A BFGS-SQP Method for Nonsmooth, Nonconvex, Constrained Optimization and its Evaluation using Relative Minimization Profiles

Frank E. Curtis†, Tim Mitchell‡, and Michael L. Overton§

(Received 00 Month 20XX; final version received 00 Month 20XX)

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 BFGS quasi-Newton Hessian approximations and an exact penalty function whose parameter is controlled using a steering strategy. We empirically validate our method using our 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 three measures, highlighting the trade-offs between the measures when comparing algorithm performance on a heterogeneous test set. For example, in our experiments, we employ our algorithm to solve challenging test problems in controller design involving both locally Lipschitz and non-locally-Lipschitz functions, using relative minimization profiles to simultaneously compare our method against others in terms of objective quality, feasibility error, and speed of progress.

Keywords: nonconvex optimization; nonsmooth optimization; constrained optimization; sequential quadratic programming; 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 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).

Unconstrained nonsmooth optimization is a well studied subject (see [14, 16] and [25, Ch. 7]), particularly in the convex case; for more references see, for example, the literature discussions in [19] and [9]. The latter paper includes discussion of some methods for...
constrained nonsmooth optimization. However, we are not aware of any work on direct methods for general problems of the form \( \text{(1)} \), excluding those that resort to penalty functions or other problem reformulations, with the exception of \([3]\), discussed further below, which is based on the gradient sampling (GS) algorithm of \([2]\) for unconstrained nonsmooth, nonconvex 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 \([17]\), it was further shown that the convergence results of GS could be strengthened by slightly modifying the algorithm, thereby obviating 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 \( O(n) \) gradients be evaluated per iteration, which can be a quite costly endeavor for expensive-to-compute functions and, as a result, GS is not a viable algorithm in many applications.

On the other hand, it has recently been strongly advocated in \([19]\) that for unconstrained minimization of nonsmooth, nonconvex, locally Lipschitz functions, a simple 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 is typically 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 \([18]\), 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 the more challenging case of constrained nonsmooth, nonconvex optimization—i.e., where \( p \geq 1 \) in \( \text{(1)} \)—a sequential quadratic programming approach employing gradient sampling (SQP-GS) was presented in \([9]\) 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 programs, but intriguingly and in contrast to the argument of \([19]\) 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. Still, the reliance on gradient sampling, both theoretically and in the implementation of the algorithm, makes SQP-GS ill-suited for optimizing with expensive-to-compute objective and constraint functions.

Our aim in proposing a method for solving \( \text{(1)} \) is to eschew a costly gradient sampling approach entirely and to instead find an efficient and effective extension of the simple BFGS approach for unconstrained problems to the domain of nonsmooth, nonconvex, constrained optimization. The result of this endeavor is an algorithm which we call BFGS-SQP, which employs BFGS Hessian approximations within a sequential quadratic programming algorithm and does not assume any special structure in the objective or constraints. To empirically validate our proposed method, we present new visualization tools called relative minimization profiles that represent an alternative to both performance profiles of Dolan and Moré \([10]\) and data profiles of Moré and Wild \([22]\).

The paper is organized as follows. In the next section, we establish a prerequisite penalty parameter approach for constrained optimization. In \([4]\) 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 proposed method. In \([4]\) we present two challenging problems—one involving non-locally-Lipschitz and one involving locally
Lipschitz functions—arising in controller design. In §5 we introduce and motivate our new visualization technique to compare algorithms using relative minimization profiles, and in §6 we use these profiles to compare our new method with competing methods on two test sets comprising problems of the forms introduced in §4. Concluding remarks are provided in §7 and additional illustrative examples are provided in the appendix.

2. An exact penalty function approach

Consider the exact nonsmooth penalty function

\[ \phi(x; \mu) = \mu f(x) + v(x), \]  

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

\[ v(x) = \sum_{i \in P_x} c_i(x) \quad \text{where} \quad P_x = \{ i \in \{1, \ldots, p \} : c_i(x) > 0 \}. \]  

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

\[ \nabla \phi(x; \mu) = \mu \nabla f(x) + \sum_{i \in P_x} \nabla c_i(x); \]  

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, \( \text{a priori} \), an acceptable value for the penalty parameter \( \mu \) such that minimizers of (2) correspond to feasible minimizers of (1) \cite{5, 6}, 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

\[ \min_{d \in \mathbb{R}^n} q(d; x_k, \mu) \]  

where

\[ q(d; x_k, \mu) := \phi(x_k; \mu) + \nabla \phi(x_k; \mu)^T d + \frac{1}{2} d^T H_k d \]  

\[ = \mu f(x_k) + \sum_{i \in P_x} c_i(x) + \left[ \mu \nabla f(x_k) + \sum_{i \in P_x} \nabla c_i(x) \right]^T d + \frac{1}{2} d^T H_k d \]  

and \( H_k \) is a BFGS approximation to the Hessian of (2) at \( x_k \). 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.

\[ \text{Note that we use} \ H_k \ \text{to denote an approximation to the Hessian at the} \ k\text{-th iterate, as opposed to the notation used in} \ [17] \ \text{and} \ [23, \ page \ 140] \ \text{where} \ H_k \ \text{is used to denote a BFGS approximation to the reverse of the Hessian.} \]
One might consider a simple strategy of using a sequential fixed penalty parameter (SFPP) restarting scheme with BFGS; i.e., one may consider iteratively lowering $\mu$ 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 $\mu$ 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 the one hand, concretely 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 [8] and [7]. 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 [11], which is obtained via solving the quadratic program (QP)

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

where the corresponding dual is given by

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

and where $e \in \mathbb{R}^p$ is a vector of ones, $s \in \mathbb{R}^p$ 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 $\lambda_k$ via the relationship

