

A Comparison of Nonsmooth, Nonconvex, Constrained Optimization Solvers for the Design of Time-Delay Compensators

Vyacheslav Kungurtsev* Tim Mitchell† Tomáš Vyhlídal‡

December 30, 2018

Abstract

We present a detailed set of performance comparisons of two state-of-the-art solvers for the application of designing time-delay compensators, an important problem in the field of robust control. Formulating such robust control mechanics as constrained optimization problems often involves objective and constraint functions that are both nonconvex and nonsmooth, both of which present significant challenges to many solvers and their end-users hoping to obtain good solutions to these problems. In our particular engineering task, the main difficulty in the optimization arises in a nonsmooth and nonconvex stability constraint, which states that the infinite spectrum of zeros of the so-called shaper should remain in the open left half-plane. To perform our evaluation, we make use β -relative minimization profiles, recently introduced visualization tools that are particularly suited for benchmarking solvers on nonsmooth, nonconvex, constrained optimization problems. Furthermore, we also introduce new visualization tools, called Global-Local Profiles, which for a given problem and a fixed computational budget, assess the tradeoffs of distributing the budget over few or many starting points, with the former getting more budget per point and latter less.

1 Introduction

Consider the following general constrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \begin{cases} c_j(x) \leq 0 & j \in \{1, \dots, m\} \\ c_j(x) = 0 & j \in \{m+1, \dots, q\} \end{cases}, \quad (1.1)$$

where the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and each individual (in)equality constraint function $c_j : \mathbb{R}^n \rightarrow \mathbb{R}$ are all continuous but may possibly be nonsmooth. We assume that the functions are still continuously differentiable almost everywhere; as such, the nonsmoothness of f and each c_j is limited to a set of measure zero in \mathbb{R}^n . To date, much of the literature for *nonsmooth* optimization has focused on unconstrained minimization, without any c_j constraint functions; e.g. see [BKM14]. However, a couple of new solvers specifically intended for nonsmooth, constrained optimization have recently appeared, opening up new possibilities for reliably solving actual applications that are modeled by (1.1) in practice, despite the challenging properties of the objective and/or constraint functions.

*Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, vyacheslav.kungurtsev@fel.cvut.cz

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

‡Department of Instrumentation and Control Engineering, Faculty of Mechanical Engineering, and Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, tomas.vyhlidal@fs.cvut.cz

In 2012, [CO12] proposed combining gradient sampling (GS), a nonsmooth optimization technique for unconstrained minimization [BLO05], with sequential quadratic programming (SQP) to additionally handle (possibly nonsmooth) constraints. The resulting algorithm, simply called SQP-GS, was shown to have provable convergence to stationary points of (1.1), assuming that all the functions f and c_j are locally Lipschitz and that such stationary points are sufficiently *calm* (for details on calmness, see [Bur91a, Bur91b]). On the other hand, the gradient-sampling procedure employed within SQP-GS requires evaluating $\mathcal{O}(n)$ gradients (for each of the particular f and c_j functions that are nonsmooth) at every iteration, which can make SQP-GS a computationally expensive method, especially if the cost to compute these functions is nontrivial. This potential high cost is further exacerbated by the fact that nonsmooth optimization methods generally have slow convergence rates (i.e. typically linear), due to the lack of exploitable smoothly varying curvature at nonsmooth minimizers, and can thus incur many iterations before satisfactorily meeting stationarity conditions.

More recently, in 2017, a philosophically different algorithm [CMO17] for solving (1.1) was proposed, i.e. one that does not rely on gradient sampling and which instead typically only requires $\mathcal{O}(1)$ gradient evaluations on average per iteration, thus potentially making it significantly faster than SQP-GS. Inspired by the observed efficiency and reliability of quasi-Newton methods for nonsmooth unconstrained optimization [LO13], specifically Broyden-Fletcher-Goldfarb-Shannon (BFGS) updating, the authors of [CMO17] proposed a new BFGS-SQP method as a fast and reliable alternative to SQP-GS. This BFGS-SQP algorithm is implemented in a code called GRANSO: GRAdient-based Algorithm for Non-Smooth Optimization [Mitb]. Although there are currently no theoretical convergence guarantees for the underlying BFGS-SQP method, it was shown that GRANSO can be a competitive solver in practice compared to SQP-GS [CMO17, Section 6]. On a large test set of challenging nonsmooth, nonconvex, constrained optimization problems [Mitc], half of which were not even locally Lipschitz, GRANSO often fared well relative to SQP-GS in terms of minimization quality and finding the feasible region, all while being many times faster than SQP-GS (typically over an order of magnitude faster, which, for those problems, was in line with the expectation of being faster by a factor of $\mathcal{O}(n)$ as the dominant cost in the running times was the function/gradient evaluations). Additionally, the “off-the-shelf” solvers (i.e., those not specifically developed for nonsmooth problems) included as baselines in the evaluation done in [CMO17] did not perform well at all, highlighting the necessity for purpose-built solvers for nonsmooth optimization.

Returning to our specific motivation of using nonsmooth, constrained optimization solvers to design time-delay compensators, so-called *input shapers*, the overall goal is to shape the movements of a main-body system (e.g. a crane trolley) in order to damp the oscillatory modes of a connected flexible subsystem (e.g. a payload suspended from a crane trolley). Cast as an optimization problem, the design of such input shapers in this paper takes the following general form:

$$\min_{x \in \mathbb{R}^n} x^\top Q x \quad \text{s.t.} \quad \begin{cases} A_1 x & \leq b_1 \\ A_2 x & = b_2 \\ s(x) & \leq 0 \end{cases}, \quad (1.2)$$

where $Q \in \mathbb{R}^{n \times n}$ is positive definite, $A_1 \in \mathbb{R}^{m \times n}$, $A_2 \in \mathbb{R}^{p \times n}$, and $s : \mathbb{R}^n \rightarrow \mathbb{R}$ is a continuous but nonsmooth and nonconvex function that is possibly also not locally Lipschitz. The quadratic objective function models desired performance characteristics in the frequency domain, while the linear constraints specify necessary structural requirements. The nonlinear and nonsmooth inequality constraint s imposes a stability constraint on the shaper, namely that its spectrum, which is infinite for the given class of time-delay systems, must entirely reside in the open left half-plane. Note however that (1.2) is applicable to a wide range of related problems in control system theory

where fast and robustly stable transient characteristics must be balanced.

Without explicitly enforcing stability of the shaper, the resulting shaper may have unstable modes which could destabilize the closed-loop system [VHKA16a]. Consequently, in [KPV⁺17], designing a *stable* input shaper via optimizing a specific form of (1.2) with constraint s enforcing stability was proposed, which the authors noted was the first use of optimization to stabilize an input shaper via moving its zeros into the open left half-plane. However, attempting to solve (1.2) with SQP-GS, the only solver specifically designed for nonsmooth, nonconvex, constrained optimization that was available to the authors at the time of those initial experiments, was not a straightforward matter. First, since the constraint function s may not be locally Lipschitz, the convergence guarantees for SQP-GS might not even apply for (1.2). Second, these convergence guarantees, even if they did apply, do not say anything about whether or not a resulting computed solution will be feasible; in other words, the convergence guarantees do not preclude SQP-GS from converging to an infeasible stationary point. Third, as the authors of [KPV⁺17, Section 4, p. 13327] noted, SQP-GS was indeed quite slow in practice. While a stable input shaper was successfully obtained by optimizing a specific example of (1.2) via SQP-GS [KPV⁺17, Section 4], the difficulties of doing so were beyond the scope of the article and thus were not reported in detail, e.g. how many starting points were required in order to find at least one stable input shaper and exactly how long did it take SQP-GS to compute these shapers. These are relevant issues for engineers actually attempting to employ this new methodology and with the recent availability of GRANSO, which might potentially solve (1.2) significantly faster than SQP-GS, there is now also the question of which solver to use in this or similar applications. As such, the questions of what is the current best practice for solving input shaper design problems of the form (1.2), and at what expense, remain. In this paper, we aim to address these concerns. Furthermore, as alluded to earlier, our evaluation of SQP-GS and GRANSO can potentially be informative for other related robust control design tasks as these often involve nonsmoothness arising from stability constraints and/or max functions.

