Optimization Methods and Software

ISSN: 1055-6788 (Print) 1029-4937 (Online) Journal homepage: https://www.tandfonline.com/loi/goms20

A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles

Frank E. Curtis, Tim Mitchell & Michael L. Overton

To cite this article: Frank E. Curtis, Tim Mitchell & Michael L. Overton (2017) A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles, Optimization Methods and Software, 32:1, 148-181, DOI: 10.1080/10556788.2016.1208749

To link to this article: https://doi.org/10.1080/10556788.2016.1208749

Published online: 20 Jul 2016.

Optimization Methods & Software, 2017
Vol. 32, No. 1, 148–181, https://doi.org/10.1080/10556788.2016.1208749

Taylor & Francis
Taylor & Francis Group

A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles

Frank E. Curtis[a], Tim Mitchell[b*] and Michael L. Overton[c]

[a]*Department of Industrial and Systems Engineering, Lehigh University, 200 West Packer Avenue, Bethlehem, PA 18015, USA;* [b]*Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg 39106, Germany;* [c]*Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA*

(*Received 23 June 2015; accepted 23 June 2016*)

We propose an algorithm for solving nonsmooth, nonconvex, constrained optimization problems as well as a new set of visualization tools for comparing the performance of optimization algorithms. Our algorithm is a sequential quadratic optimization method that employs Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton Hessian approximations and an exact penalty function whose parameter is controlled using a steering strategy. While our method has no convergence guarantees, we have found it to perform very well in practice on challenging test problems in controller design involving both locally Lipschitz and non-locally-Lipschitz objective and constraint functions with constraints that are typically active at local minimizers. In order to empirically validate and compare our method with available alternatives—on a new test set of 200 problems of varying sizes—we employ new visualization tools which we call relative minimization profiles. Such profiles are designed to simultaneously assess the relative performance of several algorithms with respect to objective quality, feasibility, and speed of progress, highlighting the trade-offs between these measures when comparing algorithm performance.

Keywords: nonconvex optimization; nonsmooth optimization; constrained optimization; sequential quadratic optimization; exact penalty methods; benchmarking; performance profiles; computational budget

1. Introduction

Consider inequality constrained optimization problems of the form

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad c_i(x) \leq 0, \quad i \in \{1,\ldots,p\}. \tag{1}$$

We propose an algorithm for solving problems of this type in which the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and constraint function $c : \mathbb{R}^n \to \mathbb{R}^p$ may be nonsmooth and nonconvex. We merely presume that the functions are continuously differentiable almost everywhere, i.e. that $f(\cdot)$ and $c_i(\cdot)$ are only nonsmooth on sets of measure zero. Our method is also applicable to solve problems that have equality constraints, but for a clearer and more concise exposition, we consider only the inequality constrained problem (1).

*Corresponding author. Email: mitchell@mpi-magdeburg.mpg.de

Unconstrained nonsmooth optimization is a well-studied subject (see [[27, 31], [45, Chapter 7]]), particularly in the convex case; for more references see, for example, the literature discussions in [13,35]. The latter paper includes discussion of some methods for constrained nonsmooth optimization. However, not many direct methods have been proposed for general problems of the form (1); indeed, most resort to penalty functions or other reformulations so that unconstrained techniques can be employed. There are exceptions, such as the subgradient-based proximal bundle method for nonconvex, nonsmooth and generally constrained multiobjective minimization implemented in the code MPBNGC [36]; see [21] for corresponding theoretical results for a closely related method for unconstrained minimization. Alternatively, there is also the sequential quadratic programming (SQP) method in [13], discussed further below, which only relies upon gradient information being available, drawing inspiration from the gradient sampling (GS) technique of Burke *et al.* [9] for unconstrained nonsmooth optimization.

The unconstrained GS algorithm has convergence guarantees that hold with probability one, assuming that $f(\cdot)$ is locally Lipschitz and its level sets are bounded. In [32], it was further shown that the convergence results of GS could be strengthened by slightly modifying the algorithm, eliminating the requirement that the level sets be bounded. In practice, GS has shown to be a reliable method on many challenging nonsmooth problems and, surprisingly, this has been evident even in cases where the objective function is not locally Lipschitz, although 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 endeavour for functions and gradients that are expensive to compute, so, as a result, GS is not a viable algorithm in many applications.

Indeed, it has been strongly advocated in [35] that for unconstrained minimization of nonsmooth, nonconvex, locally Lipschitz functions, a simple Broyden-Fletcher-Goldfarb-Shanno (BFGS) method using inexact line searches is much more efficient in practice than gradient sampling, although the BFGS Hessian approximation typically becomes very ill-conditioned and no general convergence guarantees have been established. Note that the Hessian of a nonsmooth function typically is not defined at a local minimizer; however, any locally Lipschitz nonsmooth function can be viewed as a limit of increasingly ill-conditioned smooth functions. Hence, the authors argue that the ill-conditioning of the Hessian approximation is actually beneficial. They show an example indicating that, when the objective function is partly smooth in the sense of Lewis [34], BFGS seems to be able to automatically identify the U and V spaces associated with the objective function f near the minimizer, along which f varies smoothly and nonsmoothly, respectively.

For general constrained nonsmooth, nonconvex optimization problems—i.e. where $p \geq 1$ in (1)—the aforementioned sequential quadratic optimization approach employing gradient sampling (SQP-GS) was presented in [13], with guaranteed convergence to stationary points holding with probability one. This algorithm uses a BFGS approximation to define a Hessian matrix that appears in the quadratic subproblems, but in contrast to the argument of Lewis and Overton [35] regarding the benefit of ill-conditioning for the unconstrained problem, the convergence result requires enforcing upper and lower bounds on the eigenvalues of the BFGS approximations. To the best of our knowledge, this is the only method in the literature with convergence guarantees for the broad problem class represented by (1). However, its reliance on gradient sampling, both in theory and in the implementation of the algorithm, makes SQP-GS a computationally intensive method.

As a consequence, our aim in proposing a new method for solving (1) is to eschew a costly gradient sampling approach entirely and to instead find an *efficient* and *effective* extension of the BFGS approach for nonsmooth, nonconvex unconstrained optimization to the case with nonsmooth, nonconvex constraints.

One motivation for our work is the success that BFGS has had in the domain of controller design for linear dynamical systems. In particular, the BFGS algorithm is the primary

optimization method in the open-source MATLAB toolbox HIFOO (H-infinity fixed-order optimization) [10]. The toolbox designs fixed-order controllers by optimizing stability measures that generally exhibit a high degree of nonsmoothness at minimizers (a property that we discuss and illustrate in Section 4). The HIFOO toolbox has been used successfully in a wide variety of applications, including synchronization of heterogeneous multi-agent systems and networks [29,30], design of motorized gimbals that stabilize an angular motion of an optical payload around an axis [43], flight control via static-output-feedback control [50], robust observer-based fault detection and isolation [47], influence of tire damping on control of quarter-car suspensions [1], flexible aircraft lateral flight dynamic control [22–24], optimal control of aircraft with a blended wing body [25], vibration control of a fluid/plate system [44], controller design of a nose landing gear steering system [42], bilateral teleoperation for minimally invasive surgery [14], design of an aircraft controller for improved gust alleviation and passenger comfort [49], robust controller design for a proton exchange membrane fuel cell system [48], design of power systems controllers [16], and design of winding systems for elastic web materials [33].

Most of the applications of HIFOO have involved unconstrained nonsmooth optimization, but, in 2009, HIFOO was extended [19] to handle simultaneous plant optimization, where one controller is designed to control multiple systems, some of which appear in optimization objective functions and some in constraint functions, all typically nonsmooth. Such problems can be expressed as constrained nonsmooth optimization problems of the form (1) given above. Because of the prohibitive cost of using gradient sampling, the current approach taken in HIFOO to solve these problems is to apply BFGS to a nonsmooth penalty function using a sequential fixed penalty parameter (SFPP) strategy (see Section 2 for further discussion of this method). This provides one of the main motivations for the present work; we are hoping to replace the SFPP method currently implemented in HIFOO by the new method investigated in this paper. We note that methods to compute stability measures are typically iterative with a cubic cost per iteration (with the respect to the dimension of the dynamical system, not the number of optimization variables), which motivates a strong preference for optimization methods which do not require too many function and gradient evaluations.

Our contributions in this paper are threefold. The first is naturally our new algorithm, BFGS-SQP, which employs BFGS Hessian approximations within a sequential quadratic optimization algorithm and does not assume any special structure in the objective or constraints. The second is the creation of a new heterogenous test set consisting of 200 nonsmooth, nonconvex, constrained optimization problems derived from an important application in controller design. These test problems, half of which involve locally Lipschitz functions, and half non-locally-Lipschitz functions, were specifically chosen for their challenging properties; the highly nonlinear objective and constraint functions are not only expensive to compute but they also typically display a high degree of nonsmoothness at local minimizers as well, with the constraints often being active. Our third contribution is the introduction of *relative minimization profiles* (RMPs), which are new visualization tools that we employ to empirically validate our method against available competing methods (to be described later). Given that we rely upon our new benchmarking tool to justify our algorithm's performance and utility in the absence of convergence results, this latter contribution deserves further motivation.

While performance profiles of Dolan and Moré [15] are popular benchmarking tools, they require that a binary success/failure criterion be applied to each problem in a test set. However, even in convex settings, performance profiles (and thus any conclusions made from them) can be sensitive to how exactly they are created, which is of particular concern since the success/failure criteria employed must typically be chosen arbitrarily and is often sensitive to the choice of each method's stopping parameters, difficulties which are discussed in detail in [2–4]. The additional considerations of nonconvex, nonsmooth constrained problems only intensify the difficulties of implementing a fair and informative benchmark. Quality of objective minimization, attaining

feasibility, and rate of progress with respect to computation are all important performance metrics, but they are often at odds with one another. Furthermore, their respective importance may vary for different problem classes and from user to user. Though not specifically intended for constrained and/or nonconvex optimization benchmarking, the data profiles of Moré and Wild [39] address some of the aforementioned issues, in part as they can be used as complementary tools in conjunction with performance profiles. Data profiles highlight the relative rates of progress of algorithms in attaining pre-specified levels of accuracy, but such success/failure criteria are again subject to the same pitfalls discussed above. In addition, accuracy in attaining some target success value is an ambiguous concept for nonconvex problems.

RMPs attempt to address these benchmark challenges arising in heterogenous sets of nonconvex and/or constrained problems, smooth or nonsmooth, in two key and related ways. First, RMPs permit a new *unified* visualization tool that can allow one to *simultaneously* assess the relative performance differences of competing methods with respect to objective quality, feasibility, and speed of progress, while only requiring one or a few RMP plots. Second, RMPs highlight the relative performance differences and trade-offs of methods as functions of the parameters of the RMPs themselves, *decoupled from the stopping parameters of each method*. This is done by observing the transient behaviours of each algorithm through the properties of their computed iterates.

The paper is organized as follows. In the next section, we establish a prerequisite penalty parameter approach for constrained optimization. In Section 3, we review a steering strategy for updating the penalty parameter to promote progress towards feasibility at every iteration, which is a major component for our new method. In Section 4, we present two challenging problem classes arising in controller design: one involving non-locally-Lipschitz and one involving locally Lipschitz functions. In Section 5, we introduce and further motivate our new visualization technique to compare algorithms using RMPs, and in Section 6 we use these profiles to compare our new method with competing methods on our test set comprising 200 problems of the forms introduced in Section 4. Concluding remarks are provided in Section 7 and additional illustrative examples are provided in the appendix.

2. An exact penalty function approach

We use the following notation: $\mathbb{R}^+$ denotes the set of nonnegative real numbers, $\mathbb{R}^{++}$ denotes the set of strictly positive real numbers, $\mathbb{Z}^+$ denotes the set of nonnegative integers, and e denotes a vector of ones whose length is determined by the context in which it appears.

Consider the exact nonsmooth penalty function

$$\phi(x;\mu) = \mu f(x) + v(x), \tag{2}$$

where $\mu \in \mathbb{R}^{++}$ is a penalty parameter and the function $v : \mathbb{R}^n \to \mathbb{R}$, measuring total violation cost over the constraints, is defined by

$$v(x) = \|\max\{c(x), 0\}\|_1 = \sum_{i\in\mathcal{P}_x} c_i(x) \quad \text{where } \mathcal{P}_x = \{i \in \{1,\dots,p\} : c_i(x) > 0\}. \tag{3}$$

At any x such that $c_i(x) \neq 0$ for all $i \in \{1,\dots,p\}$, the gradient of $\phi(\cdot;\mu)$ is

$$\nabla\phi(x;\mu) = \mu\nabla f(x) + \sum_{i\in\mathcal{P}_x} \nabla c_i(x); \tag{4}$$

however, the penalty function is in general nonsmooth, even if the objective and constraint functions are smooth. Thus, if we aim to solve constrained optimization problems by optimizing an

exact penalty function, we must consider optimization methods that are effective for nonsmooth functions. If we happen to know, *a priori*, an acceptable value for the penalty parameter μ such that minimizers of (2) correspond to feasible minimizers of (1) [6,7], then a straightforward BFGS method can be a practical approach. In such a method, given an iterate x_k at which the penalty function is differentiable, the search direction d_k is calculated by solving

$$\min_{d \in \mathbb{R}^n} q(d; x_k, \mu), \tag{5}$$

where

$$\begin{aligned} q(d; x_k, \mu) &:= \phi(x_k; \mu) + \nabla \phi(x_k; \mu)^\mathsf{T} d + \frac{1}{2} d^\mathsf{T} H_k d \\ &= \mu f(x_k) + \sum_{i \in \mathcal{P}_{x_k}} c_i(x_k) + \left[\mu \nabla f(x_k) + \sum_{i \in \mathcal{P}_{x_k}} \nabla c_i(x_k) \right]^\mathsf{T} d + \frac{1}{2} d^\mathsf{T} H_k d \end{aligned} \tag{6}$$

and H_k is a BFGS approximation to the Hessian of (2) at x_k.[1] Unfortunately, however, 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 (1) if the penalty parameter weighting the objective is set too high.