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

Note that when $\mu = 1$ and there are no constraints—i.e., when $p = 0$—equation (2) yields the standard BFGS search direction. In the presence of constraints, the resulting $d_k$ computed from (7) (or by solving (8) and employing (2)) provides a descent direction for (2) at $x_k$ with the current penalty parameter $\mu$. 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.,
\[ l(d; x_k) := \| \max\{ c(x_k) + \nabla c(x_k)^T d, 0 \} \|_1 \] (10)
at \( x_k \) given a direction \( d \). (Observe that the magnitude of this quantity relates to that of \( e^T s \) in the solution to (7)). Furthermore, that balance is shifted as the penalty parameter is changed: at one extreme, when \( \mu \) is set high, the search direction \( d_k \) will be heavily weighted towards minimizing the objective, regardless of whether it results in pushing towards feasibility or not, while at the other extreme when \( \mu = 0 \), \( d_k \) will almost entirely be weighted towards satisfying the linear model of constraint violation, and thus it would tend to represent a direction towards the feasible set. (Note that in the case when \( \mu = 0 \), we say “almost entirely” as \( H_k \) will contain curvature information for both the objective and violated constraints.) Thus, a good so-called “steering strategy” will dynamically adjust \( \mu \) to ensure that search directions computed by solving (7) (or (8) and employing (9)) maintain a good balance of minimizing the objective while simultaneously pushing towards feasibility over the course of the optimization.

We now present our proposed approach for updating the penalty parameter, which is based on the steering methods proposed in [7, 8]. 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
\[ l_\delta(d; x_k) := l(0; x_k) - l(d; x_k) = v(x_k) - \| \max\{ c(x_k) + \nabla c(x_k)^T d, 0 \} \|_1. \] (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 \( \mu \) 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 likelihood of divergence ensuing in the case that \( f(\cdot) \) is unbounded below.

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
The steering strategy first checks whether the computed search direction is predicted to at least reduce a reasonable fraction of the violation, which is specified by the constant $c_v$. If not, the method computes the reference direction with $\mu = 0$ to obtain search direction $\tilde{d}_k$. If the predicted reduction in violation given by $\tilde{d}_k$ is at least a reasonable fraction of the predicted reduction given by $d_k$, then $\tilde{d}_k$ is returned and the penalty parameter is not updated. However, if, by comparison, the reference direction still shows a much greater predicted reduction in violation than what is offered by $d_k$, the penalty parameter is reduced and a new search direction $d_k$ is computed iteratively until the specified sufficient fraction of predicted violation reduction for $d_k$ compared to the reference direction is achieved.

Consider an alternative stopping strategy compared to the usual criteria for optimizing smooth functions. To that end, following the approach taken in [19] for the unconstrained problem, consider the $l$ most recent iterates of the algorithm that are considered “close” in some sense of distance and define

$$G := [\nabla f(x_{k+1-1}) \ldots \nabla f(x_k)]$$

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

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 neighborhood around the current iterate. The motivation is similar in gradient sampling where one aims to approximate subdifferential information by the random sampling of gradients in a neighborhood of a given point. Here, however, we are reusing the gradients of the objective and constraints of previously computed points $\{x_{k+1-1}, \ldots, x_k\}$ 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 such as $10^{-4}$. If the solution of the resulting QP is sufficiently small in norm, then we have reason to believe that we are in a small neighborhood of a stationary point for the constrained optimization problem. For consistency with (7), we employ the $H_k$-norm,

\[
\begin{align*}
\text{Procedure 1} \quad [d_k, \mu_{\text{new}}] &= \text{s qp} \_\text{s teering} \_\text{s trategy}(x_k, H_k, \mu_{\text{current}}) \\
\text{Input:} & \\
& \quad \text{Current iterate } x_k \text{ and BFGS Hessian approximation } H_k \\
& \quad \text{Current value of the penalty parameter } \mu_{\text{current}} \\
& \quad \text{Constants: values } c_v \in (0, 1) \text{ and } c_\mu \in (0, 1) \\
\text{Output:} & \\
& \quad \text{Search direction } d_k \\
& \quad \text{Penalty parameter } \mu_{\text{new}} \in (0, \mu_{\text{current}}] \\
1: & \quad \text{Solve QP given in (7) using } \mu := \mu_{\text{current}} \text{ to obtain search direction } d_k \\
2: & \quad \text{if } l_\delta(d_k; x_k) < c_v v(x_k) \text{ then} \\
3: & \quad \text{Solve (7) using } \mu = 0 \text{ to obtain reference direction } \tilde{d}_k \\
4: & \quad \text{while } l_\delta(d_k; x_k) < c_v l_\delta(\tilde{d}_k; x_k) \text{ do} \\
5: & \quad \mu_{\text{current}} := c_\mu \mu_{\text{current}} \\
6: & \quad \text{Solve QP given in (7) using } \mu := \mu_{\text{current}} \text{ to obtain search direction } d_k \\
7: & \quad \text{end while} \\
8: & \quad \text{end if} \\
9: & \quad \mu_{\text{new}} := \mu_{\text{current}}
\end{align*}
\]
which leads to the dual formulation, written as

$$\max_{\sigma \in \mathbb{R}^l, \lambda \in \mathbb{R}^p} \sum_{i=1}^m c_i(x_k)e^T \lambda_i - \frac{1}{2} [\sigma, \lambda] [G, J_1, \ldots, J_p]^T H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}$$  \hspace{1cm} (12)

s.t. \hspace{0.5cm} 0 \leq \lambda_i \leq e, \hspace{0.5cm} e^T \sigma = \mu, \hspace{0.5cm} \sigma \geq 0

where the smallest vector in the convex hull of gradients is given explicitly by

$$d_o = H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}.$$  \hspace{1cm} (13)