Of course, comparing the performance of competing solvers on a nonsmooth, nonconvex, constrained optimization problem is a multi-faceted affair. First, just the ability of a method to consistently find feasible solutions is paramount, particularly for nonlinear constraints since feasible points may not be knowable a priori and methods may not always find the feasible region for every starting point. Given feasibility, of second importance is the quality of objective function minimization, that is, how far has the objective function been reduced. With nonconvexity, this is not a well-defined goal since methods could converge to different local minimizers. Furthermore, with a possibly non-locally-Lipschitz function present in our optimization problem, there are no guarantees that the methods will not stagnate before reaching stationary points. However, from a user's perspective, whichever feasible answer minimizes the objective function the most would be preferred. Finally, the computational effort necessary to meet feasibility requirements and sufficiently minimize the objective must also be considered. Beyond different per-iteration costs that methods may have, this metric of interest is also complicated by the facts that (a) the rate of progress of any given method may be highly variable during optimization and (b) that reasonable computational budgets of users cannot typically be reliably estimated or predicted by others. As such, to increase the utility of benchmarks for users, it can be imperative to consider multiple budgets. To handle such complicated benchmarking objectives, where the three goals of finding feasibility, objective minimization, and computational efficiency are also often at odds with one another, β -Relative Minimization Profiles (β -RMPs or just RMPs) were proposed as new visualization tools to concisely address all these facets [CMO17, Section 5]. To help assess what is the best practice for solving our input shaper design problem, we employ β -RMP benchmarking to compare SQP-GS and GRANSO.

Finally, we propose new visualization tools, called *Global-Local Profiles (GL-Profiles)*, to help

resolve a particular quandary when it comes to nonconvex and/or nonsmooth optimization, where different local minima may be found and/or where convergence might be quite slow. Given a fixed computational budget, the question is then how to best utilize that budget in order to obtain good solutions to such a problem. One could devote the budget to runs from just one or a few starting points, to allow local minimizers to be obtained precisely (i.e. reaching stationarity conditions). Alternatively, one could distribute the budget over many starting points, where the much smaller budget per starting point means that minimizers will typically not be found so accurately but nevertheless, the best minimizer amongst those obtained could be much better than what would be found by only using few starting points. GL-Profiles present this tradeoff in a concise and intuitive manner and describe not only the relative performance differences of the solvers but also speak to the ease or difficulty of finding good solutions that is inherent to a particular nonconvex/nonsmooth problem of interest itself.

This paper is organized as follows. In Section 2, we provide background and details on the input shaper design optimization problem with the nonsmooth stability constraint. In Section 3, we first give a brief overview of the SQP-GS algorithm and then the BFGS-SQP method employed within GRANSO. We describe our experimental setup for designing stable input shapers in Section 4, while also giving a preliminary analysis of our results. In Section 5, we provide a primer on β -RMPs for the convenience of the reader and then do a β -RMP benchmark comparing SQP-GS and GRANSO for our input shaper design problem. To address the question of how to best utilize a fixed computational budget for nonconvex/nonsmooth optimization problems, we first introduce our new GL-Profiles and then do a GL-Profile benchmark to compare both methods with this new complementary perspective in Section 6. Final remarks are given in Section 7.

2 The input shaper design problem using distributed delays

Input shaping is a well-known technique for damping undesirable oscillatory modes of mechanical systems, to intelligently control the motions of a main-body such that when it comes to a stop, the flexible subsystem also returns to a stand still as quickly as possible. Input shaping via the use of time-delay filters was first proposed and implemented by Smith [Smi57] in the 1950s and then further extended to robust and multi-mode design tasks [SS90, SSS94, TS94] in the 1990s. In addition to the aforementioned typical crane problem [VKS10, SKK08], input shapers have also been used in many other applications, such as position control of flexible manipulators [PTDF09] and vibration suppression in robotics [PCPL06]. For an extensive review on input shaping over the last 50 years, see [Sin09].

The specific control application represented by (1.2) that we consider here arises from the use of a *distributed-delay shaper* [VKH13, VH15], where the inverse of the designed shaper is incorporated into the feedback loop of a control system, a technique proposed in [VHKA16b] (see also [HV17]). The benefit of such an approach is that it extends the applicability of input shaping to compensating the flexible modes induced by both reference signal and disturbances. In [VH15], an optimization-based approach for designing such an input shaper was proposed with the form of (1.2) but without the constraint $s(x) < 0$ and where, for robustness, the objective function was a least-squares measure of the residual vibrations across a small frequency range around the desired oscillation to be compensated. However, by including the inverse of the shaper in the feedback loop, zeros of the shaper are likely to be “turned into” poles of the closed-loop system, as addressed in [VHKA16b]. As closed-loop stability is required, it thus makes sense to add the condition that all zeros of the input shaper must reside in the open left half-plane, which will be modeled by the $s(x) < 0$ constraint in (1.2). As mentioned in the introduction, a preliminary optimization-based

approach using SQP-GS to enforce this nonsmooth stability constraint was considered in [KPV⁺17].

In more detail, we first describe an input shaper with a distributed delay. Given $T > 0$, a constant specifying the longest delay, let $\{\tau_1, \dots, \tau_n\}$ be the set of equally distributed delays in $[0, T]$, indexed in increasing order, with $\tau_1 = 0$ and $\tau_n = T$. The specific input shaper with delays proposed in [VH15] is characterized by the equation

$$y(t) = \gamma u(t) + \int_0^T u(t - \mu) dh(\mu), \quad (2.1)$$

where u and y are respectively the shaper input and output, $\gamma \in (0, 1)$ is the gain parameter, and the distribution of the delays is prescribed by the non-decreasing function $h(\mu)$, with length T (though again, we here will only focus on equally-distributed delays). Considering the step-wise distribution of the delays, the corresponding transfer function $D : \mathbb{C} \rightarrow \mathbb{C}$ of (2.1) has the following form:

$$D(s) = \gamma + \frac{1}{s} \sum_{k=1}^n x_k e^{-s\tau_k}, \quad (2.2)$$

where x_k is the respective gain for each delay spike with delay τ_k . The set of x_k values comprise our set of optimization variables.

In the classical feed-forward interconnection of the shaper $D(s)$ and the flexible subsystem $F(s)$, which acts as $D(s)F(s)$, the mode compensation takes place via cancellation of the oscillatory pole of $F(s)$ by a zero of the shaper $D(s)$. The purpose of including the inverse shaper is to pre-compensate the oscillatory modes of the flexible subsystem $F(s)$ when controlling the main-body system $G(s)$ via the controller $C(s)$. In contrast to the classical feed-forward application of the shaper $D(s)F(s)$, incorporating the inverse of the shaper allows flexible modes that are excited by disturbances d to also be compensated. This feature results from the corresponding transfer functions from the system reference w to the system output y

$$T_{y,w}(s) = \frac{C(s)G(s)}{1 + C(s)G(s)\frac{1}{D(s)}} F(s) = \frac{C(s)G(s)}{D(s) + C(s)G(s)} D(s)F(s), \quad (2.3)$$

and from the system output disturbance d to the system output y

$$T_{y,d}(s) = \frac{1}{1 + C(s)G(s)\frac{1}{D(s)}} F(s) = \frac{1}{D(s) + C(s)G(s)} D(s)F(s). \quad (2.4)$$

Note that the term $D(s)F(s)$ appears in both the transfer functions, which confirms that the oscillatory mode compensation takes place in both of the channels. Furthermore, note that the characteristic equation determining the closed-loop poles is in both cases given by

$$D(s) + C(s)G(s) = 0. \quad (2.5)$$

If $C(s)G(s)$ is a strictly-proper transfer function, then $\lim_{|s| \rightarrow \infty} |C(s)G(s)| = 0$, and so the closed-loop poles with large magnitudes will tend to match the spectrum of $D(s)$ zeros. For more details, particularly if $C(s)G(s)$ is bi-proper, we refer the reader to [VHKA16b].

As designing a robust input shaper can lead to the shaper having a large number of zeros in the right half-plane (see [KPV⁺17, Section 2.2]), designing $C(s)$, the controller for the system $G(s)\frac{1}{D(s)}$, would also have the additional task of restricting all the poles of the closed-loop system to the open left half-plane. However, the inclusion of the aforementioned stability constraint to restrict the

zeros of the shaper to open left half-plane greatly reduces this burden on the controller. This leads to defining the so-called *zeros spectral abscissa* of the input shaper $D(s)$:

$$\alpha_D := \max\{\Re\lambda : \text{for all } \lambda \in \mathbb{C} \text{ such that } D(\lambda) = 0\}. \quad (2.6)$$

In this paper, we only consider stability from the perspective of the zeros of the shaper; extension to the full closed-loop system has not yet been in the consideration in the literature but will be the subject of future work.

