \in \mathbb{R}^{m,2}$ such that $U_{k-1}V_{k-1}^\mathsf{T}$ is at most rank two with $\|U_{k-1}V_{k-1}^\mathsf{T}\| = 1$ and λ_{k-1} is a rightmost or outermost eigenvalue of $M(\varepsilon U_{k-1}V_{k-1}^\mathsf{T})$, Procedure 2 must be adapted so that it produces $U_k \in \mathbb{R}^{p,2}$ and $V_k \in \mathbb{R}^{m,2}$ with the rank of $U_kV_k^\mathsf{T}$ being at most two and $\|U_kV_k^\mathsf{T}\| = 1$ such that λ_{k-1} can be "pushed" rightward (or outward for the discrete-time case) by changing $M(\varepsilon U_{k-1}V_{k-1}^\mathsf{T})$ to $M(\varepsilon U_kV_k^\mathsf{T})$, or merely toward $M(\varepsilon U_kV_k^\mathsf{T})$ in the case that a line search is necessary to ensure monotonicity. In the case of complex perturbations, the computed step as described in Section 2.2 is due to [GGO13] while for the case of real Frobenius norm bounded perturbations, the computed step is presented in [GGMO14].

  – Given $U_{k-1} \in \mathbb{R}^{p,2}$ and $V_{k-1} \in \mathbb{R}^{m,2}$ such that $U_{k-1}V_{k-1}^{\mathsf{T}}$ is at most rank two with $\|U_{k-1}V_{k-1}^{\mathsf{T}}\| = 1$, λ_{k-1} is a rightmost or outermost eigenvalue of $M(\varepsilon U_{k-1}V_{k-1}^{\mathsf{T}})$ and $U_k V_k^{\mathsf{T}}$ is as computed by the modified version of Procedure 2 described above, hybrid expansion-contraction also requires that some mechanism in Procedure 4 enforces monotonicity of the iterates. For the continuous-time case, if the rightmost eigenvalue λ_k of $M(\varepsilon U_k V_k^{\mathsf{T}})$ is such that $\mathrm{Re}\,(\lambda_k) > \mathrm{Re}\,(\lambda_{k-1})$ does not hold, then U_k and V_k must be suitably modified to ensure monotonicity holds. Likewise for the discrete-time case, if the outermost eigenvalue λ_k of $M(\varepsilon U_k V_k^{\mathsf{T}})$ is such that $|\lambda_k| < |\lambda_{k-1}|$ does not hold, then U_k and V_k must again be suitably modified to ensure monotonicity holds. In Section 6.3.3, we show how the line search approach of Procedure 3 for the complex case of rank one perturbations can be extended to the case of real-valued perturbations that may be up to rank two and bounded by either the Frobenius or spectral norms respectively.

- Finally, in the case of real spectral value sets, there is the possibility of encountering points on the boundary that are insensitive to changes in the perturbation level, that is, as ε changes, there can be portions of the boundary of a real spectral value set that do not move [GGMO14]. Such *static* boundary points do not prevent hybrid expansion-contraction from converging to $\lambda \in \mathbb{C}$ and $\varepsilon \in \mathbb{R}^{++}$ such that λ is a locally rightmost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ and $\mathrm{Re}\,(\lambda) = 0$ or alternatively, such that λ is a locally outermost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ and $|\lambda| = 1$, for the continuous and discrete-time cases respectively. However, if λ happens to be a static boundary point, then there is no guarantee that hybrid expansion-contraction will have converged to the minimal value of ε for this particular locally rightmost point on the imaginary axis or alternatively, this particular locally outermost point on the unit circle. In Section 6.4, we present an outline of a method to call after hybrid expansion-contraction that minimizes the perturbation level ε in the case that λ is a static boundary point, though we note that in practice, we expect to never encounter such points.

6.3 The necessary modifications for real-valued rank-2 perturbations

6.3.1 Adapting the contraction phase

We begin by stating some useful results that will be helpful to concisely extend hybrid expansion-contraction to rank-2 perturbations.

Lemma 6.8. *Let $U \in \mathbb{R}^{p,k}$ and $V \in \mathbb{R}^{m,k}$ be such that $UV^\mathsf{T} \neq 0$, $D \in \mathbb{R}^{m,p}$ and $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon\|D\|_2 < 1$. Then $\|D\varepsilon UV^\mathsf{T}\|_2 < \|UV^\mathsf{T}\|_\mathrm{F}$.*

Proof. If $\varepsilon = 0$ or $D = 0$, then the result is immediate. Otherwise, the result follows from

$$\|D\varepsilon UV^\mathsf{T}\|_2 \leq \varepsilon\|D\|_2\|UV^\mathsf{T}\|_2 < \|UV^\mathsf{T}\|_2 \leq \|UV^\mathsf{T}\|_\mathrm{F}.$$

□

Lemma 6.9. *Let $U \in \mathbb{R}^{p,k}$, $V \in \mathbb{R}^{m,k}$, and $D \in \mathbb{R}^{m,p}$ be given with $\varepsilon \in \mathbb{R}^+$ and*

$\|D\varepsilon UV^{\mathsf{T}}\|_2 < 1$. *Then the following three identities hold:*

$$V^{\mathsf{T}}(I - D\varepsilon UV^{\mathsf{T}})^{-1} = (I + T(\varepsilon))V^{\mathsf{T}} \tag{6.9}$$

$$(I - \varepsilon UV^{\mathsf{T}}D)^{-1}U = U(I + T(\varepsilon)) \tag{6.10}$$

where

$$T(\varepsilon) \coloneqq \varepsilon V^{\mathsf{T}}DU(I - \varepsilon V^{\mathsf{T}}DU)^{-1} = \varepsilon(I - \varepsilon V^{\mathsf{T}}DU)^{-1}V^{\mathsf{T}}DU. \tag{6.11}$$

Proof. We begin by noting the following equivalences due to the Sherman-Morrison-Woodbury formula [GV83]:

$$\begin{aligned}(I - D\varepsilon UV^{\mathsf{T}})^{-1} &= I + \varepsilon DU(I - \varepsilon V^{\mathsf{T}}DU)^{-1}V^{\mathsf{T}} \\ (I - \varepsilon UV^{\mathsf{T}}D)^{-1} &= I + \varepsilon U(I - \varepsilon V^{\mathsf{T}}DU)^{-1}V^{\mathsf{T}}D,\end{aligned} \tag{6.12}$$

where $(I - \varepsilon V^{\mathsf{T}}DU)^{-1} \in \mathbb{R}^{k,k}$. Using the first identity of (6.12), we have that

$$\begin{aligned}V^{\mathsf{T}}(I - D\varepsilon UV^{\mathsf{T}})^{-1} &= V^{\mathsf{T}}\left(I + \varepsilon DU(I - \varepsilon V^{\mathsf{T}}DU)^{-1}V^{\mathsf{T}}\right) \\ &= V^{\mathsf{T}} + \varepsilon V^{\mathsf{T}}DU(I - \varepsilon V^{\mathsf{T}}DU)^{-1}V^{\mathsf{T}} \\ &= \left(I + \varepsilon V^{\mathsf{T}}DU(I - \varepsilon V^{\mathsf{T}}DU)^{-1}\right)V^{\mathsf{T}} \\ &= (I + T(\varepsilon))\,V^{\mathsf{T}},\end{aligned} \tag{6.13}$$

thus demonstrating (6.9).

As $\|D\varepsilon UV^\mathsf{T}\|_2 < 1$, it follows that

$$V^\mathsf{T}DU(I - \varepsilon V^\mathsf{T}DU)^{-1} = V^\mathsf{T}DU\sum_{k=0}^{\infty}(\varepsilon V^\mathsf{T}DU)^k = (I - \varepsilon V^\mathsf{T}DU)^{-1}V^\mathsf{T}DU \quad (6.14)$$

and hence $V^\mathsf{T}DU$ commutes with $(I - \varepsilon V^\mathsf{T}DU)^{-1}$, that is, (6.11) holds.

By the second identity of (6.12) and (6.11), we have the following equivalences:

$$\begin{aligned}(I - \varepsilon UV^\mathsf{T}D)^{-1}U &= \left(I + \varepsilon U(I - \varepsilon V^\mathsf{T}DU)^{-1}V^\mathsf{T}D\right)U \\ &= U + \varepsilon U(I - \varepsilon V^\mathsf{T}DU)^{-1}V^\mathsf{T}DU \\ &= U(I + T(\varepsilon)), \end{aligned} \quad (6.15)$$

thereby proving (6.10). □

Let $U \in \mathbb{R}^{p,2}$ and $V \in \mathbb{R}^{m,2}$ be two matrices such that $\|UV^\mathsf{T}\| = 1$ where $\|\cdot\|$ is either the Frobenius or spectral norm and consider the following matrix family where U and V are fixed and $\varepsilon \in \mathbb{R}^+$ and $\varepsilon\,\|D\|_2 < 1$:

$$M_{UV}(\varepsilon) := A + B\varepsilon UV^\mathsf{T}(I - D\varepsilon UV^\mathsf{T})^{-1}C. \quad (6.16)$$

If the given ε, U and V demonstrate that $\varepsilon > \varepsilon_\star$, that is matrix $M_{UV}(\varepsilon)$ has at least one eigenvalue $\lambda_{UV}(\varepsilon)$ such that $\operatorname{Re}(\lambda_{UV}(\varepsilon)) > 0$ or $|\lambda_{UV}(\varepsilon)| > 1$ respectively, then again by Key Observation 3.2, it is clear that we may always contract ε to $\hat{\varepsilon}$ such that $\varepsilon_\star < \hat{\varepsilon} < \varepsilon$ by finding a root of the function

$$g^c_{UV}(\varepsilon) := \operatorname{Re}(\lambda_{UV}(\varepsilon)) \quad \text{or} \quad g^d_{UV}(\varepsilon) := |\lambda_{UV}(\varepsilon)| - 1. \quad (6.17)$$

For convenience, we define

$$g_{UV}(\varepsilon) \coloneqq \begin{cases} g^c_{UV}(\varepsilon) & \text{if } (A,B,C,D) \text{ is a continuous-time system} \\ g^d_{UV}(\varepsilon) & \text{if } (A,B,C,D) \text{ is a discrete-time system} \end{cases} \tag{6.18}$$

so that $\lambda_{UV}(\varepsilon)$ is on the boundary of the stability region if and only if $g_{UV}(\varepsilon) = 0$ for either the continuous or discrete-time cases depending on the context. The corresponding derivatives of (6.17) with respect to ε are:

$$g'^c_{UV}(\varepsilon) = \operatorname{Re}\left(\lambda'_{UV}(\varepsilon)\right) \quad \text{and} \quad g'^d_{UV}(\varepsilon) = \operatorname{Re}\left(\frac{\overline{\lambda_{UV}(\varepsilon)}\lambda'_{UV}(\varepsilon)}{|\lambda_{UV}(\varepsilon)|}\right). \tag{6.19}$$

In order to compute the derivatives in (6.19), we must consider the derivative of (6.16). By Lemma 6.8 and using (6.9) and (6.10) of Lemma 6.9, we may rewrite (6.16) as:

$$\begin{aligned} M_{UV}(\varepsilon) &= A + B\varepsilon U\left(I + T(\varepsilon)\right)V^{\mathsf{T}}C \\ &= A + BU\left(\varepsilon + \varepsilon T(\varepsilon)\right)V^{\mathsf{T}}C. \end{aligned}$$

Taking the derivative with respect to ε, we have that

$$\begin{aligned} M'_{UV}(\varepsilon) &= BU\left(\varepsilon + \varepsilon T(\varepsilon)\right)' V^{\mathsf{T}}C \\ &= BU\left(I + T(\varepsilon) + \varepsilon T'(\varepsilon)\right)V^{\mathsf{T}}C \end{aligned} \tag{6.20}$$

where

$$\begin{aligned}\varepsilon T'(\varepsilon) &= \varepsilon\left(\varepsilon V^\mathsf{T} DU(I-\varepsilon V^\mathsf{T} DU)^{-1}\right)' \\ &= \varepsilon\left[V^\mathsf{T} DU(I-\varepsilon V^\mathsf{T} DU)^{-1} + \varepsilon V^\mathsf{T} DU\left((I-\varepsilon V^\mathsf{T} DU)^{-1}\right)'\right] \\ &= T(\varepsilon) + \varepsilon^2 V^\mathsf{T} DU\left[(I-\varepsilon V^\mathsf{T} DU)^{-1} V^\mathsf{T} DU (I-\varepsilon V^\mathsf{T} DU)^{-1}\right] \\ &= T(\varepsilon) + T(\varepsilon)^2. \end{aligned} \tag{6.21}$$

Substituting (6.21) into (6.20), it follows that

$$\begin{aligned} M'_{UV}(\varepsilon) &= BU\left(I + 2T(\varepsilon) + \varepsilon T'(\varepsilon)\right) V^\mathsf{T} C \\ &= BU(I+T(\varepsilon))^2 V^\mathsf{T} C. \end{aligned} \tag{6.22}$$

Thus for a simple eigenvalue $\lambda_{UV}(\varepsilon)$ of $M_{UV}(\varepsilon)$ with associated right and left RP-compatible eigenvectors $x(\varepsilon)$ and $y(\varepsilon)$, again by standard first-order perturbation theory of simple eigenvalues [HJ90, Theorem 6.3.12], [GO11, Lemma 2.1], we have that

$$\lambda'_{UV}(\varepsilon) = \frac{y(\varepsilon)^* M'_{UV}(\varepsilon) x(\varepsilon)}{y(\varepsilon)^* x(\varepsilon)} = \frac{y(\varepsilon)^* BU\left(I+T(\varepsilon)\right)^2 V^\mathsf{T} C x(\varepsilon)}{y(\varepsilon)^* x(\varepsilon)} \tag{6.23}$$

using (6.22) and (6.11). Thus, the contraction phase of hybrid expansion-contraction is adapted to the case of purely real perturbations by replacing $f_{uv}(\cdot)$ appearing in Procedure 5 to

$$f_{UV}(\varepsilon) := (g_{UV}(\varepsilon), g'_{UV}(\varepsilon)) \tag{6.24}$$

using (6.18), the appropriate functions in (6.17) and (6.19), (6.23) and (6.11).

6.3.2 Convergence results

In order to adapt the assertion of Theorem 3.10 that hybrid expansion-contraction converges for the case of real structured stability radius, we must first demonstrate analogous results to Lemmas 1.17 and 1.20 that respectively relates locally rightmost and outermost points of real spectral value sets on the corresponding stability boundary and their $\mu_{\mathbb{R}}^{\|\cdot\|}$ values.

Lemma 6.10. *Let $\lambda \in \sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)\backslash\sigma(A)$ be such that $\operatorname{Re}(\lambda) = 0$ and suppose that $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\omega))$ defined for $\omega \in \mathbb{R}$ is continuous in a neighborhood of $\operatorname{Im}(\lambda)$, where $\|\cdot\|$ specifies either the spectral or Frobenius norm. If λ is a locally rightmost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$, then $\operatorname{Im}(\lambda)$ is a local maximizer of $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\omega))$.*

Proof. Suppose that $\operatorname{Im}(\lambda)$ is not a maximizer of $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\omega))$. As a consequence there exists $y_1 := \operatorname{Im}(\lambda) + \delta_1$ such that $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}y_1)) > \mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\operatorname{Im}(\lambda))) = \varepsilon^{-1}$ for some $\delta_1 \in \mathbb{R}$ with $|\delta_1| > 0$ arbitrary small. By [HP05, Theorem 5.2.9] and [GGMO14], it follows that $\mathbf{i}y_1 \in \sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ and furthermore $\mathbf{i}y_1$ must be strictly in the interior. Thus, there exists $y_2 := \delta_2 + \mathbf{i}y_1 = \delta_2 + \mathbf{i}(\operatorname{Im}(\lambda) + \delta_1)$ for some sufficiently small real value $\delta_2 > 0$ such that $y_2 \in \sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ as well. However, since $\operatorname{Re}(y_2) = \delta_2 > 0 = \operatorname{Re}(\lambda)$ and δ_1 and δ_2 can be chosen to be arbitrary small, λ cannot be a locally rightmost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ and hence we have a contradiction. □

Lemma 6.11. *Let $\lambda \in \sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)\backslash\sigma(A)$ be such that $|\lambda| = 1$ and suppose that $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}e^{\mathbf{i}\theta}))$ defined for $\theta \in [0,2\pi)$ is continuous in a neighborhood of $\angle\lambda$, where $\|\cdot\|$ specifies either the spectral or Frobenius norm. If λ is a locally outermost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$, then $\angle\lambda$ is a local maximizer of $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}e^{\mathbf{i}\theta}))$.*

Proof. The proof is a straightforward adaptation of the argument for Lemma 6.10. □

Theorem 6.12. *Let* RSVSAR *be an adaptation of Procedure 4 such that it always converges to either a locally rightmost or outermost point of* $\sigma_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ *for the continuous and discrete-time cases respectively, where* $\|\cdot\|$ *specifies the spectral or Frobenius norm and for any* $\varepsilon \in \mathbb{R}^{+}$ *with* $\varepsilon\|D\|_2 < 1$. *Given an initial perturbation* $\Delta = \varepsilon_0 UV^{\mathsf{T}}$ *such that* $\varepsilon_0 \in \mathbb{R}^{++}$, $\varepsilon\|D\|_2 < 1$ *and* $\|UV^{\mathsf{T}}\| = 1$ *holds for the specified norm and matrices* $U \in \mathbb{R}^{p\times 2}$ *and* $V \in \mathbb{R}^{m,2}$ *then, if Assumption 3.3 holds:*

1. *For the continuous-time case, if* $\alpha(M_{UV}(\varepsilon_0)) \geq 0$, *hybrid expansion-contraction converges to* $\lambda \in \mathbb{C}$ *and* $\varepsilon \in \mathbb{R}^{++}$ *such that* λ *is a locally rightmost point of* $\sigma_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ *and* $\operatorname{Re}(\lambda) = 0$. *Furthermore, if* $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\omega))$ *is continuous in a neighborhood of* $\operatorname{Im}(\lambda)$, *then* $\operatorname{Im}(\lambda)$ *is a local maximizer of* $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\omega))$ *with locally maximal value* ε^{-1}.

2. *For the discrete-time case, if* $\rho(M_{UV}(\varepsilon_0)) \geq 1$, *hybrid expansion-contraction converges to* $\lambda \in \mathbb{C}$ *and* $\varepsilon \in \mathbb{R}^{++}$ *such that* λ *is a locally outermost point of* $\sigma_{\varepsilon}^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ *and* $|\lambda| = 1$. *Furthermore, if* $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\mathrm{e}^{\mathbf{i}\theta}))$ *is continuous in a neighborhood of* $\angle\lambda$, *then* $\angle\lambda$ *is a local maximizer of* $\mu_{\mathbb{R}}^{\|\cdot\|}(G(\mathbf{i}\mathrm{e}^{\mathbf{i}\theta}))$ *with locally maximal value* ε^{-1}.

Proof. The proof of the first part of the statements for the continuous and discrete-time cases follows from a near verbatim argument from the proof of Theorem 3.10. The second part of the statements for the continuous and discrete-time cases follows from Lemmas 6.10 and 6.11 respectively. □

In order to show that the convergence rate analysis of hybrid expansion-contraction detailed in Section 3.2 also holds for the case of the real structured stability radius with Frobenius norm bounded perturbations, we need only show that an analogous version of

Lemma 3.7 holds here as well, though for brevity we only present the continuous-time case. We begin by stating the following result of [GGMO14]:

Theorem 6.13. *Let $\lambda^{\mathrm{r}}(\varepsilon)$ be a single continuous branch of locally rightmost points of $\sigma_{\hat{\varepsilon}}^{\mathbb{R},\|\cdot\|_{\mathrm{F}}}(A,B,C,D)$ defined for a neighborhood $\mathcal{N}$ of $\hat{\varepsilon} \in \mathbb{R}^{++}$ with $\hat{\varepsilon}\|D\|_2 < 1$ and where $(\lambda^{\mathrm{r}}(\hat{\varepsilon}), y, x)$ is an RP-compatible eigentriple of $M(\hat{\varepsilon}UV^{\mathsf{T}})$ for the rank-2 perturbation defined by matrices $U \in \mathbb{R}^{p,2}$ and $V \in \mathbb{R}^{m,2}$ such that $\|UV^{\mathsf{T}}\|_{\mathrm{F}} = 1$. Assuming that* $\operatorname{Re}(\lambda^{\mathrm{r}}(\cdot))$ *is smooth in $\mathcal{N}$ and* $\operatorname{Re}(uv^*) \neq 0$, *where*

$$\begin{aligned} u &:= (I - \hat{\varepsilon}UV^{\mathsf{T}}D)^{-\mathsf{T}}B^{\mathsf{T}}y \\ v &:= (I - D\hat{\varepsilon}UV^{\mathsf{T}})^{-1}Cx, \end{aligned} \tag{6.25}$$

then

$$\frac{d}{d\varepsilon}\operatorname{Re}(\lambda^{\mathrm{r}}(\varepsilon))\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{\|\operatorname{Re}(uv^*)\|_{\mathrm{F}}}{y^*x}. \tag{6.26}$$

We now turn to the equivalence of (6.26) and the continuous-time derivative specified in (6.19) for the contraction phase.

Lemma 6.14. *Let $\hat{\varepsilon} \in \mathbb{R}^{++}$, $\hat{\varepsilon}\|D\|_2 < 1$ and $\lambda_{UV} \in \sigma_{\hat{\varepsilon}}^{\mathbb{R},\|\cdot\|_{\mathrm{F}}}(A,B,C,D)$ be such that (λ_{UV}, x, y) is an RP-compatible eigentriple of $M_{UV}(\hat{\varepsilon})$ with corresponding matrices $U \in \mathbb{R}^{p,2}$ and $V \in \mathbb{R}^{m,2}$ and $\|UV^{\mathsf{T}}\|_{\mathrm{F}} = 1$. If λ_{UV} is a locally rightmost point of $\sigma_{\hat{\varepsilon}}^{\mathbb{R},\|\cdot\|_{\mathrm{F}}}(A,B,C,D)$, then*

$$g_{UV}^{\prime c}(\varepsilon)\bigg|_{\varepsilon=\hat{\varepsilon}} = \frac{\|\operatorname{Re}(uv^*)\|_{\mathrm{F}}}{y^*x} \tag{6.27}$$

where u and v are defined in (6.25).