One might consider a simple strategy of using an SFPP restarting scheme with BFGS; i.e. one may consider iteratively lowering μ and restarting BFGS repeatedly until a feasible solution has been found. An immediate pertinent issue with such an approach is how accurately BFGS should attempt to minimize (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 only lowering the penalty parameter when it has been demonstrated to be too high versus lowering the penalty parameter too aggressively and/or frequently. On one hand, confirming that the penalty parameter is set too high often entails the potentially costly process of allowing the algorithm to first converge to an infeasible point. On the other hand, lowering the penalty parameter unnecessarily comes with the risk of 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 (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 unbounded below off the feasible set.

3. A steering strategy

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 Byrd *et al.* [11,12]. 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 (5) with an alternative search direction computed by a penalty-SQP method [17], which is obtained via solving the quadratic problem (QP)

$$\begin{aligned} \min_{d \in \mathbb{R}^n,\ s \in \mathbb{R}^p} \quad & \mu(f(x_k) + \nabla f(x_k)^\mathsf{T} d) + e^\mathsf{T} s + \frac{1}{2} d^\mathsf{T} H_k d \\ \text{s.t.} \quad & c(x_k) + \nabla c(x_k)^\mathsf{T} d \leq s, \quad s \geq 0, \end{aligned} \tag{7}$$

where the corresponding dual is given by

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^p} \quad & \mu f(x_k) + c(x_k)^\mathsf{T}\lambda - \frac{1}{2}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda)^\mathsf{T} H_k^{-1}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda) \\ \text{s.t.} \quad & 0 \leq \lambda \leq e, \end{aligned} \tag{8}$$

$s \in \mathbb{R}^p$ is a vector of slack variables and $\lambda \in \mathbb{R}^p$ is a vector of dual variables. The primal solution component d_k can be recovered from the dual solution λ_k via the relationship

$$d_k = -H_k^{-1}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda_k). \tag{9}$$

Note that when $\mu = 1$ and there are no constraints (i.e. when $p = 0$) Equation (9) yields the standard BFGS search direction. In the presence of constraints, the resulting d_k computed from (7) (or by solving (8) and employing (9)) provides a descent direction for (2) at x_k with the current penalty parameter μ.

The search direction computed in this manner can be viewed as balancing the two (sometimes opposing) goals of minimizing the objective and pushing towards a feasible solution, the latter of which can be measured by the magnitude of the linear model of constraint violation, i.e.

$$l(d; x_k) := \| \max\{c(x_k) + \nabla c(x_k)^\mathsf{T} d, 0\} \|_1 \tag{10}$$

at x_k given a direction d. Observe that the magnitude of this quantity relates to that of $e^\mathsf{T} s$ in the solution to (7). Our approach for updating the penalty parameter, based on the techniques in [11,12], proceeds as follows. Let the reduction in the linear model of constraint violation given in (10) at the current iterate x_k and for any search direction d be defined as

$$\begin{aligned} l_\delta(d; x_k) &:= l(0; x_k) - l(d; x_k) \\ &= v(x_k) - \| \max\{c(x_k) + \nabla c(x_k)^\mathsf{T} d, 0\} \|_1. \end{aligned} \tag{11}$$

For any search direction d at x_k, (11) predicts how much progress towards feasibility d may make. The basic tenet of the steering strategy defined in Procedure 1 is to 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) 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) produces a search direction satisfactorily balanced in terms of progress towards the feasible set and minimizing the objective.

The benefits of the steering strategy in Procedure 1 are that the penalty parameter can be dynamically set at every iteration, where the amount that it may be reduced is determined by how difficult it appears to be to promote progress towards the feasible set from the current iterate. In contrast, with the simple fixed penalty parameter restarting scheme SFPP, one must either wait for BFGS to converge to 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 adjust the penalty parameter anyway, without knowing whether it is too high or not. Moreover, empirical evidence has shown that a steering strategy such as that in Procedure 1 decreases the

Procedure 1 $[d_k, \mu_{\text{new}}] =$ **sqp_steering_strategy**(x_k, H_k, μ)

Input:
Current iterate x_k and BFGS Hessian approximation H_k
Current value of the penalty parameter μ
Constants:
Values $c_v \in (0, 1)$ and $c_\mu \in (0, 1)$
Output:
Search direction d_k
Penalty parameter $\mu_{\text{new}} \in (0, \mu]$

1: Solve QP (8) using $\mu_{\text{new}} := \mu$ to obtain search direction d_k from (9)
2: **if** $l_\delta(d_k; x_k) < c_v v(x_k)$ **then**
3: Solve (8) using $\mu = 0$ to obtain reference direction $\tilde{d}_k$ from (9)
4: **while** $l_\delta(d_k; x_k) < c_v l_\delta(\tilde{d}_k; x_k)$ **do**
5: $\mu_{\text{new}} := c_\mu \mu_{\text{new}}$
6: Solve QP (8) using $\mu := \mu_{\text{new}}$ to obtain search direction d_k from (9)
7: **end while**
8: **end if**

NOTE: *The constant c_v specifies the fraction of either total violation $v(x_k)$ or predicted violation reduction $l_\delta(\tilde{d}_k; x_k)$ for the reference direction $\tilde{d}_k$ that is deemed to be an acceptable amount of predicted progress towards feasibility for the search direction d_k. Otherwise, the constant c_μ is used as the factor by which to iteratively reduce the penalty parameter μ_{new} and correspondingly compute new trial search directions. In practice, the while loop is terminated after a fixed number of iterations, if its termination condition continues not to be met, and the last search direction and penalty parameter are accepted regardless.*

likelihood of divergence ensuing in the case that $f(\cdot)$ is unbounded below (but is bounded below in the feasible region).

As the penalty function is generally nonsmooth at minimizers, we cannot expect the norm of its gradient to decrease as iterates approach a minimizer. 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 [35] for the unconstrained problem, consider the l most recent iterates of the algorithm that are considered 'close' in some sense of distance and define

$$G := [\nabla f(x_{k+1-l}) \cdots \nabla f(x_k)]$$
$$\text{and} \quad J_i := [\nabla c_i(x_{k+1-l}) \cdots \nabla c_i(x_k)], \quad i \in \{1, \ldots, p\}.$$

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). However, we augment the QP with previously computed gradient information (from G and the J_i's) in order to capture changes in the problem functions in a neighbourhood 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 neighbourhood of a given point. Here, however, we are reusing the gradients of the objective and constraints of previously computed points $\{x_{k+1-l}, \ldots, x_k\}$ to form G and the J_i's, provided that this set of previous iterates is sufficiently close to x_k. Note that, in practice, a set of l 'close' iterates can be maintained by purging the set any time the norm of the most recent step is sufficiently large, e.g. if it is greater than some user-provided tolerance. If the solution of the resulting QP is sufficiently small in norm, then we have reason to believe that

we are in a small neighbourhood of a stationary point for the constrained optimization problem. For consistency with (7), we employ the H_k-norm, which leads to the dual formulation, using $\lambda = [\lambda_1{}^{\mathsf{T}}, \ldots, \lambda_p{}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{pl}$ with $\lambda_i \in \mathbb{R}^l$:

$$\begin{aligned} \max_{\sigma \in \mathbb{R}^l, \lambda \in \mathbb{R}^{pl}} \quad & \sum_{i=1}^{p} c_i(x_k) e^{\mathsf{T}} \lambda_i - \frac{1}{2} \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}^{\mathsf{T}} [G, J_1, \ldots, J_p]^{\mathsf{T}} H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix} \\ \text{s.t.} \quad & 0 \leq \lambda_i \leq e, \quad e^{\mathsf{T}} \sigma = \mu, \ \ \sigma \geq 0. \end{aligned} \tag{12}$$

The termination condition is then that $d_\diamond$ is sufficiently small in norm, where

$$d_\diamond = H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}. \tag{13}$$

Overall, motivated by [13, Equation (2.7)], this measure is reasonable since $d_\diamond$ is the minimizer of a piecewise quadratic model of the penalty function about x_k. The minimizer of this model is given by the solution of a QP similar to (7), but augmented in the sense that $\nabla f(x_k)^{\mathsf{T}} d$ and $\nabla c_i(x_k)^{\mathsf{T}} d$ for $i \in \{1, \ldots, p\}$ are, respectively, replaced by the maxima of terms involving the gradients in G and the J_i's (see [13, Equation (2.6)]). In our setting, the minimizer (13) is recovered by solving the QP's dual, given by (12).

We may now present our algorithm BFGS-SQP, shown in pseudocode in Procedure 2, where the search direction is calculated using the SQP-based steering strategy of Procedure 1. Following Lewis and Overton [35] for unconstrained optimization using BFGS, we also make use of an inexact Armijo-Wolfe line search to determine the length of the actual step taken along the chosen search direction d_k. The line search returns the next iterate of BFGS-SQP, that is, the point x_{k+1} which satisfies the Armijo and weak Wolfe conditions for the penalty function, along with the corresponding values of the penalty function, its gradient and the constraint violation amount at x_{k+1}. Finally, BFGS-SQP is terminated once both $\|d_\diamond\|_2 \leq \tau$ holds for some fixed user-provided tolerance $\tau \in \mathbb{R}^{++}$, indicating that stationarity has been achieved, and the constraints at this point are also satisfied to the desired accuracy, indicating that this stationary point is also a feasible solution. We make no claims that BFGS-SQP has theoretical convergence guarantees to such points, though we believe that the results of our numerical experiments justify the algorithmic choices, which we have made in its design.

Remark 3.1 For brevity, we omit the primal form of (12) and instead refer to [13, Equation (2.6)], where the problem is stated in terms of sample points, not previous iterates.

In the remainder of the paper, we compare the performance of an implementation of BFGS-SQP against the performance of three other methods:[2] the SFPP and SQP-GS methods already described, and, in addition, Sparse Nonlinear OPTimizer (SNOPT) [18], a highly regarded general purpose code for nonlinearly constrained optimization. Although SNOPT is not intended for solving nonsmooth problems, it is worth noting that neither BFGS nor the SQP steering strategy we employ were originally intended for nonsmooth optimization either. As SNOPT is also a quasi-Newton SQP method, it is reasonable to ask whether it is effective on nonsmooth problems and, if so, how effective it is compared to our new method. Consequently, in order to empirically validate BFGS-SQP, we include SNOPT in addition to SFPP and SQP-GS in our set of competing alternatives for constrained nonsmooth optimization. For more details on the implementations and versions of these codes, see Section 6.1.

Procedure 2 $[x_\star, f_\star, v_\star] =$ **bfgs_sqp**$(f(\cdot), c(\cdot), x_0, \mu_0)$

Input:

Objective $f : \mathbb{R}^n \to \mathbb{R}$ and inequality constraints $c : \mathbb{R}^n \to \mathbb{R}^p$ given as $f(\cdot)$ and $c(\cdot)$
Initial starting point $x_0 \in \mathbb{R}^n$ and penalty parameter $\mu_0 \in \mathbb{R}^{++}$

Constants:

Constants: positive stopping tolerances $\tau_\diamond$ (stationarity) and τ_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 (2) using $f(\cdot)$ and $c(\cdot)$
Set $\nabla\phi(\cdot)$ and $v(\cdot)$ as the associated gradient (4) and violation function (3)
Evaluate $\phi_0 := \phi(x_0; \mu)$, $\nabla\phi_0 := \nabla\phi(x_0; \mu)$, and $v_0 := v(x_0)$
for $k = 0, 1, 2, \ldots$ **do**
  $[d_k, \hat{\mu}] :=$ **sqp_steering_strategy**(x_k, H_k, μ)
  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
  $[x_{k+1}, \phi_{k+1}, \nabla\phi_{k+1}, v_{k+1}] :=$ **inexact_linesearch**$(x_k, \phi_k, \nabla\phi_k, d_k, \phi(\cdot), \nabla\phi(\cdot))$
  Compute $d_\diamond$ via (12) and (13)
  if $\|d_\diamond\|_2 < \tau_\diamond$ and $v_{k+1} < \tau_v$ **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 we refer to* Lewis and Overton [35] *and* [40 pages 140–143]*, respectively. If the line search fails, then the algorithm is terminated.*

4. Nonsmooth, nonconvex, constrained optimization examples

4.1 *Static-output-feedback controller design*

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

$$x_{k+1} = Ax_k + Bw_k,$$
$$z_k = Cx_k$$

and the associated *static-output-feedback plant* [5,10,41]

$$A + BXC,$$

where the matrices $A \in \mathbb{R}^{N,N}$, $B \in \mathbb{R}^{N,M}$, and $C \in \mathbb{R}^{P,N}$ are given, $X \in \mathbb{R}^{M,P}$ is an embedded variable controller matrix, $w_k \in \mathbb{R}^M$ is the control input, and $z_k \in \mathbb{R}^P$ is the output. A well-known and

an important problem is that 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.

4.2 *Spectral radius optimization*

Our first test problem requires the following definition.

DEFINITION 4.1 *The* spectral radius *of a matrix* $A \in \mathbb{C}^{N,N}$ *is defined as*

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

where the spectrum, or set of eigenvalues of A, is

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

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

Remark 4.2 The discrete-time dynamical system $x_{k+1} = Ax_k$ is asymptotically stable—i.e. $x_k \to 0$ as $k \to \infty$ for any $x_0 \in \mathbb{R}^N$—if and only if $\rho(A) < 1$. Hence, our usage of the term 'stable' is somewhat nonstandard since if A has a multiple eigenvalue with a Jordan block and with modulus one, $\{x_k\}$ may diverge. However, it is convenient for the purposes of optimization to work with non-strict inequalities and closed feasible regions, and in practice one can always change 1 to a suitable number slightly less than 1.

Remark 4.3 Although the spectral radius function $\rho(\cdot)$ is nonsmooth at matrices with more than one eigenvalue attaining the maximum modulus, and non-Lipschitz at matrices with multiple (coinciding) eigenvalues attaining the maximum modulus [8], it is continuously differentiable almost everywhere; as such, it is a suitable challenging function for comparing gradient-based nonsmooth, nonconvex optimization methods.

We now consider the nonconvex, nonsmooth, and constrained optimization problem

$$\begin{aligned} \min_{X \in \mathbb{R}^{M,P}} \quad & \max\{\rho(A_i + B_i X C_i) : i \in \{p+1, \ldots, p+q\}\} \\ \text{s.t.} \quad & \rho(A_i + B_i X C_i) \leq 1, \quad i \in \{1, \ldots, p\} \end{aligned} \tag{14}$$

where each static-output-feedback plant $A_i + B_i X C_i$ is defined by the 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 $p+q$ plants. The goal is to compute 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 constraints remain stable.