We now present the specific optimization problem for designing the gains of the delay spikes in the input shaper. As its derivation is both technical and not directly relevant to our goal of how best to solve it in practice, we omit many of these details here and instead just refer the reader to [PMV17] for how they exactly arise. Given (2.1) as the chosen realization of the shaper, with defining constants T , n , and γ , the optimal *stable* shaper is obtained via solving

$$\min_{x \in \mathbb{R}^n} \frac{x^\top H x}{R_{\text{nom}}^2} \quad \text{s.t.} \quad \begin{cases} A_1 x & \leq b_1 \\ A_2 x & = b_2, \\ \alpha_D(x) & \leq \alpha_c \end{cases} \quad (2.7)$$

where $A_1 \in \mathbb{R}^{(n-1) \times n}$ and $b_1 \in \mathbb{R}^{n-1}$ in the linear inequality constraints are defined by

$$-\sum_{k=1}^l x_k \leq 0 \quad \text{for } l \in \{1, \dots, n-1\}, \quad (2.8)$$

$A_2 \in \mathbb{R}^{2 \times n}$ and $b_2 \in \mathbb{R}^2$ in the linear equality constraints are defined by

$$\sum_{k=1}^n x_k = 0 \quad \text{and} \quad \gamma - \sum_{k=1}^n x_k \tau_k = 1, \quad (2.9)$$

$H \in \mathbb{R}^{n \times n}$ is positive definite, $R_{\text{nom}} > 0$, $\alpha_D(x)$ is the zeros spectral abscissa of the shaper $D(s)$ as a function of optimization variables x , and constant $\alpha_c \leq 0$. Matrix H is a discretized approximation of the residual vibrations in the system while R_{nom} is the reference for residual vibrations. The linear inequality constraints (2.8) ensure the desired non-decreasing step response of the shaper while the linear equality constraints (2.9) guarantee that the static gain of the shaper $D(s)$ remains equal to one, so as to not otherwise influence the system. The choice of the nonpositive constant α_c allows one to optionally enforce that the zeros of the shaper not just merely be in the left half-plane but that they be at least $|\alpha_c|$ away from the imaginary axis.

Remark 2.10 *We note that we present (2.7) with a few technical/notational differences from what is used in the preceding control literature. First, in order to maintain consistency with (1.2) and the convention of the solvers in our evaluation, we flipped the \geq relation for linear inequalities to \leq , i.e. by implicitly multiplying them by -1 . Second, we use γ to denote the initial gain, instead of A as has been done elsewhere. Third, in previous literature, the initial gain γ was embedded as the first element of a vector \mathbf{x} also containing the individual gains to be optimized, i.e. $\mathbf{x} := [\gamma \ x_1 \ \dots \ x_n]^\top$. Rewriting the problem with γ removed from \mathbf{x} is straightforward and can still be rewritten as the quadratic program plus a nonsmooth constraint we give in (2.7). However, to accommodate this (small) reformulation of the vector of gains, it means the matrices we refer to above are correspondingly transformed versions of those defined in the aforementioned control literature, and are thus not the same.*

2.1 The zeros spectral abscissa

As a max function, the zeros spectral abscissa of a time-delay system will, in general, be nonsmooth when there are ties, that is when there is more than a single globally rightmost zero (ignoring conjugacy). Furthermore, as the spectral abscissa of a linear system is known to be non-Lipschitz when an eigenvalue attaining the spectral abscissa has multiplicity greater than one [BO01], the zeros spectral abscissa is also not Lipschitz, since a linear system is simply a special case of a time-delay system. As such, one should expect that optimizing the spectrum of a time-delay system will, in general, be at least as hard as that of a linear one.

To compute the zeros spectral abscissa of $D(s)$ as a function of the optimization variables x , that is $\alpha_D(x)$, we actually instead consider computing roots of $sD(s)$, i.e. $\lambda \in \mathbb{C}$ such that

$$\lambda D(\lambda) = \lambda\gamma + \sum_{k=1}^n x_k e^{-\lambda\tau_k} = 0. \quad (2.11)$$

The reason for this is that the roots of equation (2.11), which is of retarded quasi-polynomial form, can be determined as the poles of the associated retarded time-delay system

$$\dot{y}(t) = -\frac{1}{\gamma} \sum_{k=1}^n x_k y(t - \tau_k), \quad (2.12)$$

and so the roots of (2.11) can be computed via the method of [WM12], which is available as an open-source MATLAB code.¹ Note that considering $sD(s) = 0$ introduces an additional root at the origin, but this is simply removed from the computed set of roots.

In general, (2.11) has infinitely many solutions. However, due to $\gamma > 0$, the spectrum is retarded. For retarded systems, the number of roots satisfying $\Re\lambda > \beta$ for any $\beta \in \mathbb{R}$ is always finite [HL13]. This is in contrast to neutral time-delay systems, where the number of roots located to the right of a given vertical line specified by β can be infinite. Thus, a neutral system can have infinitely many unstable roots, which in turn makes stabilizing them more difficult than retarded systems, which only have finitely many unstable roots. Note that a classical shaper with lumped delays has a neutral spectrum. Avoiding this undesirable spectrum distribution, especially for the case when the shaper is applied in the inverse form, was the main reason why distributed-delay shapers were introduced [VKH13].

Although apparently not yet rigorously proven, the zeros spectral abscissa at least seems to be differentiable almost everywhere with respect to the delay gains x , even though it is still ultimately nonsmooth [MN07, Section 10.2.1]. Let \hat{x} be any set of specific gains such that $\alpha_D(x)$ is smooth at \hat{x} , with $\lambda_1 \in \mathbb{C}$ being a simple globally rightmost zero of (2.11) (again, excluding conjugacy). It then follows from straightforward differentiation with respect to x that the gradient of the zeros spectral abscissa at \hat{x} is given by

$$\nabla\alpha_D(x) \Big|_{x=\hat{x}} = -\Re \left(\frac{[e^{-\lambda_1\tau_1} \quad \dots \quad e^{-\lambda_1\tau_n}]^T}{\gamma - \sum_{k=1}^n \tau_k \hat{x}_k e^{-\lambda_1\tau_k}} \right). \quad (2.13)$$

Although computing the zeros of the shaper can be nontrivial, the subsequent gradient calculation is relatively straightforward and inexpensive to obtain.

¹<http://twr.cs.kuleuven.be/research/software/delay-control/roots/>

3 Descriptions of SQP-GS and GRANSO

For brevity in describing the algorithms, we will limit the discussion in this section to inequality constrained optimization but we note that both methods can also handle equality constraints and indeed, for the optimization problems we wish to solve here, we require methods that are applicable for problems with both equality and inequality constraints. Assuming that all the functions in (1.1) are locally Lipschitz, by Rademacher's Theorem [RW09], it follows that any nondifferentiability must be restricted to sets of measure zero, if at all. In our setting here, we will assume that there is always at least one function in (1.1) that is nonsmooth. Nevertheless, even when the nonsmoothness of (1.1) is restricted to just a set of measure zero, this is still often an insurmountable challenge to overcome for many optimization methods that are not specifically built for nonsmooth optimization in mind. In particular, in many applications, minimizers of (1.1) will typically be points at which one of the functions is nondifferentiable.

One potential way to solve (1.1) is by minimizing a (nonsmooth) *penalty function*, an approach employed by both SQP-GS and GRANSO. Consider the l_1 penalty function

$$\phi(x; \rho) := \rho f(x) + v(x) \tag{3.1}$$

where penalty parameter $\rho > 0$ balances the (sometimes opposing) goals of minimizing the objective and satisfying the constraints and $v : \mathbb{R}^n \rightarrow \mathbb{R}$ measures the total violation of constraints, defined as

$$v(x) := \|\max\{c(x), 0\}\|_1 = \sum_{j \in \mathcal{P}_x} c_j(x) \quad \text{where} \quad \mathcal{P}_x := \{j \in \{1, \dots, m\} : c_j(x) > 0\}. \tag{3.2}$$

If an acceptable value of ρ is known *a priori*, then solving (3.1) via a suitable *unconstrained* optimization method will also yield a solution to the constrained problem (1.1) (see [Bur91a, Bur91b]). Of course in practice, this is often not the case and the responsibility of finding a good value for ρ falls upon the constrained optimization method.

3.1 SQP-GS

In order to overcome any discontinuities in the gradients of the objective or constraint functions, respectively ∇f and ∇c^j , SQP-GS finds a stationary point of (1.1) by solving a sequence of quadratic programs augmented by additional gradient information obtained by random sampling. At each iteration, ∇f and ∇c^j are randomly sampled from ϵ -neighborhoods around the current iterate x_k ; we denote these respective sets of sampled gradients by $\mathcal{B}_{\epsilon,k}^f$ and $\{\mathcal{B}_{\epsilon,k}^{c^j}\}_{j=1}^m$. For now, we will assume that the objective and constraint functions are all nonsmooth. In order for the convergence results of SQP-GS to hold, each set must contain at least $n + 1$ sampled gradients (in addition to the gradient at x_k) but in practice, it is usually recommended to sample more than this (e.g. $2n$). Then to compute a descent direction d , the following subproblem of (1.1) is solved:

$$\begin{aligned} \min_{d,z,r} \quad & \rho z + \sum_{j=1}^m r^j + \frac{1}{2} d^\top H_k d \\ \text{s.t.} \quad & f(x_k) + \nabla f(x)^\top d \leq z, \quad \forall x \in \mathcal{B}_{\epsilon,k}^f \\ & c^j(x_k) + \nabla c^j(x)^\top d \leq r^j, \quad \forall x \in \mathcal{B}_{\epsilon,k}^{c^j}, \quad r_j \geq 0, \quad \forall j \in \{1, \dots, m\}, \end{aligned} \tag{3.3}$$

where ρ is the aforementioned penalty parameter, $z \in \mathbb{R}$ and $r^j \in \mathbb{R}$ are slack variables, and H_k is a BFGS approximation of the Lagrangian Hessian that is maintained and updated at every

iteration. Then given the descent direction d obtained by solving (1.1), a backtracking line search is performed to compute the next iterate x_{k+1} such that sufficient decrease of the penalty function (3.1) is satisfied.