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 [19] 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 \(d_k\). For brevity, we don’t explicitly give pseudocode for the routine inexact linesearch() that appears in Procedure 2 and instead refer the reader to [13, 19] for details. The inexact line search returns the next iterate of BFGS-SQP, that is, the point \(x_{k+1}\) which satisfies the weak Wolfe conditions for the penalty function, along with the values of the penalty function, its gradient and the violation amount at \(x_{k+1}\) and the length of the step taken, which we denote by the variable \(s\) in Procedure 2. Finally, BFGS-SQP is terminated once both \(||d_o||_2 \leq \tau\) 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 we have made in its design.

**Remark 3.1** For brevity, we omit the primal form of (12) and instead refer the reader to the thorough discussion in [9] for more details. As a consequence, note that we have actually defined BFGS-SQP in terms of both the primal and the dual (for the steering QP given by (7) and the stationarity QP given by (12), respectively) as we feel doing so provides a much more concise explanation of how our algorithm works. In a practical implementation of the algorithm, one should formulate and solve only the primal forms or only the dual forms. For our code, we used the dual as noted in §6.1 and thus our implementation of BFGS-SQP only needs inverse Hessian approximations.

### 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 [1, 4, 24]

\[ A + BXC, \]
Our first test problem requires the following definition.

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, it is 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, constrained optimization problem

$$\min_{X \in \mathbb{R}^{M,P}} \max_{i \in \{p+1, \ldots, p+q\}} \{\rho(A_i + B_iXC_i) : i \in \{p+1, \ldots, p+q\}\}$$

subject to $\rho(A_i + B_iXC_i) \leq 1$, $i \in \{1, \ldots, p\}$

(14)

where each static output feedback plant $A_i + B_iXC_i$ 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 [24]. Because the matrices are real, the eigenvalues are either real or occur in complex conjugate pairs. It was observed in [24] 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 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. This phenomenon also tends to occur when the spectral radius is optimized. See [24] for further discussion.

In Figure 1 we compare SFPP (where the number of iterations for each fixed value of the penalty parameter is limited to 20), BFGS-SQP, and SQP-GS for designing a
controller for (14) on a randomly generated example of dimension $N = 13$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $MP = 23 \times 2 = 46$ variables and was initially set to the zero matrix, a feasible starting point. (See §6.1 for more details on the experimental setup.) We see that SFPP immediately stepped outside of the feasible set and then, even with a relatively frequent penalty parameter update, struggled for well over 100 iterations before any real progress towards feasibility was made, due to the penalty parameter remaining too high for those iterations. In contrast, BFGS-SQP, although also initially stepping outside of the feasible set, immediately made quick progress back towards it while SQP-GS in fact maintained feasibility at almost every iterate, and both algorithms simultaneously minimized the objective significantly more than SFPP. By viewing the time at which BFGS-SQP finished its 500 iterations relative to SQP-GS, it is manifestly apparent that BFGS-SQP is well over an order of magnitude faster in terms of CPU time. For this particular problem, we see that BFGS-SQP happens to reduce the objective more than SQP-GS, which appears to reach its iteration limit before it has converged.

In Figure 2, we show the final spectral configurations of the three controllers ultimately produced by SFPP, BFGS-SQP, and SQP-GS for this particular example. These are all plots in the complex plane, symmetric about the real axis because the matrix data and controller $X$ are all real. In all configurations, in accordance with Remark 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 can’t be observed from the plot, but a close-up view indicates that, unlike the others, the positive real eigenvalue does not quite have its modulus attaining the spectral radius. Furthermore, we see that the spectral radii of the different plants in the objective are nearly the same, 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 8.1 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

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

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
Figure 1. The plots in the top row track the value of the spectral radius based objective function in terms of iteration number for SFPP (left), BFGS-SQP (middle), and SQP-GS (right) on a randomly generated example of dimension $N = 13$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $MP = 23 \times 2 = 46$ variables. The vertical dashed black line in the top right plot indicates the SQP-GS iteration number at which time BFGS-SQP already terminated. The horizontal dashed black line in the same plot indicates the value of BFGS-SQP's best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.

an upper bound $\varepsilon$ on the smallest singular value of $A - \lambda I$.

**Remark 4.7** Like the spectral radius, the pseudospectral radius is nonconvex, nonsmooth, and differentiable almost everywhere. However, in contrast to the spectral radius, the pseudospectral radius is locally Lipschitz [12] 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 [20].

**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, constrained optimization problem

$$
\min_{X \in \mathbb{R}^{M,P}} \max \{ \rho_\varepsilon(A_i + B_i X C_i) : i \in \{p+1, \ldots, p+q\} \}
$$

s.t. $\rho_\varepsilon(A_i + B_i X C_i) \leq 1, \ i \in \{1, \ldots, p\}$ \hspace{1cm} (15)

which we may view as a locally Lipschitz regularization of (14). The goal is still to still design a matrix $X$ such that stability of the static output feedback plants in the objective are all enhanced while simultaneously ensuring that the plants appearing in the constraint functions remain stable, except here stability means that any plant must
Figure 2. The three rows show the final spectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The dashed black circle on the objective plots (barely visible) corresponds to the max spectral radius of the three plants in the objective for that particular algorithm’s controller.

remain stable under any perturbation up to norm $\varepsilon = 10^{-1}$.