Remark 4.4 This problem is a challenging nonsmooth, nonconvex optimization problem even when there are no constraints and $q = 1$, in which case the objective is simply the spectral radius of a single static-output-feedback plant. Problems of this form were investigated in [41]. Because the matrices are real, the eigenvalues are either real or occur in complex conjugate pairs. It was observed in [41] that at local minimizers of the spectral radius, typically several eigenvalues (real and/or conjugate pairs) of the static-output-feedback plant have moduli that attain the spectral

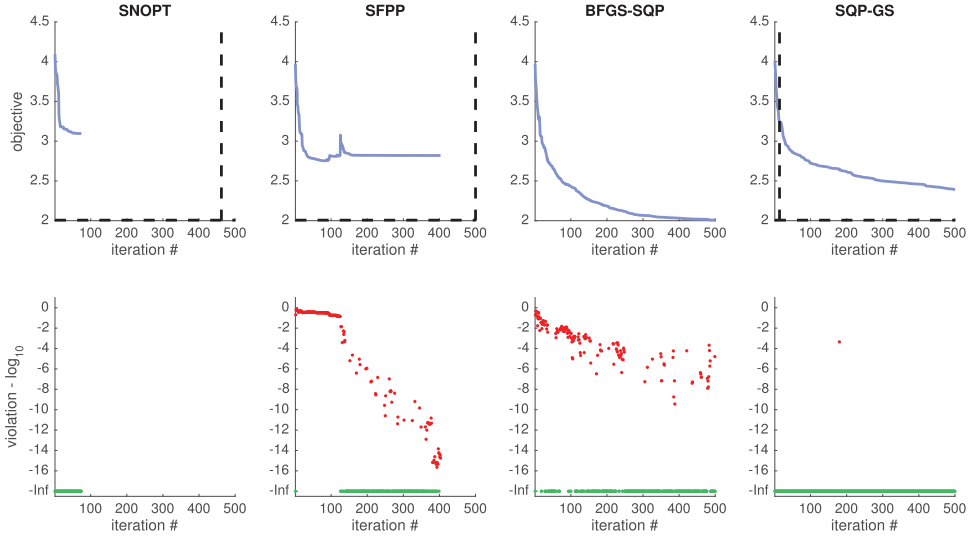

Figure 1. The plots in the top row track the value of the spectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 13$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $MP = 23 \times 2 = 46$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP's total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

radius. We can consider the number of such 'active' real or conjugate pairs of eigenvalues, minus one, as a measure of nonsmoothness at the solution: we call this the 'number of activities'. When a complex conjugate pair of active eigenvalues has multiplicity higher than one, that is two or more conjugate pairs of active eigenvalues coincide with each other, the number of activities and hence the degree of nonsmoothness is higher still, since both the real and imaginary parts coincide, not only the modulus. For a discussion of related constrained problems, see [19].

In Figure 1, we compare SNOPT, SFPP, BFGS-SQP, and SQP-GS for designing a controller for (14) on a randomly generated example comprised of three plants in the objective and one plant in the constraint, with the controller matrix initially set to zero, a feasible starting point. (See Section 6.1 for more details on the experimental set-up.) Though SNOPT's iterates are all feasible, we see that it achieves the least minimization of the objective amongst the four methods. SFPP is able to provide moderately better minimization quality compared to SNOPT, but SFPP struggled for well over 100 iterations before any real progress towards feasibility was made. Despite SFPP's penalty parameter update schedule every 20th iterate, the penalty parameter simply remained too high for too long. In contrast, BFGS-SQP immediately made progress towards finding feasible points and minimizing the objective, even as it often encountered infeasible regions as it progressed. Interestingly, SQP-GS maintained feasibility at almost every iterate, and while it and BFGS-SQP minimized the objective significantly more than either SNOPT or SFPP did, SQP-GS's slower progress per iteration is not only outclassed by BFGS-SQP's fast rate of convergence, but, we also see that CPU time per iteration is well over an order of magnitude larger for SQP-GS compared to BFGS-SQP. By the time BFGS-SQP has finished its 500 iterations, SQP-GS has barely begun. For this particular problem, we see that BFGS-SQP happens to reduce the objective the most of all four methods.

In Figure 2, for this same example, we show the final spectral configurations for the four controllers ultimately produced by SNOPT, SFPP, BFGS-SQP, and SQP-GS. These are all plots in

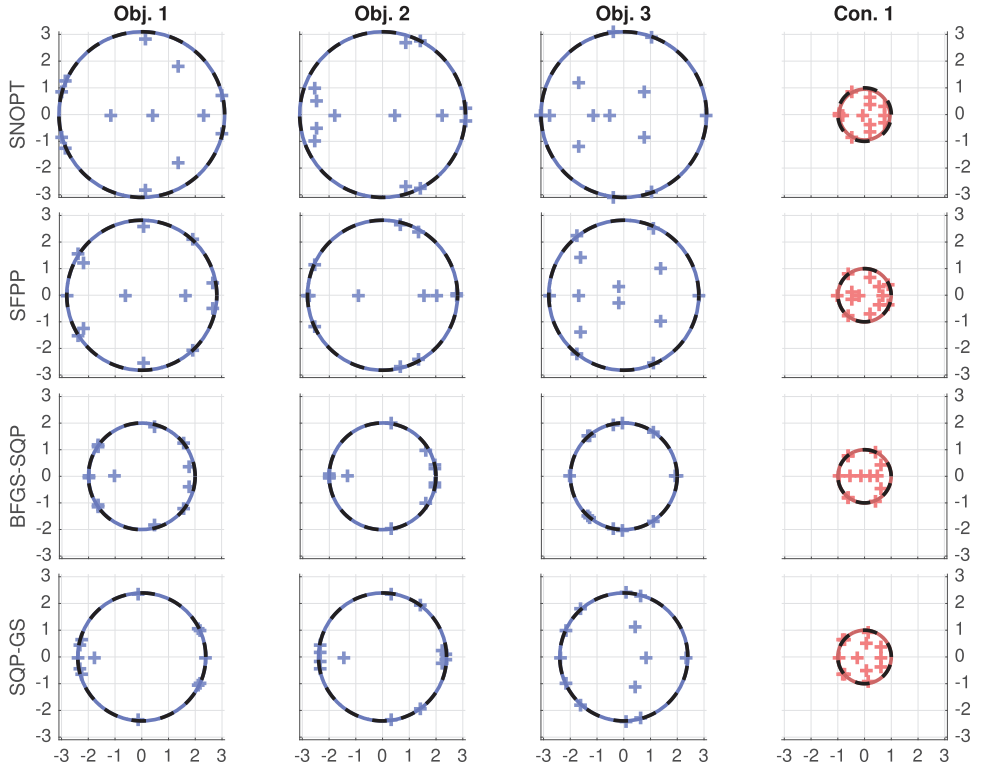

Figure 2. The four rows show the final spectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure 1. The left three columns indicate the plants in the objective while the right column indicates the single plant in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the max spectral radius of the three plants in the objective for that particular algorithm's controller. The dashed black circle on the constraint plots is the unit circle (the stability boundary). The solid lighter circles indicate the spectral radius of each plant.

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 4.4, we observe that the moduli of several eigenvalues attain the relevant spectral radius. For example, in the third plant in the objective for BFGS-SQP's controller , we see the moduli of all 13 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 cannot 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, which further increases the overall number of activities encountered at these solutions and demonstrates the inherent high degree of nonsmoothness in the optimization problem. The constraint plant also shows activities, with the moduli of several eigenvalues attaining the upper bound of one imposed by the constraint, thus demonstrating that the algorithms converged to controllers where both the objective and constraint are nonsmooth. (See Appendix A.1 for additional spectral radius examples similar to the one shown in Figures 1 and 2.)

4.3 *Pseudospectral radius optimization*

Our second test problem requires the following definition.

DEFINITION 4.5 *The* pseudospectral radius *of a matrix* $A \in \mathbb{C}^{N,N}$ *is defined as*

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

where $\rho_0(A) = \rho(A)$. *We say that* A *is stable with respect to the perturbation level* $\varepsilon \geq 0$ *if* $\rho_\varepsilon(A) \leq 1$.

Remark 4.6 The pseudospectral radius can be equivalently defined in terms of the norm of the resolvent as [46]

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

where we use the convention that $\|(A - \lambda I)^{-1}\|_2 = \infty$ when $A - \lambda I$ is singular. The inequality on the right-hand side of this equation can alternatively be expressed as imposing an upper bound ε on the smallest singular value of $A - \lambda I$.

Remark 4.7 Like the spectral radius, the pseudospectral radius is nonconvex, nonsmooth, and continuously differentiable almost everywhere. However, in contrast to the spectral radius, the pseudospectral radius is locally Lipschitz [20] and is thus potentially an easier function to optimize. For example, the known convergence rates for gradient sampling hold for minimizing the pseudospectral radius but not the spectral radius. On the other hand, the pseudospectral radius (along with its gradient, where it is differentiable) is significantly more expensive to compute than the spectral radius [37].

Remark 4.8 In contrast to the spectral radius, the pseudospectral radius can be considered a robust stability measure, in the sense that it models the case where asymptotic stability of the linear dynamical system is guaranteed under the influence of noise up to a specified amount.

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

$$\begin{aligned} \min_{X \in \mathbb{R}^{M,P}} \quad & \max\{\rho_\varepsilon(A_i + B_i X C_i) : i \in \{p+1, \ldots, p+q\}\} \\ \text{s.t.} \quad & \rho_\varepsilon(A_i + B_i X C_i) \leq 1, \quad i \in \{1, \ldots, p\}, \end{aligned} \tag{15}$$

which we may view as a locally Lipschitz regularization of (14). The goal is still 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, except here stability means that any plant must remain stable under any perturbation up to norm $\varepsilon = 10^{-1}$.

In Figure 3, we again compare SNOPT, SFPP, BFGS-SQP, and SQP-GS (using the same experimental set-up as used in Section 4.2 and described in Section 6.1) for designing a controller for (15) on a randomly generated example comprised of three plants in the objective, one in the constraint, and where the controller matrix was initially set to the zero matrix, again a feasible point. For this particular pseudospectral radius optimization problem, we again see that both SNOPT and SFPP provide the least amount of minimization of the four methods, with SNOPT appearing to stagnate before reaching a stationary point while SFPP incurred over 160 iterations before its 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 either SNOPT or SFPP once again. Interestingly, as on the spectral radius problem, BFGS-SQP found a much better controller

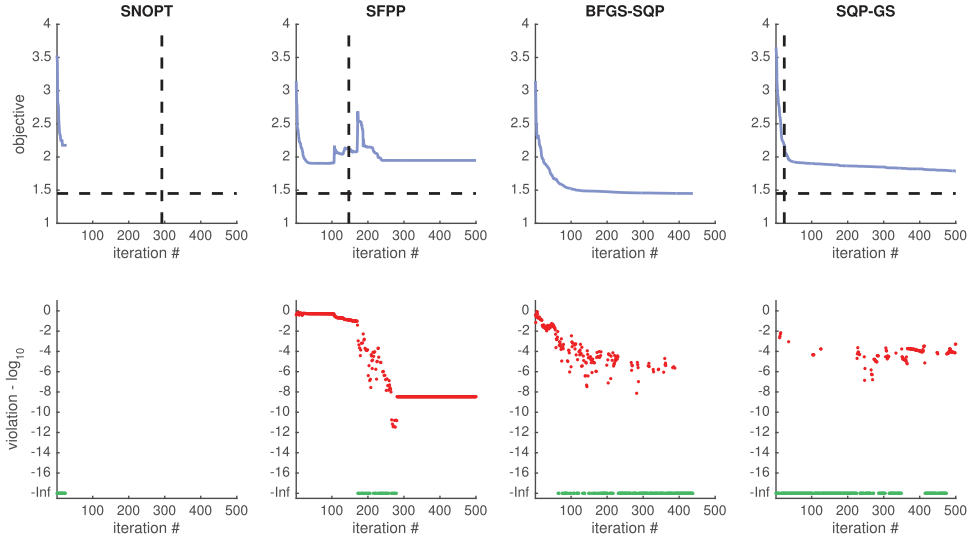

Figure 3. The plots in the top row track the value of the pseudospectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 5$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $MP = 13 \times 2 = 26$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP's total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

than SQP-GS, even though SQP-GS's convergence results hold for this particular problem 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 than BFGS-SQP with respect to CPU time.

In Figure 4, for this particular example, we show the final pseudospectral configurations[3] of the four controllers ultimately produced by SNOPT, SFPP, BFGS-SQP, and SQP-GS. For almost all the plants, we see that the resulting pseudospectral configurations show that there are multiple nonsmoothness activities, in the sense that the optimal pseudospectral radius value is attained at more than one point in the closed upper half-plane. Furthermore, we observe additional activities due to the fact that pseudospectral radii for the three plants in the objectives have approximately the same value. The constraint plant for most of the controllers also shows activities, with the pseudospectral radius attaining the upper bound value of one at more than one point in the closed upper half-plane. Thus, we see strong evidence that both the objective and the constraint are indeed nonsmooth at the solutions found by each algorithm. (See Appendix A.2 for additional pseudospectral radius examples similar to the one shown here in Figures 3 and 4.)