Assuming that all the functions in (1.1) are indeed locally Lipschitz, then with probability one, SQP-GS is guaranteed to converge to a stationary point of (1.1) but this computed stationary point may be either a feasible or infeasible one. Note that the convergence results for SQP-GS also require that the BFGS Hessian approximation remain bounded, which is enforced in the 1.2 and later versions of its implementation. As mentioned earlier, the main downside to using SQP-GS is the many gradient samples ($\mathcal{O}(n)$) that need to be evaluated on each iteration, which can make SQP-GS a very expensive method to use, perhaps prohibitively so, when the functions and/or gradients are expensive to evaluate. Unfortunately, this is often the case for applications in robust control, including the problem we consider in this paper. Nevertheless, it is only necessary to apply the gradient sampling procedure to the specific functions that are actually nonsmooth and so SQP-GS allows a user to set the number of samples individually for each individual function. While we will make use of this feature to only enable gradient sampling for the nonsmooth zeros spectral abscissa, the relative cost of computing the remaining smooth functions and their gradients is so negligible that such selected gradient sampling will only marginally improve the computational burden for our specific problem.

3.2 GRANSO

To avoid the potentially high overhead that SQP-GS can have, GRANSO completely eschews gradient sampling. Instead, inspired by the wealth of evidence that regular BFGS is a reliable and efficient method for unconstrained nonsmooth optimization (e.g. [LO13]), GRANSO uses BFGS updating as a tool for solving nonsmooth constrained problems, in the form of (1.1), via finding a minimizer of the penalty function (3.1). However, the choice of determining a good value for the penalty parameter ρ still remains an issue. To address this, GRANSO combines BFGS updating with an SQP-based *steering* strategy. Originally proposed by [BNW08, BLCN12] for smooth problems, steering attempts to dynamically determine at every iteration whether or not the current value of ρ should be lowered, and by how much, in order to promote sufficient progress towards the feasible region. This strategy works as follows.

Consider the following quadratic program:

$$\begin{aligned} \min_{d,r} \quad & \rho(f(x_k) + \nabla f(x_k)^\top d) + \sum_{j=1}^m r^j + \frac{1}{2}d^\top H_k d \\ \text{s.t.} \quad & c^j(x_k) + \nabla c^j(x_k)^\top d \leq r^j, \quad r^j \geq 0, \quad \forall j \in \{1, \dots, m\}. \end{aligned} \tag{3.4}$$

At each iterate x_k , in order to determine a new descent direction d_k , GRANSO will solve (3.4) one or more times, for different trial values of the penalty parameter. Note if $\rho = 1$ and there are no constraints, solving (3.4) would just yield the standard BFGS direction. First, GRANSO solves (3.4) for the current value of the penalty parameter to produce a candidate direction d . Using a linearized model of constraint violation, GRANSO computes the predicted reduction in total violation if the candidate direction d is used, that is,

$$l_\delta(d; x_k) := v(x_k) - \|\max\{c(x_k) + \nabla c(x_k)^\top d, 0\}\|_1. \tag{3.5}$$

If the prediction reduction given by (3.5) is deemed sufficiently large, then the computed direction d and the current value of the penalty parameter are considered to sufficiently promote progress

towards the feasible region and d is accepted as the new search direction d_k while ρ remains unaltered. Otherwise, (3.4) is re-solved with ρ temporarily set to zero to obtain \tilde{d}_k , which is called the *reference direction*. With ρ temporarily set at zero, the objective function and its gradient are removed from (3.4). Thus, the reference direction is biased towards reducing the current violation $v(x_k)$ as much as possible regardless of what effect it may have on the objective function. Using the predicted violation reduction for the reference direction, $l_\delta(\tilde{d}_k; x_k)$, which models the best case reduction in violation one might achieve, GRANSO then re-solves (3.4) for successively lower values of ρ until it produces a direction d such that $l_\delta(d; x_k)$ is some sufficiently large fraction of $l_\delta(\tilde{d}_k; x_k)$. In other words, in this latter scenario, GRANSO computes a direction d_k that is predicted to reduce the violation by at least, say 10% for example, of the amount that the reference direction \tilde{d}_k would. An inexact Armijo-Wolfe line search is then used with direction d_k to compute the next iterate and the BFGS Hessian approximation H_k is updated.

4 Experimental setup and preliminary analysis

For setting up the specific instance of (2.7), we chose to optimize an input shaper with 18 equally-spaced delays ($n = 18$) with $T = 0.8$, $\gamma = 0.01$, and $\alpha_c = -0.1$ to keep the zeros of the shaper a bit away from the stability boundary. To assess the difficulty of solving this 18-variable robust control design task and benchmark SQP-GS and GRANSO, we ran both solvers on two different sets of 1000 starting points, each constructed as follows. For the first set, we simply created random starting points generated via `randn`, a common (and often default) choice for most solvers and practitioners when no other information is known about the problem’s solution(s) a priori. The second set also contained 1000 randomly-generated starting points, but created using uniform distributions such that these initial points at least satisfied all of the linear inequality constraints and the first of the two linear equality constraints. We will herein respectively refer to these two sets as the “`randn`” and “`LC`” points, with the latter standing for “Linear Constraints”. We did not attempt to construct initial points that satisfied all linear constraints and/or the nonlinear zeros spectral abscissa constraint.

For the implementation of the optimization problem, we were provided the MATLAB code used for the experiments done in [KPV⁺17] and other related papers. However, we modified this code so that the implemented objective and constraint functions agreed with the formal shaper optimization problem we present here. In this process, we also increased the code’s efficiency and reliability when evaluating these function/gradient and did code cleanup and modularization, to facilitate our larger evaluation.

For the solvers, we specifically used SQP-GS v1.3 and GRANSO v1.6 and allowed each solver a maximum of 1000 iterations per starting point. We set both codes to use a stationarity tolerance of 10^{-8} . In order for an iterate to be considered (sufficiently) feasible, we set the tolerances of both solvers so that the inequality constraints had to be strictly satisfied while the total violation of the equality constraints could be no more than 10^{-8} . For additional consistency, we set the initial value of the penalty parameter of both SQP-GS (`rho_init`) and GRANSO (`opts.mu0`) to 1. Finally, for SQP-GS, we set the number of gradient samples for the nonsmooth spectral abscissa constraint to be $2n$, the recommended value noted in its documentation and relevant papers ([CO12, Section 4] and [BLO05, Section 4]). Otherwise, all other parameters of the two codes were left at their defaults and both codes used the builtin `quadprog` routine provided in MATLAB to solve their respective QP subproblems.

The experiments were computed over a subset of identical compute nodes of the `mechthild` cluster located at MPI Magdeburg. where each node has two Intel Xeon Silver 4110 CPUs with

	randn			LC		
	GRANSO	SQP-GS	Ratio	GRANSO	SQP-GS	Ratio
Total Wall-Clock Time (days)	2.12	83.1797	39.16	1.28	31.96	31.10
Total Iters	107,491	248,254	2.31	54,151	118,565	2.19
Average Time Per Iter (min/iter)	0.03	0.48	16.96	0.03	0.39	14.20

Table 1: Total cost statistics for each solver, on each of the two test sets of 1000 different starting points. The “Ratio” columns simply show the statistic for SQP-GS divided by the corresponding one for GRANSO.

192 GB RAM and were running MATLAB R2017b on CentOS Linux 7. Note however that very little memory was actually needed for the experiments. As both SQP-GS and GRANSO are serial programs, the only exploited parallelism was in the experimental setup, to distribute the 2000 different starting points over multiple nodes. The cumulative serial time of all our experiments (measured using `tic` and `toc`) was 118 days, with SQP-GS being accountable for 97% of the total computation time.