Proof. By the proof of Theorem 6.13 contained in [GGMO14], we have the following equivalences:

$$\begin{aligned}\| \operatorname{Re}(uv^*)\|_{\mathrm{F}} &= \operatorname{Re}\left(u^*\left(\hat{\varepsilon}UV^\mathsf{T}\right)v\right) \\ &= \operatorname{Re}\left(y^*B(I-\hat{\varepsilon}UV^\mathsf{T}D)^{-1}\hat{\varepsilon}UV^\mathsf{T}(I-D\hat{\varepsilon}UV^\mathsf{T})^{-1}Cx\right) \qquad (6.28)\end{aligned}$$

By Lemmas 6.8 and 6.9, we may substitute (6.9) and (6.10) into (6.28) yielding

$$\| \operatorname{Re}(uv^*)\|_{\mathrm{F}} = \operatorname{Re}\left(y^*B\hat{\varepsilon}U(I+T(\hat{\varepsilon}))^2V^\mathsf{T}Cx\right),$$

noting that this is the numerator of the continuous-time derivative specified in (6.19) via (6.23), thus demonstrating the result. □

As a consequence, in the case of real spectral value sets bounded by the spectral or Frobenius norm, hybrid expansion-contraction will converge at least superlinearly to a local maximizer $\tilde{\omega}$ of $\mu_{\mathbb{R}}^{\|\cdot\|_{\mathrm{F}}}(G(\mathbf{i}\omega))$, provided $\mu_{\mathbb{R}}^{\|\cdot\|_{\mathrm{F}}}(G(\mathbf{i}\omega))$ is continuous in a neighborhood of $\tilde{\omega}$. Furthermore, as explained in Section 3.2 and empirically shown in Section 5, this is a worst-case convergence rate and we can often expect hybrid expansion-contraction to converge quadratically in practice and in some cases, possibly even faster.

6.3.3 Adapting the SVSAR line search to ensure monotonicity

We now consider a line search approach for ensuring monotonicity of the iterates of SVSAR for the case where $\Delta \in \mathbb{R}^{p,m}$ may be up to rank-2. The resulting algorithm will be analogous to the line search procedure specified by Procedure 3 for the simpler case when Δ is always rank-1, with some modifications.

Recall the perturbed system matrix:

$$M(\Delta) = A + B\Delta(I - D\Delta)^{-1}C,$$

where $\Delta = \varepsilon U V^{\mathsf{T}}$ such that $\|\Delta\| = \varepsilon$ for some norm $\|\cdot\|$. Analogously to the line search derivation of [GGO13] and also presented in Section 2.2, let us consider the perturbation $\Delta(t)$ for $t \in [0,1]$:

$$\Delta(t) = \frac{\varepsilon U(t)V(t)^{\mathsf{T}}}{\|U(t)V(t)^{\mathsf{T}}\|}$$

where

$$\begin{aligned} U(t) &= tU_k + (1-t)U_{k-1} \\ V(t) &= tV_k + (1-t)V_{k-1} \end{aligned}$$

and $U_{k-1} \in \mathbb{R}^{p,2}$ and $V_{k-1} \in \mathbb{R}^{m,2}$ define the perturbed system matrix $M(\varepsilon U_{k-1}V_{k-1}^{\mathsf{T}})$ such that $\|U_{k-1}V_{k-1}^{\mathsf{T}}\| = 1$ while $U_k \in \mathbb{R}^{p,2}$ and $V_k \in \mathbb{R}^{m,2}$ are also normalized such that $\|U_kV_k^{\mathsf{T}}\| = 1$. Furthermore, let $\lambda(t)$ a rightmost or outermost eigenvalue $M(\Delta(t))$, where $\lambda(0) \in \sigma(M(\varepsilon U_{k-1}V_{k-1}^{\mathsf{T}}))$. We will show that if $\operatorname{Re}(\lambda'(0)) < 0$, then flipping the signs of both U_k and V_k will flip the sign of $\operatorname{Re}(\lambda'(0))$ from negative to positive and likewise for $|\lambda(0)|'$.

We thus consider the matrix family for $t \in [0,1]$:

$$\begin{aligned} N(t) &= A + B\Delta(t)(I - D\Delta(t))^{-1}C \\ &= A + B\widetilde{\Delta}(t)C, \end{aligned} \tag{6.29}$$

where $\widetilde{\Delta}(t) \coloneqq \Delta(t)(I - D\Delta(t))^{-1}$ has its corresponding derivative:

$$\begin{aligned}
\widetilde{\Delta}'(t) &= \frac{d}{dt}\Delta(t)(I - D\Delta(t))^{-1} \\
&= \Delta'(t)(I - D\Delta(t))^{-1} + \Delta(t)(I - D\Delta(t))^{-1} D\Delta'(t)(I - D\Delta(t))^{-1} \\
&= \left[I + \Delta(t)(I - D\Delta(t))^{-1}D\right] \Delta'(t)(I - D\Delta(t))^{-1}. \qquad (6.30)
\end{aligned}$$

For a simple eigenvalue $\lambda(t)$ of $N(t)$ with RP-compatible right and left eigenvectors x and y, we have

$$\lambda'(t) = \frac{y^* N'(t) x}{y^* x} = \frac{y^* B \widetilde{\Delta}'(t) C x}{y^* x} \qquad (6.31)$$

and thus, the sign of $\operatorname{Re}(\lambda'(t))$ is determined by the sign of $\widetilde{\Delta}'(t)$, which in turn is determined by the sign of $\Delta'(t)$ as shown in (6.30). Letting

$$\begin{array}{lcl}
f(t) = U(t)V(t)^\mathsf{T} & & f(0) = U_{k-1}V_{k-1}^\mathsf{T} \\
 & \text{where} & \\
g(t) = \|U(t)V(t)^\mathsf{T}\| & & g(0) = \|U_{k-1}V_{k-1}^\mathsf{T}\| = 1,
\end{array} \qquad (6.32)$$

we see that:

$$\Delta'(t) = \frac{d}{dt}\frac{\varepsilon U(t)V(t)^\mathsf{T}}{\|U(t)V(t)^\mathsf{T}\|} = \frac{d}{dt}\frac{\varepsilon f(t)}{g(t)} = \frac{\varepsilon[f'(t)g(t) - f(t)g'(t)]}{[g(t)]^2} = \frac{\varepsilon P(t)}{[g(t)]^2} \qquad (6.33)$$

and thus the sign of $\Delta'(t)$ is determined by:

$$P(t) \coloneqq f'(t)g(t) - f(t)g'(t). \qquad (6.34)$$

Differentiating both equations in (6.32) yields:

$$f'(t) = U'(t)V(t)^\mathsf{T} + U(t)V'(t)^\mathsf{T}$$

so evaluating this at $t = 0$, we have

$$f'(0) = (U_k - U_{k-1})V_{k-1}^{\mathsf{T}} + U_{k-1}(V_k - V_{k-1})^{\mathsf{T}}$$
$$= U_k V_{k-1}^{\mathsf{T}} + U_{k-1}V_k^{\mathsf{T}} - 2U_{k-1}V_{k-1}^{\mathsf{T}}. \tag{6.35}$$

Using (6.32) and (6.35), it follows that (6.34) evaluated at $t = 0$ is

$$P(0) = (U_k V_{k-1}^{\mathsf{T}} + U_{k-1}V_k^{\mathsf{T}} - 2U_{k-1}V_{k-1}^{\mathsf{T}}) - (U_{k-1}V_{k-1}^{\mathsf{T}})g'(0). \tag{6.36}$$

The Frobenius norm case

In the case that $\|\cdot\|$ is the Frobenius norm, the derivative of $g(t)$ given in (6.32) is

$$g'(t) = \frac{d}{dt}\|U(t)V(t)^{\mathsf{T}}\|_{\mathrm{F}} = \frac{d}{dt}\left(\mathrm{Tr}\left((U(t)V(t)^{\mathsf{T}})^{\mathsf{T}}U(t)V(t)^{\mathsf{T}}\right)\right)^{\frac{1}{2}}$$
$$= \frac{\mathrm{Tr}\left(\frac{d}{dt}U(t)^{\mathsf{T}}U(t)V(t)^{\mathsf{T}}V(t)^{\mathsf{T}}\right)}{2\|U(t)V(t)^{\mathsf{T}}\|_{\mathrm{F}}}.$$

Denoting $\widetilde{U}(t) := U(t)^{\mathsf{T}}U(t)$, then

$$\widetilde{U}(t) = t^2 U_k^{\mathsf{T}}U_k + (1-t)^2 U_{k-1}^{\mathsf{T}}U_{k-1} - (1-t)t\left(U_{k-1}^{\mathsf{T}}U_k + U_k^{\mathsf{T}}U_{k-1}\right)$$
$$\widetilde{U}'(t) = 2tU_k^{\mathsf{T}}U_k - 2(1-t)U_{k-1}^{\mathsf{T}}U_{k-1} + (2t-1)\left(U_{k-1}^{\mathsf{T}}U_k + U_k^{\mathsf{T}}U_{k-1}\right)$$

and

$$\widetilde{U}(0) = U_{k-1}^{\mathsf{T}}U_{k-1}$$
$$\widetilde{U}'(0) = -2U_{k-1}^{\mathsf{T}}U_{k-1} - U_{k-1}^{\mathsf{T}}U_k - U_k^{\mathsf{T}}U_{k-1}$$

and for $\widetilde{V}(t) \coloneqq V(t)^\mathsf{T} V(t)$, we also have the analogous equations for $\widetilde{V}(t)$ and $\widetilde{V}'(t)$ evaluated at $t = 0$. Recalling that $\|U_{k-1}V_{k-1}^\mathsf{T}\|_\mathrm{F} = 1$, we have that

$$
\begin{aligned}
g'(0) = \frac{1}{2} \operatorname{Tr}\left(\frac{d}{dt}\bigg|_{t=0} \widetilde{U}(t)\widetilde{V}(t)^\mathsf{T} \right) &= \frac{1}{2} \operatorname{Tr}\left(\widetilde{U}'(0)\widetilde{V}(0)^\mathsf{T} + \widetilde{U}(0)\widetilde{V}'(0)^\mathsf{T} \right) \\
&= \frac{1}{2} \operatorname{Tr}\left(W\right) \qquad (6.37)
\end{aligned}
$$

where

$$
\begin{aligned}
W = &\left(-2U_{k-1}^\mathsf{T} U_{k-1} - U_{k-1}^\mathsf{T} U_k - U_k^\mathsf{T} U_{k-1}\right) V_{k-1}^\mathsf{T} V_{k-1} \\
&+ U_{k-1}^\mathsf{T} U_{k-1} \left(-2V_{k-1}^\mathsf{T} V_{k-1} - V_{k-1}^\mathsf{T} V_k - V_k^\mathsf{T} V_{k-1}\right)^\mathsf{T}.
\end{aligned}
$$

Regrouping terms, we see that:

$$
\begin{aligned}
W = &-4U_{k-1}^\mathsf{T} U_{k-1} V_{k-1}^\mathsf{T} V_{k-1} - \left(U_{k-1}^\mathsf{T} U_k + U_k^\mathsf{T} U_{k-1}\right) V_{k-1}^\mathsf{T} V_{k-1} \\
&- U_{k-1}^\mathsf{T} U_{k-1} \left(V_{k-1}^\mathsf{T} V_k + V_k^\mathsf{T} V_{k-1}\right)
\end{aligned}
$$

and thus, since $\operatorname{Tr}(4U_{k-1}^\mathsf{T} U_{k-1} V_{k-1}^\mathsf{T} V_{k-1}) = 4\|U_{k-1}V_{k-1}^\mathsf{T}\|_\mathrm{F}^2 = 4$,

$$
\begin{aligned}
\operatorname{Tr}(W) = -4 &- \operatorname{Tr}\left(\left(U_{k-1}^\mathsf{T} U_k + U_k^\mathsf{T} U_{k-1}\right) V_{k-1}^\mathsf{T} V_{k-1}\right) \\
&- \operatorname{Tr}\left(U_{k-1}^\mathsf{T} U_{k-1} \left(V_{k-1}^\mathsf{T} V_k + V_k^\mathsf{T} V_{k-1}\right)\right).
\end{aligned} \qquad (6.38)
$$

Using (6.37), we can rewrite (6.36) for the Frobenius norm case as:

$$
P(0) = \left(U_k V_{k-1}^\mathsf{T} + U_{k-1} V_k^\mathsf{T} - 2U_{k-1} V_{k-1}^\mathsf{T}\right) - U_{k-1} V_{k-1}^\mathsf{T} \left(\frac{1}{2} \operatorname{Tr}\left(W\right)\right). \qquad (6.39)
$$

Substituting (6.38) into (6.39) cancels the $\pm 2U_{k-1}V_{k-1}^{\mathsf{T}}$ terms, yielding

$$P(0) = U_k V_{k-1}^{\mathsf{T}} + U_{k-1} V_k^{\mathsf{T}} + \eta^{\mathbb{R},\mathrm{F}} U_{k-1} V_{k-1} \tag{6.40}$$

where

$$\begin{aligned} \eta^{\mathbb{R},\mathrm{F}} = -\frac{1}{2}\Big[& \operatorname{Tr}\left(\left(U_{k-1}^{\mathsf{T}} U_k + U_k^{\mathsf{T}} U_{k-1}\right) V_{k-1}^{\mathsf{T}} V_{k-1}\right) \\ & + \operatorname{Tr}\left(U_{k-1}^{\mathsf{T}} U_{k-1}\left(V_{k-1}^{\mathsf{T}} V_k + V_k^{\mathsf{T}} V_{k-1}\right)\right)\Big]. \end{aligned} \tag{6.41}$$

It is now clear that the sign of (6.40) flips if the signs of U_k and V_k are both flipped and thus for the Frobenius norm case, we have that the sign of

$$\Delta'(0) = \varepsilon\left[U_k V_{k-1}^{\mathsf{T}} + U_{k-1} V_k^{\mathsf{T}} - \eta^{\mathbb{R},\mathrm{F}} U_{k-1} V_{k-1}^{\mathsf{T}}\right]. \tag{6.42}$$

also flips, thereby demonstrating the result.

The spectral norm case

In the case that $\|\cdot\|$ is the spectral norm, the derivative of $g(t)$ given in (6.32) is

$$g'(t) = \frac{d}{dt}\|U(t)V(t)^{\mathsf{T}}\|_2. \tag{6.43}$$

Let

$$\begin{aligned} R(t) &\coloneqq U(t)V(t)^{\mathsf{T}} && (6.44) \\ &= \left(tU_k + (1-t)U_{k-1}\right)\left(tV_k + (1-t)V_{k-1}\right)^{\mathsf{T}}, && (6.45) \end{aligned}$$

which has the corresponding derivative

$$R'(t) = 2tU_kV_k^\mathsf{T} + (1-2t)\left[U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T}\right] + (2t-2)U_{k-1}V_{k-1}^\mathsf{T}. \tag{6.46}$$

Consider the function $\sigma_R(t) \coloneqq \|R(t)\|_2$ where $\sigma_R(0) = \|R(0)\|_2 = \|U_{k-1}V_{k-1}^\mathsf{T}\|_2 = 1$ and let u_R and v_R denote the right and left singular vectors of $\sigma_R(0)$. Assuming $\sigma_R(0)$ is a simple singular value, by standard perturbation theory for singular values, [HJ90, Theorem 7.3.7], [GO11, Lemma 2.3], we have that

$$\begin{aligned} g'(0) = \frac{d}{dt}\sigma_R(t)\bigg|_{t=0} &= u_R^* R'(0) v_R \\ &= u_R^*\left(U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T} - 2U_{k-1}V_{k-1}^\mathsf{T}\right)v_R \\ &= u_R^*\left(U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T}\right)v_R - 2u_R^*R(0)v_R \\ &= u_R^*\left(U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T}\right)v_R - 2. \end{aligned} \tag{6.47}$$

Substituting (6.47) into (6.36) and noting that the $\pm 2U_{k-1}V_{k-1}^\mathsf{T}$ now cancel, we have for the spectral norm case that:

$$P(0) = U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T} - \eta^{\mathbb{R},2}U_{k-1}V_{k-1}^\mathsf{T} \tag{6.48}$$

where

$$\eta^{\mathbb{R},2} = u_R^*\left(U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T}\right)v_R. \tag{6.49}$$

As in the Frobenius norm case, it is now clear that the sign of (6.48) flips if the signs of U_k and V_k are both flipped and thus for the spectral norm case, we have that the sign of

$$\Delta'(0) = \varepsilon\left[U_kV_{k-1}^\mathsf{T} + U_{k-1}V_k^\mathsf{T} - \eta^{\mathbb{R},2}U_{k-1}V_{k-1}^\mathsf{T}\right]. \tag{6.50}$$

also flips, thereby demonstrating the result.

Calculating the line search derivative

In order to derive the exact formula for (6.31), that is the corresponding value to ψ_k defined in (2.12) when the perturbations are instead restricted to be real and may thus be up to rank-2, it will be useful to have the following definitions.

First define

$$\begin{aligned} S_1(t) &\coloneqq I + \Delta(t) \left(I - D\Delta(t)\right)^{-1} D \\ S_2(t) &\coloneqq \Delta'(t) \left(I - D\Delta(t)\right)^{-1} \end{aligned}$$

so that we may rewrite (6.30) as

$$\begin{aligned} \widetilde{\Delta}'(t) &= \left[I + \Delta(t)(I - D\Delta(t))^{-1} D\right] \Delta'(t)(I - D\Delta(t))^{-1} \\ &= S_1(t) S_2(t). \end{aligned} \tag{6.51}$$

We further define

$$\widehat{U} \coloneqq U_k - \eta^{\mathbb{R}} U_{k-1} \tag{6.52}$$

$$T(\varepsilon) \coloneqq \varepsilon \left(V_{k-1}^{\mathsf{T}} D U_{k-1}\right) \left(I - \varepsilon V_{k-1}^{\mathsf{T}} D U_{k-1}\right)^{-1} \tag{6.53}$$

$$\widehat{T}(\varepsilon) \coloneqq \varepsilon \left(V_k^{\mathsf{T}} D U_{k-1}\right) \left(I - \varepsilon V_{k-1}^{\mathsf{T}} D U_{k-1}\right)^{-1}, \tag{6.54}$$

where $\eta^{\mathbb{R}}$ is either defined by (6.41) or (6.49) and noting that only difference between $T(\varepsilon)$ and $\widehat{T}(\varepsilon)$ is that the first occurrence of V_{k-1} is replaced with V_k. Therefore, we generically have for the Frobenius and spectral norm cases that

$$\Delta'(0) = \varepsilon \left[U_k V_{k-1}^\mathsf{T} + U_{k-1} V_k^\mathsf{T} - \eta^\mathbb{R} U_{k-1} V_{k-1}^\mathsf{T}\right]. \tag{6.55}$$

By Lemma 6.8 along with (6.9) of Lemma 6.9 and (6.53), a straightforward calculation shows that

$$\begin{aligned} S_1(0) &= I + \Delta(0) \left(I - D\Delta(0)\right)^{-1} D \\ &= I + \varepsilon U_{k-1} V_{k-1}^\mathsf{T} \left(I - D\varepsilon U_{k-1} V_{k-1}^\mathsf{T}\right)^{-1} D \\ &= I + \varepsilon U_{k-1} \left(I + T(\varepsilon)\right) V_{k-1}^\mathsf{T} D. \end{aligned} \tag{6.56}$$

Also by a straightforward calculation, using the identity of (6.12) and substituting in (6.52) and (6.55), we correspondingly have that

$$\begin{aligned} S_2(0) &= \Delta'(0) \left(I - D\Delta(0)\right)^{-1} \\ &= \varepsilon \left[U_k V_{k-1}^\mathsf{T} + U_{k-1} V_k^\mathsf{T} - \eta^\mathbb{R} U_{k-1} V_{k-1}^\mathsf{T}\right] \left(I - D\varepsilon U_{k-1} V_{k-1}^\mathsf{T}\right)^{-1} \\ &= \varepsilon \left[\widehat{U} V_{k-1}^\mathsf{T} + U_{k-1} V_k^\mathsf{T}\right] \left(I - D\varepsilon U_{k-1} V_{k-1}^\mathsf{T}\right)^{-1} \\ &= \varepsilon \left[\widehat{U} V_{k-1}^\mathsf{T} + U_{k-1} V_k^\mathsf{T}\right] \left(I + \varepsilon D U_{k-1} \left(I - \varepsilon V_{k-1}^\mathsf{T} D U_{k-1}\right)^{-1} V_{k-1}^\mathsf{T}\right). \end{aligned}$$

Carrying out the multiplication, regrouping terms, and substituting in (6.53) and (6.54), we see that

$$\begin{aligned}
S_2(0) &= \varepsilon\widehat{U}V_{k-1}^{\mathsf{T}}\left(I + \varepsilon D U_{k-1}\left(I - \varepsilon V_{k-1}^{\mathsf{T}} D U_{k-1}\right)^{-1} V_{k-1}^{\mathsf{T}}\right) \\
&\qquad + \varepsilon U_{k-1} V_k^{\mathsf{T}}\left(I + \varepsilon D U_{k-1}\left(I - \varepsilon V_{k-1}^{\mathsf{T}} D U_{k-1}\right)^{-1} V_{k-1}^{\mathsf{T}}\right) \\
&= \varepsilon\widehat{U}\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}} + \varepsilon U_{k-1}\left[V_k^{\mathsf{T}} + \widehat{T}(\varepsilon) V_{k-1}^{\mathsf{T}}\right]. \qquad (6.57)
\end{aligned}$$

Substituting both (6.56) and (6.57) into (6.51) and then carrying out the multiplication and regrouping terms yields

$$\begin{aligned}
\widetilde{\Delta}'(0) &= S_1(0) S_2(0) \\
&= \varepsilon\widehat{U}\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}} + \varepsilon U_{k-1}\left[V_k^{\mathsf{T}} + \widehat{T}(\varepsilon) V_{k-1}^{\mathsf{T}}\right] \\
&\qquad + \varepsilon U_{k-1}\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}} D\left(\varepsilon\widehat{U}\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}}\right) \\
&\qquad + \varepsilon U_{k-1}\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}} D\left(\varepsilon U_{k-1}\left[V_k^{\mathsf{T}} + \widehat{T}(\varepsilon) V_{k-1}^{\mathsf{T}}\right]\right) \\
&= \varepsilon\left[\widehat{U} + \varepsilon U_{k-1}\left(I + T(\varepsilon)\right)\left(V_{k-1}^{\mathsf{T}} D\widehat{U}\right)\right]\left(I + T(\varepsilon)\right) V_{k-1}^{\mathsf{T}} \qquad (6.58) \\
&\qquad + \varepsilon U_{k-1}\left(I + \varepsilon\left(I + T(\varepsilon)\right)\left(V_{k-1}^{\mathsf{T}} D U_{k-1}\right)\right)\left[V_k^{\mathsf{T}} + \widehat{T}(\varepsilon) V_{k-1}^{\mathsf{T}}\right].
\end{aligned}$$

Letting

$$\begin{aligned}
\bar{\bar{U}} &\coloneqq \left[\widehat{U} + \varepsilon U_{k-1}\left(I + T(\varepsilon)\right)\left(V_{k-1}^{\mathsf{T}} D\widehat{U}\right)\right] \qquad (6.59) \\
\bar{\bar{V}} &\coloneqq \left[V_k^{\mathsf{T}} + \widehat{T}(\varepsilon) V_{k-1}^{\mathsf{T}}\right]^{\mathsf{T}}, \qquad (6.60)
\end{aligned}$$

and noting that $\bar{\bar{U}} \in \mathbb{R}^{p,2}$ and $\bar{\bar{V}} \in \mathbb{R}^{m,2}$, using (6.58), (6.59), and (6.60), we can finally completely derive the numerator of (6.31) evaluated at $t = 0$ as

$$\begin{aligned}\psi_k^{\mathbb{R}} &= y^* B \widetilde{\Delta}'(0) C x \\ &= y^* B \left[\varepsilon \bar{\bar{U}} \left(I + T(\varepsilon) \right) V_{k-1}^{\mathsf{T}} \right] C x \\ &\qquad + y^* B \left[\varepsilon U_{k-1} \left(I + \varepsilon \left(I + T(\varepsilon) \right) \left(V_{k-1}^{\mathsf{T}} D U_{k-1} \right) \right) \bar{\bar{V}}^{\mathsf{T}} \right] C x.\end{aligned}$$

Regrouping terms was once again yields:

$$\begin{aligned}\psi_k^{\mathbb{R}} &= \varepsilon \left[y^* \left(B \bar{\bar{U}} \right) \right] \left[I + T(\varepsilon) \right] \left[\left(V_{k-1}^{\mathsf{T}} C \right) x \right] \\ &\quad + \varepsilon \left[y^* \left(B U_{k-1} \right) \right] \left[I + \varepsilon \left(I + T(\varepsilon) \right) \left(V_{k-1}^{\mathsf{T}} D U_{k-1} \right) \right] \left[\left(\bar{\bar{V}}^{\mathsf{T}} C \right) x \right],\end{aligned} \tag{6.61}$$

using (6.52), (6.53), (6.54), (6.59), (6.60) and either (6.41) or (6.49) for the Frobenius and spectral norm cases respectively. For the special case of $D = 0$, (6.61) simplifies to

$$\psi_k^{\mathbb{R}} = \varepsilon \left[y^* \left(B \widehat{U} \right) \right] \left[\left(V_{k-1}^{\mathsf{T}} C \right) x \right] + \varepsilon \left[y^* \left(B U_{k-1} \right) \right] \left[\left(V_k^{\mathsf{T}} C \right) x \right]. \tag{6.62}$$

While we note that (6.61) is somewhat complicated, it is actually quite efficient to compute. Assuming A, B, C and D are all sparse matrices, the specified multiplication groupings allow (6.61) to be efficiently computed in $\mathcal{O}(n + p + m)$ operations and furthermore, only requires two matrix vector products per matrices B, C^{T}, and D.