5. Comparing nonsmooth, nonconvex, and constrained optimization algorithms

For comparing codes for solving convex optimization problems, a popular visualization tool is a performance profile [15], which is often used to simultaneously depict each code's efficiency and reliability on a test set of problems. For such a profile, made for a given performance measure related to solving each problem, whether it be total CPU time, number of function evaluations, etc., a code's efficiency is measured on a per-problem basis by comparing its performance against the performance of the 'best' code observed in the comparison for that problem, meaning the

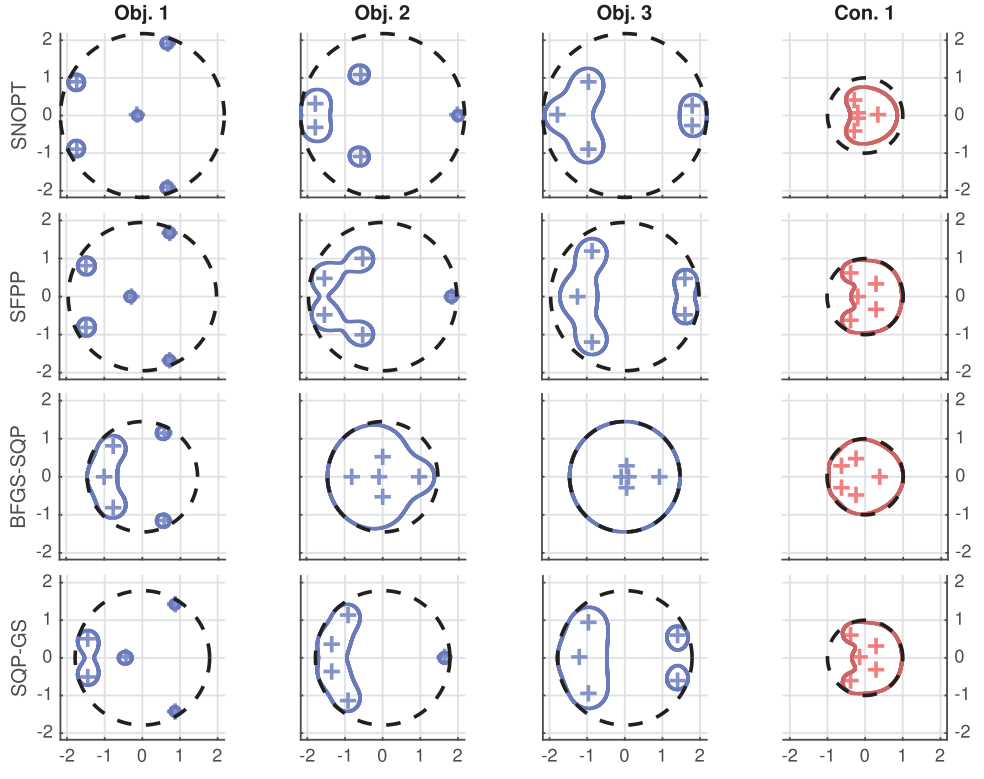

Figure 4. The four rows show the final pseudospectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure 3. The left three columns indicate the plants in the objective while the right column indicates the single plant in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm's controller. The dashed black circle on the constraint plots is the unit circle (the stability boundary). The solid lighter circles indicate the pseudospectral boundaries of each plant.

code which successfully solves that problem the fastest. Reliability, on the other hand, is measured in terms of the percentage of problems on which a given code is deemed to have been successful. Mathematically, if the plot associated with a code passes through the point $(a, b/100)$ in a performance profile, then that code solves $b\%$ of the problems in the test set when its performance measure of interest (say CPU time) is restricted to be less than or equal to a times that of the performance measure of the 'best' code for each problem.

The ingenuity of performance profiles is that they capture such information about relative efficiency and reliability in an easily read plot, while additionally ensuring that the illustrated performance for a particular code is not skewed too much by its performance on one (or a small subset) of problems. For example, if a code were to perform disastrously on one problem, then an assessment based on overall running time to complete the test set could cause the code to appear inferior, even if it is better than its competition on the remaining problems in the test set. Performance profiles, by considering performance on a per-problem basis, allow a reader to distinguish such behaviour.

However, by only focusing on whether or not a solver has *solved* each problem, performance profiles do not distinguish the transient behaviours of each algorithm. Indeed, this deficiency was a key motivator for the introduction of data profiles [39], where the authors explicitly state that 'the percentage of problems that can be solved (for a given tolerance τ) with a given number of function evaluations ... is essential to users with expensive optimization problems'. For

particularly expensive optimization problems, permitting an algorithm to run until its convergence criteria are met may be neither computationally possible nor efficient, especially if an approximate solution will suffice. Even if the optimization problems are not expensive, algorithms which generally make good initial progress but then plateau with little to no further progress until they terminate will be correspondingly, and perhaps unfairly, penalized in a performance profile benchmark in a nontransparent fashion. In such cases, it is often unclear whether plateauing performance is a characteristic of the algorithm in question or merely the result of the specific parameters chosen by the user (or by default). By plotting the percentage of problems that a method has solved, to some predetermined level of accuracy, as a function of the number of functions/gradients evaluated, data profiles elucidate how the relative rankings of methods may change as the computational budget varies. Furthermore, unlike performance profiles where the depicted performance of algorithms is influenced by their respective stopping parameters, data profiles depict the performance of algorithms with respect to a consistent computational budget metric, which significantly lessens the influence of the stopping parameters in the benchmark. Lastly, by creating multiple data profiles for different levels of acceptable inaccuracy, one can readily assess which algorithms are fastest for obtaining only approximate solutions versus ones which are fastest for resolving accurate solutions to a high precision.

Nonetheless, despite the high utility of both performance and data profiles, either of which can be used stand-alone or in conjunction with each other, implementing an informative and *fair* benchmark remains a challenging prospect, even in convex, unconstrained settings. As mentioned in Section 1, not only does the choice of stopping parameters and success/failure criteria influence the resulting performance profiles, but furthermore, these choices must often be made arbitrarily. Data profiles help alleviate some of these issues, but one must still define success in terms of what is considered a satisfactory solution in terms of some maximum permissible level of inaccuracy.

When solving nonconvex problems with gradient-based local search methods, one can neither expect that codes will find global minimizers nor that they will find the same local minimizers for a given problem. Indeed, the codes might not find 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 approximate stationary point. While one could define the binary success/failure criteria needed for performance and data profiles as (sufficiently) reaching any stationary point, it is questionable whether such a metric would be either fair or informative for scenarios where not all, or perhaps even none, of the algorithms have convergence theory, such as is the case for the benchmark we show in Section 6.2 using only non-locally-Lipschitz problems of the form described in Section 4.2. To this point, for the consideration of a success/failure criteria for constrained optimization, in [4, Chapter 14] the authors make the case that finding a feasible point that is (approximately) stationary or finding a feasible point which obtains the lowest value of the objective both have reasonable rationale and the priorities of users and optimization method designers may be quite different. Indeed, in [2,3] it is advocated to employ performance profiles generated using final objective values (compared to some best found per problem), but this approach still has the pitfalls of having to prescribe some binary success/failure criterion, which we seek to avoid doing here. 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. Furthermore, there is always an ambiguity as to whether the time to run an optimization code is attributable to the algorithm itself, its implementation, or to the particular ease or difficulty inherent in the problem itself for finding any given candidate solution. In the case of nonsmooth, nonconvex, and constrained optimization, where the key performance metrics are quality of objective minimization, attaining feasibility, and to what level those goals can be met for given computational resources, there is no single agreeable definition of success. The above

challenges and ambiguities, which are not addressed by performance or data profiles, suggest that an informative and fair benchmark should allow for *different interpretations of success/failure*, depending on the priorities of the reader, and evaluators should aim not to define success but present as much relevant benchmark data as possible in a concise and intuitive manner. We aim to address these challenges.

5.1 *Relative minimization profiles*

Given a test set of problems, let us initially consider evaluating algorithms in term of quality (amount) of objective minimization achieved and feasibility of iterates, without focusing on cost, be it running time, number of function evaluations, or memory use. We propose producing a plot, which we call an *RMP*, that relates the percentage of feasible iterates[4] that each algorithm found over the entire data set (measured on the y-axis) with the amount of per-problem objective minimization achieved by the individual algorithms, measured as relative differences to some prescribed per-problem target values Ω (measured on the x-axis).

To define an RMP precisely, we begin with the following definitions:

$$\begin{aligned}
\mathcal{M} &:= \text{set of methods to compare}\\
\mathcal{T} &:= \text{ordered set of test problems}\\
\omega_i &:= \text{target objective value for problem } p_i \in \mathcal{T}\\
\Omega &:= \{\omega_i : p_i \in \mathcal{T}\}\\
f_i(x) &:= \text{objective function for problem } p_i \in \mathcal{T}\\
v_i(x) &:= \text{violation function (recall (3)) for problem } p_i \in \mathcal{T}\\
\{x_k\}_i^m &:= \text{iterates produced by method } m \in \mathcal{M} \text{ on problem } p_i \in \mathcal{T}.
\end{aligned}$$

For method m on problem $p_i \in \mathcal{T}$, we define its *best computed objective value over all feasible iterates*, in terms of some violation tolerance $\tau_v \in \mathbb{R}^+$, as

$$f_i^m(\tau_v) := \min\{f_i(x) : x \in \{x_k\}_i^m,\ v_i(x) \leq \tau_v\}, \tag{16}$$

with the understanding that the value is ∞ if the set is empty. Furthermore, we define the following relative residual function and its associated indicator function:[5]

$$r(\varphi, \tilde{\varphi}) := \begin{cases} \infty & \text{if } \varphi = \infty \text{ or } \tilde{\varphi} = \infty \\ \left| \dfrac{\varphi - \tilde{\varphi}}{\varphi} \right| & \text{otherwise,} \end{cases}$$

$$\mathbb{1}_r(\varphi, \tilde{\varphi}, \gamma) := \begin{cases} 1 & \text{if } r(\varphi, \tilde{\varphi}) \leq \gamma \\ 0 & \text{otherwise.} \end{cases}$$

We may now formally define a *RMP curve*.

Definition 5.1 *Given a fixed violation tolerance $\tau_v \in \mathbb{R}^+$, per-problem target objective value set $\Omega := \{\omega_i\}$, and method $m \in \mathcal{M}$, a* relative minimization profile curve $r^m_{\Omega,\tau_v} : \mathbb{R}^+ \to [0, 1]$ *is that given by*

$$r^m_{\Omega,\tau_v}(\gamma) := \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \mathbb{1}_r(\omega_i, f_i^m(\tau_v), \gamma), \tag{17}$$

where γ specifies the maximum relative difference allowed to the target set values Ω. (Note that $r^m_{\Omega,\tau_v}(\gamma)$ is well defined at $\gamma = \infty$.)

The RMP curve, which like performance and data profiles is also visually interpreted like a receiver operating characteristic (ROC) (see, e.g. [28]) curve, is the cumulative proportion of the number of best feasible (to violation tolerance τ_v) points returned by method m, as defined by (16), that are each within a relative residual γ of their respective target values $\omega_i \in \Omega$. However, unlike performance and data profiles where the success/failure criterion is fixed and the rankings of the algorithms are shown dependent on how quickly they were able to find acceptable solutions, the RMP curve defined in (17) instead shows the *entire range over which the success/failure criterion can be tuned*, from requiring the best minimization possible on the feasible set to merely obtaining feasibility, regardless of the objective value. An RMP curve portrays how the percentage of feasible points found increases for each method as the maximum allowed relative difference to the target values is relaxed. Of course, the utility of an RMP is influenced by how the per-problem target value set Ω is defined, an issue we will now address.

If optimal objective values for the problems in $\mathcal{T}$ are known, then they may be used to define Ω. However, in general, such data are not necessarily available or attainable. Moreover, using such a defined target set can be problematic if none of the codes find points that (nearly) attain the target values. Thus, we instead propose defining Ω as the set of *best computed objective values at (nearly) feasible points encountered across all methods in* $\mathcal{M}$. To do so, we first define the best computed objective value encountered by any method on problem p_i:

$$f_i^{\mathcal{M}}(\tau_v) := \min_{m \in \mathcal{M}} f_i^m(\tau_v) \tag{18}$$

for a fixed violation $\tau_v \in \mathbb{R}^+$, and then define the per-problem target set as

$$\Omega := \{\omega_i : \omega_i = f_i^{\mathcal{M}}(\tau_v)\}. \tag{19}$$

By using (19), the RMP curve defined in (17) measures the quality of the points obtained by method $m \in \mathcal{M}$ relative to the best objective values *known to be achievable in practice* on the sets of feasible and nearly feasible points for each problem.

In order to view the effect of the relative difference tolerance γ in an idiomatic and compact fashion, we suggest that γ should be plotted using a base-10 log scaling, ranging from machine precision to the highest relative difference observed. If this latter uppermost range is lessened, one should still include $\gamma = \infty$ as the rightmost tick mark so that RMP curves will still depict the ability of codes to find (nearly) feasible points, regardless of the amount of minimization achieved. Thus, a key feature of an RMP's design is that it highlights when a code's performance is subpar, such as when it either frequently fails to satisfy feasibility or tends to stagnate, while simultaneously showing which codes most frequently and closely match the largest amount of minimization observed. From an RMP, it can be inferred with what frequency codes are either finding the same *quality of points* (and to what precision) or *different points* altogether. While a single RMP does not compare methods in terms of their computational costs, in the next section we will show how computational cost can be additionally and simultaneously considered in a benchmark by employing a panel of multiple, related RMP plots.

Though an RMP has similarities to a conventional performance profile, there are crucial differences. For a given performance metric, a conventional performance profile curve for a given code is defined using *relative ratios* of the performance measurement of the given code to the performance measurement of the 'best' code for each particular problem. While a relative ratio is generally appropriate and makes for easily-understood plots for cost-based performance measures (such as running time), it is not necessarily a natural choice for assessing the amount of per-problem objective minimization over a test set, which is one of the reasons we have proposed using relative differences for defining RMPs. Furthermore, outside of the mild requirement to set a cut-off tolerance for the amount of violation allowed for each point, no such hard cut-off line

must be drawn to determine whether an algorithm was successful in terms of its minimization quality. An RMP shows the entire continuum of per-problem relative minimization performance achieved over the (nearly) feasible sets and how it affects the percentage of (nearly) feasible points found. In contrast, conventional performance profiles have the drawback that the points computed by each code are subjected to a binary classification into successes or failures, which can oversimplify and even misleadingly skew the resulting plot, particularly if classification is very sensitive around the success/failure boundary being chosen. Though RMPs still have this problem with respect to a violation tolerance, it is a far more natural choice to have a hard limit for amount of constraint violation allowed than it is for classifying accuracy with respect to some measure of stationarity into successes and failures.

5.2 *Benchmarking efficiency via multiple β-RMPs*

In order to extend RMPs for additionally comparing the cost of each method, we will generalize (17) to what we call a β-RMP curve. The idea is to use the parameter β to specify a particular set of per-problem computational budgets such that the modified RMP curve only considers the subset of all iterates computed by each method within the limits specified by the budget. Then, multiple β-RMP plots can be produced for various values of β to create a *β-RMP benchmark panel* that depicts how the performance of each code's *relative rate of progress* changes and compares to one another as the per-problem computational budgets are increased/decreased.