4.1 Preliminary performance analysis

In Table 1, we provide basic cost statistics for performing the experiments, for each pair of solver and starting point set. The relative high cost of running SQP-GS is clearly evident, with it requiring about 31 times longer than GRANSO to complete the LC test set and needing 39 times longer on the `randn` test set. While GRANSO completed both test sets in 3.4 days, SQP-GS took over 115 days. Interestingly, part of this large difference was not just the additional cost for SQP-GS to obtain $\mathcal{O}(n)$ gradient samples every iteration but also because SQP-GS also took over twice the number of iterations compared to GRANSO. Compared to the `randn` starting points, we see that using LC starting points reduces the total time and iterations for both solvers, by a factor of two, roughly speaking. We also see that more often than not both solvers are terminating well before their 1000 maximum allowed iteration limit.

In Figure 1a, for each of the `randn` starting points, we plot the best objective value encountered by each solver. The plot is sorted in decreasing order with respect to these best objective values, with the infeasible solutions plotted first. An infeasible solution means that a solver was not able to ever find the feasible set when initialized at that particular starting point. A dramatic performance difference between GRANSO and SQP-GS is immediately apparent, namely that SQP-GS only finds the feasible sets for just less than half (477 of 1000) of the `randn` test set while GRANSO finds the feasible set for 922 of the 1000 `randn` starting points. On the other hand, of the feasible solutions found, we see that SQP-GS found a larger portion of the best solutions, even though the very best solution was computed by GRANSO.

In Figure 1b, we provide the same type of plot but now for the LC starting points. We more or less see the same overall picture, with a couple notable differences. First, initializing from the LC points allows both solvers to slightly increase their respective success rates in finding the feasible region, which is not so surprising given the construction of the LC points to partially satisfy feasibility. Second, both solvers on average seem to return better solutions, ones corresponding to somewhat lower objective values, when initialized from the LC points, with this positive change being slightly more prominent for SQP-GS. Nevertheless, the very best solution was also computed by GRANSO on the LC test set.

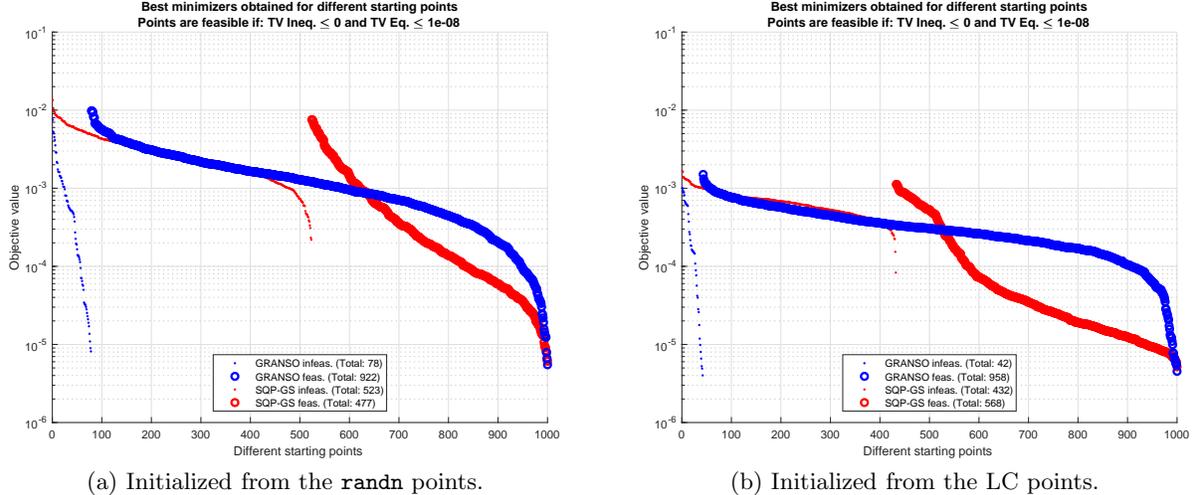


Figure 1: Best solutions, in terms of the value of the objective function evaluated at those solutions, obtained from each starting point. A solution is considered feasible if its total violation for the inequality constraints (TV Ineq.) is less than zero and its total violation for the equality constraints (TV Eq.) is at most 10^{-8} . Note that use of log scale on the y -axis, as the best objective values span a great range.

Plots like the ones we show in Figures 1a and 1b are useful for assessing how many different stationary points a nonconvex problem may have, which would appear as plateaus due to the sorting of the solutions by objective value. However, neither plot seems to show any evidence of plateaus. Instead, both show a rather smoothly-varying decrease in the objective values. One possible explanation is that there is only one feasible minimizer and perhaps a second infeasible one. In this case, the smooth changes seen in the objective value could be due to inconsistencies in the computed solutions, due to scaling issues, choices of stopping tolerances for the solvers, `quadprog` and its parameters, inaccuracy in the computation of the zeros spectral value abscissa, high sensitivity of the functions to the variables near minimizers, etc. On the other hand, given that the zeros spectral value abscissa is likely not Lipschitz, and thus neither solver would have convergence guarantees, the smoothly-varying objective value plots may simply be an indicator that the solvers are simply terminating before actually reaching stationary points. In this case, it may be that there are multiple stationary points present (even though they were not detected in Figures 1a and 1b). Unfortunately, it is difficult to determine which of the many possible combinations of explanations is correct.

5 A comparison using Relative Minimization Profiles

The preceding analysis comparing SQP-GS and GRANSO has so far only considered the quality of the best answers they produced. Neither Figure 1a or 1b takes into account how much computation was actually necessary to obtain these best solutions. For assessing the relative overall efficiency and speed of progress of these two methods, when computation is not an unlimited resource, we will employ what are called β -relative minimization profiles (β -RMPs or RMPs), new benchmarking visualization tools first introduced in [Mit14, Section 7.5] and then more formally presented in [CMO17, Section 5]. To aid the reader who may not be familiar with these tools, we will first provide the prerequisite description of β -RMPs; for brevity, we only provide a high-level overview

sufficient for understanding our comparisons done here and refer to [CMO17, Section 5] for the formal mathematical description of β -RMPs and further notes on their usage. Following that, we will then describe how we tailor the RMP benchmark for the purposes of our comparison (since RMPs can be considered more of a methodology, not just a single fixed specification) and then present our numerical evaluation of SQP-GS and GRANSO.

5.1 An RMP primer

RMPs are typically presented as a small selection of related plots, say four, called a β -RMP *benchmark panel*, that simultaneously compares methods in terms of amount of minimization achieved, ability to find feasible solutions, and speed of progress. Each particular RMP plot in the selection is defined for its own value of parameter β , which must be positive and scales the computational budget allowed per each problem in a given test set. Individual budgets may be assigned to each problem differently (for example, to account for its relative difficulty, size, etc). Then, these budget assignments are scaled by the choices of β to determine the actual budgets for a specific profile; the set of such RMPs are referred to as β -RMPs.

When $\beta = \infty$, the corresponding ∞ -RMP compares methods without any regard to their respective computational costs. Each method has its own ∞ -RMP *curve* on the ∞ -RMP plot and for a given point (x, y) on such a curve, y is the *percentage of problems solved* such that the solutions computed by this method were both feasible (to tolerances) and the relative differences of the corresponding objective values to the *best known objective values* (smallest and obtained on the feasible region) were at most x for each problem. On the right side of the ∞ -RMP, typically $x = \infty$ is shown, which gives the percentage of problems that a method was able to find feasible solutions for, regardless of their corresponding objective values (since $x = \infty$ means that there is no limit on what is an acceptably small relative difference). The left side of the ∞ -RMP, say for $x = 10^{-16}$, instead gives the percentage of problems that a method found solutions that were both feasible and agreed to the best known objective values (again, over the feasible set) to essentially machine precision. Thus, the ∞ -RMP curve for each method shows exactly how the percentage of its *acceptably solved problems* increases as the relative difference tolerance x is *loosened* from machine precision to ∞ .

When β is a finite positive value, the corresponding β -RMP compares methods when a per-problem computational budget, scaled by β , is forced upon them. Thus, instead of being compared with respect to their best computed answers, they are compared with respect to the best answers they were able to find within some given per-problem budgets, scaled by β . When $\beta = 1$, the per-problem budgets are not scaled and the methods are judged with respect to however the per-problem budgets are specified. As β is increased, the allowable computational budgets per problem is increased, giving each method more computation resources (perhaps measured in wall-clock time, number of iterations, number of function evaluations, etc) for a chance to improve upon their best known answers so far.