In Figure 3, we again compare SFPP, BFGS-SQP, and SQP-GS (using the same experimental setup as used in §4.2 and described in §6.1) for designing a controller for (14) on a randomly generated example of dimension $N = 5$ comprised of three plants in the objective and one in the constraint and where the controller matrix has $MP = 13 \times 2 = 26$ variables and was initially set to the zero matrix, again a feasible point. For this particular pseudospectral radius optimization problem, we see that SFPP performed even worse than it did on the spectral radius example problem after initially stepping outside of the feasible set, with over 160 iterations incurred before the penalty parameter was sufficiently lowered to promote any significant progress towards satisfying feasibility. BFGS-SQP again initially stepped outside the feasible region but quickly progressed back towards it while SQP-GS maintained feasibility a majority of the time, and both algorithms simultaneously minimized the objective more than SFPP once again. Interestingly, as on the spectral radius problem, BFGS-SQP found a much better controller than SQP-GS, even though SQP-GS’s convergence results hold for this particular problem 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, we show the final pseudospectral configurations of the three controllers ultimately produced by SFPP, BFGS-SQP, and SQP-GS for this particular example. For almost all the plants, we see that the resulting pseudospectral configurations show that

\[^{2}\text{Generated via MATLAB’s \texttt{contour} by evaluating the smallest singular value of } A - \lambda I \text{ on a grid in the complex plane, exploiting Remark 4.6. Note that because the matrices are real, the pseudospectra boundaries are symmetric with respect to the real axis of the complex plane.}\]
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 also shows activities, with the pseudospectral radius attaining the upper bound value 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 8.2 for additional pseudospectral radius examples similar to the one shown here in Figures 3 and 4.)

5. Comparing nonsmooth, nonconvex, constrained optimization algorithms

For comparing codes for solving convex optimization problems, a popular visualization tool is a performance profile \([10]\), 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 such as 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 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.
Figure 4. The three rows show the final pseudospectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm’s controller.

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 behavior.

Making any comparison of optimization codes for nonconvex problems, however, can be quite a challenge, let alone attempting to make a fair comparison. When solving nonconvex problems with gradient-based local search methods, one can neither expect that the codes will find global minimizers for the test problems nor can one expect the codes to find the same local minimizers for a given problem or even local minimizers at all. The general state of affairs is that for any nonconvex problem, the codes being evaluated may each return their own distinct approximate stationary point. Thus, there is an inherent ambiguity as to whether the time to run a code on a nonconvex, nonsmooth problem is attributable to the algorithm’s implementation/design or to the particular ease or difficulty inherent in the problem itself for finding any given stationary point. As a consequence, even in the event that one algorithm consistently finds better (lower) minimizers and/or stationary points across a test set of nonconvex problems, there is often little to no grounds for attributing such a success to the properties and design of the algorithm. Unfortunately, performance profiles do not address these issues. Furthermore, our setting is not just nonconvex optimization, but nonsmooth, nonconvex, constrained optimization where we must additionally assess the feasibility of the different candidate
minimizers produced by different solvers in context of how much computational cost was spent to obtain these solutions and how well they minimize their respective objectives.

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 those solutions, without focusing on cost, be it running time, number of function evaluations, or memory use. We propose producing a plot, which we call a relative minimization profile (RMP), that relates the percentage of feasible solutions 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:

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

For method \( m \) on problem \( p_i \in T \), we define its best computed objective value, in terms of some violation tolerance \( \tau_v \in \mathbb{R}^+ \):

\[ f_i^m(\tau_v) := \min_{x \in \{x_k\}_i^m} f_i(x) \text{ s.t. } v_i(x) \leq \tau_v. \tag{16} \]

Furthermore, we define the following relative residual function and its associated indicator function:

\[ r(\varphi, \tilde{\varphi}) := \begin{cases} \infty & \text{if } \varphi = \infty \text{ or } \tilde{\varphi} = \infty \\ \frac{\varphi - \tilde{\varphi}}{\varphi} & \text{otherwise}, \end{cases} \]
\[ 1_{r}(\varphi, \tilde{\varphi}, \gamma) := \begin{cases} 1 & \text{if } r(\varphi, \tilde{\varphi}) \leq \gamma \\ 0 & \text{otherwise}. \end{cases} \]

We may now formally define a relative minimization profile 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 \( m \in M \), a relative minimization profile curve \( r_{\Omega, \tau_v}^m \):
\( \mathbb{R}^+ \to [0,1] \) is that given by

\[
r_{m,\infty}^{\Omega,\tau_v}(\gamma) := \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \mathbb{1}_\tau(\omega_i, f^m_i(\tau_v), \gamma),
\]

(17)

where \( \gamma \) specifies the maximum relative difference allowed to the target set values \( \Omega \). (Note that \( r_{m,\infty}^{\Omega,\tau_v}(\gamma) \) is well-defined at \( \gamma = \infty \).)

The RMP curve, which is visually interpreted like a receiver operating characteristic (ROC) (e.g., see [15]) curve, is the cumulative proportion of the number of best feasible (to violation tolerance \( \tau_v \)) solutions returned by method \( m \), as defined by (16), that are each within a relative residual \( \gamma \) of their respective target values \( \omega_i \in \Omega \). In other words, an RMP curve portrays how the percentage of feasible solutions 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 \( \Omega \) 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 \( \Omega \). However, in general, such data is not necessarily available or attainable. Moreover, using such a defined target set can be problematic if none of the codes find solutions that (nearly) attain the target values. Thus, we instead propose defining \( \Omega \) 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 \( p_i \):

\[
f_i^\mathcal{M}(\tau_v) := \min_{m \in \mathcal{M}} f_i^m(\tau_v)
\]

(18)