To that end, for method $m \in \mathcal{M}$ on problem $p_i \in \mathcal{T}$, let us first define a generic function $t_i^m : \mathbb{Z}^+ \to \mathbb{R}^+$ that represents the cumulative cost (in the performance metric of one's choosing) to compute method m's first j iterates:

$$t_i^m(j) := \text{cumulative cost to compute } \{x_0, \ldots, x_j\} \subseteq \{x_k\}_i^m.$$

We may then define the set of iterates encountered by method m on problem p_i, subject to some cost limit $t \in \mathbb{R}^+$:

$$\mathcal{X}_i^m(t) := \begin{cases} \{x_k\}_i^m & \text{if } t = \infty, \\ \{x_j : x_j \in \{x_k\}_i^m \text{ and } t_i^m(j) \leq t\} & \text{otherwise.} \end{cases} \tag{20}$$

Note that even if $t = \infty$, the set of iterates is finite due to whatever stopping conditions were employed in the actual experiments.

Now, for method m on problem p_i, we redefine its *best computed objective value over all feasible iterates* to be in terms of some violation tolerance $\tau_v \in \mathbb{R}^+$ and some computational cost limit $t \in \mathbb{R}^+$:

$$f_i^m(\tau_v, t) := \min\{f_i(x) : x \in \mathcal{X}_i^m(t),\ v_i(x) \leq \tau_v\}, \tag{21}$$

with the two conventions that $f_i^m(\tau_v, t) := \infty$ if $v_i(x) > \tau_v$ for all $x \in \mathcal{X}_i^m(t)$ and when argument t is omitted, it is taken to be ∞, that is, $f_i^m(\tau_v, \infty) = f_i^m(\tau_v)$. However, it is not prudent to just replace the $f_i^m(\tau_v)$ term appearing in the RMP curve defined in (17) with $f_i^m(\tau_v, t)$ as doing so would assign the same computational budget for all problems.

A beautiful property of conventional performance profiles curves is that they prevent any one problem (or small subset) in the test set from exerting too much influence on the overall performance implied by the curve, a property which is obtained by leveraging the observed efficiency (or performance) in the experiments for the codes on each problem as statistics to approximate the relative difficulty of each problem. We will adapt a similar strategy for defining

β-RMPs, which will necessitate defining a computational budget $\mathcal{B}$ in a per-problem fashion:

$$\mathcal{B} := \{b_i : b_i \text{ is the maximum computational cost allowed for problem } p_i \in \mathcal{T}\}.$$

However, unlike conventional performance profiles, it is no longer as straightforward to define the per-problem computational budgets relative to the fastest successful method for each problem because an RMP neither relates efficiency to success nor distinguishes what is specifically meant by success and failure. While the fastest code for a particular test problem may be the most efficient and successful method in the comparison, it is also possible that its apparent efficiency is instead an indication that it terminated due to early stagnation. To combat such pitfalls, we propose defining a relative per-problem budget using the statistics of either the average or median efficiency per problem across all the codes being considered. Yet, if one's purpose is to benchmark a single new method $m_\star$ in the context of competing methods, as opposed to doing a multi-way comparison, then it is more sensible to use per-problem efficiencies of method $m_\star$ to define the budget $\mathcal{B}$ so that efficiency of all other methods will be assessed relative to the efficiency of method $m_\star$. Mathematically, we define the *baseline maximum computational cost for problem $p_i \in \mathcal{T}$ relative to a chosen method $m_\star \in \mathcal{M}$* as follows:

$$b_i := \max_{x_j \in \{x_k\}_i^{m_\star}} t_i^{m_\star}(j), \tag{22}$$

that is, b_i is set to the total computational cost needed for method $m_\star$ on problem p_i.

We may now define a family of β-RMP curves relative to the budgets in $\mathcal{B}$.

DEFINITION 5.2 *Given a fixed violation tolerance $\tau_v \in \mathbb{R}^+$, per-problem target objective value set $\Omega := \{\omega_i\}$, per-problem computational budget set $\mathcal{B} := \{b_i\}$, constant factor $\beta \in \mathbb{R}^{++}$, and method $m \in \mathcal{M}$, a β-RMP curve $r^{m,\beta}_{\Omega,\tau_v} : \mathbb{R}^+ \to [0,1]$ is that given by*

$$r^{m,\beta}_{\Omega,\tau_v}(\gamma) := \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \mathbb{1}_r(\omega_i, f_i^m(\tau_v, \beta b_i), \gamma). \tag{23}$$

Note that if $\beta \geq 1$ and $\mathcal{B}$ is defined by using (22), then we have the following equivalence:

$$r^{m_\star,\beta}_{\Omega,\tau_v}(\gamma) \equiv r^{m_\star}_{\Omega,\tau_v}(\gamma).$$

In other words, for this specific case, (23) is equivalent to (17) since increasing the per-problem budgets with respect to each $b_i \in \mathcal{B}$ has no effect as $\mathcal{B}$ is the set of per-problem budgets method $m_\star$ required on the test set. Like the RMP curve defined in (17), a β-RMP curve is still the cumulative proportion of the number of best objective values found by method $m \in \mathcal{M}$ in the (nearly) feasible sets, which are within a relative residual γ of their respective target values $\omega_i \in \Omega$, but, with the additional constraint that these best iterates computed by method m are obtained within the multiplicative factor β applied to the per-problem budgets b_i given by $\mathcal{B}$.

A natural choice for defining the target set Ω for a β-RMP benchmark panel is to reuse the best objective values known to be achievable on the (nearly) feasible sets, that is, to set each ω_i to the values given by (18). However, while this has the benefit of consistent y-axis scaling for all β-RMP plots, the desired β-RMP plots may not all be conducive to a single scaling for visualization purposes. As an alternative, we proceed by proposing a *rolling target value set* which is updated as the parameter β is changed.

Analogous to (20) and (21), where each method $m \in \mathcal{M}$ is subject to a fixed computational limit of $t \in \mathbb{R}^+$, we define the *set of iterates computed by all methods*, namely

$$\mathcal{X}_i^{\mathcal{M}}(t) := \bigcup_{m \in \mathcal{M}} \mathcal{X}_i^m(t), \tag{24}$$

and similarly define *the best computed objective value over all methods* being compared:

$$f_i^{\mathcal{M}}(\tau_v, t) := \min\{f_i(x) : x \in \mathcal{X}_i^{\mathcal{M}}(t),\ v_i(x) \leq \tau_v\}. \tag{25}$$

We may now define the rolling target value set by specifying each target value additionally in terms of parameter β as follows:

$$\omega_i := f_i^{\mathcal{M}}(\tau_v, \beta b_i), \tag{26}$$

that is, each problem's target value is the best known objective value at a feasible iterate encountered by any of the methods *so far within the computational limit* given by βb_i. The effect of using the rolling target value set defined by (26) is that each β-RMP plot is scaled to maximize the separation of each method's β-RMP curve.

For $\beta = \infty$, a β-RMP depicts how the codes compare given *no computational constraint whatsoever*. In the case where (22) is employed to define β-RMPs relative to a chosen method $m_\star$, a β-RMP plot for $\beta = 1$ shows how the other methods perform relative to method $m_\star$ when each method is only allowed a computational budget equal to the per-problem cost needed by method $m_\star$. For intermediate values, say $\beta = 5$, we see how the codes' performances compare when their computational budgets are allowed up to five times the total computational cost of method $m_\star$ on any given problem while for $\beta < 1$, all methods are only given a fraction of the method $m_\star$'s computational budget. Generally, $\beta = 1$ and $\beta = \infty$ (assuming they yield different plots) should always be included in a panel of β-RMP plots; other values of β must be chosen or generated automatically but it is usually straightforward and fairly effortless to manually find a handful of additional β values to make an illuminating β-RMP benchmark panel. A well-made β-RMP benchmark panel will highlight how relative rankings and performance gaps of the algorithms may, or may not, change as the relative per-problem computational budgets are increased/decreased via changing parameter β.

5.3 *Practical considerations for β-RMPs*

As β-RMPs effectively simulate stopping criteria based on the costs to optimize and attain feasibility, which can be restricted by respectively adjusting β and the violation tolerance, it is important to encourage all algorithms to run as long as possible when performing an evaluation. Typically, this means setting extremely tight termination tolerances, which would be quite unusual in practice. However, doing so not only allows for the most data to be collected for generating the β-RMPs but, crucially, helps to decouple a β-RMP evaluation from each method's own specific stopping criteria. Instead, each method's relative progress, and how it changes, is judged on equal footing in terms of the computational budget, obviating any need to consider how to fairly set each method's stopping parameters for the benchmark, beyond setting the maximum number of iterations allowed for each method.

While RMPs depict the relative amount of minimization achieved versus the frequency of satisfying (near) feasibility, the effect of changing the computational cost allowed per problem is only coarsely approximated across two or more β-RMPs. This is due to several reasons. First, measuring computational cost is usually inherently variable, such as measuring CPU-time, in contrast to assessing feasibility of a point and its corresponding objective value, which can usually be done very precisely. 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 computational limit. Third, depending on the codes being compared, accurate histories of the iterates, and with what cost that they were obtained (such as elapsed CPU-time) may not always be practically obtainable.

In our experience, it is often sufficient to use an average computational cost per iterate as a proxy metric for the purposes of generating data for plotting β-RMPs, such as average elapsed CPU-time per iterate, and especially so when the test sets contain numerous problems and each method typically requires a high number of iterations before converging. If the methods are indeed significantly different in terms of speed, then that property itself will most likely be the dominant factor in determining what is presented by the plots, not the limitation of estimating the timing data. Furthermore, if comparing the relative speeds of the algorithms requires high precision, then it suggests that running time is not a particularly discriminating or even reproducible performance characteristic amongst the codes being compared. In such cases, a single RMP, without specifying a computational limit, should be sufficient, perhaps with a supplemental table listing the overall time reported for each algorithm. As a result, we typically envision using a β-RMP benchmark panel when there are significant differences in computational costs between algorithms, where the distances between the chosen values of β are well above any level of error in the computational cost data. In case a more exact history of timing data is desired, we refer to [38, pages 161–162] for further discussion.

Finally, it is worth noting that while data profiles are precise in terms of the cost to run the algorithms, they give only an approximate indication of the accuracy of the candidate solutions achieved unless many plots are made for various specified accuracy levels. By contrast, RMPs are precise in terms of both the frequency of feasible points found and the quality of minimization achieved over the data set (and their relationship to each other) while only a handful of β-RMPs plots are needed to highlight performance differences of algorithms as the computational budget is changed. This alternative prioritization of how algorithms are compared via β-RMPs is a new benchmarking tool that we believe many will find useful, at least in nonconvex and/or nonsmooth settings and potentially in unconstrained convex settings as well.

6. Numerical results

6.1 *Experimental set-up*

We created two test sets comprised of 100 problems each, where the problems in the first set are spectral radius optimization problems of the form (14) while the problems in the second set are pseudospectral radius optimization problems of the form (15). For both test sets, each problem was comprised of two to five plants, split randomly between the objective and constraints (ensuring both were assigned at least one plant each). In order to compare to SQP-GS, for which 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. A candidate value for MP was picked by randomly choosing an integer Q between $\lceil (N/4)(p + 2q) \rceil$ and $\lceil (N/4)(3p + 6q) \rceil$ inclusively. In order to provide variety in the shapes of controller matrices X generated, M was set to a random integer chosen from $[1, \lceil \sqrt{Q} \rceil]$ and then P was set to Q/M rounded to the nearest integer. Normally distributed matrices A_i were generated using the `randn` function and were subsequently destabilized (for the objective) or stabilized (for the constraints) by successively multiplying the matrices by a constant greater or less than one, respectively. Each B_i matrix was set to either (a) consist of a column of ones followed by a uniform distribution of randomly generated positive and negative ones over the remaining entries of the matrix or (b) a normally

distributed random matrix, with the choice between (a) and (b) being made randomly with a probability of one half for each. Similarly, each C_i matrix was set to either (a) the first m rows of the identity matrix or (b) a normally distributed random matrix. Since the origin is a feasible point for all the problems, each algorithm was initialized with X set to the zero matrix. We also initialized 20% of the problems in our tests sets from multiple infeasible starting points generated via `randn()`, to assess whether or not the choice of always initializing from the feasible origin biased our results. While we noticed some minor differences when starting from randomly generated infeasible points, we omit the details for the sake of brevity since the overall conclusions made from these additional experiments agree with our results and interpretation given here in the paper. We used $\mu = 16$ as the initial penalty parameter value.[6]

As noted earlier, we include four methods in our comparisons. We used version 7.5-1.4 of SNOPT [18]. SFPP was implemented using the BFGS routine from the 2.02 version of the MATLAB HANSO optimization package [26], with a limit of 20 iterations per BFGS call before lowering the penalty parameter if the 20th iterate was infeasible. We used version 1.1 of SQP-GS, available on the first author's homepage. We implemented our BFGS-SQP method, supporting both inequality and equality constraints, in a new open-source MATLAB code called GRANSO: GRadient-based Algorithm for Non-Smooth Optimization, utilizing a modified version of the Armijo–Wolfe line search routine from HANSO. For parameters for our implementation of Procedure 1, we set $c_v = 0.1$ and $c_\mu = 0.9$, limited its loop to at most 10 iterations, and only applied steering at infeasible iterates, while returning the direction computed in line 1 for feasible iterates. Both SQP-GS and BFGS-SQP require a QP solver and for these experiments we used MOSEK 7. Note that SNOPT is the only compiled, non-interpreted code in the evaluation and hence has a significant advantage in terms of speed.

As per the guidelines given in Section 5.3 for implementing a β-RMP benchmark, we aimed to encourage the algorithms to continue optimizing as long as possible by using very tight stopping parameters. To that end, we set BFGS-SQP's termination tolerance, $\tau_\diamond$, to machine precision (2.22×10^{-16}) and its violation tolerance, τ_v, to zero (noting that zero can be a reasonable violation tolerance when there are only inequality constraints present, as is the case for our test sets). Similarly, we respectively set the analogous termination and violation tolerances for SFPP and SQP-GS to machine precision and zero as well. We also used machine precision for SNOPT's 'major optimality tolerance' but the code would then not allow its 'major feasibility tolerance' to be set to zero, or even machine precision, so we instead used a slightly bigger number (2.23×10^{-16}). All four methods were allowed to run for up to 500 iterations each; in the case of SNOPT, we allowed 500 major iterations and 100 minor iterations.