There is no prescribed choice or methodology for specifying the computational budgets used in β -RMPs. Instead, the per-problem budgets are allowed to be defined in any way the user wishes, as different budget choices may reflect different goals of the benchmark. Several example choices and their possible respective usage cases were discussed in [CMO17, Sections 5.2 and 5.3], when introducing the formal description of β -RMPs. For example, budgets could be specified a priori by the user, which may be appropriate if the user has either known limited computational resources or good estimates for the difficulties of each problem in the test set. Alternatively, the per-problem budgets can be constructed using data collected from the evaluation itself. In a multi-way comparison, one might use the average or median of the runtimes of each method for a

given problem as the budget for that problem, as this may be a good estimate of that problem’s difficulty. Another viable choice is to use the runtime per problem of a single method, which can be appropriate when one, for example, is benchmarking a new method against existing others (and as was done in the evaluation of GRANSO [CMO17, Section 6]).

The best known objective values used by the RMPs, called the *target values*, can also be determined in multiple ways. If the problems are convex and the global minimizers are known, then the globally minimal objective values could be supplied directly by the user. For nonconvex problems, minima may not be known and the methods may converge to different stationary points, or perhaps not to any (e.g. if the problems are not Lipschitz). As such, RMPs set the target value for a problem to the best (smallest) objective value encountered at a feasible point (to tolerances), by any of the methods, at any iterate in their entire iteration histories, for the given problem. RMPs also give the user the choice of defining these target values by restricting the pool of candidate iterates to consider to be those that are computed within β times the allotted budget for that problem; this feature is called *target scaling* and alters the target values to be the best known objective values *so far*. Enabling target scaling typically increases the separation of the different RMP curves for each method in each particular β -RMP plot (which can aid in the visualization) and judges the methods with respect to results that are known to be *achievable within the particular budgets given for that β -RMP plot*.

To produce a β -RMP benchmark panel, it is necessary to collect several pieces of information. For each method-problem pair, one must collect a history of iterates. The history must track the objective values, constraints values, and the computational cost (in whatever metric one has chosen to use) needed to obtain each iterate by that method. Note that the violation values can be used in lieu of constraint values but then one must take care the violation values across methods are computed in identical ways. A second of benefit of collecting the constraint values is that it allows the definition of acceptably feasible to be easily changed without having to rerun the experiments. It is worth noting that the objective values do not necessarily need to be the values of the objective function of (1.1); using a measure of stationarity is also a perfectly fine “objective” value to track.

5.2 Our β -RMP benchmark

To create our β -RMP benchmark, we recorded the objective and constraint function trajectories of the methods for every starting point. For the elapsed time to obtain each iterate, we simply recorded the total running time and then used the average elapsed time per iteration to estimate the cumulative elapsed time, as recommended by documentation of the open-source `betaRMP` software for making β -RMPs [Mita].

Since we have evaluated SQP-GS and GRANSO on a single test problem, but one initialized from many different points, we instead specified budgets and target values a bit differently than what would be done in a RMP benchmark over a heterogenous set of test problems. First, we chose a single fixed budget value (to be scaled by β) for each starting point, since initialization technically does not change the underlying optimization problem. For this, we set the fixed budget value as the median of the GRANSO running times over the LC points (about 25.6 seconds), as these GRANSO LC runs were, on average, significantly shorter than the times of GRANSO on the `randn` points or SQP-GS on either set. Second, we forwent the use of target scaling and simply specified a single target value, namely the median of all the best solutions computed by both solvers, over all starting points, computed at any time; this target value is 6.915×10^{-5} . We selected this particular target since very few runs came close to the best computed solution, as seen in Figures 1a and 1b. Third, we plotted separate RMP curves for both sets of starting points, for both methods, since there were noticeable differences in each of the methods’ behaviors depending on which starting set was used.

6 Global-Local Profiles

For a user of optimization software, typically there is a particular desired window of time in which they would like to obtain the best possible solution to a problem. For nonconvex problems, one important consideration is how much effort to devote to “global” versus “local” optimization, that is, to either distribute a fixed budget over a large number of starting points or only a few. If there are a large number of local minimizers of significantly different minimal objective values, then it is expected that the ideal choice might be to start from as many points as possible, with the hope that a solver might encounter at least one of the better local minimizers, even if only approximately. Otherwise, if there are few local minimizers or they all have similar minimal objective values (as is often the case in machine learning applications, for instance), it would likely be better to use only few starting points but to obtain these minimizers to a higher precision.

While such considerations will partially depend on an algorithm’s speed and accuracy, it will primarily depend on the problem itself. The quantity of local minimizers and the complexity and curvature of the regions around each local minimizer will vary problem to problem, and thus reward different strategies of budget distribution. To address this problem, we propose new visualization tools called Global-Local Profiles (GL-Profiles).

In order to create GL-Profiles, we will need the same collected data as needed for β -RMPs. An optimization algorithm has been run N times from N different starting points, with all the objective and constraint values recorded for every iterate, along with the cumulative cost to obtain each iterate; the cumulative cost is with respect to each starting point, i.e. the cost is zero at every x_0 . Consider the following basic definitions regarding all the iterates of a single method started from N initial points:

$$\begin{aligned} x_0^{(i)} &:= \text{the } i\text{th starting point in the set of } N \text{ initial points} \\ \{x_k\}^{(i)} &:= \text{the sequence of iterates computed by the given solver initialized at } x_0^{(i)} \\ t_i(j) &:= \text{cumulative cost to compute } \{x_0^{(i)}, x_1, \dots, x_j\} \subseteq \{x_k\}^{(i)}. \end{aligned}$$

Now, for a solver initialized at $x_0^{(i)}$, we define the subset of (sufficiently) *feasible iterates* that were computed within a fixed computational limit given by $t > 0$:

$$\mathcal{F}_i := \{x_j : x_j \in \{x_k\}^{(i)}, x_j \text{ is considered feasible, and } t_i(j) \leq t\}.$$

Let $S_l \subseteq \{1, \dots, N\}$ denote some selection of the initial points, taken by their indices; the subscript l will come into effect later, when we consider multiple selections. Then we consider the subset of those for which the solver computed at least one feasible iterate within the computational limit given by t :

$$S_l^{\mathcal{F}} := \{i : i \in S_l \text{ and } \mathcal{F}_i \neq \emptyset\}.$$

Given some total budget T and a selection of starting points given by some set of indices S_l , the budget allotted per starting point is simply $\frac{T}{|S_l|}$. Then, the best (lowest) objective value obtained on the feasible set by running a solver from the points given by S_l , each with budget $t = \frac{T}{|S_l|}$, is

$$f_l^* := \min_{i \in S_l^{\mathcal{F}}} \min\{f(x) : x \in \mathcal{F}_i\},$$

where f_l^* is taken to be ∞ if $S_l^{\mathcal{F}}$ is empty.

Let $P := \{S_1, \dots, S_q\}$ be a set of equally-sized nonoverlapping subsets of the N starting points, that is, with $|S_i| = |S_j|$ and $S_i \cap S_j = \emptyset$ for all $i \neq j \in \{1, \dots, q\}$. We now define the subset of sets $S_l \in P$ that have nonempty associated $S_l^{\mathcal{F}}$ subsets:

$$P^{\mathcal{F}} := \{l : S_l \in P \text{ such that } S_l^{\mathcal{F}} \neq \emptyset\}.$$

In other words, $P^{\mathcal{F}}$ is the subset of P such that $f_l^* \neq \infty$ for all $l \in P^{\mathcal{F}}$. Thus, given some P , we can now calculate a statistic for the expected best (lowest) objective value obtained among the feasible iterates if we were to run the solver initialized at $M = |S_1|$ different starting points, with each given a budget of $t = \frac{T}{M}$:

$$\tilde{f}^* := \frac{1}{|P^{\mathcal{F}}|} \sum_{l \in P^{\mathcal{F}}} f_l^*,$$

along with the standard error for this statistic:

$$\tilde{f}^{\text{err}} := \frac{1}{\sqrt{|P^{\mathcal{F}}|}} \sqrt{\frac{\sum_{l \in P^{\mathcal{F}}} (f_l^* - \tilde{f}^*)^2}{|P^{\mathcal{F}}| - 1}},$$

i.e. the usual standard error for estimating a mean of a value given a set of sample means, with estimated (rather than a priori known) population standard deviation. Thus, given a fixed budget, a *GL-Profile curve* for a method plots \tilde{f}^* as the number of starting points is increased, along with the error bars given by \tilde{f}^{err} . Of course, as the number of starting points is increased, the number of possible subsets in P will decrease, and thus error bars will grow, as the \tilde{f}^* statistic becomes less reliable.

We can also create GL-Profile curves with error bars for just the expectation of finding the feasible set, regardless of the objective value:

$$\tilde{c}^* := \frac{|P^{\mathcal{F}}|}{|P|},$$

also along with standard error for this statistic:

$$\tilde{c}^{\text{err}} := \sqrt{\frac{\tilde{c}^*(1 - \tilde{c}^*)}{|P|}}.$$

As one might expect, a GL-Profile curve will generally be more or less “U”-shaped. If few runs are performed, then the risk is high for them to be bad starting points, that either only result in infeasible iterates and/or high objective values. However, this risk can quickly decrease as more initial points are added. But, if too many points are used, any increased benefit will be outweighed by the effect of the decreasing budget allotted per starting point. At the very extreme, with little budget devoted to each starting point, running optimization might not be so different than just randomly sampling the variable space. Often, the optimal strategy, denoted by a global minimum of GL-Profile curve, will be somewhere in between the extremes but this will depend on both the available total budget, the particular solvers, and the properties of the problem itself.