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

\[
\Omega := \{ \omega_i : \omega_i = f_i^\mathcal{M}(\tau_v) \}.
\]

(19)

By using (19), the RMP curve defined in (17) measures the quality of the solutions 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 \( \gamma \) in an idiomatic and compact fashion, we suggest that \( \gamma \) should be plotted using a base-10 log scaling, initially 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 solutions, 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 to what frequency codes are either finding the same quality of solutions (and to what precision) or different solutions 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 plot 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 to use relative differences for defining RMPs. Furthermore, outside of the mild requirement to set a cutoff tolerance for the amount of violation allowed for each solution, no such hard cutoff 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 solutions found. In contrast, conventional performance profiles have the drawback that each code’s solutions are entirely subjected to a binary classification into successes or failures, which can oversimplify and possibly even misleading 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 $\beta$-RMPs

In order to extend RMPs for additionally comparing the cost of each method, we will generalize (17) into what we call a $\beta$-RMP curve. The idea is to use the parameter $\beta$ 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 $\beta$-RMP plots can be produced for various values of $\beta$ to create a $\beta$-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 $m \in M$ on problem $p_i \in T$, let us first define a generic function $t_i^m : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$ that represents the cumulative cost (in the performance metric of one’s choosing) to compute method $m$’s first $j$ iterates:

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

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

$$X_i^m(t) := \begin{cases} \{x_k\}^m_i & \text{if } t = \infty \\ \{x_j : x_j \in \{x_k\}^m_i \text{ and } t_i^m(j) \leq t\} & \text{otherwise.} \end{cases} \quad (20)$$

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 to be in terms of some violation tolerance $\tau_v \in \mathbb{R}^+$ and some computational cost limit $t \in \mathbb{R}^+$:

$$f_i^m(\tau_v, t) := \min_{x \in X_i^m(t)} f_i(x) \text{ s.t. } v_i(x) \leq \tau_v \quad (21)$$

with the two conventions that $f_i^m(\tau_v, t) := \infty$ if $v_i(x) > \tau_v$ for all $x \in X_i^m(t)$ and when argument $t$ is omitted, it is taken to be $\infty$, 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 born 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 $\beta$-RMPs, which will necessitate defining a computational budget $B$ in a per-problem fashion:

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

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 $B$ so that efficiency of all other methods will be assessed relative relative to the efficiency of method $m_\star$. Mathematically, we define the baseline maximum computational cost for problem $p_i \in T$ relative to a chosen method $m_\star \in M$ as follows:

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

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 $\beta$-RMP curves relative to the budgets in $B$.

**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 $B := \{b_i\}$, constant factor $\beta \in \mathbb{R}^{++}$, and method $m \in M$, a $\beta$-relative minimization profile curve $r_{\Omega, \tau_v}^{m, \beta} : \mathbb{R}^+ \to [0, 1]$ is that given by

$$r_{\Omega, \tau_v}^{m, \beta}(\gamma) := \frac{1}{|T|} \sum_{i=1}^{|T|} \mathbb{1}_v(\omega_i, f_i^m(\tau_v, \beta b_i), \gamma).$$

Note that if $\beta \geq 1$ and $B$ is defined by using (22) with $m = m_\star$, then we have the following equivalence:

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

In other words, for this specific case, (23) is equivalent to (17) since increasing the per-problem budgets with respect to each $b_i \in B$ has no effect as $B$ is the set of per-problem budgets method $m$ required on the test set. In contrast to the earlier $\infty$-RMP curve defined in (17), a $\beta$-RMP curve is still the cumulative proportion of the number of best objective values found by method $m \in M$ in the (nearly) feasible sets, which
are within a relative residual $\gamma$ 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 $\beta$ applied to the per-problem budgets $b_i$ given by $B$.

A natural choice for defining the target set $\Omega$ for a $\beta$-RMP benchmark panel is to reuse the best objective values known to be achievable on the (nearly) feasible sets, that is, to set each $\omega$ to the values given by (18). However, while this has the benefit of consistent y-axis scaling for all $\beta$-RMP plots, the desired $\beta$-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 updates as parameter $\beta$ is changed.

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

$$X_i^M(t) := \bigcup_{m \in M} X_i^m(t),$$

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

$$f_i^M(\tau_v, t) := \min_{x \in X_i^M(t)} f_i(x) \text{ s.t. } v_i(x) \leq \tau_v.$$ 

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

$$\omega_i := f_i^M(\tau_v, \beta b_i),$$

that is, each problem’s target value is the best known feasible objective value encountered by any of the methods so far within the computational limit given by $\beta b_i$. The effect of using the rolling target value set defined by (26) is that each $\beta$-RMP plot is automatically and individually scaled to maximize the separation of each method’s $\beta$-RMP curve.

For $\beta = \infty$, a $\beta$-RMP depicts how the codes compare given no computational constraint whatsoever. In the case where (22) is employed to define $\beta$-RMPs relative to a chosen method $m_\star$, a $\beta$-RMP plot for $\beta = 1$ shows how the other methods perform relative to $m_\star$ 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 $\beta$-RMP plots; other values of $\beta$ must be chosen or generated automatically but it is usually straightforward and fairly effortless to manually find a handful of additional $\beta$ values to make an illuminating $\beta$-RMP benchmark panel. A well-made $\beta$-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 $\beta$.

