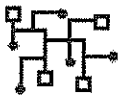

ISE

Industrial and
Systems Engineering

# A Robust Sequential Quadratic Programming Algorithm for Nonconvex, Nonsmooth Constrained Optimization

Frank E. Curtis
Lehigh University

Michael L. Overton
New York University

Report: 09T-018

# A ROBUST SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHM FOR NONCONVEX, NONSMOOTH CONSTRAINED OPTIMIZATION

FRANK E. CURTIS* AND MICHAEL L. OVERTON†

**Abstract.** We consider optimization problems where the objective and constraint functions may be nonconvex and nonsmooth. Problems of this type arise in many important applications, and in particular many have solutions at points of nondifferentiability of the problem functions. We present a line search algorithm for situations when the objective and constraints are locally Lipschitz and continuously differentiable on open dense subsets of $\mathbb{R}^n$. Our method is based on a sequential quadratic programming (SQP) method that uses an $\ell_1$ penalty to regularize the constraints, where a process of gradient sampling (GS) is employed to make the search direction computation robust in nonsmooth regions. We analyze the global convergence properties of our SQP-GS method and illustrate its effectiveness with a MATLAB implementation.

**Key words.** nonconvex optimization, nonsmooth optimization, constrained optimization, sequential quadratic programming, gradient sampling, exact penalization

**AMS subject classifications.** 49M37, 65K05, 65K10, 90C26, 90C30, 90C55

## 1. Introduction.

In this paper, we consider optimization problems of the form

$$\begin{aligned} &\min_x \ f(x) \\ &\text{subject to (s.t.) } c(x) \leq 0 \end{aligned} \tag{1.1}$$

where the objective $f : \mathbb{R}^n \to \mathbb{R}$ and constraint functions $c : \mathbb{R}^n \to \mathbb{R}^m$ are locally Lipschitz and continuously differentiable on open dense subsets of $\mathbb{R}^n$. Due to a subsequent regularization strategy that we apply to problem (1.1), the solution method that we propose can also be applied to problems with equality constraints $c^{\mathcal{E}}(x) = 0$ if they are formulated as the inequalities $c^{\mathcal{E}}(x) \leq 0$ and $-c^{\mathcal{E}}(x) \leq 0$. Thus, for ease of exposition, we consider only inequality constraints in (1.1). Moreover, since we make no convexity assumptions about $f$ and/or $c$, we are concerned only with finding local solutions to (1.1).

A wealth of research on the solution of *smooth constrained* optimization problems has been produced in recent decades. In particular, the methodology known as sequential quadratic programming (SQP) has had one of the longest and richest histories [19, 32]. Its many variations are still widely used and studied throughout the optimization community as new techniques are developed to confront the issues of nonconvexity, ill-conditioned constraints, and the solution of large-scale applications. At a given iterate $x_k$, the main feature of SQP algorithms is the following quadratic programming (QP) subproblem used to compute a search direction:

$$\begin{aligned} &\min_d \ f(x_k) + \nabla f(x_k)^T d + \tfrac{1}{2} d^T H_k d \\ &\text{s.t. } c^j(x_k) + \nabla c^j(x_k)^T d \leq 0, \quad j = 1, \ldots, m. \end{aligned} \tag{1.2}$$

*Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA. This author was supported in part by National Science Foundation grant DMS 0602235 while at the Courant Institute of Mathematical Sciences, New York University, New York, NY, USA. E-mail: frank.e.curtis@gmail.com

†Courant Institute of Mathematical Sciences, New York University, New York, NY, USA. This author was supported by National Science Foundation grant DMS 0714321. E-mail: overton@cs.nyu.edu

This subproblem is constructed by forming a local (and usually convex) quadratic model of the objective function and linearizations of the constraints, which makes the search direction calculation intuitively appealling as well as computationally robust and efficient in practice.

There has also been a great deal of work on the solution of *nonsmooth unconstrained* optimization problems (i.e., problem (1.1) with $m = 0$). For instance, in a collection of recent papers [6, 7, 27], an algorithm known as gradient sampling (GS) has been developed and analyzed for the minimization of locally Lipschitz $f$. The GS method is interesting theoretically in that convergence guarantees hold with probability one, but is also widely applicable, robust, and has been used to solve a variety of interesting problems [5, 13, 18, 17, 37]. Indeed, unlike the many variations of bundle methods [21, 24], GS does not require the user to compute subgradients as the iterates and sample points remain in the open dense set $\mathcal{D}^f \subset \mathbb{R}^n$ over which $f$ is continuously differentiable. The main computational expenses at a given iterate $x_k$ are simply $O(n)$ gradient evaluations and the computation of an approximate steepest descent direction. Defining the multifunction

$$G(x_k, \epsilon) := \text{cl conv } \nabla f(\mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^f),$$

where

$$\mathbb{B}(x_k, \epsilon) := \{x \mid \|x - x_k\|_2 \leq \epsilon\}$$

is the closed ball centered at $x_k$ with radius $\epsilon$, we have the representation