6.1 Our GL-Profile benchmark

Similar to our β -RMP benchmark, we plotted GL-Profile curves for each pairing of the two solvers and the two starting sets, and using the same tolerances to determine whether an iterate is feasible or not. We created two different panels, one for the expected objective value and one for the

expectation of just finding feasible solutions, respectively shown in Figures 3 and 4. For each panel, we created four separate GL-Profiles for the following four different budgets: 20, 40, 80, and 160 minutes of wall-clock time. Each GL-Profile divides their respective budget over 2^r starting points, for $r = 1, \dots, 9$. Since our initial points were in no particular order, we simply constructed P sets by assigning the first 2^r points to S_1 , the next to S_2 , etc. Since $N = 1000$ is not a power of two, the less than 2^r remaining initial points were simply omitted from each P . Of course, as r increases, $|P|$ decreases and for $r = 9$, P will contain only one selection, namely the first 512 starting points.

In Figure 3, our GL-Profiles consider the expected best objective value to be obtained on the feasible set. For a small budget of just 20 minutes total, we see that GRANSO initialized from 32 of the LC points is the best strategy. SQP-GS is so slow that it actually does not make sense to run more than one single point, since even running from just two points for 10 minutes each returns a worse result. But with the total budget increased to 40 minutes, we see that running SQP-GS from two or four starting is better than just one. Still, from the `randn` test set, GRANSO provides a better optimal strategy for budget allocation than SQP-GS does. However, on the LC test set, we see that optimal allocations for GRANSO and SQP-GS offer similar solution quality. For budgets of 80 minutes or longer, we now see that SQP-GS initialized from four to eight LC points provides the best strategy. This slightly larger sample size significantly increases the probability that one of the starting points might be from the right side of the plots shown in Figures 1a and 1b, where SQP-GS generally found a larger percentage of higher quality solutions than GRANSO did.

In Figure 4, the GL-Profiles only consider the probability of finding the feasible region. Here, the scenario is quite different compared to the objective value criterion. Now we see that GRANSO is always better than SQP-GS at finding the feasible set, regardless of budget. This is of course not surprising given feasible/infeasible distributions shown in Figures 1a and 1b.

7 Concluding Remarks

In our evaluation of using nonsmooth, nonconvex, constrained optimization solvers, namely SQP-GS and GRANSO, to efficiently design input shapers that effectively damp undesirable oscillatory modes of mechanical systems, we have observed noticeable performance differences between them, some expected and some surprising. At a high-level, and as we would expect, SQP-GS was at least a magnitude of order slower than GRANSO, highlighting that the large computational burden of SQP-GS may simply make it a too impractical choice for many users. As illustrated by our β -RMP benchmark, even when SQP-GS was given up to twelve times longer to run than the median running time for GRANSO on the LC starting points, SQP-GS was still not at all competitive with GRANSO in terms of minimizing the objective on the feasible set. More surprising is that SQP-GS was not particularly effective at finding the feasible set at all, regardless of the budget, having a success rate of only about 50%. Meanwhile, GRANSO was particularly adept at finding the feasible set, rarely failing to do so. Nevertheless, SQP-GS still managed to return the majority of the best feasible solutions, even though GRANSO found the very best solution. From this limited perspective, one might be hard pressed to choose which of the two solvers would be better for the task of designing input shapers.

However, our newly introduced GL-Profiles do shed further light on how to select an optimal strategy that maximizes the likelihood of getting the best quality solution to a given problem under a fixed computational budget. For those with who either need solutions quickly, or have limited computational resources, GRANSO started with 32 or more of the LC starting points provides the best strategy. Otherwise, for users that are less restricted in their computational budget (specifically, at least 160 minutes total compute time on fast modern hardware), the best strategy

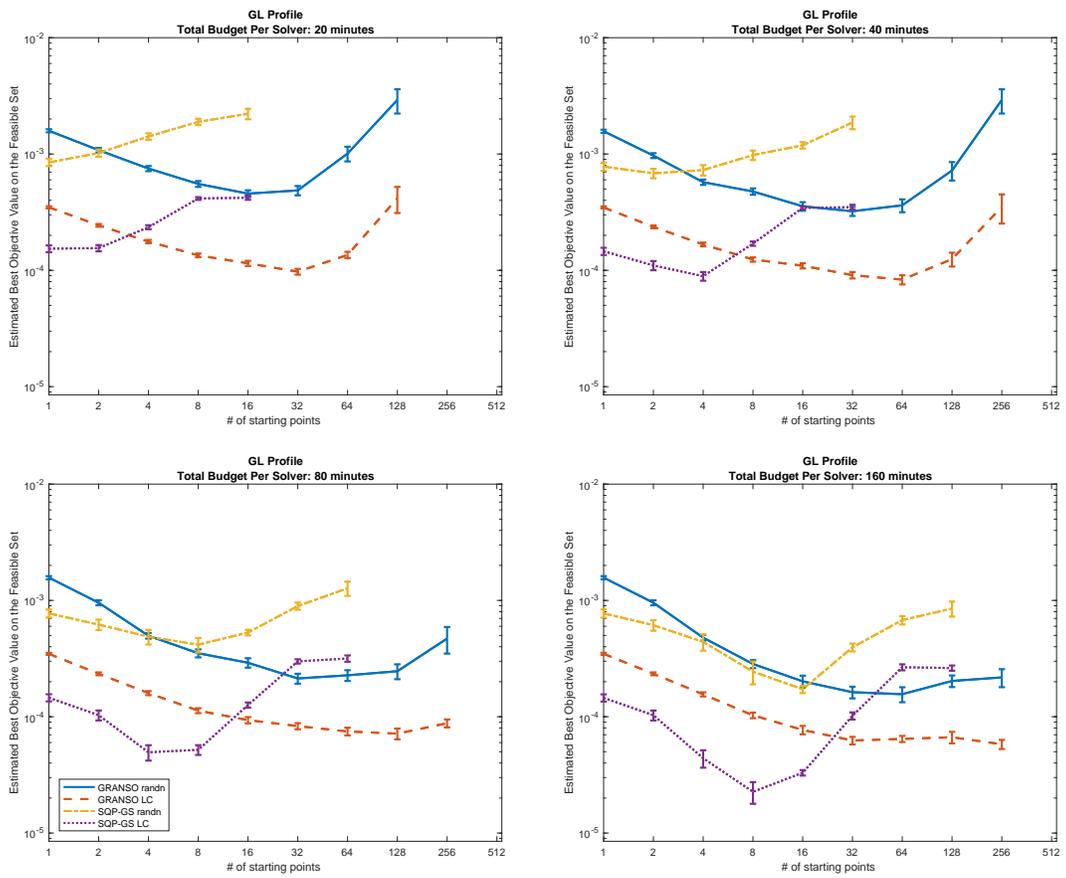


Figure 3: GL-Profiles for estimated best objective value found on the feasible set.