5.3 Practical considerations for $\beta$-RMPs

While relative minimization profiles precisely 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 $\beta$-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
the solution 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 the case where one wishes to use CPU-time as the computational cost, and the codes being compared do not provide detailed timing data of iterates, it can sometimes be practical to couple the implementations of the objective and constraint functions to user-accessible persistent storage (e.g., via object-oriented programming) which can record each evaluation and the time at which it occurred. However, doing so may introduce significant overhead and there is often little to no way of reliably determining which evaluations correspond to the actual iterates accepted by the algorithm at every step. Provided that the codes at least return a history of the iterates, calculating the average time per iterate and using it as a proxy to determine what progress an algorithm has made by any given time can often be a suitable and easily attained approximation. Though there may be variability in the time each iteration takes, if the data set is sufficiently large and a high number of iterations is typically needed for any method to converge on the problems, then such an approximation is likely to be good enough for qualitative running time comparisons for a set of plots made with different per-problem time limits. Similar approaches can be used to get or approximate the requisite data for each iterate for other computational cost metrics.

Continuing the discussion with the example of CPU-time cost, despite any inaccuracies in the timing data, if the methods are indeed significantly different in terms of speed, that property itself will most likely be the dominant factor in determining what is presented by the plots, not the inherent limitation of having to estimate the exact timing data. On the other hand, if comparing the relative speeds of the algorithms requires high precision, then it suggests that running time is not a particularly discriminating performance characteristic amongst the codes being compared. In such cases, a single RMP, without 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.

Finally, some comments on data profiles [22] are also in order. Data profiles are visualization tools that are also interpreted as ROC curves and are thus similar in appearance to performance profiles, but instead attempt to measure the overall progress of an algorithm towards solutions in a test set with respect to some constraint on the computational resources. Specifically, they propose using the number of function and/or gradient evaluations to assess the cost of the algorithms and measure the percentage of problems in the test set that are solved to some specified level of accuracy. As the allowed number of function and/or gradient evaluations is increased, the algorithms can potentially return more and more accurate solutions, and thus the percentage of problems considered to be solved sufficiently well monotonically increases. By creating multiple data profiles, one plot for each different level of accuracy permitted to be considered an acceptable solution to any given problem, one can assess which algorithms are fastest for obtaining only approximate solutions versus which are fastest for resolving accurate solutions to a high precision. However, data profiles do not provide a direct way for additionally assessing feasibility for constrained problems nor do they address the possibility of obtaining different quality of solutions in the nonconvex setting.

Data profiles are precise in terms of the cost to run the algorithms but give only an
approximate indication of the accuracy of the 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 solutions found and the quality of minimization achieved over the data set (and their relationship to each other) while only a handful of $\beta$-RMPs plots are needed to highlight performance differences of algorithms as the computational budget is changed. We believe this latter approach is typically a more useful prioritization for comparing algorithms, at least in nonconvex and/or nonsmooth settings and potentially in unconstrained convex settings as well.

6. Numerical results

6.1 Experimental setup

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. To generate the number of controller variables, $MP$, such that a higher number of variables would usually be chosen for problems that either had larger-dimensional $A_i$ matrices or which had more plants in the objective function, a candidate value for $MP$ was picked by randomly choosing an integer $Q$ between $\left\lceil \frac{4}{3}(p + 2q) \right\rceil$ and $\left\lceil \frac{4}{3}(3p + 6q) \right\rceil$ inclusively. In order to provide variety in the shapes of controller matrices $X$ generated, $M$ was set to a random integer chosen from $[1, \left\lceil \sqrt{Q} \right\rceil]$ 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 used $\mu = 16$ as the initial penalty parameter value.

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

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.
To implement BFGS-SQP, we modified the BFGS routine from HANSO, replacing the standard BFGS search direction computation with the steering strategy of Procedure 1. However, as the BFGS version from HANSO actually approximates the inverse of the Hessian, we chose to adapt Procedure 1 and Procedure 2 using the dual form of the QPs, instead of the primal as stated in the pseudocodes. For tolerances, we set $c_v := 0.1$ and $c_\mu := 0.5$ in Procedure 1 and limited its loop to at most ten iterations.

As $\beta$-RMPs effectively simulate stopping criteria based on the amount of cost spent to optimize, which can be changed via adjusting the plots’ violation tolerance and varying the values used for parameter $\beta$, we set the termination and violation tolerances for BFGS-SQP, SQP-GS, and SFPP to machine precision and zero respectively ($\tau_\epsilon$ and $\tau_v$ for BFGS-SQP’s Procedure 2). Even though such tight tolerances are quite unusual in practice, encouraging all algorithms to run as long as possible not only allows for the most data to be collected for generating the $\beta$-RMPs but crucially, it decouples the 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 scaling computational budget defined by $\beta$-RMPs themselves. Using $\beta$-RMPs in this fashion also obviates any need to consider how to fairly set each method’s parameters for the benchmark, outside of setting the prerequisite max allowed number of iterations for each method. All three methods were allowed to run for up to 500 iterations each.

To specifically validate our proposed algorithm BFGS-SQP, we chose to define the computational budget $B$ so that the $\beta$-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 $\Omega$ based on the best-encountered iterate by any algorithm so far, as defined by $B$ and the particular value of $\beta$ using (26). We made $\beta$-RMP plots for $\beta = 1, 5, 10, \infty$ using a violation tolerance of zero since all the problems were only inequality constrained. Neither SQP-GS or HANSO’s BFGS codes provide per-iterate timing data, so, as outlined and justified in §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 $\beta$-RMP benchmark panel for comparing the relative rates of progress for the algorithms.