6.3.4 Normalizing large-scale rank-2 perturbations

In the case of complex perturbations, where we have rank-1 perturbations defined by uv^* with $u \in \mathbb{C}^p$ and $v \in \mathbb{C}^m$, it is efficient and straightforward to normalize u and v such that $\|u\|_2 = \|v\|_2 = 1$ to ensure that $\|uv^*\|_2 = 1$ holds. However, by restricting perturbations to be real, we must now consider normalizing rank-2 perturbations UV^T to have unit norm, where $U \in \mathbb{R}^{p,2}$ and $V \in \mathbb{R}^{m,2}$.

For the Frobenius case, $\|UV^\mathsf{T}\|_\mathrm{F}$ has an efficient formulation with only $\mathcal{O}(p+m)$ operations by the following equivalences:

$$\begin{aligned}
\|UV^\mathsf{T}\|_\mathrm{F} &= \left[\mathrm{Tr}\left(\left(UV^\mathsf{T}\right)^* \left(UV^\mathsf{T}\right)\right)\right]^{\frac{1}{2}} \\
&= \left[\mathrm{Tr}\left(VU^\mathsf{T}UV^\mathsf{T}\right)\right]^{\frac{1}{2}} \\
&= \left[\mathrm{Tr}\left(\left(U^\mathsf{T}U\right)\left(V^\mathsf{T}V\right)\right)\right]^{\frac{1}{2}}.
\end{aligned} \tag{6.63}$$

In the case of the spectral norm, there does not seem to be a simple analogous formula. The routine `svds` in MATLAB would perform a sparse SVD decomposition of UV^T by computing the eigenvalues and eigenvectors of the matrix

$$\begin{bmatrix} 0 & UV^\mathsf{T} \\ (UV^\mathsf{T})^* & 0 \end{bmatrix} = \begin{bmatrix} 0 & UV^\mathsf{T} \\ VU^\mathsf{T} & 0 \end{bmatrix} \tag{6.64}$$

via `eigs`. Unfortunately, and in contrast to `eigs`, the MATLAB R2014a version of `svds` does not currently provide a facility to supply the matrix as a function handle for performing the matrix-vector product and instead must be provided with an explicit matrix as input. However, it is potentially very expensive to form (6.64) explicitly, as UV^T is typically dense and both p and m might be large. Fortunately, it is fairly easy to create a simple routine to compute the largest singular value of UV^T, along with its right and left singular vectors, by providing a function handle to `eigs` that multiples a vector by (6.64) without forming UV^T. Each matrix-vector product is $\mathcal{O}(p+m)$ operations and we have found that `eigs` consistently converges to the largest singular value in only one iteration when using the default Krylov subspace dimension of 20, that is a total of 20 matrix-vector products, for randomly generated $U \in \mathbb{R}^{p,2}$ and $V \in \mathbb{R}^{m,2}$ matrices. Though we cannot empirically assess the accuracy of this approach when p and m are both large, we have found it to be accurate on smaller dimensional test problems where we can tractably call `svd` for comparison while it is also able to efficiently handle cases up to when $p = m = 10^6$, which was the largest we tested.

6.3.5 Vector extrapolation for rank-2 perturbations

For the real Frobenius-bounded perturbation case, we must consider the sequences of perturbations defined by sequences $\{U_1, \ldots, U_k\}$ and $\{V_1, \ldots, V_k\}$, which may be mixed sequences of single and double column matrices and where $\|U_l V_l\|^\mathsf{T} = 1$ holds for $l = 1, \ldots, k$. In the case that the sequences are comprised entirely of single column matrices, we may just use the rank-1 perturbation extrapolation procedure described in Section 4.2.1. We also exclude the case where either $m = 1$ or $p = 1$, as the perturbation matrices must then also all be rank-1 in that case. Otherwise, if the sequences contain double column matrices, then we must consider the possibility that the extrapolated matrix $\widehat{\Delta}_\star = U_\star V_\star^\mathsf{T}$ may be rank-2 instead of rank-1.

Let

$$\begin{aligned} r_\star^{(1)} &= \widehat{\Delta}_\star(i_1, :) \quad & c_\star^{(1)} &= \widehat{\Delta}_\star(:, j_1) \\ r_\star^{(2)} &= \widehat{\Delta}_\star(i_2, :) \quad & c_\star^{(2)} &= \widehat{\Delta}_\star(:, j_2) \end{aligned}$$

for some $\{i_1, i_2\} \subset \{1, \ldots, p\}$ such that $i_1 \neq i_2$ and similarly, for some $\{j_1, j_2\} \subset \{1, \ldots, m\}$ such that $j_1 \neq j_2$. Since $\widehat{\Delta}_\star$ may be rank-2, we construct the two row vectors

$$r_k^{(1)} = \widehat{\Delta}_k(i_1, :) = u_k^{(1)}(i_1)v_k^{(1)\mathsf{T}} + u_k^{(2)}(i_1)v_k^{(2)\mathsf{T}}$$
$$r_k^{(2)} = \widehat{\Delta}_k(i_2, :) = u_k^{(1)}(i_2)v_k^{(1)\mathsf{T}} + u_k^{(2)}(i_2)v_k^{(2)\mathsf{T}}$$

and two column vectors

$$c_k^{(1)} = \widehat{\Delta}_k(:, j_1) = v_k^{(1)}(j_1)u_k^{(1)} + v_k^{(2)}(j_1)u_k^{(2)}$$
$$c_k^{(2)} = \widehat{\Delta}_k(:, j_2) = v_k^{(1)}(j_2)u_k^{(1)} + v_k^{(2)}(j_2)u_k^{(2)}$$

where

$$\begin{matrix} u_k^{(1)} = U_k(:,1) \\ v_k^{(1)} = V_k(:,1) \end{matrix} \quad \text{and} \quad \begin{pmatrix} u_k^{(2)} = U_k(:,2) & \text{or} & u_k^{(2)} = 0 \in \mathbb{R}^{p\times 1} \text{ if } U_k \in \mathbb{R}^{p\times 1} \\ v_k^{(2)} = V_k(:,2) & & v_k^{(2)} = 0 \in \mathbb{R}^{m\times 1} \text{ if } V_k \in \mathbb{R}^{m\times 1} \end{pmatrix},$$

and similarly construct the row vectors $r_1^{(1)}, \ldots, r_{k-1}^{(1)}$ and $r_1^{(2)}, \ldots, r_{k-1}^{(2)}$ and the column vectors $c_1^{(1)}, \ldots, c_{k-1}^{(1)}$ and $c_1^{(2)}, \ldots, c_{k-1}^{(2)}$. We then apply vector extrapolation to each of the four sequences:

$$\{r_1^{(1)}, \ldots, r_k^{(1)}\} \quad \{c_1^{(1)}, \ldots, c_k^{(1)}\}$$
$$\{r_1^{(2)}, \ldots, r_k^{(2)}\} \quad \{c_1^{(2)}, \ldots, c_k^{(2)}\}$$

to yield $r_\star^{(1)}$, $r_\star^{(2)}$, $c_\star^{(1)}$ and $c_\star^{(2)}$. As

$$r_\star^{(1)} = u_\star^{(1)}(i_1)v_\star^{(1)\mathsf{T}} + u_\star^{(2)}(i_1)v_\star^{(2)\mathsf{T}} \tag{6.65}$$
$$r_\star^{(2)} = u_\star^{(1)}(i_2)v_\star^{(1)\mathsf{T}} + u_\star^{(2)}(i_2)v_\star^{(2)\mathsf{T}} \tag{6.66}$$

and

$$c_\star^{(1)} = v_\star^{(1)}(j_1)u_\star^{(1)} + v_k^{(2)}(j_1)u_\star^{(2)} \tag{6.67}$$

$$c_\star^{(2)} = v_\star^{(1)}(j_2)u_\star^{(1)} + v_k^{(2)}(j_2)u_\star^{(2)}, \tag{6.68}$$

recovering $U_\star \in \mathbb{R}^{p\times 2}$ and $V_\star \in \mathbb{R}^{m\times 2}$ requires determining $2(m+p)$ unknowns. However, (6.65), (6.66), (6.67) and (6.68) only provide $2(m+p)-4$ equations since $r_\star^{(1)}$, $r_\star^{(2)}$, $c_\star^{(1)}$, and $c_\star^{(2)}$ combined share four entries of $\widehat{\Delta}_\star$. Thus, we are free to choose four entries of say $V_\star$ and assume them to be as follows:

$$\begin{aligned} v_\star^{(1)}(j_1) = 1 \quad v_\star^{(2)}(j_1) = 1 \\ v_\star^{(1)}(j_2) = 1 \quad v_\star^{(2)}(j_2) = -1. \end{aligned} \tag{6.69}$$

Substituting (6.69) into (6.67) and (6.68) and then adding (6.67) and (6.68) yields:

$$u_\star^{(1)} = \frac{c_\star^{(1)} + c_\star^{(2)}}{2}$$

and likewise subtracting (6.68) from (6.67) yields:

$$u_\star^{(2)} = \frac{c_\star^{(1)} - c_\star^{(2)}}{2}.$$

With $u_\star^{(1)}$ and $u_\star^{(2)}$ both determined numerically, we may now recover

$$v_\star^{(1)} = \frac{u_\star^{(2)}(i_2)r_\star^{(1)\mathsf{T}} - u_\star^{(2)}(i_1)r_\star^{(2)\mathsf{T}}}{u_\star^{(1)}(i_1)u_\star^{(2)}(i_2) - u_\star^{(1)}(i_2)u_\star^{(2)}(i_1)} \tag{6.70}$$

and

$$v_\star^{(2)} = \frac{u_\star^{(1)}(i_2) r_\star^{(1)\mathsf{T}} - u_\star^{(1)}(i_1) r_\star^{(2)\mathsf{T}}}{u_\star^{(1)}(i_2) u_\star^{(2)}(i_1) - u_\star^{(1)}(i_1) u_\star^{(2)}(i_2)}, \tag{6.71}$$

provided that $u_\star^{(1)}$ and $u_\star^{(2)}$ are rank two, as otherwise (6.70) and (6.71) will be undefined as their denominators will be zero. If this is indeed the case, we may instead fallback to just attempting a rank one extrapolation via the method in Section 4.2.1. Otherwise, assuming that both $u_\star^{(1)}$ and $u_\star^{(2)}$ are rank two, we recover

$$U_\star = \left[\begin{array}{cc} u_\star^{(1)} & u_\star^{(2)} \end{array} \right] \quad \text{and} \quad V_\star = \left[\begin{array}{cc} v_\star^{(1)} & v_\star^{(2)} \end{array} \right].$$

Analogously to the rank-1 case, there is no guarantee that $\|U_\star V_\star^\mathsf{T}\| = 1$ so we must finally renormalize by computing

$$\hat{\beta} \coloneqq \sqrt{\|U_\star V_\star^\mathsf{T}\|}$$

and setting

$$U_\star \coloneqq \frac{U_\star}{\hat{\beta}} \quad \text{and} \quad V_\star \coloneqq \frac{V_\star}{\hat{\beta}}.$$

Given the divisions necessary to compute (6.70) and (6.71), we thus will strive to choose a set of indices $\{i_1, i_2, j_1, j_2\}$ such that

$$|u_\star^{(1)}(i_1) u_\star^{(2)}(i_2) - u_\star^{(1)}(i_2) u_\star^{(2)}(i_1)| \gg 0.$$

By noting the equivalence

$$\begin{aligned} 2\left(u_\star^{(1)}(i_1) u_\star^{(2)}(i_2) - u_\star^{(1)}(i_2) u_\star^{(2)}(i_1)\right) = \\ \widehat{\Delta}_\star(i_1, j_1)\widehat{\Delta}_\star(i_2, j_2) - \widehat{\Delta}_\star(i_1, j_2)\widehat{\Delta}_\star(i_2, j_1), \end{aligned} \tag{6.72}$$

we could our indices so that the absolute value of the righthand side of (6.72) is sufficiently larger than 0. We might consider using U_k and V_k as approximations to $U_\star$ and $V_\star$, and then select $\{i_1, j_1\}$ as is done in the rank-1 case (which attempts to maximize $|\widehat{\Delta}_k(i_1, j_1)|$) and then randomly choose $\{i_2, j_2\}$ randomly and compute:

$$\tau := \left|\widehat{\Delta}_k(i_1, j_1)\widehat{\Delta}_k(i_2, j_2) - \widehat{\Delta}_k(i_1, j_2)\widehat{\Delta}_k(i_2, j_1)\right|. \tag{6.73}$$

If τ is not sufficiently large enough, we choose another set of indices randomly and recompute τ until we have found an acceptable choice of pivot indices. However, U_k and V_k may be single vectors, even if $U_\star$ and $V_\star$ are rank-2. Thus, it may just be more efficient to choose all four indices randomly and only discard them and choose again if (6.73) is too small.

Unfortunately, we do not know *a priori* whether the extrapolated row pair and column pair will constitute a rank-1 or rank-2 perturbation. For example, if the sequence of perturbation matrices is alternating between rank-1 and rank-2, it may be unclear whether it is converging to a rank-1 or rank-2 perturbation. However, we note that constructing a rank-1 extrapolation is a fairly minimal cost if a rank-2 construction has already been attempted (even if it failed), since we can reuse the extrapolated vectors $r_\star^{(1)}$ and $c_\star^{(1)}$ for the rank-1 recovery from $r_\star$ and $c_\star$ with $i = i_1$ and $j = j_1$. Thus, it may be wise to just always compute both reconstructions and then choose whichever one better approximates the extrapolated pair of rows and pair of columns of the perturbation matrix we have attempted to extrapolate implicitly. We thus compare the residual for the rank-2 reconstruction:

$$\left\| \begin{bmatrix} r_\star^{(1)} \\ r_\star^{(2)} \end{bmatrix} - \begin{bmatrix} U_\star(i_1,:) \\ U_\star(i_2.\,:) \end{bmatrix} V_\star^\mathsf{T} \right\| + \left\| \begin{bmatrix} c_\star^{(1)} c_\star^{(2)} \end{bmatrix} - U_\star \begin{bmatrix} V_\star(j_1,:) \\ V_\star(j_2.\,:) \end{bmatrix}^\mathsf{T} \right\|$$

with the residual of the corresponding rank-1 reconstruction:

$$\|r_\star - u_\star(i) v_\star^*\| + \left\| c_\star - \overline{v_\star(j)} u_\star \right\|$$

and select whichever yields a smaller residual.

6.4 Handling static boundary points of real spectral value sets

We now present an outline of a procedure to handle minimizing the perturbation level if in fact hybrid expansion-contraction converges to a static boundary point. For brevity, we only present the continuous-time case as it is straightforward to adapt the procedure for the discrete-time case.

Let $\lambda_{UV} \in \mathbb{C}$ and $\varepsilon \in \mathbb{R}^{++}$ with $\varepsilon\|D\|_2 < 1$ be the pair of values that hybrid expansion-contraction converges to, where λ_{UV} is an eigenvalue of $M(\varepsilon UV^\mathsf{T})$ and a locally rightmost point of $\sigma_\varepsilon^{\mathbb{R},\|\cdot\|}(A,B,C,D)$ with $\operatorname{Re}(\lambda_{UV}) = 0$. However, assume that λ_{UV} is a static boundary point and furthermore, that ε is not its corresponding minimal value. That is, there exists some minimal value $0 < \varepsilon_{\tilde\star} < \varepsilon$ such that λ is also a boundary point of $\sigma_{\varepsilon_{\tilde\star}}^{\mathbb{R},\|\cdot\|}(A,B,C,D)$. As in the contraction phase of hybrid expansion-contraction, let $\lambda_{UV}(t)$ be the path of eigenvalues for fixed U and V, that is $\lambda_{UV}(t) \in \sigma(M(tUV^\mathsf{T}))$, and consider running RSVSAR initialized at $\lambda_{UV}(\hat\varepsilon)$ for some $\hat\varepsilon < \varepsilon$ to find a locally rightmost of $\sigma_{\hat\varepsilon}^{\mathbb{R},\|\cdot\|}(A,B,C,D)$. Under the assumption that RSVSAR always converges to a locally rightmost point, there are three possible outcomes:

1. RSVSAR converges to a locally rightmost point $\hat{\lambda}$ strictly in the right-half plane. This is in fact the best case scenario as it means that hybrid expansion-contraction can be restarted to further reduce to a perturbation level smaller than $\hat{\varepsilon}$. However, it is possible that hybrid expansion-contraction may again converge to a static boundary point. So we must still consider contracting $\hat{\varepsilon}$ to some even smaller value $\tilde{\varepsilon}$.

2. RSVSAR converges to a locally rightmost point $\hat{\lambda}$ on the imaginary axis. In this case, we have successfully reduced ε to some smaller value $\hat{\varepsilon}$ but $\hat{\lambda}$ may still also be a static boundary point. As above, we must again consider contracting $\hat{\varepsilon}$ to even smaller value $\tilde{\varepsilon}$.

3. RSVSAR converges to a locally rightmost point $\hat{\lambda}$ strictly in the left half-plane. In this case and similar to the breakdown case of the GGO algorithm, we do not know if $\hat{\varepsilon} < \varepsilon_{\star}$ holds. Furthermore, we do not know if there exists a single continuous branch of locally rightmost points $\lambda^{\mathrm{r}}(t)$ such that $\lambda^{\mathrm{r}}(t)$ goes through both λ_{UV} and $\hat{\lambda}$, that is $\lambda^{\mathrm{r}}(\varepsilon) = \lambda_{UV}$ and $\lambda^{\mathrm{r}}(\hat{\varepsilon}) = \hat{\lambda}$. Thus, we reject both $\hat{\varepsilon}$ and $\hat{\lambda}$, and return to λ_{UV} and consider a lesser contraction, that is $\lambda_{UV}(\tilde{\varepsilon})$ where $\hat{\varepsilon} < \tilde{\varepsilon} < \varepsilon$.

If we repeat this process of contraction into the left half-plane and expanding rightward in a loop to continually reduce and update ε, as prescribed by outcomes one, two, and three, it follows that the process must eventually terminate, due to the facts that the optimization problem is bounded below and the perturbation level is monotonically decreasing. Furthermore, the method still converges to a locally rightmost point of a real spectral value set along the imaginary axis, since it has to throw away any iterate $\hat{\lambda}$ that is in the left half-plane while if it is in the right half-plane, hybrid expansion-contraction is restarted, thereby producing a new locally rightmost point on the imaginary axis. Finally, the process cannot stagnate as at every iteration of the loop, the method is free to try any reduction of the current perturbation level to attempt to move the static boundary point to the left of the imaginary axis. As a consequence of monotonicity and local continuity of the spectral value set at λ, the method will eventually converge to a point where even the slightest contraction will break the static boundary point off of the imaginary axis, thereby indicating that the method has converged to $\hat{\varepsilon}$, assuming outcome one did not occur. Otherwise, if outcome one does occur, the method converges to some other locally minimal value.

One potential problem of this method for handling static boundary points is that convergence will most likely be slow and furthermore, it is unclear how to choose how much to contract by, beyond guessing and bisection steps, since indeed the boundary point is static. In the case of outcome three, it may sometimes be possible to use a Newton step computed using the point $\hat{\lambda}$ which is strictly in the left half-plane to help locate $\varepsilon_{\tilde{\star}}$ but it must be that for such a Newton step ε^{N} that $\varepsilon^{\mathrm{N}} < \varepsilon$ holds for the current value of perturbation level ε, and furthermore, the locally rightmost point computed for the ε^{N} real spectral value set must also be either be on the imaginary axis or reside in the right half-plane, for which in the latter case again hybrid expansion-contraction can be restarted. Finally, as these static boundary points seem to be a degenerate case and we do not expect to encounter them in practice, it makes sense to initially try only the smallest of contraction step sizes above the termination tolerance of the procedure and to only increase the contraction step size if RSVSAR converges back to the imaginary axis, indicating that λ was a static boundary point. If λ is not a static boundary point, the initial tiny steps allow the method to quickly detect this and thus permit a near immediate termination without incurring a lengthy convergence process to confirm that ε was actually the minimal value $\varepsilon_{\tilde{\star}}$. Thus, the potentially slow method is at least only slow for actual static boundary points while it is optimized to be as inexpensive as possible for the typical non-degenerate case we expect to only ever observe in practice.

6.5 A randomized SVSAR method for the real-valued spectral norm case

We note that though we don't yet have an explicit adaption of Procedure 2 for the real spectral norm case, we can instead employ the same random sampling strategy used in Section 4.4. Though this is not necessarily a practical method by itself, it nonetheless could be interesting to develop these ideas further.

7 Simultaneous optimization of multiple stability measures

We now turn our focus to the problem of optimizing stability measures in a multi-objective setting, that is, optimization subject to constraints. This is an important application in control where it is often desirable to design and build a single controller that can *simultaneously* govern and enhance the robustness of multiple dynamical systems [GHMO09]. However, as in the case of the H_∞ norm, stability measures are often both nonconvex and nonsmooth, and sometimes not even locally Lipschitz, making optimization potentially difficult, and even more so in the constrained setting. On the other hand, the nonsmooth points of these measures are typically limited to a set of measure zero, implying that gradient-based methods can still be effective. Even so, we expect, and in fact do observe in practice, that such methods may encounter the nonsmooth manifolds during the course of optimization as well as at minimizers, which can potentially cause slowdown, stagnation, and even breakdown of such methods. Without assuming any special structure, we consider employing BFGS (Broyden-Fletcher-Goldfarb-Shanno) Hessian approximations within a sequential quadratic programming method for general constrained optimization where both the objective and constraints may be nonsmooth and nonconvex. We call the resulting algorithm BFGS SQP. This chapter is joint work with Frank E. Curtis and Michael L. Overton.