To specifically validate our proposed algorithm BFGS-SQP, we chose to define the computational budget $\mathcal{B}$ so that the β-RMP benchmark panels would be relative to BFGS-SQP's efficiency on each problem in the test set, using (22). As no optimal values were known for our test sets, we used a rolling target value set Ω based on the best-encountered iterate by any algorithm so far, as defined by $\mathcal{B}$ and the particular value of β using (26). We made β-RMP plots for $\beta = 1, 5, 10$, and ∞, using a violation tolerance of zero, since all the problems were only inequality constrained, and excluded the feasible origin starting points from the histories of iterates from every method (since otherwise, each method would have a feasible point for every problem). Neither SQP-GS nor HANSO's BFGS codes provide per-iterate timing data, so, as outlined and justified in Section 5.3, for each method, and for every problem, we chose to compute an average time per-iterate statistic and used it to approximate the necessary cost data for plotting an RMP. Given the large differences in running times between the codes in our experiments, and the fact that the convergence is generally slow in terms of the number of iterations on the problems created for our test sets, it is unlikely that using an average time per-iterate proxy would significantly skew the β-RMP benchmark panel for comparing the relative rates of progress for the algorithms.

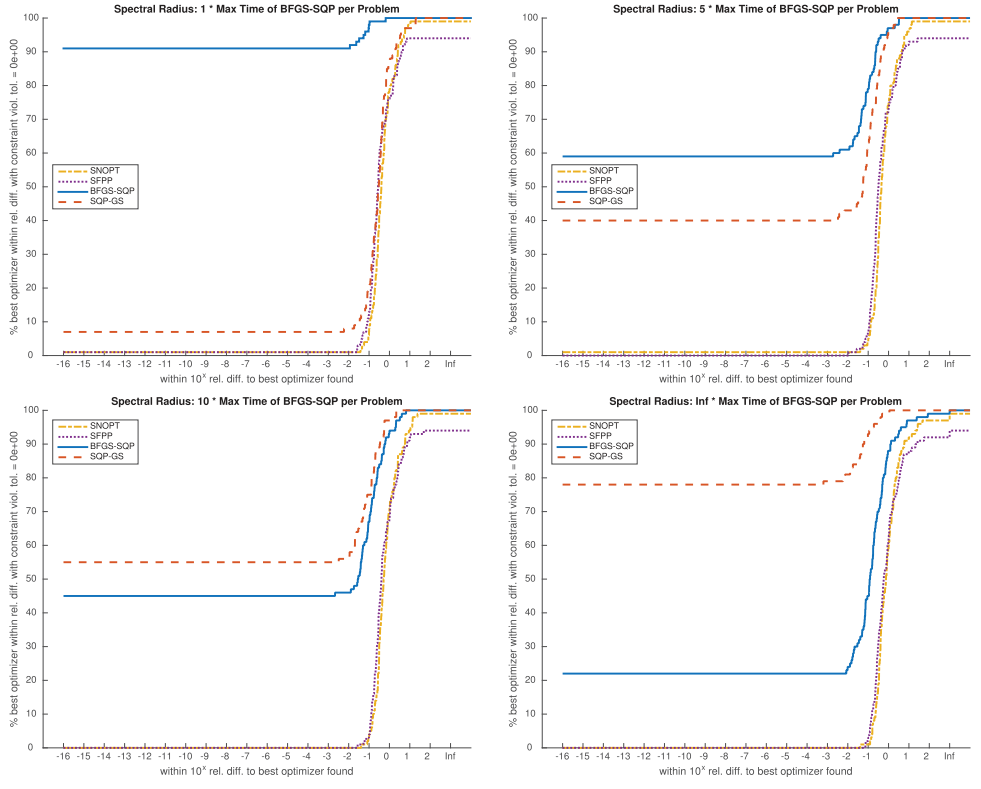

Figure 5. Performance benchmark using β-RMPs, on the spectral radius optimization problems, of SNOPT, SFPP, BFGS-SQP, and SQP-GS. For $\beta = 1$ (top left) and 5 (top right), where the other methods are, respectively, given one and then five times the per-problem computational budget as BFGS-SQP's per-problem total elapsed CPU-time, we observe that BFGS-SQP provides the greatest percentage of best optimizers on the test set, readily increasing its performance margin as β is decreased towards one. Only as β is increased to 10 and higher, where little to no restriction is put on computational budgets, do we see SQP-GS outperforming BFGS-SQP over the test, but this accomplishment is only attained by respectively incurring a higher computational overhead that is ten times (bottom left) or 26.6 times (bottom right) greater than that of BFGS-SQP.

All experiments described in this section were run on an Intel Xeon X5650 2.67 Ghz CPU using MATLAB R2012b. MATLAB source code for GRANSO and for generating RMPs, along with the data and function files defining our test set, are publicly available.[7]

6.2 *Spectral radius optimization*

We begin analysing the relative performance of the four codes by first turning our attention to the bottom right ∞-RMP in Figure 5, where the methods are tested using the non-locally-Lipschitz spectral radius set and where none of them have convergence guarantees. Here, each algorithm was allowed to run without any time restriction and we see that despite the promising example shown earlier in Figure 1 for BFGS-SQP, SQP-GS ended up finding 78% of the best optimizers while BFGS-SQP found 22% of the best optimizers. However, we see that BFGS-SQP's performance over the data set starts to look better if we relax how much of the minimization of the best known candidate solution is necessary to be considered a success, indicating that BFGS-SQP generally either converged to different (worse) points or perhaps was stagnating early compared to SQP-GS. By comparison, SNOPT and SFPP appear to be completely uncompetitive, though it is notable that SNOPT does at least find more feasible points compared to SFPP,

even if neither are able to do so for every problem, unlike BFGS-SQP and SQP-GS. While the impression given by this ∞-RMP does not initially appear promising for BFGS-SQP, the fact remains that SQP-GS actually took 26.6 times longer to run to obtain these better results, which is a computational cost difference that is likely not be readily absorbed by many users.

In fact, by now turning our attention to the bottom left 10-RMP in Figure 5, where SQP-GS was allowed to run up to 10 times longer than BFGS-SQP, a still significant and not usually dismissed computational cost increase, we see that the performance gap between BFGS-SQP and SQP-GS has been greatly narrowed, with BFGS-SQP finding about 45% of the best optimizers and SQP-GS finding 55%. If we further consider the 5-RMP in the top right, where SQP-GS is now allowed up to five times as much time as BFGS-SQP per problem, we are now able to observe that BFGS-SQP actually found 59% of the best optimizers, compared to SQP-GS's 40%, while only expending a fraction of the computational resources spent by SQP-GS. Finally, in the top left 1-RMP, where SNOPT, SFPP, and SPQ-GS are assessed at the per-problem times at which BFGS-SQP terminated, we see that BFGS-SQP completely outclasses all the other methods, finding 92% of the best optimizers, with SQP-GS only finding 7% of the best minimizers and SNOPT and SFPP faring even worse.

Since SNOPT and SFPP, respectively ran over the test set in only 13.0% and 74.7% of the elapsed CPU-time that BFGS-SQP required, we also generated two panels of β-RMPs relative to SNOPT and SFPP, where BFGS-SQP's computational budget is determined by factors of the generally faster per-problem running times of SNOPT and SFPP. Since both panels of β-RMPs showed BFGS-SQP greatly outperforming SNOPT and SFPP, we omit both panels for brevity. For even $\beta = 1$, where BFGS-SQP is allowed only a small fraction of its normal total running time, it respectively found about 70% and 86% of the best optimizers for SNOPT-based 1-RMP and SFPP-based 1-RMP.

6.3 *Pseudospectral radius optimization*

On the pseudospectral radius problems, we see that even in the bottom right ∞-RMP in Figure 6, where no running time restrictions were put on the algorithms, BFGS-SQP significantly outperformed SQP-GS, with 59% of the best optimizers found by BFGS-SQP and only 34% found by SQP-GS (in terms of matching the best observed objective values to at least 12 digits, i.e. 10^{-12} on the RMPs). The other RMPs for smaller values of β only paint BFGS-SQP in a significantly more favourable light. To put that in context, BFGS-SQP more frequently produced better minimizers over the test set compared to SQP-GS, regardless of computational budget limits. This is a remarkable outcome given that the convergence results for SQP-GS hold for the problems in the pseudospectral radius test set (as they are locally Lipschitz) while BFGS-SQP provides no theoretical guarantees yet still manages to provide better overall performance while simultaneously being 14.4 times faster. Indeed, it is in stark contrast to the spectral radius test set results, where SQP-GS could at least pull ahead compared to BFGS-SQP when SQP-GS was allowed to run 10–26.6 times longer.

Interestingly, we also see that SNOPT and SFPP both do noticeably better on the Lipschitz problems, though both are still outperformed by BFGS-SQP at all computational budgets considered. Only SNOPT ran faster than BFGS-SQP over the test set, taking 30.0% of BFGS-SQP's elapsed CPU-time. For this reason, we generated panels of β-RMPs relative to SNOPT's per-problem running times, but these still showed BFGS-SQP outperforming SNOPT for $\beta \geq 1$, though the performance gap between the two was less than observed in the spectral radius case. Again for brevity, we omit these β-RMPs and merely remark that for $\beta = 1$, where BFGS-SQP is generally given about 30% of its normal per-problem running time, it still found 63% of the best minimizers compared to SNOPT's 30%.

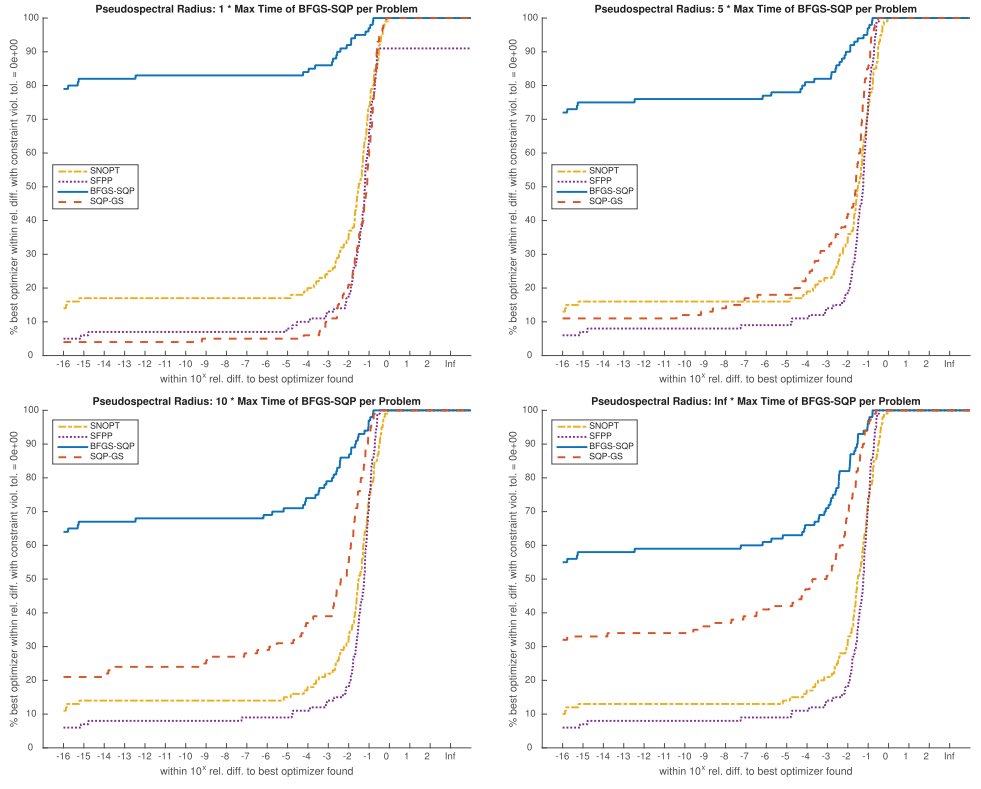

Figure 6. Performance benchmark using β-RMPs, on the pseudospectral radius optimization problems, of SNOPT, SFPP, BFGS-SQP, and SQP-GS. For $\beta = 1$ (top left), 5 (top right), and 10 (bottom left), where the other methods are respectively given 1, 5, and then 10 times the per-problem computational budget as BFGS-SQP's per-problem total elapsed CPU-time, we again observe that BFGS-SQP provides the greatest percentage of best optimizers on the test set, readily increasing its performance margin as β is decreased. Even when no computational budget is applied to SQP-GS, where SQP-GS took 14.4 times longer than BFGS-SQP to run over the test set, the overall fraction of best minimizers returned by SQP-GS is still significantly less than that of BFGS-SQP, shown for $\beta = \infty$ (bottom right).

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 [35], we explore the effect of regularizing the Hessian approximation in BFGS-SQP. However, as BFGS-SQP requires solving QPs, there is also a potential tradeoff of regularizing to produce easier-to-solve QPs (or ones for which the QP solver could produce more accurate solutions) versus not regularizing to retain the ill-conditioning in the BFGS Hessian approximation. We note that on the non-locally-Lipschitz spectral radius test set, the Hessian approximations formed in BFGS-SQP were generally more ill-conditioned than those observed on the locally Lipschitz pseudospectral radius test set, with the max condition numbers over all problems and iterations, respectively, being 2.03×10^{20} and 3.06×10^{17}. Thus, to assess the effect of regularization, we reran the BFGS-SQP experiments multiple times, where the Hessian approximations appearing in the steering and termination QPs at every iteration were, if necessary, regularized to ensure that their condition numbers did not exceed our specified limits. We chose condition number limits of 10^{2j}, for $j \in \{0, \ldots, 10\}$ for the spectral radius test set and for $j \in \{0, \ldots, 8\}$ for the pseudospectral radius test set, where the case of $j = 0$ corresponds to replacing the Hessian approximation by a multiple of the identity.