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

### 6.2 Spectral radius optimization

We begin analyzing the relative performance of the three codes over the spectral radius test set by first turning our attention to the bottom right RMP in Figure 5. Here, each algorithm was allowed to run without any time restriction and we see that despite the promising example shown earlier in Figure 4, SQP-GS ended up finding over 80% of the best optimizers while BFGS-SQP only found just under 20% of the best optimizers. Furthermore, we see that BFGS-SQP’s performance over the data set only starts to look better if we greatly relax how much of the minimization of the best known solution is necessary to be considered a success, indicating that BFGS-SQP in general either converged to different (worse) points or perhaps was stagnating early compared to SQP-GS. On the other hand, SFPP appears to be completely uncompetitive by comparison to either BFGS-SQP and SQP-GS. Though this doesn’t initially appear promising for BFGS-SQP, the fact remains that SQP-GS actually took 27.4 times longer to run.
Figure 5. Overall performance assessment on the spectral radius optimization problems of BFGS-SQP, SQP-GS, and SFPP, with relative time factors of 1, 5, 10, and ∞ compared to the per-problem times of BFGS-SQP. In the top left panel for a time factor of 1, we see that the other two algorithms have barely made progress towards feasibility and minimizing the objective function when BFGS-SQP has terminated. In the top right, even with the opportunity to run 5 times longer per problem than BFGS-SQP, we see that SQP-GS and SFPP are still nowhere near as competitive. In the bottom left, when given the opportunity to run 10 times longer per problem than BFGS-SQP, we see that SQP-GS is starting to come close to matching BFGS-SQP. In the bottom right panel, where all the algorithms have no limits on running time besides the 500 max iteration count, we finally see SQP-GS outperforming BFGS-SQP and by a significant margin, though it took 27.4 times longer to do so. Even without a time restriction, SFPP is indeed extremely ineffective and, interestingly, ran in only 0.435 of the time that BFGS-SQP took, perhaps indicating that SFPP often suffered from early stagnation and/or breakdown on this test set of non locally Lipschitz optimization problems.

To this point, observing the progress that SQP-GS made when it was only allowed to run 10 times longer than BFGS-SQP per problem, we see in the bottom left RMP in Figure 5 that SQP-GS performed worse than BFGS-SQP, even though it had a tenfold time advantage. If we further consider the RMP in the top right, where SQP-GS is only allowed up to five times as much time as BFGS-SQP per problem, we see that the relative performance of the two algorithms has practically reversed, where it is now clear that BFGS-SQP performed significantly better than SQP-GS, even though SQP-GS still is being evaluated with a fivefold time advantage. Finally, in the top left RMP, where SQP-GS and SFPP are assessed at the per-problem times at which BFGS-SQP terminated, we see that BFGS-SQP completely outclasses SQP-GS due to its much higher efficiency. Impressively for both BFGS-SQP and SQP-GS is that they are able to be effective at all, and in fact significantly so, when attempting to optimize functions that are non-Lipschitz, although neither algorithm has convergence guarantees in this setting.

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 slightly outperformed SQP-GS over the test set. To put that in context, BFGS-SQP produced just as good or better minimizers compared to SQP-GS, and in fact did so
more frequently, while also being 29.6 times faster than SQP-GS. This is a remarkable outcome given that the convergence results for SQP-GS hold for the problems in the pseudospectral radius test set (as they are locally Lipschitz) while BFGS-SQP provides no theoretical guarantees. Indeed, it is a stark contrast to what we observed on the spectral radius test set, where SQP-GS could ultimately still pull ahead compared to BFGS-SQP, when allowed to run 27.4 times longer. Interestingly, we also see that SFPP appears to do better on the Lipschitz problems though it is still quite unimpressive to say the least. As we look at smaller relative time factors for the other RMPs in Figure 6, we see that the comparison is even more in favor of BFGS-SQP. In other words, SQP-GS was not only 29.6 times slower than BFGS-SQP on the pseudospectral radius problems, but it apparently needed that much of a time advantage to be at all competitive in terms of the quality of the computed solutions.

### 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 [19], we explore the effect of regularizing the Hessian approximation in BFGS-SQP. However, as BFGS-SQP requires solving QPs inside the BFGS iteration, there is also a potential tradeoff of regularizing to improve the QP solves for determining a search direction versus not regularizing to
retain the ill-conditioning in the BFGS Hessian approximation. To assess the effect of regularization, we reran the BFGS-SQP experiments multiple times, where for each the Hessian approximation was regularized such that its condition number was no more than $10^{2j}$, for $j \in \{0, \ldots, 8\}$, where the case of $j = 0$ corresponds to replacing the Hessian approximation by a multiple of the identity.

For the spectral radius problems, as shown in Figure 7, 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 Figure 7 that moderate levels of regularization (such as $10^6$) seems to have a strikingly clear beneficial effect compared to not regularizing at all. It is hard to say why we observe this behavior but it is conceivable that the regularization helped improve the accuracy of the QP solves more than it hurt the effectiveness of BFGS. However, it is apparent that the observed effects of regularization is problem dependent since there is no parallel to the regularization results for the spectral radius test set. Certainly more investigation into this matter would be a worthwhile pursuit.