7.1 Nonsmooth, nonconvex constrained optimization

Consider the functions $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ in the general inequality constrained optimization problem:

$$\min_x f(x) \quad \text{s.t.} \quad c_i(x) \leq 0, \quad i = 1, \ldots, p. \tag{7.1}$$

Here, we presume that $f(\cdot)$ and $c_i(\cdot)$ are both nonconvex and nonsmooth yet are still differentiable *almost everywhere*. We make note that our proposed method is also applicable to problems that have equality constraints, via relaxation with slack variables, but for the purpose of clearer and more concise exposition, we only consider the inequality constrained problem.

For the unconstrained nonsmooth problem, that is where $m = 0$ in (7.1), the authors of [BLO05] proposed a gradient sampling (GS) algorithm with provable global convergence results that hold with probability one, provided that $f(\cdot)$ is locally Lipschitz and its level sets are bounded. In [Kiw07], it was further shown that the convergence results of GS could be strengthened by removing the requirement that the level sets be compact. In practice, GS has been shown to be an extremely robust method on many challenging nonsmooth problems and surprisingly, this holds even in cases where the objective function is not locally Lipschitz, even though the convergence results do not extend to such problems. However, the GS technique requires that $\mathcal{O}(n)$ gradients be evaluated per iteration, which can be a quite costly endeavor for expensive-to-compute functions, such as the H_∞ norm, and as a result in many applications, GS is not a viable algorithm.

On the other hand, it has recently been strongly advocated in [LO13] that for nonsmooth unconstrained optimization, a simple BFGS method using inexact line searches is actually much more efficient in practice than gradient sampling, although the BFGS approximation to the Hessian typically becomes very ill-conditioned and no general convergence results are known. In fact, the authors argue that the ill-conditioning of the Hessian approximation is actually beneficial, analogous to the ill-conditioning that occurs in interior point methods, and tantalizingly show an example indicating that, when the objective function is partly smooth in the sense of [Lew03], BFGS seems to be able to automatically identify the smooth U and nonsmooth V-spaces associated with the objective function near the minimizer.

For the more challenging case of constrained nonsmooth, nonconvex optimization, where $m \geq 1$ in (7.1), a sequential quadratic programming approach employing gradient sampling (SQP-GS) was presented in [CO12] with convergence to stationary points provably holding with probability one. This algorithm uses a BFGS approximation to define a Hessian matrix that appears in the quadratic programs, but intriguingly and in contrast to the argument of [LO13] regarding the benefit of ill-conditioning for the unconstrained problem, the constrained case convergence result required enforcing upper and lower bounds on the eigenvalues of the BFGS approximations used in the algorithm. Still, the reliance on gradient sampling, both theoretically for the convergence results but more importantly, in the actual algorithm, again makes SQP-GS ill-suited for optimizing expensive-to-compute objective and constraint functions.

Thus, our aim is to eschew a costly gradient sampling approach entirely and to instead find an *efficient* and *effective* extension of the simple BFGS approach for unconstrained problems to the domain of constrained nonsmooth, nonconvex optimization.

7.2 A nonsmooth penalty parameter approach

Consider the fixed nonsmooth penalty function and its gradient

$$\phi(x;\mu) = \mu f(x) + v(x) \tag{7.2}$$

$$\nabla\phi(x;\mu) = \mu\nabla f(x) + \sum_{i\in\mathcal{P}_x} \nabla c_i(x), \tag{7.3}$$

where $\mu \in \mathbb{R}^{++}$ is the penalty parameter and $v(\cdot)$ is the total violation cost over the constraints, that is:

$$v(x) = \sum_{i\in\mathcal{P}_x} c_i(x) \quad \text{where} \quad \mathcal{P}_x = \{i \in \{1,\ldots,m\} : c_i(x) > 0\}. \tag{7.4}$$

We note that the penalty function itself is inherently nonsmooth, even if the objective and constraint functions are smooth, and thus if we are to use such a penalty scheme, we must consider corresponding optimization algorithms for nonsmooth functions. If we happen to know *a priori* of an acceptable value for the penalty parameter μ such that minimizers of (7.2) correspond to feasible minimizers of (7.1), then a straightforward BFGS method can be a practical approach, where at each iterate x_k, the standard search direction d_k is calculated by solving the quadratic program:

$$\min_d q(d; x_k, \mu) \tag{7.5}$$

where

$$
\begin{aligned}
q(d; x_k, \mu) &\coloneqq \phi(x_k; \mu) + \nabla \phi(x_k; \mu)^{\mathsf{T}} d + \tfrac{1}{2} d^{\mathsf{T}} H_k d \\
&= \mu f(x_k) + \sum_{i \in \mathcal{P}_x} c_i(x) + \left[\mu \nabla f(x_k) + \sum_{i \in \mathcal{P}_x} \nabla c_i(x) \right]^{\mathsf{T}} d + \tfrac{1}{2} d^{\mathsf{T}} H_k d \quad (7.6)
\end{aligned}
$$

and H_k is the BFGS approximation to the Hessian of (7.2).[1] Unfortunately, we often do not know what value the penalty parameter should take and as such, it is typical that BFGS will converge to a stationary point (assuming it converges at all) that is actually infeasible for the original problem of (7.1) if the penalty parameter weighting the objective is set too high.

One might consider a simple strategy of using a sequential fixed penalty parameter (SFPP) restarting scheme with BFGS, that is, to lower μ and restart BFGS continually in a loop until a feasible solution has been found. An immediate pertinent issue with such an approach is the question of how accurately BFGS should attempt to minimize (7.2) for a given value of μ before deciding whether it is necessary to lower the penalty parameter. There is a delicate balance here between the cost of computation to concretely confirm that the penalty parameter needs to be lowered to obtain a feasible solution versus too aggressively lowering the penalty parameter so that a feasible solution is found but potentially increasing the difficulty of optimizing the penalty function itself and thus perhaps lowering the rate of progress of the method. Furthermore, if $f(\cdot)$ is unbounded below, then (7.2) may also be unbounded below, even if $f(\cdot)$ is bounded below on the feasible set. One goal of our proposed algorithm is to address the case when $f(\cdot)$ is

[1]Note that we use H_k here to denote an approximation to the Hessian at the k-th iterate, as opposed to the notation used in [LO13] and [NW06, page 140], where H_k is used to denote a BFGS approximation to the *inverse* of the Hessian.

unbounded below off the feasible set.

7.3 A steering strategy for nonsmooth constrained problems

As a potential solution to the issues of when to adjust the penalty parameter and handling the case of $f(\cdot)$ being potentially unbounded below, we consider adapting the steering strategy of [BNW08] and [BLCN12]. Although originally intended for the case where the objective and constraints are both smooth, we propose using such a steering strategy to permit a modified BFGS search direction calculation in our present setting where both the objective and constraint functions may be nonsmooth. Specifically, we replace the standard BFGS search direction given in (7.5) by a penalty-SQP method [Fle87] where the search directions are computed via solving

$$\begin{aligned} \min_{d,s}\ & \mu(f(x_k) + \nabla f(x_k)^\mathsf{T} d) + e^\mathsf{T} s + \tfrac{1}{2} d^\mathsf{T} H_k d \\ \text{s.t. } & c(x_k) + \nabla c(x_k)^\mathsf{T} d \leq s, \quad s \geq 0, \end{aligned} \tag{7.7}$$

which has the corresponding dual

$$\begin{aligned} \max_{\lambda}\ & \mu f(x_k) + c(x_k)^\mathsf{T}\lambda - \tfrac{1}{2}(\mu\nabla f(x_k) + \nabla c(x_k)\lambda)^\mathsf{T} H_k^{-1}(\mu\nabla f(x_k) + \nabla c(x_k)\lambda) \\ \text{s.t. } & 0 \leq \lambda \leq e, \end{aligned}$$

and where e is a vector of m ones, $s \in \mathbb{R}^m$ is a vector of slack variables and $\lambda \in \mathbb{R}^m$. The primal solution component d_k can be recovered from the dual solution λ_k via the relationship $d_k = -H_k^{-1}(\mu\nabla f(x_k) + \nabla c(x_k)\lambda_k)$. Note that we recover the usual BFGS search direction when there are no constraints so μ can be taken as one. In the presence of constraints, the resulting d_k computed by solving (7.7) indeed provides a descent direction for (7.2) at x_k with the current penalty parameter μ. The search direction computed from (7.7) can be viewed as balancing the two, sometimes opposing, goals of minimizing the objective and pushing towards a feasible solution, the latter of which is measured by the size of the linear model of constraint violation

$$l(d; x_k) \coloneqq \| \max\{c(x_k) + \nabla c(x_k)^{\mathsf{T}} d, 0\} \|_1 \tag{7.8}$$

at x_k given direction d (in other words, how small $e^{\mathsf{T}} s$ is at the solution to (7.7)). Furthermore, that balance is shifted as the penalty parameter is changed: at one extreme, when μ is set high, the search direction d_k will be heavily weighted towards minimizing the objective, regardless of whether it results in pushing towards feasibility or not, while at the other extreme when $\mu = 0$, d_k will almost entirely be weighted towards satisfying the linear model of constraint violation, and thus hopefully produce a direction towards the feasible set. (Note that in the case when $\mu = 0$, we say "almost entirely" as H_k will contain curvature information for both the objective and violated constraints.) Thus, a good so-called "steering strategy" will dynamically adjust μ to ensure that search directions d_k computed by solving (7.7) maintain a good balance of minimizing the objective while simultaneously pushing towards feasibility over the course of the optimization.

Let the reduction in the linear model of constraint violation given in (7.8) at the current iterate x_k and for any search direction d be defined as

$$\begin{aligned} l_\delta(d; x_k) &\coloneqq l(0; x_k) - l(d; x_k) \\ &= v(x_k) - \| \max\{c(x_k) + \nabla c(x_k)^{\mathsf{T}} d, 0\} \|_1. \end{aligned} \tag{7.9}$$

For any search direction d at x_k, (7.9) predicts how much progress towards feasibility d may make. The basic tenet of the steering strategy defined in Procedure 7 is to overall promote progress towards feasibility during every iteration, which it does by first evaluating the predicted violation reduction for the search direction d_k produced for the current value of the penalty parameter. If the resulting predicted violation reduction for d_k seems inadequate, the steering strategy alternatively assesses the predicted violation reduction for the reference search direction $\tilde{d}_k$, which is the direction resulting from solving (7.7) with μ set to zero. By essentially biasing the search direction calculation to the extreme of only promoting progress towards feasibility regardless of the effect on the objective, the predicted violation reduction given for $\tilde{d}_k$ gives an indication of the largest violation reduction the algorithm may hope to achieve when taking a step from x_k. If the predicted violation reduction for d_k is still inadequate compared to the predicted reduction given by the reference direction, then the steering strategy iteratively lowers the current value of the penalty parameter until (7.7) produces a search direction satisfactorily balanced in terms of progress towards the feasible set and minimizing the objective.

The benefits of such a strategy are that the penalty parameter can be dynamically updated at every iteration (though not necessarily) and the amount of reduction in the penalty parameter is dynamically determined by how difficult it appears to be to promote progress towards the feasible set any given iterate. In contrast, with the simple fixed penalty parameter restarting scheme SFPP, one must either wait for BFGS to converge a stationary point (which can be slow for nonsmooth problems) to assess whether it is necessary to lower the penalty parameter and restart, or terminate BFGS early and *blindly* adjust the penalty parameter. Moreover, the steering strategy of Procedure 7 decreases the likelihood of divergence ensuing in the case that $f(\cdot)$ is unbounded below.

As we expect the penalty function to be nonsmooth at minimizers, we cannot expect the norm of its gradient to decrease as iterates approach these nonsmooth minimizers.

```
**Procedure 7** $[d_k, \mu_{\text{new}}]$ = **sqp_steering_strategy**$(x_k, H_k, \mu_{\text{current}})$
**Input:**
  Current iterate $x_k$ and BFGS Hessian approximation $H_k$
  Current value of the penalty parameter $\mu_{\text{current}}$
  Constants: fixed values $c_v \in (0,1)$ and $c_\mu \in (0,1)$
**Output:**
  Search direction $d_k$
  $\mu_{\text{new}} \leq \mu_{\text{current}}$

  Solve QP given in (7.7) using $\mu := \mu_{\text{current}}$ to obtain search direction $d_k$
  if $l_\delta(d_k; x_k) < c_v v(x_k)$ then
    Solve (7.7) using $\mu = 0$ to obtain "reference direction" $\tilde{d}_k$
    while $l_\delta(d_k; x_k) < c_v l_\delta(\tilde{d}_k; x_k)$ do
      $\mu_{\text{current}} := c_\mu \mu_{\text{current}}$
      Solve QP given in (7.7) using $\mu := \mu_{\text{current}}$ to obtain search direction $d_k$
    end while
  end if
  $\mu_{\text{new}} := \mu_{\text{current}}$
```

NOTE: *The steering strategy first checks whether the computed search direction d_k for the initial value of μ is predicted to at least reduce a reasonable fraction of the violation, which is specified by constant c_v. If not, the method alternatively computes the "reference direction" with $\mu = 0$ in order to estimate how much the violation might actually be reduced by taking any step from x_k, since a reduction of $c_v v(x_k)$ or more may not be obtainable. If the predicted reduction in violation given by d_k is at least a reasonable fraction of the predicted reduction given by $\tilde{d}_k$, then d_k is returned and the penalty parameter is not updated. However, if by comparison, the reference direction still shows a much greater predicted reduction in violation than what is offered by d_k, the penalty parameter is then iteratively reduced and new search directions d_k are computed until the specified sufficient fraction of predicted violation reduction for d_k compared to the reference direction is achieved.*

Consequently, we must consider an alternative stopping strategy compared to the usual criteria for optimizing smooth functions. To that end, following the approach taken in [LO13] for the unconstrained problem, consider l consecutive iterates of the algorithm that are considered “close” in some sense of distance and define:

$$G \coloneqq \left[\nabla f(x_{k+1-l}) \dots \nabla f(x_k)\right]$$
$$J_i \coloneqq \left[\nabla c_i(x_{k+1-l}) \dots \nabla c_i(x_k)\right].$$

Our stationarity measure is derived by first forming a QP subproblem designed to compute a step toward minimizing the penalty function along the same lines as (7.7). However, we augment the QP with previously computed gradient information (from G and J_i) in order to capture changes in the problem functions in a neighborhood around the current iterate. The motivation is similar in gradient sampling where one aims to approximate subdifferential information by the random sampling of gradients in a neighborhood of a given point. Here however, we are reusing the gradients of the objective and constraints of previously computed points $\{x_{k+1-l}, \dots, x_k\}$ to form G and J_i, provided that this set of previous iterates is sufficiently close to the x_k. If the solution of the resulting QP is sufficiently small in norm, then we have reason to believe that we are in a small neighborhood of a stationary point for the constrained optimization problem. In the dual formulation, this is written as

$$\max_{\sigma,\lambda} \quad \sum_{i=1}^{m} c_i(x_k) e^{\mathsf{T}} \lambda_i - \tfrac{1}{2} [\sigma, \lambda] [G, J_1, \dots, J_m]^{\mathsf{T}} H_k^{-1} [G, J_1, \dots, J_m] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix} \tag{7.10}$$
$$\text{s.t.} \quad 0 \le \lambda_i \le e$$
$$e^{\mathsf{T}} \sigma = \mu$$
$$\sigma \ge 0$$

where the smallest vector is given explicitly by

$$d_\diamond = H_k^{-1} \left[G, J_1, \ldots, J_m \right] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}. \tag{7.11}$$

Remark 7.1. *For brevity, we omit the primal form of* (7.10) *and instead refer the reader to the thorough discussion in* [CO12] *for more details. As a consequence, note that we have actually defined* BFGS SQP *in terms of both the primal and the dual* (*for the steering QP given by* (7.7) *and the stationarity QP given by* (7.10) *respectively*) *as we feel doing so provides a much more concise explanation of how our algorithm works. In an actual implementation, we must choose one convention or the other and for our code, we used the dual as noted in Section 7.6.1.*

For some fixed tolerance $\tau \in \mathbb{R}^{++}$, we terminate the BFGS SQP iteration once $\|d_\diamond\|_2 \leq \tau$ is satisfied, provided that the constraints are also satisfied to a desired accuracy. Furthermore, we discard the current set of l "close" iterates and begin collecting a new set of "close" iterates any time BFGS takes a sufficiently large step.

We present pseudocode for the BFGS SQP method in Procedure 8.

Procedure 8 $[x_\star, f_\star, v_\star]$ = **bfgs_sqp**$(f(\cdot), c(\cdot), x_0, \mu_0)$

```
Input:
  Objective $f : \mathbb{R}^n \to \mathbb{R}$ and inequality constraints $c : \mathbb{R}^n \to \mathbb{R}^m$ given as $f(\cdot)$ and $c(\cdot)$
  Initial starting point $x_0 \in \mathbb{R}^n$ and finite penalty parameter $\mu_0 \in \mathbb{R}^{++}$
  Constants: positive stopping tolerances $\tau_\diamond$ (stationarity) and $\tau_v$ (violation)
Output:
  Best solution $x_\star$ encountered with corresponding objective value $f_\star$ and violation $v_\star$

  Set $H_0 := I$ and $\mu := \mu_0$
  Set $\phi(\cdot)$ as the penalty function given in (7.2) using $f(\cdot)$ and $c(\cdot)$
  Set $\nabla\phi(\cdot)$ and $v(\cdot)$ as the associated gradient (7.3) and violation function (7.4)
  for $k = 0, 1, 2, \ldots$ do
    Evaluate $\phi_k := \phi(x_k; \mu)$, $\nabla\phi_k := \nabla\phi(x_k; \mu)$, and $v_k := v(x_k)$
    $[d_k, \hat{\mu}] :=$ sqp_steering_strategy$(x_k, H_k, \mu)$
    if $\hat{\mu} \leq 0$ then
      // $\hat{\mu} \leq 0$ is used to indicate steering failed - terminate algorithm
      break
    else if $\hat{\mu} < \mu$ then
      // Penalty parameter has been lowered by steering - update current iterate
      Set $\mu := \hat{\mu}$
      Reevaluate $\phi_k := \phi(x_k; \mu)$, $\nabla\phi_k := \nabla\phi(x_k; \mu)$, and $v_k := v(x_k)$
    end if
    $[x_k, \phi_k, \nabla\phi_k, v_k, s] :=$ inexact_linesearch$(x_k, \phi_k, \nabla\phi_k, d_k, \phi(\cdot), \nabla\phi(\cdot))$
    if $s \leq 0$ then
      // Step size $s \leq 0$ is used to indicate line search failed - terminate algorithm
      break
    end if
    Compute $d_\diamond$ via (7.10) and (7.11) // Note the use of dual form here using $H_k^{-1}$
    if $\|d_\diamond\|_2 < \tau_\diamond$ and $v_k < \tau_v$ then
      // Stationarity and feasibility sufficiently attained - terminate successfully
      break
    end if
    Set $H_{k+1}$ using BFGS update formula
  end for
```

NOTE: *For brevity, we omit the specifics for tracking the best optimizer encountered so far and the set of previous gradients which are considered "close" to the current iterate. For details on the inexact line search method and the* BFGS *update formula for the Hessian, not its inverse, we refer to* [LO13, Han] *and* [NW06, pages 140-143] *respectively.*

7.4 Nonsmooth, nonconvex constrained optimization examples

7.4.1 Static output feedback controller design

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

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

and the associated *static output feedback plant* [Blo99, BHLO06, Ove14]

$$A + BXC,$$

where matrices $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$, and $C \in \mathbb{R}^{p,n}$ are fixed, $X \in \mathbb{R}^{m,p}$ is an embedded variable controller matrix and $w_k \in \mathbb{R}^m$ is the control input and $z_k \in \mathbb{R}^p$ is the output. A well known and an important problem in applications is the goal of designing the controller matrix X such that with respect to some chosen measure of stability for dynamical systems, the stability of the static output feedback plant is enhanced as much as possible. Typically controller design for static output feedback plants amounts to solving a nonsmooth, nonconvex optimization problem, one which may not even be locally Lipschitz depending on which stability measure is specified in the problem. Furthermore, as the controller is optimized, we typically observe that an *increasing degree* of nonsmoothness in the problem is encountered by the algorithm as it progresses.

7.4.2 Multi-objective spectral radius optimization

Definition 7.2. *The* spectral radius *of a square matrix A is defined as*

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

where

$$\sigma(A) = \{\lambda \in \mathbb{C} : \det(A - \lambda I) = 0\}.$$

We say that A is (Schur) stable if $\rho(A) < 1$.

Remark 7.3. *The discrete-time dynamical system $x_{k+1} = Ax_k$ with $x_0 \in \mathbb{R}^n$ is (asymptotically) stable, that is $x_k \to 0$ as $k \to \infty$ holds for all x_0, if and only if $\rho(A) < 1$.*

Remark 7.4. *Though the spectral radius is nonconvex, nonsmooth, and furthermore not locally Lipschitz* [BO01], *it is in fact still differentiable almost everywhere and as such, makes for a good and challenging candidate function for comparing gradient-based nonsmooth, nonconvex optimization methods.*

Remark 7.5. *If the task of optimizing a controller for a single static output feedback plant is done with respect to the spectral radius, then the stability for the system will be enhanced by decreasing the spectral radius to as close to zero as possible. However, as the spectral radius is decreased, we generally observe that more and more eigenvalues of the static output feedback plant have moduli that attain the spectral radius. We can consider the number of such "active" eigenvalues, minus one, (the number of "activities") as a measure of nonsmoothness at the solution. When complex active eigenvalues coincide with each other, the degree of nonsmoothness is higher still, since both the real and imaginary parts coincide, not only the modulus: this also tends to happen as the* *spectral radius is decreased. See* [Ove14] *for further discussion of this phenomenon.*

We now consider the following multi-objective nonconvex and nonsmooth constrained optimization problem

$$\begin{aligned} \min_X \quad & \max\{\rho(A_i + B_i X C_i) : i = 1, \ldots, p\} \\ \text{s.t.} \quad & \rho(A_i + B_i X C_i) < 1, \; i = p+1, \ldots, k \end{aligned} \tag{7.12}$$

where each static output feedback plant $A_i + B_i X C_i$ is defined by the fixed matrices $A_i \in \mathbb{R}^{n,n}$, $B_i \in \mathbb{R}^{n,m}$, and $C_i \in \mathbb{R}^{p,n}$ and the matrix $X \in \mathbb{R}^{m,p}$ is an embedded variable controller for all k plants. The goal is to design a matrix X such that stability of the static output feedback plants in the objective are all enhanced while simultaneously ensuring that the plants appearing in the constraint functions remain stable, as measured by the spectral radius.