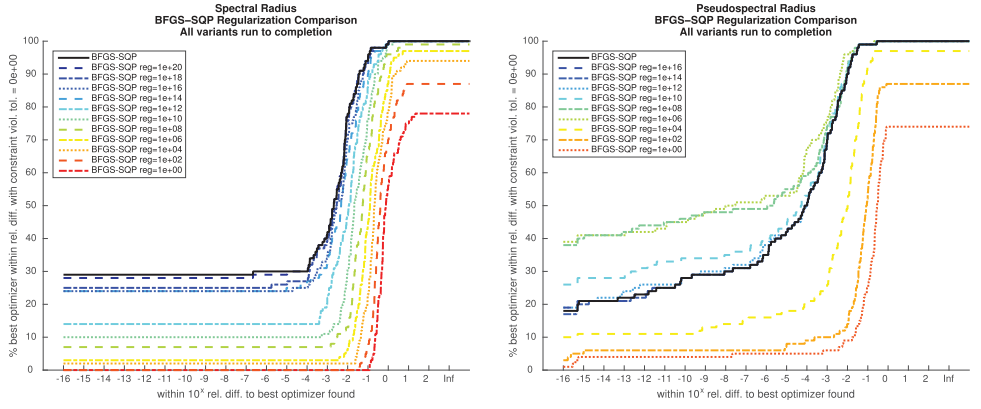

Figure 7. Left: the spectral radius test set. Right: the pseudospectral radius test set. Both RMPs show how the performance of BFGS-SQP changes over the test sets when increasing amount of regularization is applied to the inverse of the Hessian approximation.

For the spectral radius problems, as shown in the left plot in Figure 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 the 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 the right plot in Figure 7 that moderate levels of regularization (such as 10^6 – 10^8) seem to have a strikingly clear beneficial effect compared to not regularizing at all. It is hard to say why we observe this behaviour but it is conceivable that the regularization helped improve the accuracy of the QP solves more than it hurt the effectiveness of BFGS. In any case, the large differences between the spectral radius RMP and the pseudospectral radius RMP in Figure 7 make it apparent that the effects of applying regularization in BFGS-SQP are problem dependent. Certainly more investigation into this matter would be a worthwhile pursuit.

7. Conclusion

We have empirically validated our new BFGS-SQP method against competing alternatives, demonstrating BFGS-SQP's efficiency and reliability, both in terms of satisfying feasibility and minimizing the objective, on our test set of 200 challenging nonsmooth, nonconvex constrained optimization problems arising in controller design. Indeed, not only did BFGS-SQP completely outperform SNOPT and the SFPP method on the entire test set, it also outperformed SQP-GS in terms of solution quality as well as efficiency on the locally Lipschitz problems, despite the fact that BFGS-SQP has no theoretical convergence guarantees while SQP-GS does in this domain. On the harder non-Lipschitz problems, where none of the algorithms have convergence results, we see that only in the absence of computational budget concerns does SQP-GS outperform BFGS-SQP; in scenarios where devoting computational resources many times larger than that needed by BFGS-SQP is intractable, we actually observe our new method outperforming SQP-GS, demonstrating BFGS-SQP's high efficiency and practicality. The challenging properties inherent to optimization problems arising in controller design, which are embodied in our test sets, raise the question of whether BFGS-SQP's performance demonstrated in this domain extends to other nonconvex, constrained optimization problem classes; we leave such investigations for future work. Finally, facilitating the algorithmic comparison done here, we

have proposed RMPs as a new tool for benchmarking optimization software which allow for a concise, yet detailed visualization for comparing pertinent performance characteristics of algorithms and how they interrelate.

Notes

1. Note that we use H_k to denote an approximation to the Hessian at the kth iterate, as opposed to the notation used in [35] and [40, page 140] where H_k is used to denote a BFGS approximation to the *inverse* of the Hessian.
2. We excluded the Fortran-based MPBNGC software from our experiments since it is not equipped to provide the information needed for our comparisons. In particular, MPBNGC does not have a publicly available MATLAB interface, which would have been needed since our test problems are coded in MATLAB and rely on third-party MATLAB software. In addition, MPBNGC does not provide a complete history of the iterate sequence, which is necessary to perform the types of comparisons performed in this paper.
3. Generated via MATLAB's `contour` by evaluating the smallest singular value of $A - \lambda I$ on a grid in the complex plane, exploiting Remark 4.6. Note that because the matrices are real, the pseudospectra boundaries are symmetric with respect to the real axis of the complex plane.
4. In fact, RMPs 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. See [38, Remarks 7.12 and 7.16] for a brief discussion on the applicability of RMPs to other settings.
5. Note that the case when $\phi = 0$ must be handled specially, as $r(\varphi, \tilde{\varphi})$ is otherwise undefined. However, such special treatment is not applicable here as ϕ is always strictly positive for all test problems in this paper.
6. Note that we use a power of two for the initial penalty parameter and, when reducing it, always halve it since otherwise, in floating point arithmetic, we noticed that $d_k = -H_k^{-1}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda_k)$ was only approximately equal to $-\mu H_k^{-1} f(x_k) - H_k^{-1} \nabla c(x_k)\lambda_k$, and the latter form is what we use for efficiency reasons.
7. http://www.cims.nyu.edu/∼tmitchell/

Acknowledgements

The authors are grateful to the two anonymous referees for carefully reading the paper and for their helpful comments. They are also very grateful to Philip Gill and Elizabeth Wong for making an improved version of SNOPT's MATLAB interface available so that the iteration history needed to create β-RMPs could be collected.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

Supported by Department of Energy Grant [DE–SC0010615] and National Science Foundation Grant [DMS–1319356]. Supported by National Science Foundation Grant [DMS-1317205]. Supported by National Science Foundation Grant [DMS-1317205].

ORCiD

Frank E. Curtis http://orcid.org/0000-0001-7214-9187
Tim Mitchell http://orcid.org/0000-0002-8426-0242
Michael L. Overton http://orcid.org/0000-0002-6563-6371

References

[1] H. Akcay and S. Turkay, *Influence of tire damping on actively controlled quarter-car suspensions*, J. Vib. Acoust. 133 (2011), p. 054501.
[2] E.G. Birgin and J.M. Gentil, *Evaluating bound-constrained minimization software*, Comput. Optim. Appl. 53 (2012), pp. 347–373. Available at http://dx.doi.org/10.1007/s10589-012-9466-y.

[3] E.G. Birgin and J. Martínez, *Practical Augmented Lagrangian Methods for Constrained Optimization*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014. Available at http://epubs.siam.org/doi/abs/10.1137/1.9781611973365.
[4] E.G. Birgin, L.F. Bueno, and J.M. Martínez, *Assessing the reliability of general-purpose inexact restoration methods*, J. Comput. Appl. Math. 282 (2015), pp. 1–16. Available at http://www.sciencedirect.com/science/article/pii/S0377042714005871.
[5] V. Blondel, *Three problems on the decidability and complexity of stability*, in *Open Problems in Mathematical Systems and Control Theory*, V. Blondel and J. Tsitsiklis, eds., Springer, London, 1999, pp. 53–56.
[6] J.V. Burke, *Calmness and exact penalization*, SIAM J. Control Optim. 29 (1991), pp. 493–497. Available at http://dx.doi.org/10.1137/0329027.
[7] J.V. Burke, *An exact penalization viewpoint of constrained optimization*, SIAM J. Control Optim. 29 (1991), pp. 968–998. Available at http://dx.doi.org/10.1137/0329054.
[8] J.V. Burke and M.L. Overton, *Variational analysis of non-Lipschitz spectral functions*, Math. Program. 90 (2001), pp. 317–352.
[9] J.V. Burke, A.S. Lewis, and M.L. Overton, *A robust gradient sampling algorithm for nonsmooth, nonconvex optimization*, SIAM J. Optim. 15 (2005), pp. 751–779.
[10] 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.
[11] R.H. Byrd, J. Nocedal, and R.A. Waltz, *Steering exact penalty methods for nonlinear programming*, Optim. Methods Softw. 23 (2008), pp. 197–213. Available at http://dx.doi.org/10.1080/10556780701394169.
[12] R.H. Byrd, G. Lopez-Calva, and J. Nocedal, *A line search exact penalty method using steering rules*, Math. Program. 133 (2012), pp. 39–73. Available at http://dx.doi.org/10.1007/s10107-010-0408-0.
[13] F.E. Curtis and M.L. Overton, *A sequential quadratic programming algorithm for nonconvex, nonsmooth constrained optimization*, SIAM J. Optim. 22 (2012), pp. 474–500. Available at http://dx.doi.org/10.1137/090780201.
[14] T. Delwiche, *Contribution to the design of control laws for bilateral teleoperation with a view to applications in minimally invasive surgery*, Ph.D. thesis, Free University of Brussels, 2009.
[15] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213. Available at http://dx.doi.org/10.1007/s101070100263.
[16] D. Dotta, A.S. e Silva, and I.C. Decker, *Design of power systems controllers by nonsmooth, nonconvex optimization*, in *IEEE Power and Energy Society General Meeting*, Calgary, 2009.
[17] R. Fletcher, *Practical Methods of Optimization*, 2nd ed., John Wiley, Chichester and New York, 1987.
[18] P.E. Gill, W. Murray, and M.A. Saunders, *SNOPT : An SQP algorithm for large-scale constrained optimization*, SIAM J. Optim. 12 (2002), pp. 979–1006. Available at http://dx.doi.org/10.1137/S1052623499350013.
[19] 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.
[20] M. Gürbüzbalaban and M.L. Overton, *Some regularity results for the pseudospectral abscissa and pseudospectral radius of a matrix*, SIAM J. Optim. 22 (2012), pp. 281–285. Available at http://dx.doi.org/10.1137/110822840.
[21] N. Haarala, K. Miettinen, and M.M. Mäkelä, *Globally convergent limited memory bundle method for large-scale nonsmooth optimization*, Math. Program. 109 (2007), pp. 181–205. Available at http://dx.doi.org/10.1007/s10107-006-0728-2.
[22] T. Hanis and M. Hromcik, *Flexible aircraft lateral control law of low order*, in Czech Aerospace Proceedings, 2011.
[23] T. Hanis and M. Hromcik, *Lateral control for flexible BWB high-capacity passenger aircraft*, in Proceedings of the 18th IFAC World Congress, Milan, 2011, pp. 7233–7237.
[24] T. Hanis and M. Hromcik, *Lateral flight dynamic controller for flexible BWB aircraft*, in Proceedings of the 18th International Conference on Process Control, Bratislava, Slovenska technicka univerzita, 2011, pp. 333–337.
[25] T. Hanis, V. Kucera, and M. Hromcik, *Low order H-infinity optimal control for ACFA blended wing body*, in 4th European Conference for Aerospace Sciences (EUCASS), St. Petersburg, 2011.
[26] HANSO (Hybrid algorithm for non-smooth optimization), http://www.cs.nyu.edu/overton/software/hanso/, [Online; accessed 10 September 2014].
[27] J. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms*, Springer, New York, 1993. two volumes.
[28] F. Hsieh and B.W. Turnbull, *Nonparametric and semiparametric estimation of the receiver operating characteristic curve*, Ann. Statist. 24 (1996), pp. 25–40. Available at http://dx.doi.org/10.1214/aos/1033066197.
[29] S. Khodaverdian and J. Adamy, *A noninteracting control strategy for the robust output synchronization of linear heterogeneous networks*, Control Theory Technol. 12 (2014), pp. 234–249.
[30] S. Khodaverdian and J. Adamy, *Robust output synchronization for heterogeneous multi-agent systems based on input-output decoupling*, in 11th IEEE International Conference on Control & Automation, Taichung, Taiwan, 2014, pp. 607–613.
[31] K.C. Kiwiel, *Methods of Descent for Nondifferentiable Optimization*, Lecture Notes in Mathematics, Vol. 1133, Springer-Verlag, Berlin, 1985.
[32] K.C. Kiwiel, *Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization*, SIAM J. Optim. 18 (2007), pp. 379–388.
[33] D. Knittel, D. Henrion, M. Millstone, and M. Vedrines, *Fixed-order and structure H-infinity control with model based feedforward for elastic web winding systems*, in Proceedings of the IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems, Gdansk, Poland, 2007.
[34] A.S. Lewis, *Active sets, nonsmoothness and sensitivity*, SIAM J. Optim. 13 (2003), pp. 702–725.

[35] A.S. Lewis and M.L. Overton, *Nonsmooth optimization via quasi-Newton methods*, Math. Program. 141 (2013), pp. 135–163. Available at http://dx.doi.org/10.1007/s10107-012-0514-2.
[36] M.M. Mäkelä, *Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0*, Reports of the Department of Mathematical Information Technology, Series B, Scientific Computing No. B 13/2003, University of Jyväskylä, 2003.
[37] E. Mengi and M.L. Overton, *Algorithms for the computation of the pseudospectral radius and the numerical radius of a matrix*, IMA J. Numer. Anal. 25 (2005), pp. 648–669.
[38] T. Mitchell, *Robust and efficient methods for approximation and optimization of stability measures*, Ph.D. thesis, New York University, New York University, 2014.
[39] J.J. Moré and S.M. Wild, *Benchmarking derivative-free optimization algorithms*, SIAM J. Optim. 20 (2009), pp. 172–191. Available at http://dx.doi.org/10.1137/080724083.
[40] J. Nocedal and S. Wright, *Nonlinear Optimization*, 2nd ed., Springer, New York, 2006.
[41] M.L. Overton, *Stability optimization for polynomials and matrices*, in *Nonlinear Physical Systems: Spectral Analysis, Stability and Bifurcations*, O. Kirillov and D. Pelinovsky, eds., Chapter 16, Wiley-ISTE, London, 2014, pp. 351–375.
[42] G. Pouly, J.P. Lauffenburger, and M. Basset, *Reduced order H-infinity control design of a nose landing gear steering system*, in 12th IFAC Symposium on Control in Transportation Systems, 2010.
[43] M. Rezac and Z. Hurák, *Structured MIMO H^∞ design for dual-stage inertial stabilization: case study for HIFOO and Hinfstruct solvers*, Mechatronics 23 (2013), pp. 1084–1093.
[44] B. Robu, V. Budinger, L. Baudouin, C. Prieur, and D. Arzelier, *Simultaneous H-infinity vibration control of fluid/plate system via reduced-order controller*, in Proceedings of CDC 2010, Atlanta, 2010, pp. 3146–3151.
[45] A. Ruszczynski, *Nonlinear Optimization*, Princeton University Press, Princeton, NJ, 2011.
[46] L.N. Trefethen and M. Embree, *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*, Princeton University Press, Princeton, NJ, 2005.
[47] A. Wahrburg, S. Khodaverdian, and J. Adamy, *Robust observer-based fault detection and isolation in the standard control problem framework*, in Proceedings of the 7th IFAC Symposium on Robust Control Design, Aalborg, Denmark, 2012, pp. 473–478.
[48] F.C. Wang and H.T. Chen, *Design and implementation of fixed-order robust controllers for a proton exchange membrane fuel cell system*, Int. J. Hyd. Energy 34 (2009).
[49] A. Wildschek, R. Maier, M. Hromcik, T. Hanis, A. Schirrer, M. Kozek, C. Westermayer, and M. Hemedi, *Hybrid controller for gust load alleviation and ride comfort improvement using direct lift control flaps*, in Third European Conference for Aerospace Sciences (EUCASS), Versailles, 2009.
[50] I. Yaesh and U. Shaked, *H-infinity optimization with pole constraints of static output-feedback controllers – a non-smooth optimization approach*, IEEE Trans. Control Syst. Technol. 20 (2012), pp. 1066–1072.