7. Conclusion

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

[1] 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 Verlag, London, 1999, pp. 53–56.
[2] J. Burke, A. Lewis, and M. Overton, A robust gradient sampling algorithm for nonsmooth, nonconvex optimization, SIAM Journal on Optimization 15 (2005), pp. 751–779.
[3] J. Burke and M. Overton, Variational analysis of non-Lipschitz spectral functions, Mathematical Programming 90 (2001), pp. 317–352.
[4] J. Burke, D. Henrion, A. Lewis, and M. Overton, HIFOO - A MATLAB Package for Fixed-Order Controller Design and H∞ Optimization, in Fifth IFAC Symposium on Robust Control Design, Toulouse, 2006.
[5] 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.
[6] 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.
[7] R. 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.
[8] R. Byrd, J. Nocedal, and R. 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.
[9] F. Curtis and M. 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.
[10] E. Dolan and J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2002), pp. 201–213, Available at http://dx.doi.org/10.1007/s101070100263.
[11] R. Fletcher, Practical Methods of Optimization, 2nd ed., John Wiley, Chichester and New York, 1987.
[12] M. Gürbüzbalaban and M. Overton, Some regularity results for the pseudospectral abscissa and pseudospectral radius of a matrix, SIAM Journal on Optimization 22 (2012), pp. 281–285, Available at http://dx.doi.org/10.1137/110822840.
[13] HANSO (Hybrid algorithm for non-smooth optimization), http://www.cs.nyu.edu/overton/software/hanso/ [Online; accessed 10-September-2014].
[14] J. Hiriart-Urruty and C. Lemaréchal, Convex Analysis and Minimization Algorithms, Springer-Verlag, New York, 1993, two volumes.
[15] F. Hsieh and B. 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/aoas/1033066197.
[16] K. Kiwiel, Methods of Descent for Nondifferentiable Optimization, Lecture Notes in Mathematics 1133, Springer-Verlag, Berlin and New York, 1985.
[17] K. Kiwiel, Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization, SIAM Journal on Optimization 18 (2007), pp. 379–388.
[18] A. Lewis, Active sets, nonsmoothness and sensitivity, SIAM Journal on Optimization 13 (2003), pp. 702–725.
[19] A. Lewis and M. 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.
[20] E. Mengi and M. Overton, Algorithms for the computation of the pseudospectral radius and the numerical radius of a matrix, IMA Journal on Numerical Analysis 25 (2005), pp. 648–669.
[21] T. Mitchell, Robust and efficient methods for approximation and optimization of stability measures, Ph.D. thesis, New York University, New York University, 2014.
[22] J. Moré and S. Wild, Benchmarking derivative-free optimization algorithms, SIAM J. Optim. 20 (2009), pp. 172–191, Available at http://dx.doi.org/10.1137/080724083.
[23] J. Nocedal and S. Wright, Nonlinear Optimization, 2nd ed., Springer, New York, 2006.
[24] M. Overton, Stability optimization for polynomials and matrices, in Nonlinear Physical Systems: Spectral Analysis, Stability and Bifurcations, O. Kirillov and D. Pelinovsky, eds., chap. 16, Wiley-ISTE, London, 2014, pp. 351–375.
[25] A. Ruszczynski, Nonlinear optimization, Princeton University Press, 2011.
[26] L. Trefethen and M. Embree, Spectra and Pseudospectra: the Behavior of Nonnormal Matrices and Operators, Princeton University Press, 2005.
8. Appendix

8.1 Spectral Radius Additional Examples

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

Figure 9. The three rows show the final spectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The dashed black circle on the objective plots (barely visible) corresponds to the max spectral radius of the three plants in the objective for that particular algorithm’s controller.
Figure 10. The plots in the top row track the value of the spectral radius based objective function in terms of iteration number for SFPP (left), BFGS-SQP (middle), and SQP-GS (right) on a randomly generated example of dimension $N = 10$ comprised of one plant in the objective and four in the constraint and where the controller matrix has $MP = 16 \times 1 = 16$ variables. The vertical dashed black line in the top right plot indicates the SQP-GS iteration number at which BFGS-SQP terminated while the horizontal dashed black line indicates the value of BFGS-SQP’s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.

Figure 11. The three rows show the final spectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the plants in the objective while red is used to indicate plants in the constraints, with the plus signs indicating the eigenvalues and the colored circles indicating the spectral radius of each plant. The dashed black circle on the plots for constraints is the unit circle (the stability boundary). The dashed black circle on the objective plots (barely visible) corresponds to the max spectral radius of the three plants in the objective for that particular algorithm’s controller.
8.2 Pseudospectral Radius Additional Examples

Figure 12. The plots in the top row track the value of the pseudospectral radius based objective function in terms of iteration number for SFPP (left), BFGS-SQP (middle), and SQP-GS (right) on a randomly generated example of dimension $N = 6$ comprised of one plant in the objective and one in the constraint and where the controller matrix has $MP = 7 \times 1 = 7$ variables. The vertical dashed black line in the top right plot indicates the SQP-GS iteration number at which BFGS-SQP terminated while the horizontal dashed black line indicates the value of BFGS-SQP’s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.

Figure 13. The three rows show the final pseudospectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm’s controller.
Figure 14. The plots in the top row track the value of the pseudospectral radius based objective function in terms of iteration number for SFPP (left), BFGS-SQP (middle), and SQP-GS (right) on a randomly generated example of dimension $N = 7$ comprised of two plants in the objective and three in the constraint and where the controller matrix has $MP = 5 \times 4 = 20$ variables. The vertical dashed black line in the top right plot indicates the SQP-GS iteration number at which BFGS-SQP terminated while the horizontal dashed black line indicates the value of BFGS-SQP’s best feasible solution. The log-scaled plots in the bottom row show the amount of violation tracking with the iteration counts with green and red indicating whether each iterate is feasible or infeasible respectively.

Figure 15. The three rows show the final pseudospectral configurations of the three controllers found by SFPP (top), BFGS-SQP (middle), and SQP-GS (bottom) respectively 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. Blue is used to indicate the pseudospectral boundaries for the plants in the objective while red is used to indicate the pseudospectral boundaries of the plants in the constraints, with the plus signs indicating the eigenvalues. The dashed black circle on the plots for constraints is the unit circle, that is the stability boundary. The dashed black circle on the objective plots corresponds to the max pseudospectral radius of the three plants in the objective for that particular algorithm’s controller.