In Figure 7.1, we compare SFPP (where the number of iterations for each fixed value of the penalty parameter is limited to 20), BFGS SQP, and SQP-GS for designing a controller for (7.12) on a randomly generated example of dimension $n = 13$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $mp = 23 \times 2 = 46$ variables and was initially set to 0, a feasible starting point. (See Section 7.6.1 for more details on the experimental setup.) We see that SFPP immediately stepped outside of the feasible set and then, even with a relatively frequent penalty parameter update, struggled for well over 100 iterations before any real progress towards feasibility was made, due to the penalty parameter remaining too high for those iterations. In contrast, BFGS SQP, although also initially stepping outside of the feasible set, immediately made quick progress back towards it while SQP-GS in fact maintained feasibility at almost every iterate, and both algorithms simultaneously minimized the objective significantly more than SFPP. By viewing the time at which BFGS SQP finished its 500 iterations relative to SQP-GS, it is manifestly apparent that BFGS SQP is well over an order of magnitude faster in terms of CPU time. For this particular problem, we see that BFGS SQP happens to reduce the objective more than SQP-GS, which appears to reach its max iteration count before it has actually converged.

In Figure 7.2, we show the final spectral configurations of the three controllers ultimately produced by SFPP, BFGS SQP and SQP-GS for this particular example. These are all plots in the complex plane, symmetric about the real axis because the matrix data and controller X are all real. In all configurations, in accordance with Remark 7.5, we observe that the moduli of several eigenvalues attain the relevant spectral radius value. For example, in the third plant in the objective for BFGS SQP's controller we see the moduli of all thirteen eigenvalues close to attaining the spectral radius, with two complex conjugate pairs being close to coincident. A count of the number of eigenvalues in this case shows that two real eigenvalues must actually be coincident, though this can't be observed from the plot, but a close-up view indicates that, unlike the others, the positive real eigenvalue does not quite have its modulus attaining the spectral radius. Furthermore, we see that the spectral radii of the different plants in the objective are nearly the same, further increasing the overall number of activities encountered at these solutions and demonstrating the inherent high degree of nonsmoothness in the optimization problem. The plants in the constraint also show activities, with the moduli of several eigenvalues attaining the value one imposed by the constraint, thus demonstrating that the algorithms converged to controllers where both the objective and constraint are nonsmooth. (See Appendix B for additional spectral radius examples similar to the one shown here in Figures 7.1 and 7.2.)

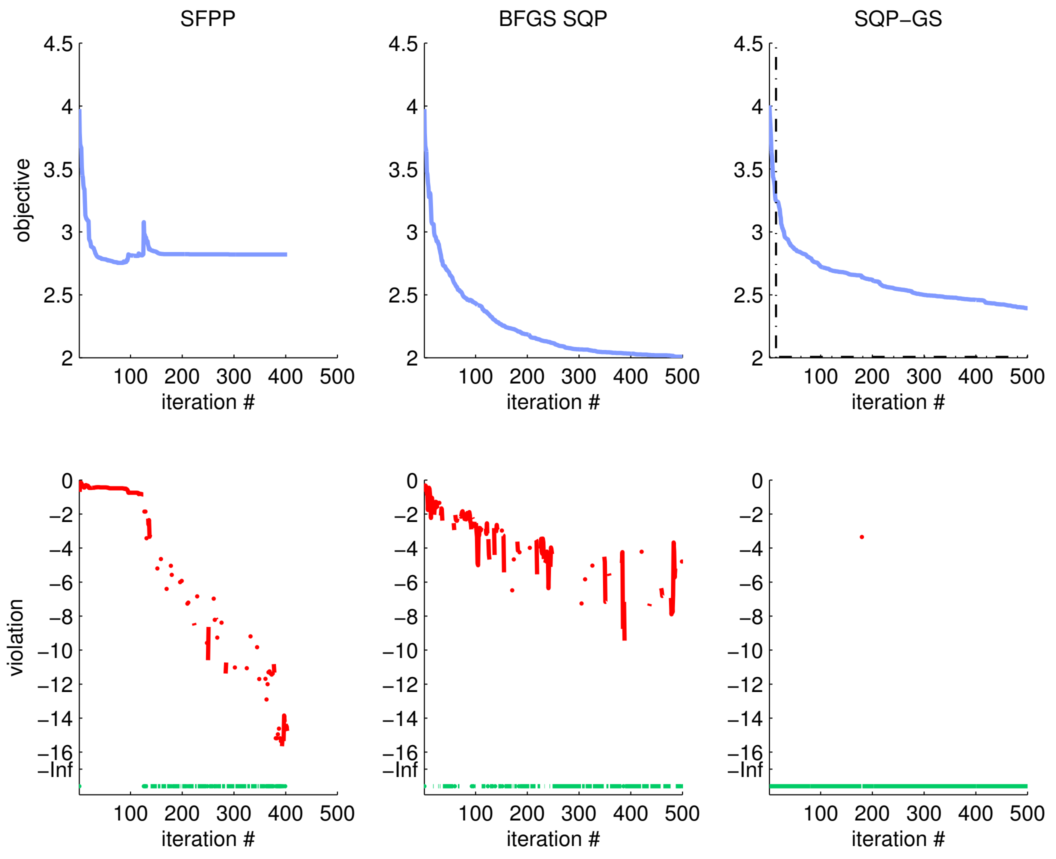

FIGURE 7.1: *The plots in the top row track the value of the spectral radius based objective function in terms of iteration number for* SFPP (*left*), BFGS SQP (*middle*), *and* SQP-GS (*right*) *on a randomly generated example of dimension* $n = 13$ *comprised of three plants in the objective and one in the constraint and where the controller matrix has* $mp = 23 \times 2 = 46$ *variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

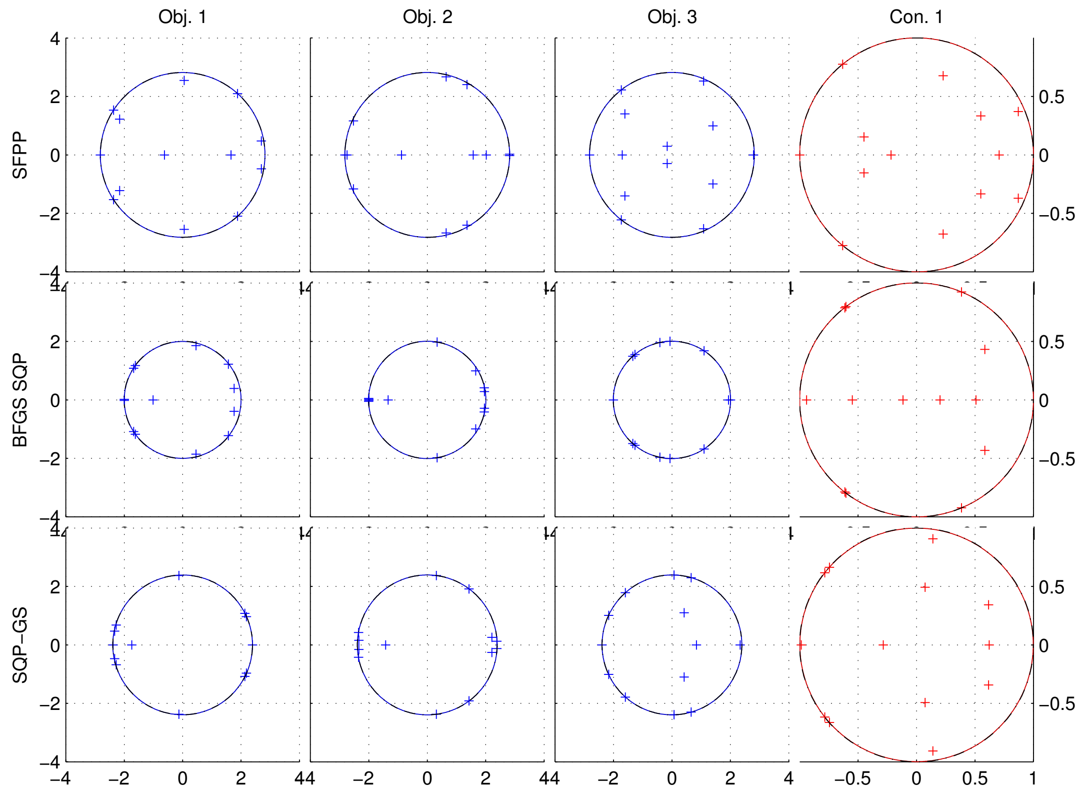

FIGURE 7.2: *The three rows show the final spectral configurations of the three controllers found by* SFPP (*top*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension* $n = 13$ *comprised of three plants in the objective and one in the constraint and where the controller matrix has* $mp = 23 \times 2 = 46$ *variables. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle* (*the stability boundary*)*. The dashed black circle on the objective plots* (*barely visible*) *corresponds to the max spectral radius of the three plants in the objective for that particular algorithm's controller.*

7.4.3 Multi-objective pseudospectral radius optimization

Definition 7.6. *The* pseudospectral radius *of a matrix $A \in \mathbb{C}^{n,n}$ is defined as*

$$\rho_\varepsilon(A) := \max\{|\lambda| : \lambda \in \sigma(A + \Delta), \Delta \in \mathbb{C}^{n,n}, \|\Delta\|_2 \leq \varepsilon\}.$$

We say that A is considered (Schur) stable with respect to perturbation level $\varepsilon \geq 0$ if $\rho_\varepsilon(A) < 1$, noting that $\rho_0(A) = \rho(A)$.

Remark 7.7. *Note that the pseudospectral radius can be equivalently defined in terms of the norm of the resolvent as* [TE05]

$$\rho_\varepsilon(A) := \max\left\{|\lambda| : \lambda \in \mathbb{C}, \|(A - \lambda I)^{-1}\|_2 \geq \varepsilon^{-1}\right\}$$

where we use the convention that $\|(A - \lambda I)^{-1}\|_2 = \infty$ when $A - \lambda I$ is singular.

Remark 7.8. *Like the spectral radius, the pseudospectral radius is also nonconvex, nonsmooth, and differentiable almost everywhere. However and in contrast to the spectral radius, the pseudospectral radius is also locally Lipschitz* [GO12] *and is thus potentially an easier function to optimize. For example, the known convergence rates for gradient sampling hold for the pseudospectral radius but not the spectral radius. On the other hand, the pseudospectral radius (along with its gradient) is significantly more expensive to compute than the spectral radius* [MO05].

Remark 7.9. *In contrast to the spectral radius, the pseudospectral radius can be considered a robust stability measure, in the sense that it can account for where asymptotic stability of the linear dynamical system is guaranteed under the influence of noise up to a specified amount.*

For a given perturbation level of $\varepsilon = 10^{-1}$, we now consider the following multi-objective nonconvex and nonsmooth constrained optimization problem

$$\begin{aligned} \min_{X} \quad & \max\{\rho_\varepsilon(A_i + B_i X C_i) : i = 1, \ldots, p\} \\ \text{s.t.} \quad & \rho_\varepsilon(A_i + B_i X C_i) < 1, \; i = p+1, \ldots, k \end{aligned} \tag{7.13}$$

which we may view as a locally Lipschitz regularization of (7.12). The goal is still to still design a matrix X such that stability of the static output feedback plants in the objective are all enhanced while simultaneously ensuring that the plants appearing in the constraint functions remain stable, except here stability means that any plant must remain stable under any perturbation up to norm $\varepsilon = 10^{-1}$.

In Figure 7.3, we again compare SFPP, BFGS SQP, and SQP-GS (using the same experimental setup as used in Section 7.4.2 and described in Section 7.6.1) for designing a controller for (7.12) on a randomly generated example of dimension $n = 5$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $mp = 13 \times 2 = 26$ variables and was initially set to 0, again a feasible point. For this particular pseudospectral multi-objective optimization problem, we see that SFPP performed even worse than it did on the multi-objective spectral radius example problem after initially stepping outside of the feasible set, where here over 160 iterations were incurred before the penalty parameter was sufficiently lowered to promote any significant progress towards satisfying feasibility. BFGS SQP again initially stepped outside the feasible region but quickly progressed back towards it while SQP-GS maintained feasibility a majority of the time, and both algorithms simultaneously minimized the objective more than SFPP once again. Interestingly, as on the multi-objective spectral radius problem, BFGS SQP found a much better controller than SQP-GS, even though

SQP-GS's convergence results hold for this particular problem (as it's locally Lipschitz) while BFGS SQP provides no guarantees. As we expect, the higher number of function evaluations per iteration required by SQP-GS cause it to be dramatically slower the BFGS SQP with respect to CPU time.

In Figure 7.4, we show the final pseudospectral configurations (generated via MATLAB's `contour` by evaluating the norm of the resolvent on a grid) of the three controllers controlled ultimately produced by SFPP, BFGS SQP and SQP-GS for this particular example. For almost all the plants, we see that the resulting pseudospectral configurations show that there are multiple nonsmoothness activities with respect to the pseudosepctral radius (without counting conjugacy) and we see further activities due to the fact that pseudospectral radii for the three plants in the objectives are all tied as well. The plants in the constraint also show activities, demonstrating that both the objective and the constraint are indeed nonsmooth at the solutions found by each algorithm. (See Appendix C for additional pseudospectral radius examples similar to the one shown here in Figures 7.3 and 7.4.)

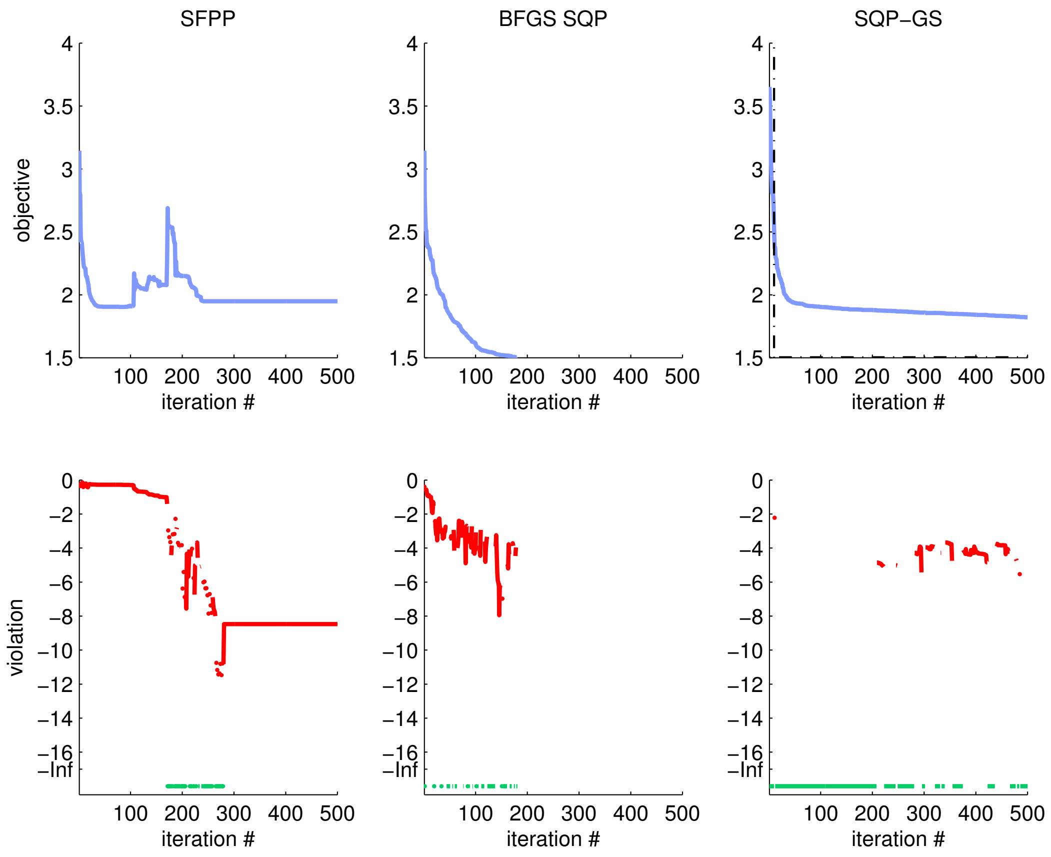

FIGURE 7.3: *The plots in the top row track the value of the pseudospectral radius based objective function in terms of iteration number for* SFPP (*left*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*right*) *on a randomly generated example of dimension* $n = 5$ *comprised of three plants in the objective and one in the constraint and where the controller matrix has* $mp = 13 \times 2 = 26$ *variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

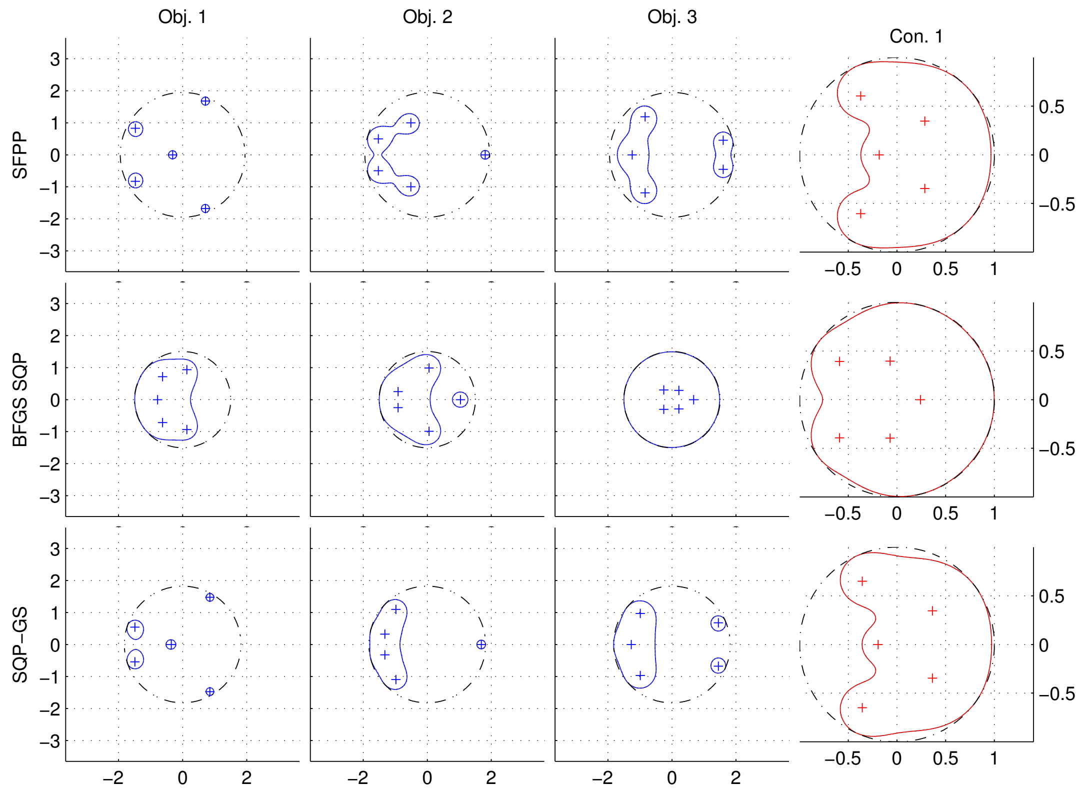

FIGURE 7.4: *The three rows show the final pseudospectral configurations of the three controllers found by* SFPP (*top*), BFGS SQP (*middle*), *and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension* $n = 5$ *comprised of three plants in the objective and one in the constraint and where the controller matrix has* $mp = 13 \times 2 = 26$ *variables. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm's controller.*

7.5 Comparing nonsmooth, nonconvex optimization algorithms

For the case of evaluating convex optimization codes, a popular visualization tool is that of performance profiles [DM02], which simultaneously give an indication of how well any given algorithm is doing on a test set, in terms of the percentage of problems it has successfully globally minimized, as well as how fast the algorithm is making its way through the test relative to the fastest algorithm *per problem*. The ingenuity of performances profiles is that they allow both the success rate and speed of computation to be concisely combined in an easily read plot, while additionally providing information on how the computation time of each algorithm changes with respect to the individual difficulty of each problem. As data sets can be quite varied in the level of difficulty of the problems, performance profiles allow the reader to gauge the range and frequency of the different difficulty levels in the test set in terms of each algorithm's specific ability for those problems, that is, the easy problems for one algorithm are not necessarily the same set of problems for a second algorithm. If an algorithm performs disastrously badly on one particular problem but exceptionally well on all the others, an assessment based on the overall running time to complete the test set could cause the algorithm to look inferior, even if it is better than its competition 99% of the time, whereas performance profiles would make this important distinction clear.

Making any comparison of optimization codes for nonconvex problems however can be quite a challenge, yet alone attempting to make a *fair* comparison. In the nonconvex setting, we can neither expect that the codes will find global minimizers for the test problems nor can we expect the codes to find the same local minimizers for a given problem or even local minimizers at all. The general state of affairs is that for any nonconvex problem, the codes being evaluated may each return their own distinct stationary point and in nonsmooth settings, the algorithms may stagnate before convergence is reached. Thus there is an inherent ambiguity, at least at a fine level, as to whether the time to run a code on a nonconvex, nonsmooth problem is attributable to the algorithm's implementation/design or to the particular ease or difficulty inherent in the problem itself for finding any given stationary point. As a consequence, even in the event that one algorithm consistently finds better (lower) minimizers and/or stationary points across a test set of nonconvex problems, there is often little to no grounds for attributing such a success to the properties and design of the algorithm. Unfortunately, performance profiles do not address these issues. Furthermore, our setting is not just nonconvex optimization, but constrained nonconvex optimization where we must also assess feasibility of the different candidate minimizers that are found, in addition to both the amount of minimization of the objective achieved as well as the overall cost incurred.

7.5.1 Relative minimization profiles

Let us first consider evaluating algorithms with respect to the quality (amount) of objective minimization achieved and feasibility of solutions over a test set of multiple problems, without considering running time. We propose producing a plot, which we call a *relative minimization profile* (RMP), where the amount of per problem objective minimization achieved for individual algorithms, relative to the lowest objective value found on the feasible set across all the algorithms (in terms of the number of digits that agree) and measured on the x-axis, is related with the percentage of feasible solutions[2] that the algorithms found over the entire data set, measured on the y-axis. At the far

[2]In fact, relative minimization profiles can be useful for comparing methods in unconstrained and/or convex settings. However, as our main focus in here is to compare algorithms for nonsmooth, nonconvex, constrained optimization, we present and motivate RMPs for this specific context. We refer the reader to Remarks 7.12 and 7.16 for brief discussions on the applicability of RMPs to other settings.

right side of an RMP, the algorithms are purely assessed on the percentage of problems in the data set for which they found feasible solutions, regardless of the objective values achieved at these solutions or even if the objective values were in fact increased from the initial starting points. However, on the far left side, an algorithm's feasible solutions are only counted in its "percentage of feasible solutions found" score if the amount of per problem objective minimization attained matches the lowest objective value found on the feasible set across all the algorithms to 16 digits. In the middle, an RMP depicts how the percentage of feasible solutions increases as the required number of digits that agree is relaxed.