Appendix

A.1. Spectral radius additional examples

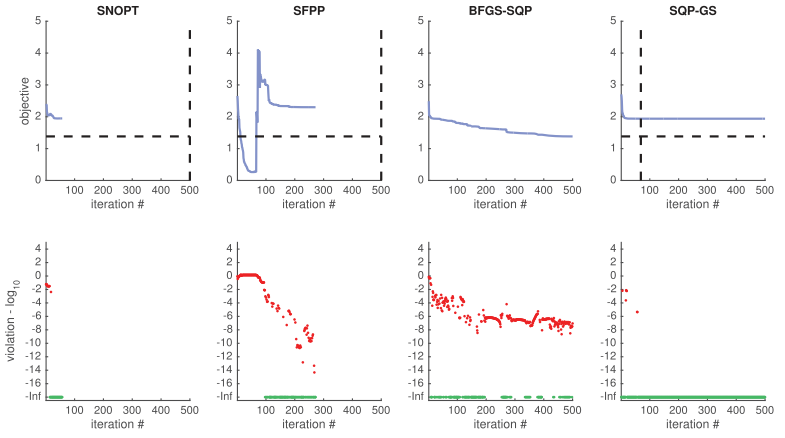

Figure A1. The plots in the top row track the value of the spectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 5$ comprised of one plant in the objective and one in the constraint and where the controller matrix has $MP = 4 \times 2 = 8$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

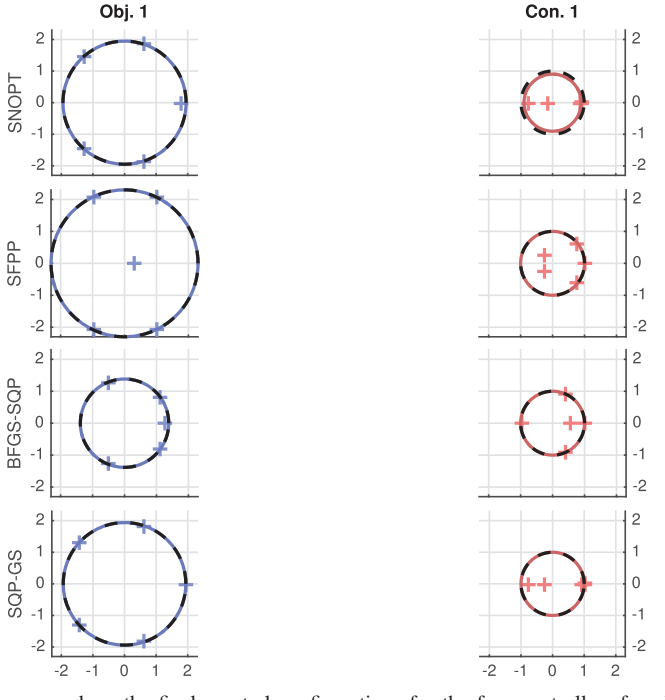

Figure A2. The four rows show the final spectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure A1. The left column indicates the single plant in the objective while the right column indicates the single plant in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the spectral radius of the plant in the objective for that particular algorithm's controller. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The solid lighter circles indicate the spectral radius of each plant.

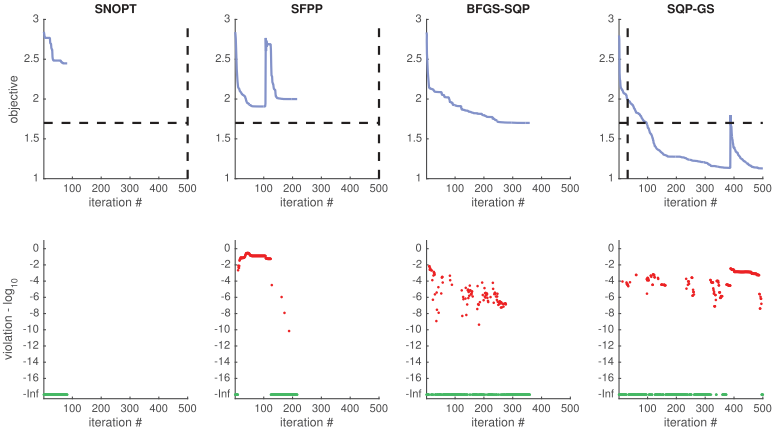

Figure A3. The plots in the top row track the value of the spectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 10$ comprised of one plant in the objective and four in the constraint and where the controller matrix has $MP = 16 \times 1 = 16$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

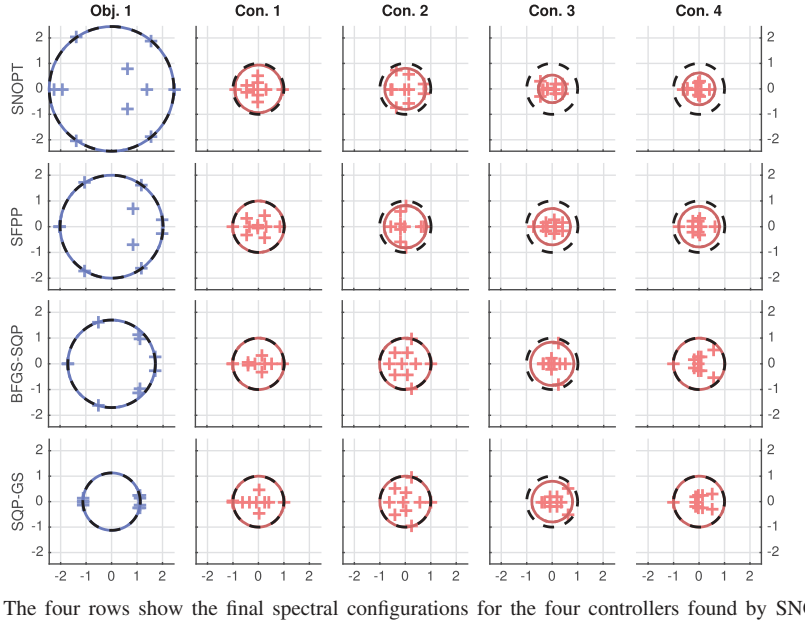

Figure A4. The four rows show the final spectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure A3. The left column indicates the single plant in the objective while the right four columns indicate the plants in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the spectral radius of the plant in the objective for that particular algorithm's controller. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The solid lighter circles indicate the spectral radius of each plant.

A.2. Pseudospectral radius additional examples

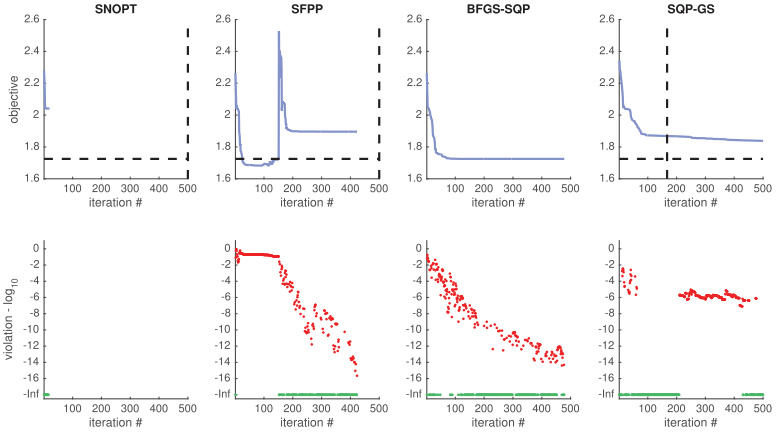

Figure A5. The plots in the top row track the value of the pseudospectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 6$ comprised of one plant in the objective and one in the constraint and where the controller matrix has $MP = 7 \times 1 = 7$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

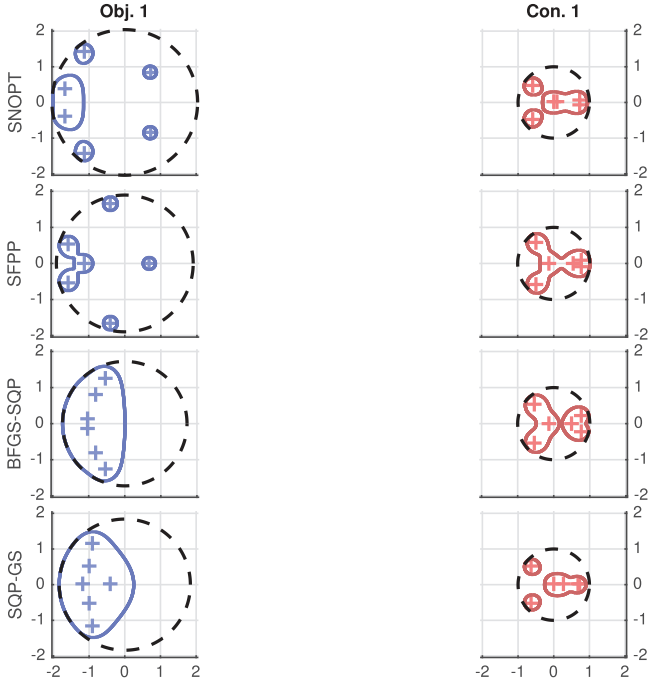

Figure A6. The four rows show the final pseudospectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure A5. The left column indicates the single plant in the objective while the right column indicates the single plant in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the pseudospectral radius of the plant in the objective for that particular algorithm's controller. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The solid lighter circles indicate the pseudospectral boundaries of each plant.

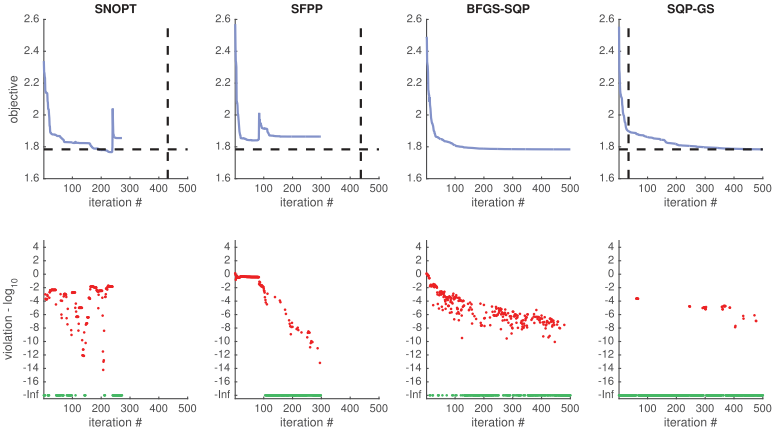

Figure A7. The plots in the top row track the value of the pseudospectral-radius-based objective function in terms of iteration number for SNOPT, SFPP, BFGS-SQP, and SQP-GS (left to right) on a randomly generated example of dimension $N = 7$ comprised of two plants in the objective and three in the constraint and where the controller matrix has $MP = 5 \times 4 = 20$ variables. In the top row of plots, the vertical dashed line indicates the iteration number whose elapsed CPU-time was closest to BFGS-SQP total elapsed CPU-time while the horizontal dashed line indicates the value of BFGS-SQP's best feasible solution. The $\log_{10}$-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with '-Inf' indicating zero violation (feasible points).

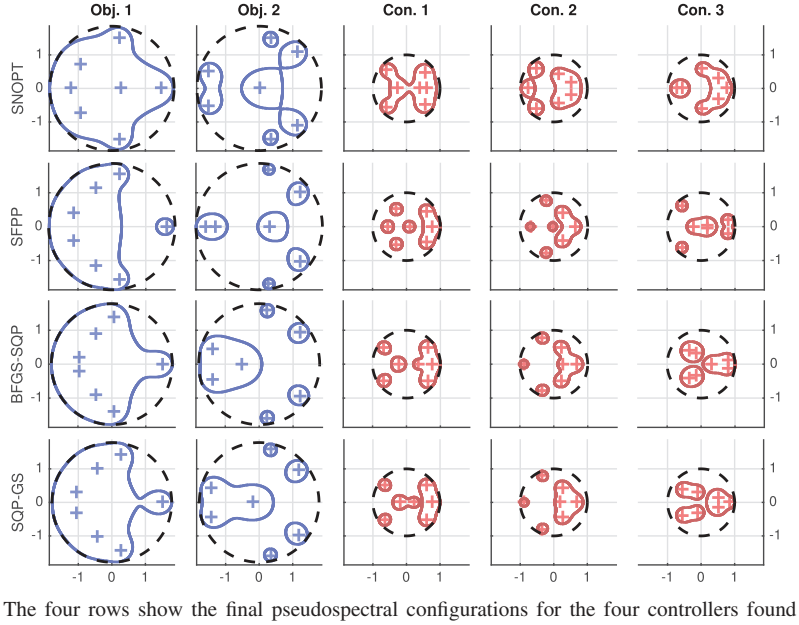

Figure A8. The four rows show the final pseudospectral configurations for the four controllers found by SNOPT, SFPP, BFGS-SQP, and SQP-GS (top to bottom) for the problem described in Figure A7. The left two columns indicate the plants in the objective while the right three columns indicate the plants in the constraint, with the plus signs indicating the eigenvalues. On the objective plots, the dashed black circle corresponds to the max pseudospectral radius of the two plants in the objective for that particular algorithm's controller. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The solid lighter circles indicate the pseudospectral boundaries of each plant.