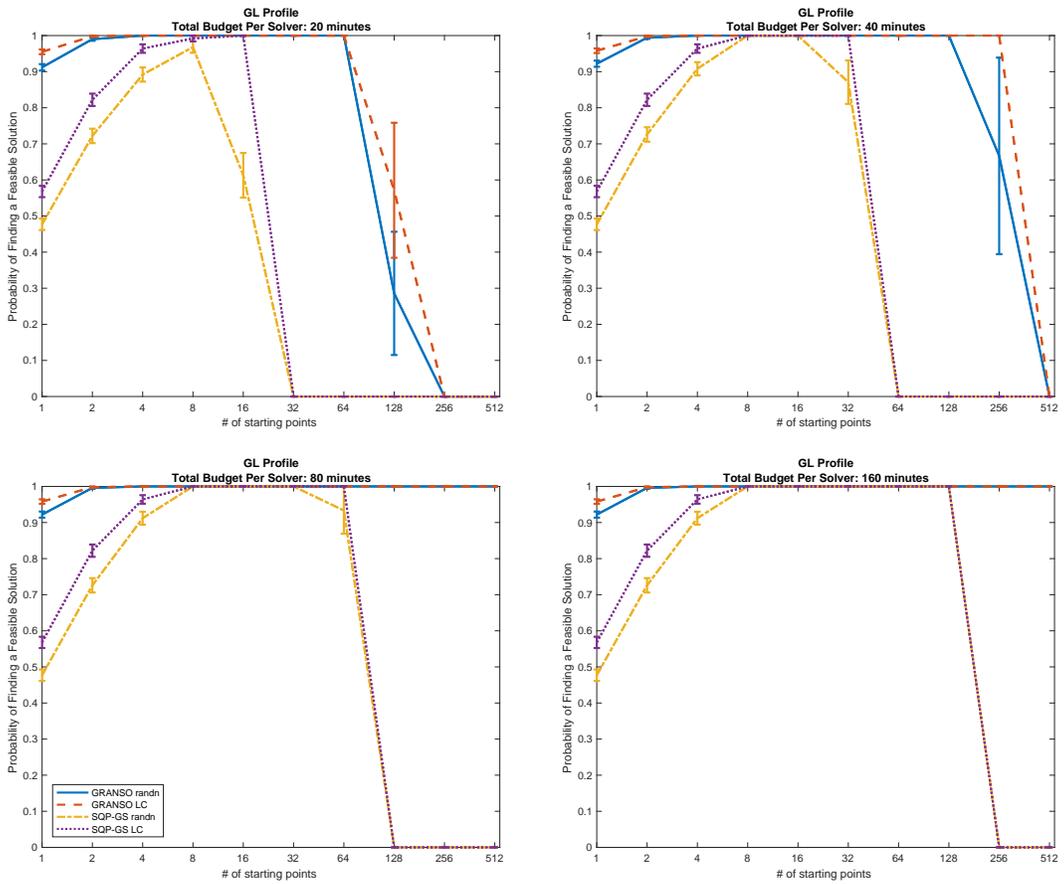


Figure 4: GL-Profiles for probability of finding at least one feasible solution.

is to use SQP-GS, dividing the budget equally amongst from eight or so initial points, constructed to satisfy as many of the constraints as possible (i.e. the LC test set). Thus, with aid of both β -RMPs and our new GL-Profiles, budget-dependent best practices for designing input shapers via nonsmooth optimization solvers are apparent.

8 Acknowledgement

The presented research was supported by the Czech Science Foundation under the project 16-17398S.

References

- [BKM14] A. Bagirov, N. Karmita, and M. M. Mäkelä. *Introduction to Nonsmooth Optimization: Theory, Practice and Software*. Springer-Verlag, Cham Heidelberg New York Dordrecht London, 2014.
- [BLCN12] R. H. Byrd, G. Lopez-Calva, and J. Nocedal. A line search exact penalty method using steering rules. *Math. Program.*, 133(1–2, Ser. A):39–73, 2012.
- [BLO05] J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM J. Optim.*, 15(3):751–779, 2005.
- [BNW08] R. H. Byrd, J. Nocedal, and R. A. Waltz. Steering exact penalty methods for nonlinear programming. *Optim. Methods Softw.*, 23(2):197–213, 2008.
- [BO01] J. V. Burke and M. L. Overton. Variational analysis of non-Lipschitz spectral functions. *Math. Program.*, 90(2, Ser. A):317–352, 2001.
- [Bur91a] J. V. Burke. Calmness and exact penalization. *SIAM J. Control Optim.*, 29(2):493–497, 1991.
- [Bur91b] J. V. Burke. An exact penalization viewpoint of constrained optimization. *SIAM J. Control Optim.*, 29(4):968–998, 1991.
- [CMO17] F. E. Curtis, T. Mitchell, and M. L. Overton. A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. *Optim. Methods Softw.*, 32(1):148–181, 2017.
- [CO12] F. E. Curtis and M. L. Overton. A sequential quadratic programming algorithm for nonconvex, nonsmooth constrained optimization. *SIAM J. Optim.*, 22(2):474–500, 2012.
- [HL13] Jack K Hale and Sjoerd M Verduyn Lunel. *Introduction to functional differential equations*, volume 99. Springer Science & Business Media, 2013.
- [HV17] Martin Hromčík and Tomas Vyhlídal. Inverse feedback shapers for coupled multibody systems. *IEEE Transactions on Automatic Control*, 62(9):4804–4810, 2017.
- [KPV⁺17] V. Kungurtsev, D. Pilbauer, T. Vyhlídal, M. Hromčík, and W. Michiels. Input shaper optimization with a constraint on the spectrum distribution. *IFAC-PapersOnLine*, 50(1):13324–13329, 2017. 20th IFAC World Congress.

- [LO13] A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-Newton methods. *Math. Program.*, 141(1–2, Ser. A):135–163, 2013.
- [Mita] T. Mitchell. betaRMP. <http://timitchell.com/software/betaRMP>. See also [CMO17].
- [Mitb] T. Mitchell. GRANSO: GRAdient-based Algorithm for Non-Smooth Optimization. <http://timitchell.com/software/GRANSO>. See also [CMO17].
- [Mitc] T. Mitchell. PSARNOT: (Pseudo)Spectral Abscissa|Radius Nonsmooth Optimization Test. <http://timitchell.com/software/PSARNOT>. See also [CMO17].
- [Mit14] T. Mitchell. *Robust and efficient methods for approximation and optimization of stability measures*. PhD thesis, New York University, New York, NY 10003, USA, September 2014.
- [MN07] W. Michiels and S. Niculescu. *Stability and Stabilization of Time-Delay Systems: An Eigenvalue-Based Approach*, volume 12 of *Advances in Design and Control*. SIAM, Philadelphia, PA, 2007.
- [PCPL06] Juyi Park, Pyung-Hun Chang, Hyung-Soon Park, and Eunjeong Lee. Design of learning input shaping technique for residual vibration suppression in an industrial robot. *IEEE/ASME Transactions on Mechatronics*, 11(1):55–65, 2006.
- [PMV17] D. Pilbauer, W. Michiels, and T. Vyhřídál. Distributed delay input shaper design by optimizing smooth kernel functions. *Journal of the Franklin Institute*, 354(13):5463–5485, 2017.
- [PTDF09] Emiliano Pereira, Juan Ramón Trapero, Iván M Díaz, and V Feliu. Adaptive input shaping for manoeuvring flexible structures using an algebraic identification technique. *Automatica*, 45(4):1046–1051, 2009.
- [RW09] R. T. Rockafellar and R. J. B. Wets. *Variational Analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin Dordrecht Heidelberg London New York, 3rd corrected printing of 1st edition, 2009.
- [Sin09] William Singhose. Command shaping for flexible systems: A review of the first 50 years. *International Journal of Precision Engineering and Manufacturing*, 10(4):153–168, 2009.
- [SKK08] William Singhose, Dooroo Kim, and Michael Kenison. Input shaping control of double-pendulum bridge crane oscillations. *Journal of Dynamic Systems, Measurement, and Control*, 130(3):034504, 2008.
- [Smi57] Otto JM Smith. Posicast control of damped oscillatory systems. *Proceedings of the IRE*, 45(9):1249–1255, 1957.
- [SS90] Neil C Singer and Warren P Seering. Preshaping command inputs to reduce system vibration. *Journal of dynamic systems, measurement, and control*, 112(1):76–82, 1990.
- [SSS94] William Singhose, Warren Seering, and Neil Singer. Residual vibration reduction using vector diagrams to generate shaped inputs. *Journal of Mechanical Design*, 116(2):654–659, 1994.

- [TS94] Timothy D Tuttle and Warren P Seering. A zero-placement technique for designing shaped inputs to suppress multiple-mode vibration. In *American Control Conference, 1994*, volume 3, pages 2533–2537. IEEE, 1994.
- [VH15] Tomáš Vyhlídal and Martin Hromčík. Parameterization of input shapers with delays of various distribution. *Automatica*, 59:256–263, 2015.
- [VHKA16a] T. Vyhlídal, M. Hromčík, V. Kučera, and M. Anderle. On feedback architectures with zero-vibration signal shapers. *IEEE Trans. Autom. Control*, 61(8):2049–2064, August 2016.
- [VHKA16b] Tomáš Vyhlídal, Martin Hromčík, Vladimír Kučera, and Milan Anderle. On feedback architectures with zero vibration signal shapers. *IEEE Transactions on Automatic Control*, 2016.
- [VKH13] Tomáš Vyhlídal, Vladimír Kučera, and Martin Hromčík. Signal shaper with a distributed delay: Spectral analysis and design. *Automatica*, 49(11):3484–3489, 2013.
- [VKS10] Joshua Vaughan, Dooroo Kim, and William Singhose. Control of tower cranes with double-pendulum payload dynamics. *IEEE Transactions on Control Systems Technology*, 18(6):1345–1358, 2010.
- [WM12] Zhen Wu and Wim Michiels. Reliably computing all characteristic roots of delay differential equations in a given right half plane using a spectral method. *Journal of Computational and Applied Mathematics*, 236(9):2499–2514, 2012.