Remark 7.10. *Requiring a 16 digit agreement of an algorithm's per problem objective minimization to the objective value attained by the best feasible point found across all the algorithms might seem excessive. However, the intention of a relative minimization profile is to demonstrate how the percentage of feasible solutions varies in the range between requiring that an algorithm always demonstrates the largest reduction in the minimization objective that is known to be possible while remaining feasible, to requiring only feasibility, regardless of the resulting objective values.*

Remark 7.11. *As our setting is for inequality constrained optimization, it is reasonable to require strict feasibility of the solutions for them to be counted in an algorithm's percentage score. For the case of equality constrained optimization, a tolerance should be used to only require a sufficiently close to feasibility condition since attaining exact feasibility is unlikely to be possible.*

A relative minimization profile, which is visually interpreted like a receiver operating characteristic [Wik14] or ROC curve, highlights when an algorithm's performance is subpar, such as when it either frequently fails to satisfy feasibility or tends to stagnate, while simultaneously showing which algorithms most frequently and closely match the most amount of minimization that is known to be attainable on the feasible sets for the test problems. In fact, an RMP is quite similar to that of a normal performance profile, but with a twist in the sense that feasibility is used to determine whether an algorithm succeeded or failed while the performance metric compares the algorithms against each other in terms of the number of digits that they were able to match in the objective value being minimized compared to the best known solution per problem. By contrast, regular performance profiles use a relative ratio with respect to the selected performance metric (usually running time) of the best performing algorithm per problem. However, this is a poor choice of scaling for assessing how closely two or more algorithms are able to minimize objectives over the data set, which is a crucial consideration when comparing optimization methods for nonconvex and/or nonsmooth problems where methods may either converge to completely different stationary points or to ones that provide approximately the same amount of minimization but only to a certain degree of precision. By instead using a base 10 logarithmic scaling of the *relative differences* from the lowest known feasible objective value for each problem, it can be clearly seen with what frequency the algorithms are either finding the same *quality of solutions* (and to what precision) or *different solutions* altogether and how the algorithms' scores in terms of percentage of feasible solutions found over the test set improve as the required precision for matching the best known feasible objective value for each problem is reduced. As a result, there is no need to set an arbitrary limit on how much minimization is required in order to determine an algorithm's success / failure on any particular problem, which is an issue when using performance profiles, even for convex problems where typically the solution value is used as the success metric and running time is used as the performance metric. Furthermore, there is no need for a probabilistic interpretation (as is done in performance profiles) as the pertinent statistics of each algorithms' behavior relating frequency of success over the data set and amount of minimization is explicit in the FPM itself.

Remark 7.12. *As alluded to, relative minimization profiles are not just applicable for comparing optimization methods on constrained nonconvex problems but also on constrained convex problems, where it is important to assess the accuracy of the solutions, and/or the amount of minimization achieved before reaching a solution, in concert with how frequently feasibility was satisfied. This is particularly important in the nonsmooth setting where algorithms may slow down or stagnate as they approach a minimizer of a convex constrained problem or for cases when just finding a point in the feasible set itself is difficult* (*e.g. when the feasible set has measure zero*) *and there thus may be little consistency in the corresponding objective values found by the algorithms being compared.*

7.5.2 Comparing time via multiple relative minimization profiles

As a single relative minimization profile makes no indication of the relative running times of the algorithms, we may instead generate several of them, where each RMP corresponds to a particular set of maximum time limits allowed for each problem. For example, if we wish to assess how well algorithm A compares to algorithms B and C, we can generate an RMP where each algorithm is only scored upon the best solution it has found so far *at the time* that algorithm A finishes for each particular problem. Thus, algorithm A is not held to any time restriction while algorithms B and C potentially are, specifically on the problems for which they take longer to converge than algorithm A does. If we then suspect that while algorithm A might usually be fastest but that it also often stagnates compared to the other algorithms, we can generate a second RMP where we allow algorithms B and C to have, say 10 times the amount of CPU time per problem as algorithm A needed per problem, to see whether algorithms B and C, when given a time advantage, make additional progress beyond what is achieved by algorithm A. Thus, for a relative specified amount of CPU time per problem compared to one algorithm's running times, we can assess which algorithm has made the most progress across the entire data set, both in terms of whether feasible solutions have been attained and by how much they reduce the objectives. A combination of several of these plots, for different relative amounts of CPU time, allows us to roughly gauge the performance of a specific algorithm to the competition in terms of relative *rates of progress* for satisfying feasibility and amount of minimization of the objectives. The actual relative time factors chosen to highlight the key performance differences between algorithms will of course be specific to any given evaluation and as such, the time factors must not only be chosen sensibly but also in an unbiased manner.

Remark 7.13. *It is not necessary to use relative time factors with respect to the times per problem for one chosen algorithm when generating a set of relative minimization profiles. For example, one could consider using relative time factors with respect to the fastest time per problem over all the algorithms. A potential concern with this approach however is when the comparison contains an algorithm which is particularly quick but is also particularly bad, which could require very large relative time factors to be chosen for each additional RMP and could thus be potentially misleading. Using the average or median time per problem is most likely a good overall choice for any general comparison using multiple RMPs. However, when the purpose is to just evaluate a single algorithm in comparison to one or more other algorithms, as opposed to an all-pairs comparison,* *using relative factors with respect to just the running times of the algorithm in question is quite appropriate.*

Remark 7.14. *In fact, it is not necessary to use relative time factors at all with respect to a chosen algorithm in a set of relative minimization profiles. However, doing so avoids the problem of how to choose specific per problem time limits on a test set that may be comprised of instances of varying dimension as well as difficulty, the latter of which may not even be a consistent property across algorithms and could be a complete unknown, such as when a test set is used for the first time.*

Note that while relative minimization profiles precisely depict the amount of minimization achieved versus the frequency of satisfying feasibility, the effect of changing the maximum amount of time allowed per problem is only coarsely approximated across two or more RMPs. This is due to several reasons. First, timing information is inherently variable, in contrast to assessing feasibility of the solution and its corresponding objective value. Second, by the discrete nature of the iterates, there will also be an unavoidable quantization effect when trying to compare two or more algorithms at any given point in time. Third, depending on the codes being compared, obtaining accurate histories of the iterates and at what time in the computation they were computed may not always be a practical endeavor.

In the case when a code does not return timing data, coupling the implementations of the objective and constraints functions to some persistent storage (e.g. via object-oriented programming) that records each evaluation and the time at which it occurred is typically an effective strategy to obtain timing data unbeknownst to the optimization code. However, this can introduce significant overhead and affect the timing data. Furthermore, there is little to no way of reliably determining which evaluations correspond to the actual iterates accepted by the algorithm at every step though this is not necessarily a problem for producing RMPs. Provided that the codes at least return a history of the iterates, calculating the average time per iterate and using it as a proxy to determine what progress an algorithm has made by any given time can however often be a suitable and easily attained approximation. Though there may be variability in the time each iteration takes, if the data set is sufficiently large and a high number of iterations is typically needed for any method to converge on the problems, then such an approximation is likely to be good enough for qualitative running time comparisons for a set of plots made with different per problem time limits.

Despite any inaccuracies in the timing data, if the methods are indeed significantly different in terms of speed, that property itself will most likely be the dominant factor in determining what is presented by the plots, not the inherent limitation of having to estimate the exact timing data. On the other hand, if comparing the relative speeds of the algorithms requires high precision, then it suggests that running time is not a particularly discriminating performance characteristic amongst the codes being compared. In such cases, a single RMP without a per problem time limit specified should be sufficient, perhaps with the overall time reported for each algorithm to complete the entire data set. As a result, we typically envision using a set of multiple RMPs to show significant differences in running times between algorithms, where the relative time factors selected are well above the level of error in the timing data. For example, any issues in the timing data would most likely be inconsequential if the relative time factors were chosen to be one, two and four.

Remark 7.15. *Some comments on data profiles* [MW09] *are in order. Data profiles are visualization tools that are also interpreted as ROC curves and are thus similar in ap-*

pearance to performance profiles, but instead attempt to measure the overall progress of an algorithm towards solutions in a test set with respect to some constraint on the computational resources. Specifically, they propose using the number of function and/or gradient evaluations to assess the cost of the algorithms and measure the percentage of problems in the test set that are solved to some specified level of accuracy. As the allowed number of function and/or gradient evaluations is increased, the algorithms can potentially return more and more accurate solutions, and thus the percentage of problems considered to be solved sufficiently well monotonically increases. By creating multiple data profiles, one plot for each different level of accuracy permitted to be considered an acceptable solution to any given problem, one can assess which algorithms are fastest for obtaining only approximate solutions versus which are fastest for resolving accurate solutions to a high precision. However, data profiles do not provide a direct way for additionally assessing feasibility for constrained problems nor do they address the possibility of obtaining different quality of solutions in the nonconvex setting.

Data profiles are precise in terms of the cost to run the algorithms but only give an approximate indication of the accuracy of the solutions achieved unless many plots are made for various specified accuracy levels. By contrast, relative minimization profiles are precise in terms of both the frequency of feasible solutions found and the quality of minimization achieved over the data set (*and their relationship to each other*) *while the number of RMPs produced and what relative time factors are used determine how fine-grained the timing comparison is amongst the algorithms being compared. We believe this latter approach is typically a more useful prioritization for comparing algorithms, at least in a nonconvex and/or nonsmooth setting.*

Remark 7.16. *In fact, even in an unconstrained convex setting, a set of relative mini-*

mization profiles could be used in lieu of data profiles to compare the relative rates of progress of algorithms for different computational budgets (with an RMP produced for each budget specified), particularly useful when one desires a fine-grained assessment of the relative level of minimizations achieved per problem at different relative times to thus track which algorithm is best for each computational budget and level of accuracy specified.

7.6 Numerical results

7.6.1 Experimental setup

In order to empirically validate that BFGS SQP is both an efficient and reliable method for nonsmooth, nonconvex constrained optimization, we created two test sets comprised of 100 problems each, where the problems for one set are of the form given in (7.12) while the problems in the second set are of the form given in (7.13). For both the multi-objective spectral radius and pseudospectral radius test sets, each problem was comprised of two to five plants, split randomly between the objective and constraints (ensuring at least one plant one was assigned to each). In order to compare to SQP-GS, since gradient sampling is so expensive, we chose to generate small dimensional test problems, where n was chosen randomly from $\{4, \ldots, 20\}$ for the spectral radius problems and from $\{4, \ldots, 8\}$ for the more expensive-to-compute pseudospectral radius problems. The dimensions for m and p were loosely generated with a preference to be in the interval $(0.5kn, 1.5kn)$. Matrices A_i were generated using `randn(n)` and were subsequently destabilized if they were in the objective and stabilized otherwise, by successively multiplying the matrices by a constant greater/lesser than one respectively. Many of the B_i and C_i matrices were similarly generated via `randn(n,p)` and `randn(m,n)` respectively, but we also generated B_i matrices that consisted of a column of ones followed by a randomly generated distribution of positive and negative ones over the remaining entries of the matrix while C_i was sometimes set to consist of rows of the identity matrix. Since the origin is a feasible point for all the problems, each algorithm was initialized with $X = 0$ and $\mu = 16$.

We used the 1.1 version of SQP-GS so we could compare BFGS SQP to a method that provides provable convergence guarantees for nonsmooth, nonconvex optimization, even though its expensive but robust gradient sampling technique makes its unlikely to be efficient. As both SQP-GS and BFGS SQP require a QP solver and for these experiments we used MOSEK 7. We implemented SFPP using the unmodified BFGS routine from the 2.02 version of the HANSO optimization package [Han] implemented in MATLAB, with a set limit of 20 iterations per BFGS call before lowering the penalty parameter if the 20th iterate was infeasible.

To implement BFGS SQP, we modified the BFGS routine from HANSO, replacing the standard BFGS search direction computation with the steering strategy of Procedure 7. However, as the BFGS version from HANSO actually approximates the inverse of the Hessian, we chose to adapt Procedure 7 and Procedure 8 using the dual form of the QPs, instead of the primal as stated in the pseudo codes. For tolerances, we set $c_v := 0.1$ and $c_\mu := 0.5$ in Procedure 7 and limited its loop to a most ten iterations, while for Procedure 8 we set $\tau_\diamond$ to machine precision (even though this is quite unusual in practice) and $\tau_v := 0$.

In order to encourage the algorithms to run as long as possible, as per our choice of tolerances for BFGS SQP, we also set the termination and violation tolerances of SQP-GS and SFPP to machine precision and zero respectively and set a max iteration limit of 500 for all, since creating multiple relative minimization profiles allows us to evaluate the algorithms at particular points in time that we specify. For the experiments here, we chose relative time factors of 1, 5, 10 and ∞ with respect to the per problem running times of BFGS SQP. For example, for a time factor of 5, SQP-GS and SFPP were each given five times the amount of time BFGS SQP took to converge per problem, while the time factor of ∞ means that the SQP-GS and SFPP were not restricted by time specifically but only by the maximum iteration count of 500 or whether they terminated themselves before that. As per iterate timing data was not a feature available in either the SQP-GS or HANSO's BFGS codes, we chose to approximate the timing data by using an average time per iterate proxy. Given the large difference in running times between the algorithms and the fact the convergence is generally slow on the nonconvex, nonsmooth constrained optimization problems created for our test sets, it is unlikely that using an average time per iterate proxy would skew a set of RMPs for comparing the relative rates of progress for the algorithms.

All experiments in this chapter were run on an Intel Xeon X5650 2.67 Ghz CPU using MATLAB R2012b.

7.6.2 Multi-objective spectral radius optimization

We begin analyzing the relative performance of the three codes over the multi-objective spectral radius test set by first turning our attention to the bottom right RMP in Figure 7.5. Here, each algorithm was allowed to run without any time restriction and we see that despite the promising example shown earlier in Figure 7.1, SQP-GS ended up finding over 80% of the best optimizers while BFGS SQP only found just under 20% of the best optimizers. Furthermore, we see that BFGS SQP's performance over the data set only starts to look better if we greatly relax how much of the minimization of the best known solution is necessary to be considered a success, indicating that BFGS SQP in general either converged to different optimizers (and worse ones at that for this test set) or perhaps was stagnating early compared to SQP-GS. On the other hand, SFPP appears to be completely uncompetitive by comparison to either BFGS SQP and SQP-GS. Though this doesn't initially appear promising for BFGS SQP, the fact remains that SQP-GS actually took 27.4 times longer to run.

However, if we look at the progress SQP-GS has made when it is allowed to run 10 times longer than BFGS SQP per problem, as shown in the bottom left RMP, we see that SQP-GS actually did slightly worse than BFGS SQP, even though it had a tenfold time advantage. If we further consider the RMP in the top right, where SQP-GS is only allowed up to five times as much time as BFGS SQP per problem, we see that the relative performance of the two algorithms has practically reversed, where it is now clear that BFGS SQP performed significantly better than SQP-GS, even though SQP-GS still is being evaluated with a fivefold time advantage. Finally, in the top left RMP, where SPQ-GS and SFPP are assessed at the per problems times at which BFGS SQP terminated, we that BFGS SQP completely outclasses SQP-GS due to its much higher efficiency. Impressively for both BFGS SQP and SQP-GS is that they are able to be effective at all, and in fact quite so, when attempting to optimize functions that are not locally Lipschitz, even though neither algorithm has convergence results in this setting.

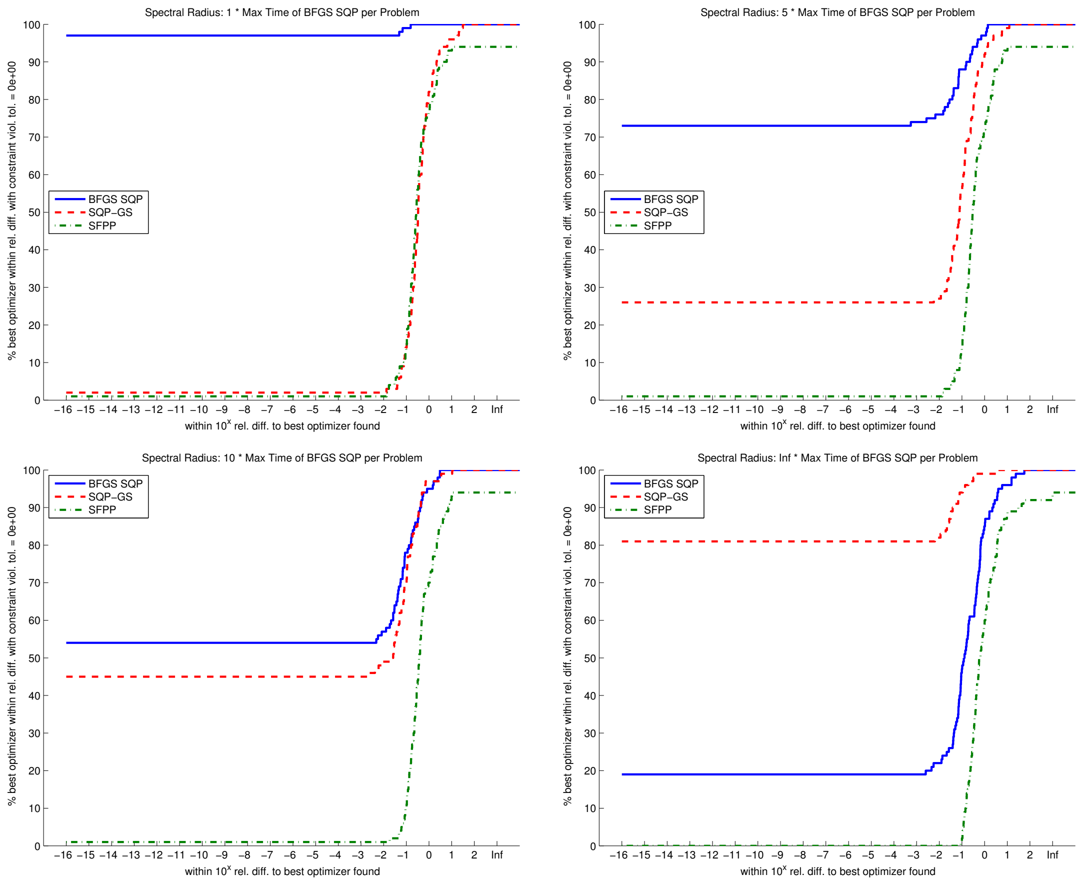

FIGURE 7.5: *Overall performance assessment on the multi-objective constrained spectral radius optimization problems of* BFGS SQP, SQP-GS, *and* SFPP, *with relative time factors of 1, 5, 10, and* ∞ *compared to the per problem times of* BFGS SQP. *In the top left panel for a time factor of 1, we see that the other two algorithms have barely made progress towards feasibility and minimizing the objective function when* BFGS SQP *has terminated. In the top right, even with the opportunity to run 5 times longer per problem than* BFGS SQP, *we see that* SQP-GS *and* SFPP *are still nowhere near as competitive. In the bottom left, when given the opportunity to run 10 times longer per problem than* BFGS SQP, *we see that* SQP-GS *is starting to come close to matching* BFGS SQP. *In the bottom right panel, where all the algorithms have no limits on running time besides the 500 max iteration count, we finally see* SQP-GS *outperforming* BFGS SQP *and by a significant margin, though it took 27.4 times longer to do so. Even without a time restriction,* SFPP *is indeed extremely ineffective and, interestingly, ran in only 0.435 of the time that* BFGS SQP *took, perhaps indicating that* SFPP *often suffered from early stagnation and/or breakdown on this test set of non locally Lipschitz optimization problems.*

7.6.3 Multi-objective pseudospectral radius optimization comparison

On the multi-objective pseudospectral radius problems, we see that even in the bottom right RMP, where no running time restrictions were put on the algorithms, BFGS SQP slightly outperformed SQP-GS over the test set. To put that in context, BFGS SQP produced just as good or better minimizers compared to SQP-GS, and in fact did so more frequently, while also being 29.6 times faster than SQP-GS. This is a remarkable outcome given that the convergence results for SQP-GS hold for the problems in the multi-objective pseudospectral radius test set (as they are locally Lipschitz) while BFGS SQP provides no theoretical guarantees. Indeed, it is a stark contrast to what we observe on the multi-objective spectral radius test set, where SQP-GS could ultimately still pull ahead compared to BFGS SQP, albeit when allowed to run 27.4 times longer. Interestingly, we also see that SFPP appears to do better on the Lipschitz problems though it is still quite unimpressive to say the least. As we look at smaller relative time factors for the other RMPs in Figure 7.6, we see that the comparison is ever more in favor of BFGS SQP. In other words, SQP-GS was not only just 29.6 times slower than BFGS SQP on the pseudospectral radius problems, but it apparently needed that much of a time advantage to be at all competitive in terms of the quality of the computed solutions.

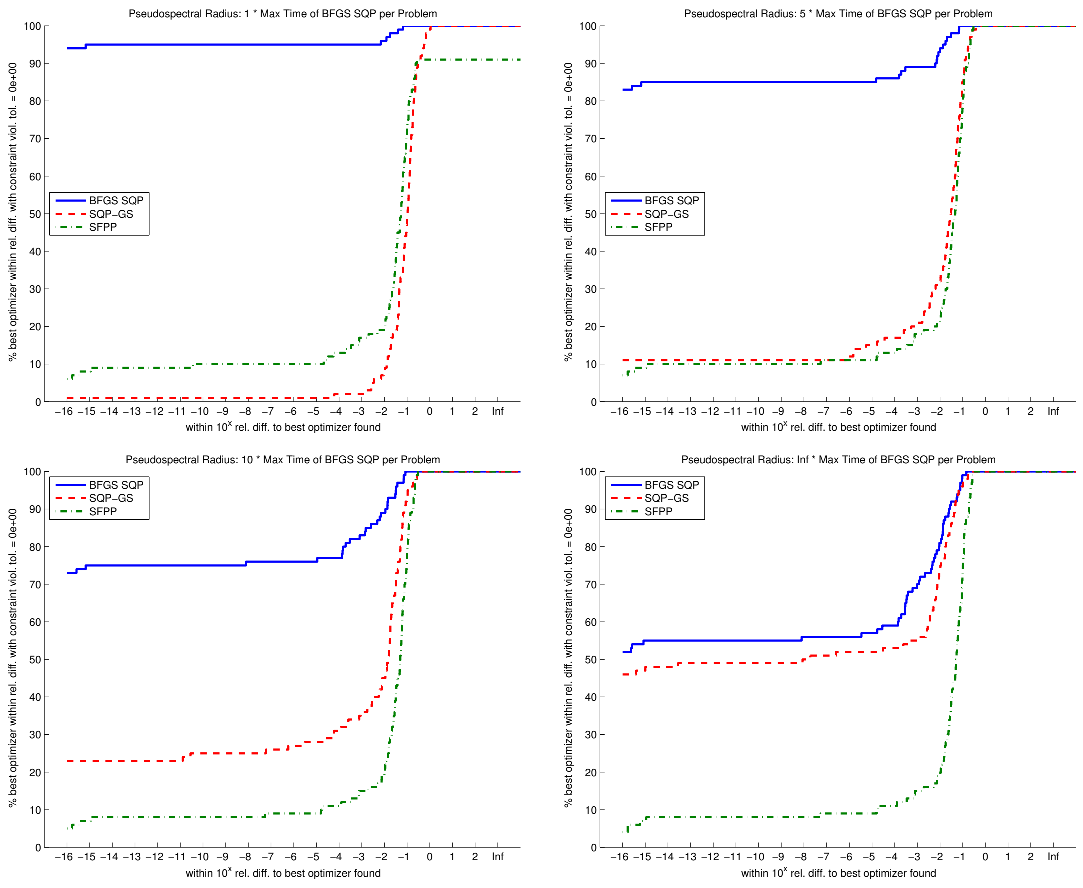

FIGURE 7.6: *Overall performance assessment on the multi-objective pseudospectral radius optimization problems of* BFGS SQP, SQP-GS, *and* SFPP, *with relative time factors of 1, 5, 10, and* ∞ *compared to the per problem times of* BFGS SQP. *In the top left panel for a time factor of 1, we see that the other two algorithms have barely made progress towards feasibility and minimizing the objective function when* BFGS SQP *has terminated. In the top right, even with the opportunity to run 5 times longer per problem than* BFGS SQP, *the other algorithms* SQP-GS *and* SFPP *are still nowhere near as competitive. In the bottom left, when given the opportunity to run 10 times longer per problem than* BFGS SQP, *we see that performance of* SQP-GS *is improving but is still nowhere near to matching* BFGS SQP. *In the bottom right panel, where all the algorithms have no limits on running time besides the 500 max iteration count, we finally see that* SQP-GS *starts to approach the performance of* BFGS SQP *though it takes 29.6 times longer to do so. On the other,* SFPP *is still extremely ineffective and even takes 1.35 times as long to complete the test set compared to* BFGS SQP.

7.6.4 The effect of regularizing the Hessian approximation

As the enforced limits on the conditioning of the Hessian approximation required for SQP-GS's convergence results seem to be at odds with the argument in the unconstrained case that ill-conditioning is actually beneficial [LO13], we explore the effect of regularizing the Hessian approximation in BFGS SQP. However, as BFGS SQP requires solving QPs inside the BFGS iteration, there is also a potential tradeoff of regularizing to improve the QP solves for determining a search direction versus not regularizing to retain the ill-conditioning in the BFGS Hessian approximation. To assess the effect of regularization, we reran the BFGS SQP experiments multiple times, where for each the Hessian approximation was regularized such that its condition number was no more than 10^{2j}, for $j = 0, \ldots, 8$, where the case of $j = 0$ corresponds to replacing the Hessian approximation by a multiple of the identity.

For the spectral radius problems, as shown in Figure 7.7, we generally see that any regularization hurts performance and more regularization is worse. We suspect that regularization can make BFGS stagnation more likely and certainly this is case when the Hessian approximation is completely regularized to be a multiple of the identity, where BFGS reduces to a steepest descent method. For the locally Lipschitz problems however, where pseudospectral radii are being minimized, we see in Figure 7.7 that moderate levels of regularization (such as 10^6) seems to have a strikingly clear beneficial effect compared to not regularizing at all. It is hard to say why we observe this behavior but it is conceivable that the regularization helped improve the accuracy of the QP solves more than it hurt the effectiveness of BFGS. However, it is apparent that the observed effects of regularization is problem dependent since there is no parallel to the regularization results for the spectral radius test set. Certainly more investigation into this matter would

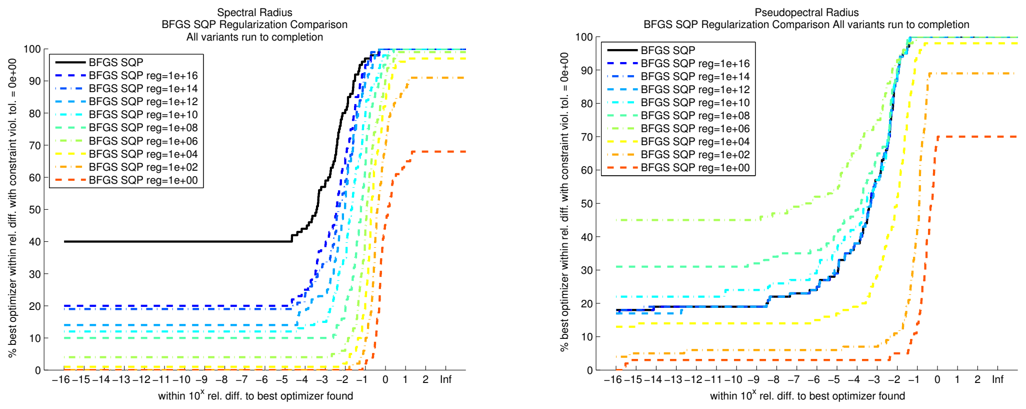

FIGURE 7.7: *Left: the spectral radius test set. Right: the pseudospectral radius test set. Both panels show how the performance of* BFGS SQP *changes over the test sets when increasing amount of regularization is applied to the the inverse of the Hessian approximation.*

be a worthwhile pursuit.

7.7 Conclusion

We have shown that our proposed BFGS SQP constrained optimization method for the case of nonsmooth, nonconvex objective and constraints is not only highly efficient but is also very reliable at finding good solutions, both in terms of satisfying feasibility and minimizing the objective compared to the competing algorithms. Not only did BFGS SQP completely outperform the baseline SFPP method, it also outperformed SQP-GS on a challengingly test set comprised of locally Lipschitz problems, while performing quite respectably compared to SQP-GS on an even harder test test, where the objectives and constraints are not locally Lipschitz. Our findings indicate that despite a lack of convergence results for BFGS SQP, it is a highly practical method due to its cheap cost per iteration, ability to quickly promote progress towards the feasible region and its generally robustness on exceptionally difficult problems in nonsmooth, nonconvex constrained optimization. Finally, we have proposed relative minimization profiles (RMPs) as a new tool for comparing algorithms for nonsmooth, potentially nonconvex, constrained optimization, which allow for a concise yet detailed visualization for comparing the pertinent performance characteristics of algorithms, and how they interrelate, and which facilitated the algorithmic comparison done here.

CONCLUSION

In this thesis, we have developed two exceptionally fast and robust algorithms, one for approximating stability measures with provable convergence guarantees and another more general routine, with applications outside the scope of this thesis, intended for optimizing nonsmooth stability measures in a constrained setting. As the first algorithm, hybrid expansion-contraction, allows for efficiently approximating the H_∞ norm for large-dimensional systems, an interesting subject for future work would be to build on this further to develop a new version of HIFOO [Hif] for designing and optimizing fixed-order controllers for large-dimensional dynamical systems. Furthermore, the extension of the algorithm to stability measures restricted to real-valued perturbations, bounded by the Frobenius or spectral norm, could add useful new performance measures to the HIFOO toolbox.

The second algorithm, BFGS SQP, by contrast offers no theoretical guarantees, but nonetheless appears to be a remarkably reliable and efficient optimization method in practice, often performing as well or better than the provably robust gradient sampling based algorithm SQP-GS, while being many times faster as well. As the latest release of HIFOO actually uses a version of the simple SFPP algorithm to design and optimize multi-objective controllers, there is likely much to be gained by replacing it with BFGS SQP in a future version.

For comparing numerical algorithms, we have proposed a novel visualization tool called relative minimization profiles that allows for evaluating multiple algorithms with respect to three important performance characteristics, highlighting how these measures interrelate to one another and compare to the other competing algorithms. In the comparison done here, we evaluated the algorithms on test sets comprised of nonsmooth, nonconvex, constrained optimization problems, where we used the profiles to evaluate attaining feasibility, amount of minimization achieved, and relative rates of progress of the algorithms. However, we have noted that relative minimization profiles could be useful analysis tools in other settings, such as for the unconstrained or convex cases, and it would be interesting to compare their utility with other more established comparison tools available, such as performance and data profiles [DM02, MW09], in these problem spaces.

Many important considerations remain open. As hybrid expansion-contraction can only be expected to find local maxima, it is worthwhile to investigate strategies for encouraging convergence to global maximizers or at least good local maximizers, which is an important consideration if the algorithm is to be used for designing controllers. While the numerous optimizations presented in this thesis have greatly reduced the potential cost incurred by the slow linear convergence of SVSAR, we note that for the special case of pseudospectra, our proposed vector extrapolation based technique, though quite good, is still often slightly outperformed by the subspace acceleration method of [KV14]. Though it indeed seems very difficult to extend the latter method to the general case of spectral value sets, any improved acceleration technique for SVSAR could further increase the efficiency of hybrid expansion-contraction. Finally, while much of the algorithm has been adapted to the real-valued spectral norm case, it remains to be seen how to compute the next perturbation vectors so that each iterate of the SVSAR subroutine makes guaranteed rightward/outward progress in the spectral value set. Though our randomized SVSAR variant can be used as a substitute in this case and appears to be effective in terms of reliability, it is generally too slow to be of much practical use.

APPENDICES

A
Contributions to Third-Party Codes

- ARPACK

  In 2011, I discovered and patched a nasty bug in ARPACK where if eigenvectors were requested and the number of converged Ritz values returned was less than number of eigenvalues requested, then ARPACK might erroneously return wrong eigenpairs. For example, if ARPACK is set to find eigenvalues with the largest real parts, under the aforementioned conditions of this bug, ARPACK could erroneously return Ritz values from the Krylov subspace that did not correspond to the desired ordering, and thus the eigenvalues of largest real part would not necessarily be returned to the user, if those Ritz values had converged. Furthermore, some of the returned pairs might not even be accurate eigenpairs since the values returned to the user might not even have converged. The bug was due to a reordering procedure not being properly called to first sort the output so that the desired converged pairs were in the beginning of the array, which was where the output Ritz values were copied from. I submitted my patch along with demo codes in MATLAB and Fortran to replicate the bug to the ARPACK authors, The MathWorks, and the community maintained repository, `arpack-ng`, which has been replacing the official ARPACK source in many Linux distributions. My ARPACK patch ships by default with MATLAB versions R2012b and later.

- `eigsPlus` v1.0 / `eigs`

  The routine `eigsPlus` is my own small fork of the standard MATLAB routine `eigs`, motivated by the fact the standard version of `eigs` used to return zeros for Ritz values that didn't converge, which the user may unwittingly mistake as actual eigenvalues. For example, if the rightmost eigenvalue of the matrix in question resides strictly in the left half-plane and if say 6 eigenvalues are requested but not all of them converge, then `max(real(eigs(A,6,'LR')))` would misleading return 0 as the spectral abscissa, even if the largest real part eigenvalue had converged. Even more problematic is that this erroneous value of 0 may then be silently propagated to subsequent code without the user or programmer realizing it. At my urging that this is an important issue, The MathWorks have since updated `eigs` such that it instead uses `NaN`'s as placeholders for Ritz values so this is no longer an issue for the standard `eigs` routine in MATLAB releases R2013b and later. However, `eigsPlus` still has some additional features that are not as of yet available in the latest release of `eigs`. For one, `eigsPlus` only returns the converged Ritz values and it always returns them in a vector, rather than a matrix, even if the eigenvectors are requested. As the number of eigenvalues returned (and correspondingly, the number of eigenvectors returned, if requested) will indicate whether all the requested Ritz values converged or not, `eigsPlus` removes the optional third output argument `flag` and instead returns the number of ARPACK iterations incurred until convergence. The `eigsPlus` routine is used by `dtiApprox` (the upcoming release of the hybrid expansion-contraction algorithm), `hinfnorm` v1.02, `psapsr` v1.3, and `subspace_pspa` codes.

- `hinfnorm` v1.02

  The v1.0 release of the routine `hinfnorm` is Mert Gürbüzbalaban's original implementation of the GGO algorithm, which was the first fast algorithm for approximating the H-infinity norm of large-scale dynamical systems, under the assumptions that the corresponding matrices are sparse and that the system has relatively few inputs and outputs. I discovered several bugs in the v1.0 code and released a patched v1.02 version that addressed many of these issues, one of which was critical enough to cause the v1.0 code to sometimes stagnate (different from the breakdown case). I additionally made some minor algorithmic improvements in the v1.02 release and ran the numerical experiments for the revised version of [GGO13] using the updated v1.02 code. Of note, it was this project that led to me to discover the breakdown case of the GGO algorithm and ultimately, the development of hybrid expansion-contraction.

- `psapsr` v1.3

  The v1.01 release of the routine `psapsr` is Michael Overton's implementation of the pseudospectral abscissa or radius approximation algorithm presented in [GO11] and is the predecessor to the generalized subroutine `svsar` for approximating the abscissa or radius of spectral value sets and used in `hinfnorm`. I discovered that the v1.01 release (as well as the v1.2 release) mistakenly ignored the user-supplied starting vectors and furthermore, even without that issue, the code would have proceeded incorrectly from the user-supplied initial vectors. I released `psapsr` v1.3 which addressed both of these issues and provided another fix for an unrelated minor bug. I additionally refactored some of the included subroutines for efficiency as well as clarity for the v1.3 code.

- `pspr` / EIGTOOL

  The routine `pspr` is Emre Mengi's code for computing the pseudospectral radius [MO05]. In 2013, I noticed and reported that the included copy of `pspr` that is bundled with EIGTOOL is actually out-of-date and may sometimes give incorrect answers. The problem and its solution is discussed in [BLO03, pages 371-373] for the related criss-cross algorithm for computing the pseudospectal abscissa. Fortunately, Emre Mengi's newer routine `pspr` properly handles this issue and gives accurate results, and will hopefully be bundled in EIGTOOL in the near future. At this time, I also pointed out and proposed a solution for a very minor plotting related bug in Emre Mengi's version of `pspr`, which he promptly fixed.

- `subspace_pspa`

  The routine `subspace_pspa` is Daniel Kressner's and Bart Vandereycken's implementation of the subspace acceleration method for approximating the pseudospectral abscissa [KV14]. As noted in Section 4.2.2, I recently discovered that `subspace_pspa`, v0.1 and v0.2, erroneously reports an incorrect value for its pseudospectral abscissa approximation, which appears to be in fact some intermediate value in the computation rather than the actual approximation computed. Fortunately, the correct value can apparently be obtained from the optional second argument, that is, by calling `[f,info] = susbspace_pspa(·)`, ignoring the incorrect return value of `f` and instead using `real(info.z_pspa(end))`. The issue has been reported to the authors, along with a comment that the console printing may contain a bug since it intriguingly doesn't always print out updates at every iteration.

B

Multi-Objective Spectral Radius Additional Examples

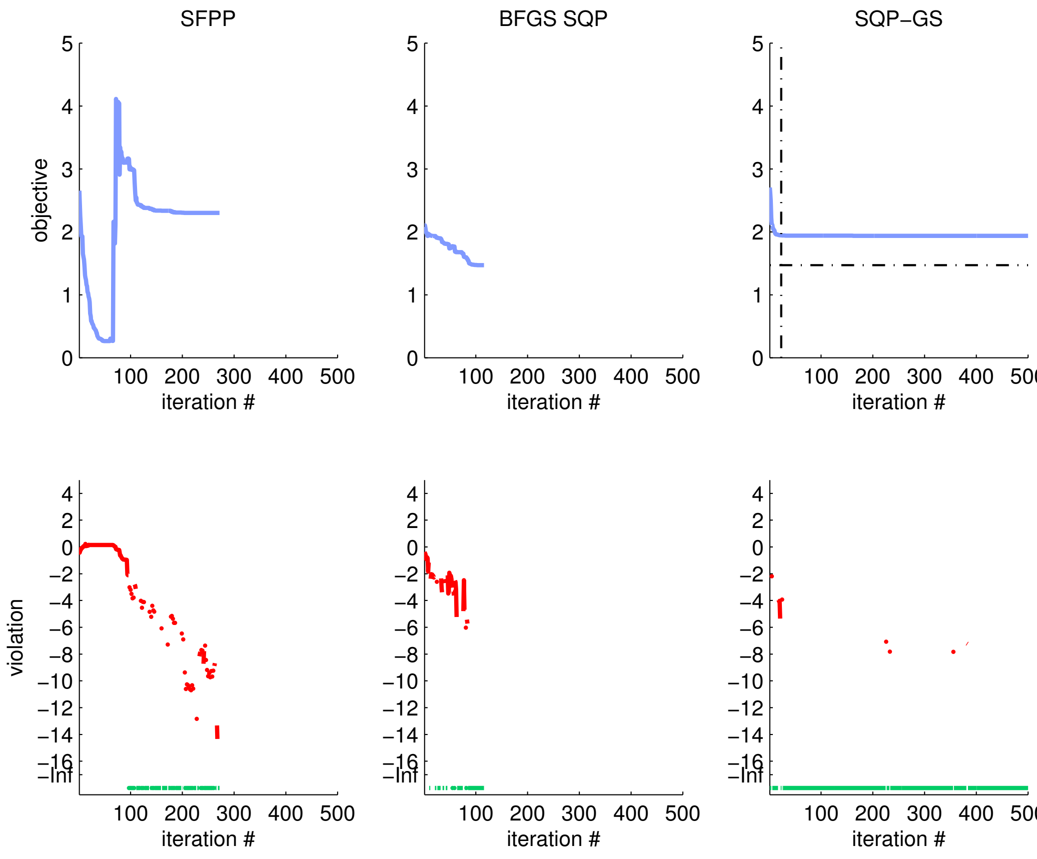

FIGURE B.1: *The plots in the top row track the value of the spectral radius based objective function in terms of iteration number for* SFPP (*left*), BFGS SQP (*middle*), *and* SQP-GS (*right*) *on a randomly generated example of dimension $n = 5$ comprised of one plant in the objective and one in the constraint and where the controller matrix has $mp = 4 \times 2 = 8$ variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

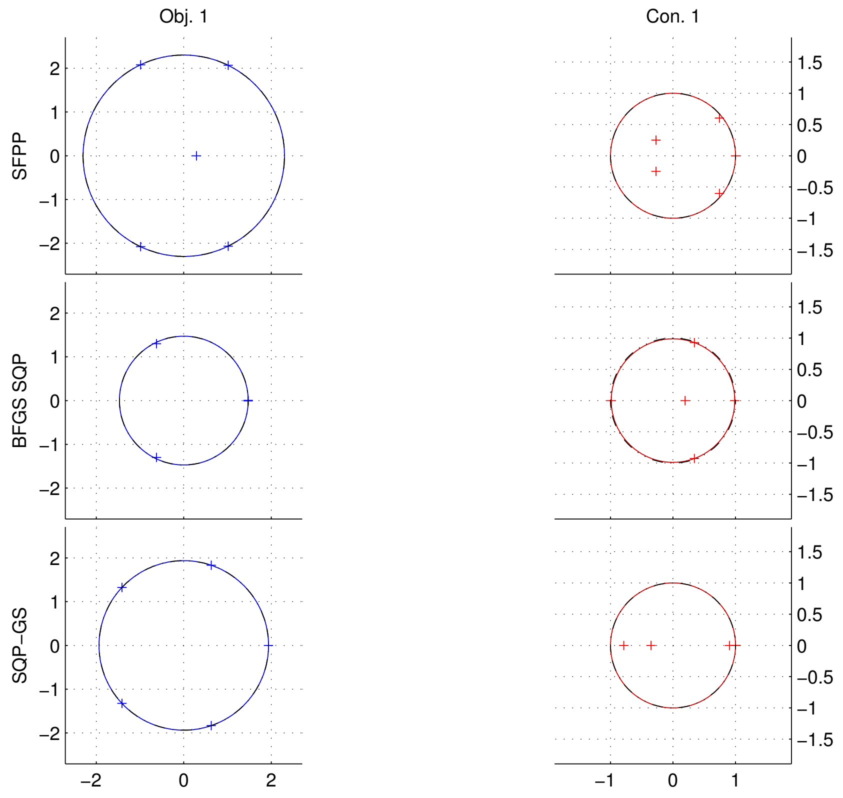

FIGURE B.2: *The three rows show the final spectral configurations of the three controllers found by* SFPP (*top*), BFGS SQP (*middle*), *and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension $n = 5$ comprised of one plant in the objective and one in the constraint and where the controller matrix has $mp = 4 \times 2 = 8$ variables. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle* (*the stability boundary*)*. The dashed black circle on the objective plots* (*barely visible*) *corresponds to the max spectral radius of the three plants in the objective for that particular algorithm's controller.*

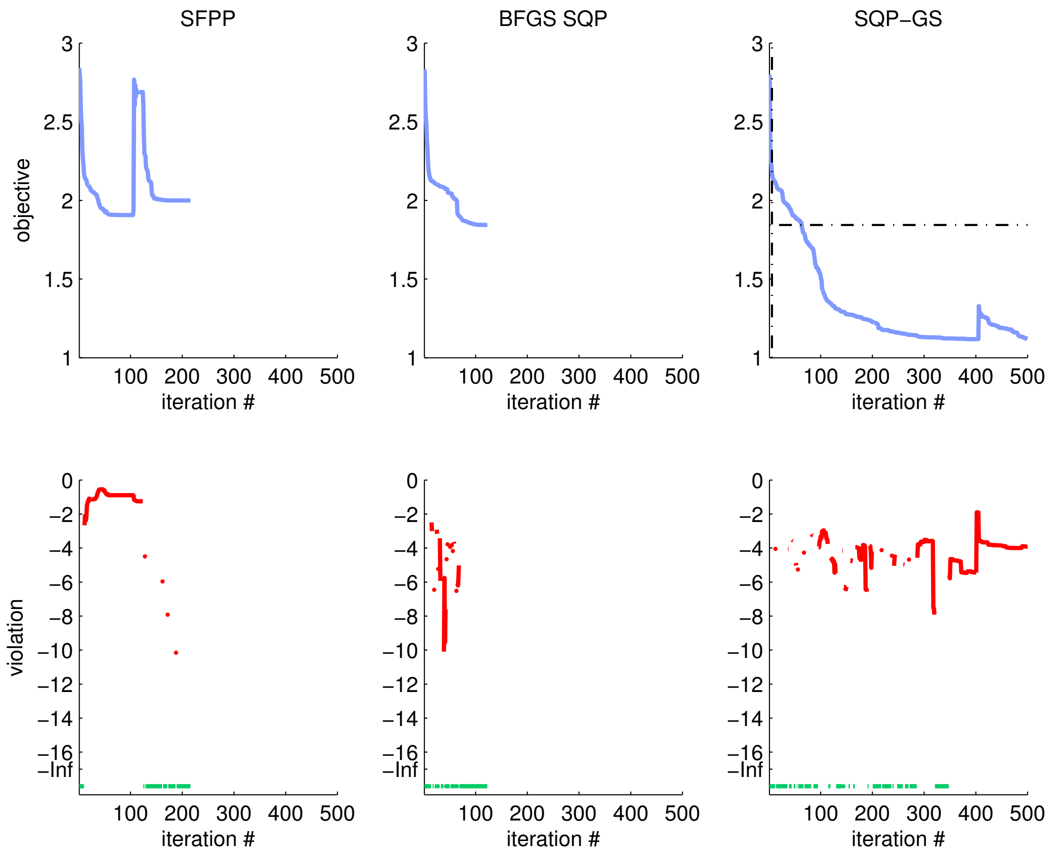

FIGURE B.3: *The plots in the top row track the value of the spectral radius based objective function in terms of iteration number for* SFPP (*left*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*right*) *on a randomly generated example of dimension $n = 10$ comprised of one plant in the objective and four in the constraint and where the controller matrix has $mp = 16 \times 1 = 16$ variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

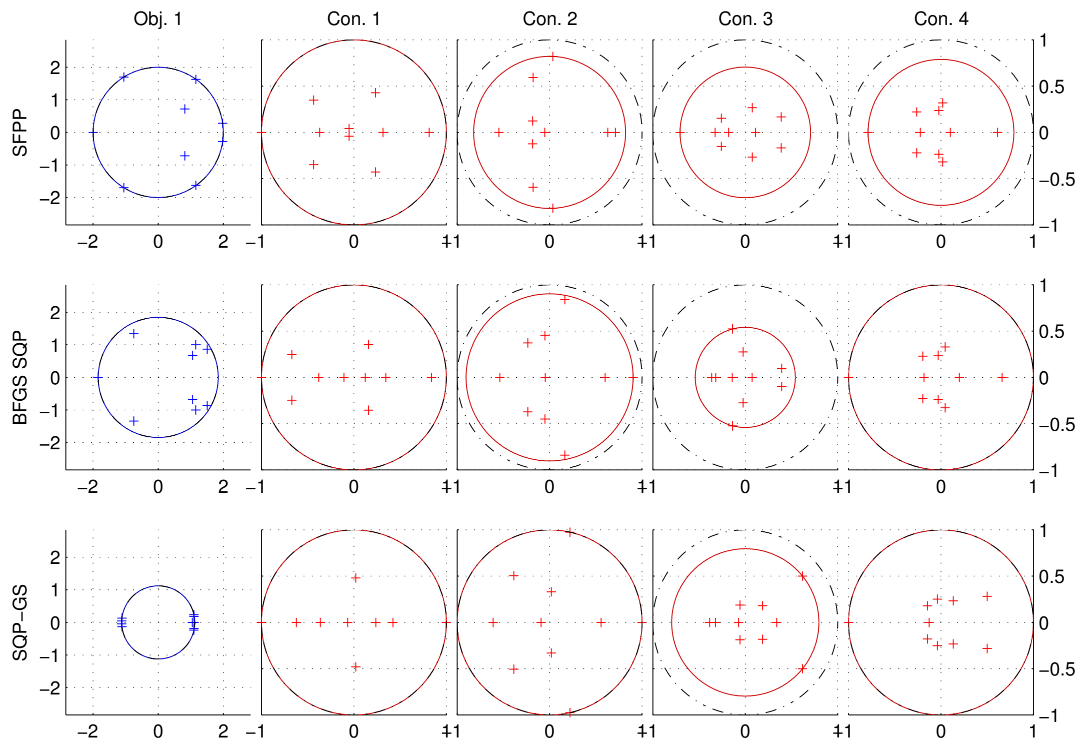

FIGURE B.4: *The three rows show the final spectral configurations of the three controllers found by* SFPP (*top*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension* $n = 10$ *comprised of one plant in the objective and four in the constraint and where the controller matrix has* $mp = 16 \times 1 = 16$ *variables. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle* (*the stability boundary*)*. The dashed black circle on the objective plots* (*barely visible*) *corresponds to the max spectral radius of the three plants in the objective for that particular algorithm's controller.*

C

MULTI-OBJECTIVE PSEUDOSPECTRAL RADIUS

ADDITIONAL EXAMPLES

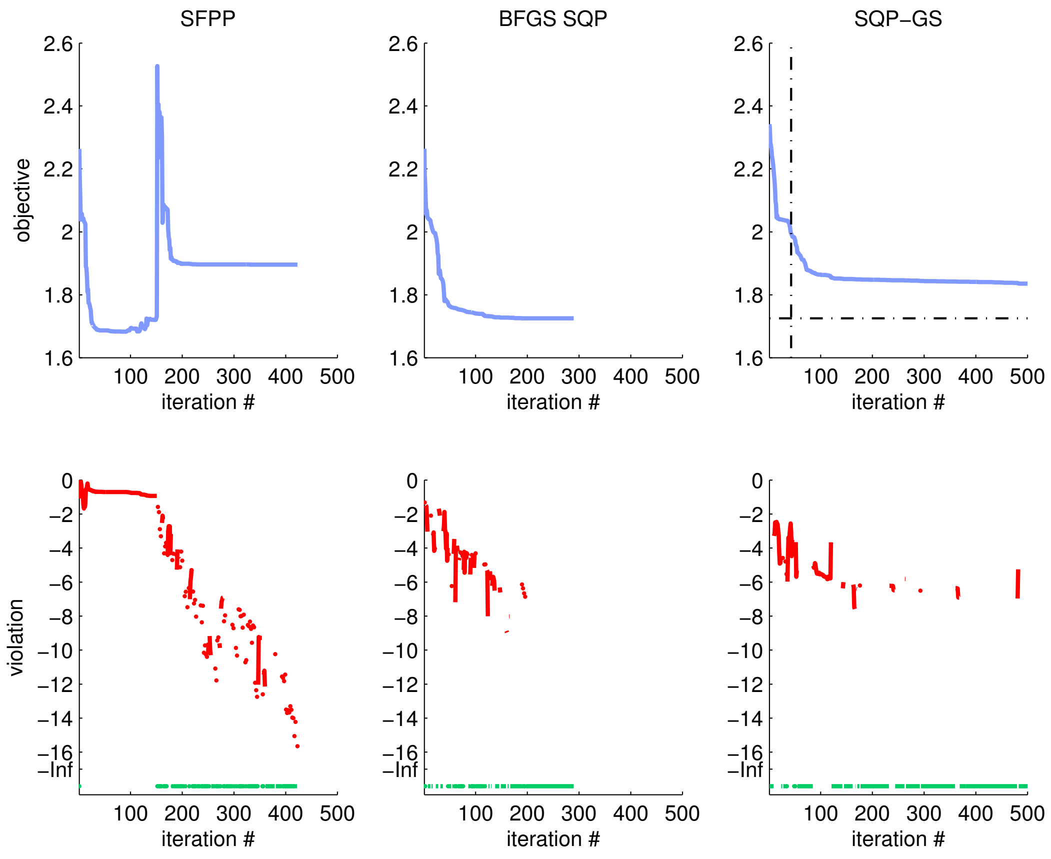

FIGURE C.1: *The plots in the top row track the value of the pseudospectral radius based objective function in terms of iteration number for* SFPP (*left*), BFGS SQP (*middle*), *and* SQP-GS (*right*) *on a randomly generated example of dimension* $n = 6$ *comprised of one plant in the objective and one in the constraint and where the controller matrix has* $mp = 7 \times 1 = 7$ *variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

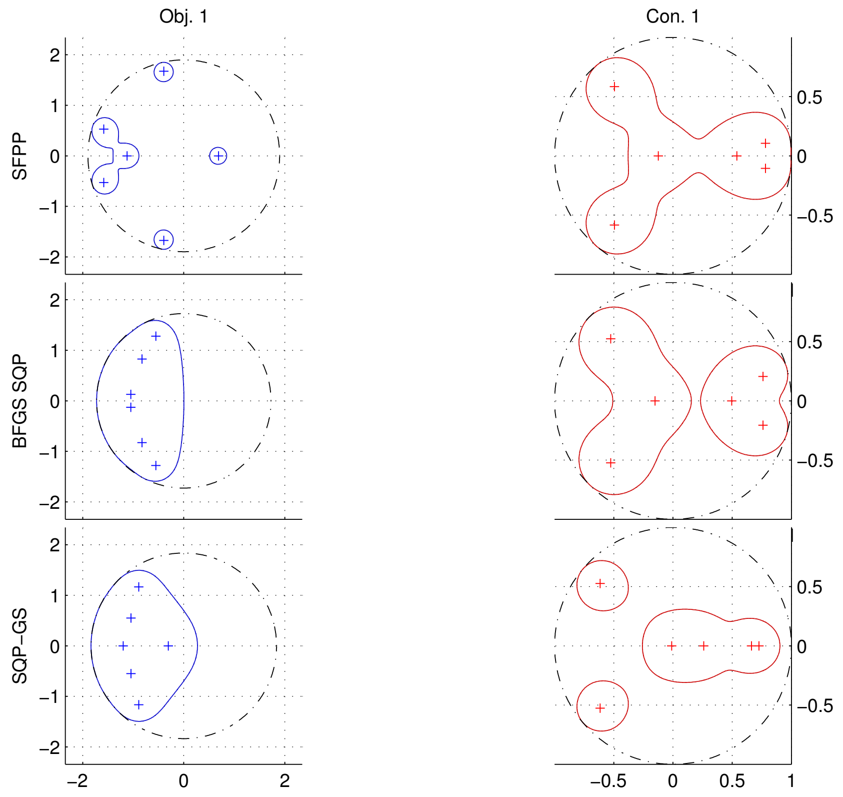

FIGURE C.2: *The three rows show the final pseudospectral configurations of the three controllers found by* SFPP (*top*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension* $n = 6$ *comprised of one plant in the objective and one in the constraint and where the controller matrix has* $mp = 7 \times 1 = 7$ *variables. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm's controller.*

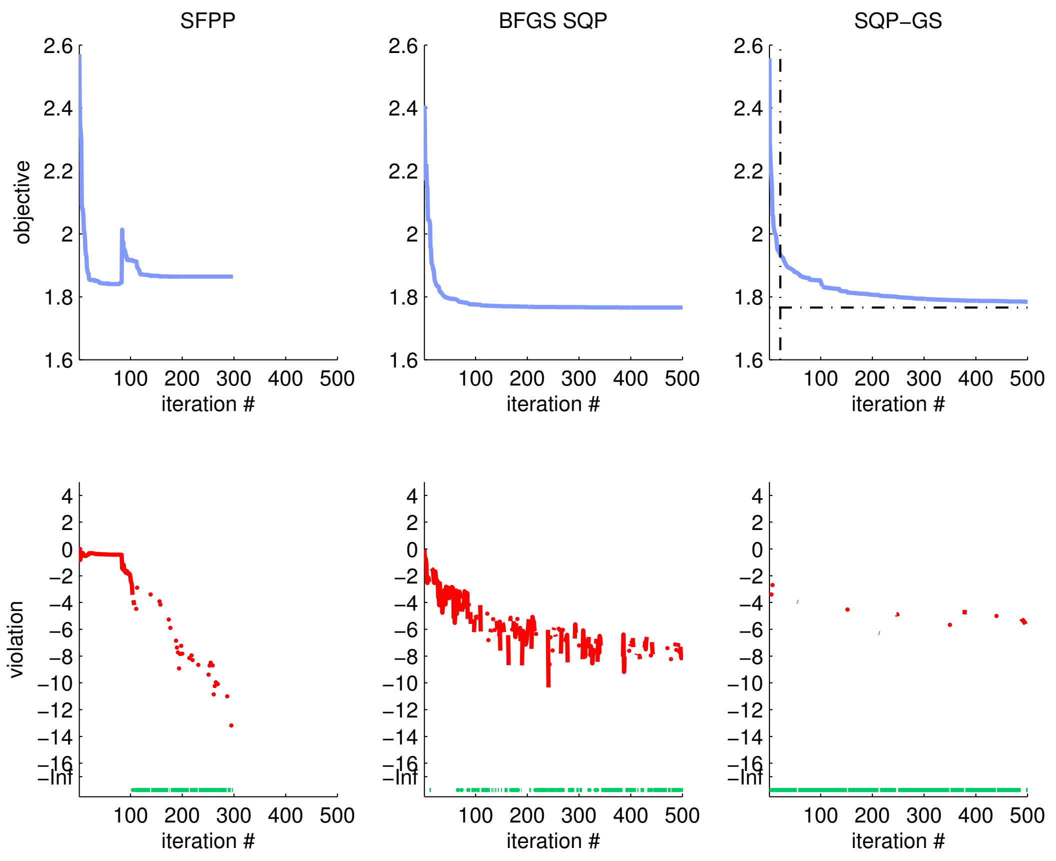

FIGURE C.3: *The plots in the top row track the value of the pseudospectral radius based objective function in terms of iteration number for* SFPP (*left*), BFGS SQP (*middle*), *and* SQP-GS (*right*) *on a randomly generated example of dimension* $n = 7$ *comprised of two plants in the objective and three in the constraint and where the controller matrix has* $mp = 5 \times 4 = 20$ *variables. The vertical dashed black line in the top right plot indicates the time at which* BFGS SQP *terminated while the horizontal dashed black line indicates the value of* BFGS SQP*'s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.*

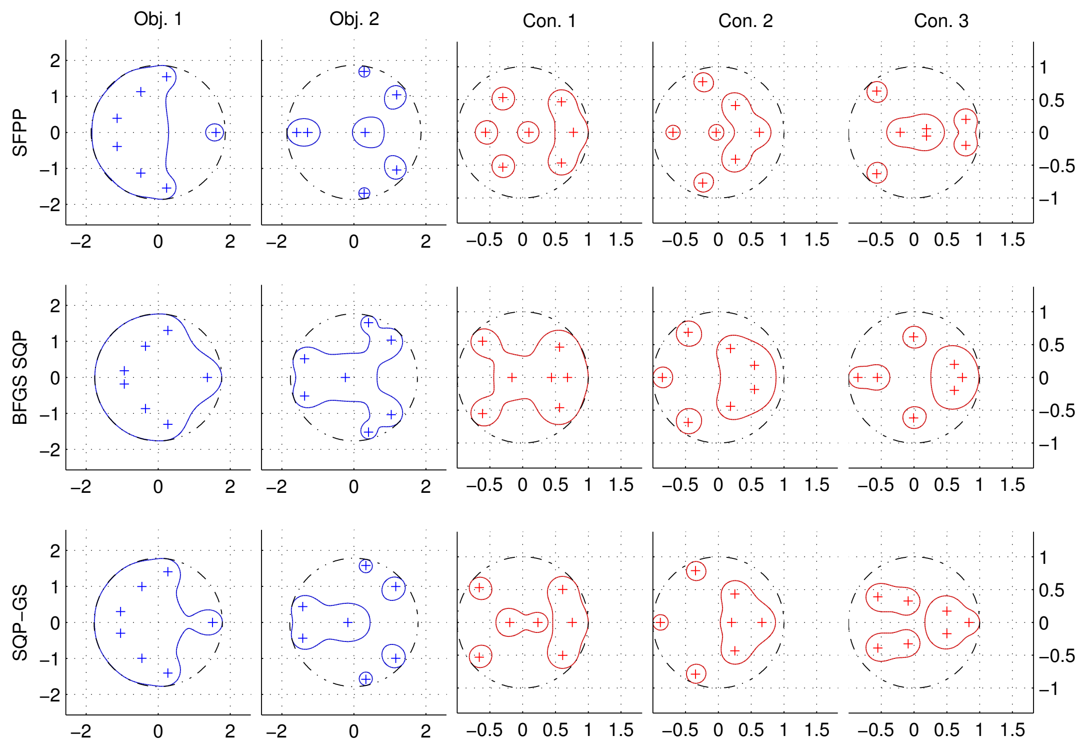

FIGURE C.4: *The three rows show the final pseudospectral configurations of the three controllers found by* SFPP (*top*)*,* BFGS SQP (*middle*)*, and* SQP-GS (*bottom*) *respectively a randomly generated example of dimension $n = 7$ comprised of two plants in the objective and three in the constraint and where the controller matrix has $mp = 5 \times 4 = 20$ variables. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm's controller.*

BIBLIOGRAPHY

[Ait26] A. Aitken. On Bernoulli's numerical solution of algebraic equations. In *Proceedings of the Royal Society of Edinburgh*, pages 289–305, 1926.

[BB90] S. Boyd and V. Balakrishnan. A regularity result for the singular values of a transfer matrix and a quadratically convergent algorithm for computing its $\mathbf{L}_\infty$-norm. *Systems Control Lett.*, 15(1):1–7, 1990.

[BHLO06] 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 *Fifth IFAC Symposium on Robust Control Design, Toulouse*, 2006.

[BLCN12] R.H. Byrd, G. Lopez-Calva, and J. Nocedal. A line search exact penalty method using steering rules. *Math. Program.*, 133(1-2, Ser. A):39–73, 2012.

[Blo99] V. Blondel. Three problems on the decidability and complexity of stability. Open Problems in Mathematical Systems and Control Theory (V. Blondel and J.N. Tsitsiklis, eds.), pages 53–56. Springer Verlag, London, 1999.

[BLO03] J.V. Burke, A.S. Lewis, and M.L. Overton. Robust stability and a criss-cross algorithm for pseudospectra. *IMA J. Numer. Anal.*, 23:359–375, 2003.

[BLO05] J.V. Burke, A.S. Lewis, and M.L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15:751–779, 2005.

[BNW08] R.H. Byrd, J. Nocedal, and R.A. Waltz. Steering exact penalty methods for nonlinear programming. *Optim. Methods Softw.*, 23(2):197–213, 2008.

[BO01] J.V. Burke and M.L. Overton. Variational analysis of non-Lipschitz spectral functions. *Mathematical Programming*, 90:317–352, 2001.

[BS90] N.A. Bruinsma and M. Steinbuch. A fast algorithm to compute the H^{∞}-norm of a transfer function matrix. *Systems Control Letters*, 14:287–293, 1990.

[BV14] P. Benner and M. Voigt. A structured pseudospectral method for H_{∞}-norm computation of large-scale descriptor systems. *Math. Control Signals Systems*, 26(2):303–338, 2014.

[CJ76] S. Cabay and L.W. Jackson. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM J. Numer. Anal.*, 13(5):734–752, 1976.

[CO12] F.E. Curtis and M.L. Overton. A sequential quadratic programming algorithm for nonconvex, nonsmooth constrained optimization. *SIAM J. Optim.*, 22(2):474–500, 2012.

[Dai89] L. Dai. *Singular control systems*, volume 118 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Berlin, 1989.

[DES82] R.S. Dembo, S.C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982.

[DH99] T.A. Davis and W.W. Hager. Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(3):606–627 (electronic), 1999.

[DH09] T.A. Davis and W.W. Hager. Dynamic supernodes in sparse Cholesky update/downdate and triangular solves. *ACM Trans. Math. Softw.*, 35(4):27:1–27:23, February 2009.

[DM02] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John Wiley, Chichester and New York, second edition, 1987.

[FSVD14] M.A. Freitag, A. Spence, and P. Van Dooren. Calculating the H_∞-norm using the implicit determinant method. *SIAM J. Matrix Anal. Appl.*, 35(2):619–635, 2014.

[GGMO14] N. Guglielmi, M. Gürbüzbalaban, T. Mitchell, and M.L. Overton. Approximating the real structured stability radius with frobenius-bounded perturbations via spectral value sets. *In preparation*, 2014.

[GGO13] N. Guglielmi, M. Gürbüzbalaban, and M.L. Overton. Fast approximation of the H_∞ norm via optimization over spectral value sets. *SIAM Journal on Matrix Analysis and Applications*, 34(2):709–737, 2013.

[GHMO09] S. Gumussoy, D. Henrion, M. Millstone, and M.L. Overton. Multiobjective robust control with HIFOO 2.0. In *Sixth IFAC Symposium on Robust Control Design, Haifa*, 2009.

[GM94] P.R. Graves-Morris. A review of Padé methods for the acceleration of convergence of a sequence of vectors. *Applied Numerical Mathematics*, 15(2):153–174, 1994.

[GO11] N. Guglielmi and M.L. Overton. Fast algorithms for the approximation of the pseudospectral abscissa and pseudospectral radius of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1166–1192, 2011.

[GO12] M. Gürbüzbalaban and M.L. Overton. Some regularity results for the pseudospectral abscissa and pseudospectral radius of a matrix. *SIAM Journal on Optimization*, 22(2):281–285, 2012.

[GV83] G.H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1983.

[Han] HANSO (Hybrid algorithm for non-smooth optimization). `http://www.cs.nyu.edu/overton/software/hanso/`. [Online; accessed 10-September-2014].

[Hif] HIFOO (H-infinity fixed-order optimization). `http://www.cs.nyu.edu/overton/software/hifoo/`. [Online; accessed 10-September-2014].

[Hin] `hinfnorm` (H-infinity norm approximation). `http://www.cims.nyu.edu/~mert/software/hinfinity.html`. [Online; accessed 10-September-2014].

[HJ90] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1990. Corrected reprint of the 1985 original.

[HP05] D. Hinrichsen and A.J. Pritchard. *Mathematical Systems Theory I: Modelling, State Space Analysis, Stability and Robustness*. Springer, Berlin, Heidelberg and New York, 2005.

[JS00] K. Jbilou and H. Sadok. Vector extrapolation methods: Applications and numerical comparison. *Journal of Computational and Applied Mathematics*, 122(1–2 (Special issue on Numerical Analysis in the 20th Century, Vol II: Interpolation and Extrapolation)):149–165, 2000.

[Kar03] M. Karow. *Geometry of Spectral Value Sets*. PhD thesis, Universität Bremen, 2003.

[Kiw07] K.C. Kiwiel. Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 18:379–388, 2007.

[KV14] D. Kressner and B. Vandereycken. Subspace methods for computing the pseudospectral abscissa and the stability radius. *SIAM J. Matrix Anal. Appl.*, 35(1):292–313, 2014.

[Lew03] A.S. Lewis. Active sets, nonsmoothness and sensitivity. *SIAM Journal on Optimization*, 13:702–725, 2003.

[LO13] A.S. Lewis and M.L. Overton. Nonsmooth optimization via quasi-Newton methods. *Math. Program.*, 141(1-2, Ser. A):135–163, 2013.

[LS96] R.B. Lehoucq and D.C. Sorensen. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl.*, 17(4):789–821, 1996.

[MO05] E. Mengi and M.L. Overton. Algorithms for the computation of the pseudospectral radius and the numerical radius of a matrix. *IMA Journal on Numerical Analysis*, 25:648–669, 2005.

[MW09] J.J. Moré and S.M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20(1):172–191, 2009.

[NW06] J. Nocedal and S.J. Wright. *Nonlinear Optimization.* Springer, New York, second edition, 2006.

[Ove14] M.L. Overton. Stability optimization for polynomials and matrices. Nonlinear Physical Systems: Spectral Analysis, Stability and Bifurcations (O.N. Kirillov and D.E. Pelinovsky, eds.), chapter 16, pages 351–375. Wiley-ISTE, London, 2014.

[Psp] `pspa` (pseudospectral abscissa). `http://home.ku.edu.tr/~emengi/software/robuststability.html`. [Online; accessed 10-September-2014].

[QBR+95] L. Qiu, B. Bernhardsson, A. Rantzer, E.J. Davison, P.M. Young, and J.C. Doyle. A formula for computation of the real stability radius. *Automatica J. IFAC*, 31(6):879–890, 1995.

[SFS87] D.A. Smith, W.F. Ford, and A. Sidi. Extrapolation methods for vector sequences. *SIAM Review*, 29(2):199–233, 1987.

[Sps] `subspace_psa` (a subspace method for computing the pseudospectral abscissa). `http://web.math.princeton.edu/~bartv/subspace_pspa/index.html`. [Online; accessed 10-September-2014].

[TE05] L.N. Trefethen and M. Embree. *Spectra and Pseudospectra: the Behavior of Nonnormal Matrices and Operators.* Princeton University Press, 2005.

[VL85] C. Van Loan. How near is a stable matrix to an unstable matrix? Linear algebra and its role in systems theory (Brunswick, Maine, 1984), volume 47 of *Contemp. Math.*, pages 465–478. Amer. Math. Soc., Providence, RI, 1985.

[Wik14] Wikipedia. Receiver operating characteristic — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`, 2014. [Online; accessed 10-September-2014].