$$\bar{\partial} f(x_k) = \bigcap_{\epsilon > 0} G(x_k, \epsilon)$$

of the Clarke subdifferential of $f(x)$ at $x_k$ [11]. Then, with $G_k$ set as the convex hull of a finite number of gradients sampled randomly in $G(x_k, \epsilon)$, the approximate steepest descent direction computed by GS is $d_k = -g_k/\|g_k\|_2$, where $g_k$ solves the QP

$$\min_g \ \tfrac{1}{2}\|g\|_2^2, \quad \text{s.t.} \ \ g \in G_k.$$

More generally, a search direction $d_k$ of this type can be defined by the solution to

$$\begin{aligned} &\min_{d,z} \ z + \tfrac{1}{2} d^T H_k d \\ &\text{s.t. } f(x_k) + \nabla f(x)^T d \leq z, \quad \forall x \in \mathcal{B}_k^f \end{aligned} \tag{1.3}$$

where $H_k$ is defined as in subproblem (1.2) and $\mathcal{B}_k^f \subset \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^f$ is a randomly generated set that includes $x_k$ ([7], Lemma 2.1).

Altogether, the successes of these and other methods for smooth constrained problems and nonsmooth unconstrained applications have laid the groundwork for *nonsmooth constrained* optimization algorithms. Indeed, the purpose of this paper is to present and analyze such an approach. Our method combines techniques from SQP and GS for the development of a robust SQP-GS algorithm. At the heart of our discussion is a subproblem related to

$$\begin{aligned} &\min_{d,z} \ z + \tfrac{1}{2} d^T H_k d \\ &\text{s.t.} \ \begin{cases} f(x_k) + \nabla f(x)^T d \leq z, & \forall x \in \mathcal{B}_k^f \\ c^j(x_k) + \nabla c^j(x)^T d \leq 0, & \forall x \in \mathcal{B}_k^{c^j}, \ \ j = 1, \ldots, m \end{cases} \end{aligned} \tag{1.4}$$

which can be seen as a natural extension of the SQP subproblem (1.2) when combined with the GS techniques that led to (1.3). Similar to $\mathcal{B}_k^f$ for the objective function, the sets $\mathcal{B}_k^{c^j} \subset \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^{c^j}$, $j = 1, \ldots, m$, are randomly generated to sample gradients of the constraint functions at points near $x_k$. Enhancements to subproblem (1.4) and other algorithmic features are necessary to ensure that the method is well-defined and yields convergence guarantees, but we believe the structure of this subproblem gives our approach both intuitive appeal as well as practical advantages.

Let us close this introduction by referencing previous work on nonsmooth constrained optimization algorithms. One popular approach has been to solve constrained problems through unconstrained optimization methods, either by using problem-specific reformulations (e.g., see §5.9 in [28]) or by penalizing constraint violations in the objective (e.g., see §4.2 in [7], §14.3 in [14], and [23, 31]). Researchers have also considered special classes of constrained problems for which methods such as the SQP approach in [15] or the techniques in [29] can be applied. In contrast, our approach is designed for general problems of the form (1.1). Overall, our method is most similar to the bundle techniques in [22] and [25, 26], though these methods and ours differ significantly in detail.

We organize our discussion as follows. Motivation and a description of our algorithm are presented in §2. The main components of the approach are the search direction computation and line search. Global convergence guarantees for our method under certain common assumptions are provided in §3. In §4 we describe a MATLAB implementation of our algorithm. Our implementation includes certain practical enhancements to the basic framework discussed in §2, such as a dynamic update of the penalty parameter value, the incorporation of (approximate) second order information, and the special handling of functions that are known to be smooth. Numerical results are presented in §5, and concluding remarks are provided in §6.

*Notation.* All norms are considered $\ell_2$ (i.e., Euclidean) unless otherwise indicated. We generally use subscripts to denote iteration number in an algorithm and superscripts to denote element number in a vector or set. The expression $M \succ 0$ is used to indicate that the matrix $M$ is symmetric and positive definite.

**2. Sequential Quadratic Programming with Gradient Sampling.** In this section, we motivate and present our algorithm. We begin by developing an enhanced, yet practical version of subproblem (1.4). Our complete algorithm is specified in detail at the end of this section, though some practical features that are inconsequential for our convergence analysis in §3 are left for the discussion in §4.

**2.1. Search direction calculation.** The search direction calculation in our algorithm is inspired by subproblem (1.4). This subproblem lacks safeguards to make it well-defined at any given iterate $x_k$, so we begin by proposing a particular safeguard: regularization through the use of a penalty function. We also need to formally define the set of sample sets

$$\mathcal{B}_k := (\mathcal{B}_k^f, \mathcal{B}_k^{c^1}, \ldots, \mathcal{B}_k^{c^m})$$

and line search in such a way that the method possesses convergence guarantees. In this subsection we discuss both of these issues.

Consider the $\ell_1$ penalty function

$$\phi(x; \rho) := \rho f(x) + v(x),$$

where

$$v(x) := \sum_{j=1}^{m} \max\{c^j(x), 0\}$$

is a measure of infeasibility and $\rho > 0$ is a penalty parameter. If $x_*$ is a minimizer of $\phi(x;\rho)$ with $\rho$ sufficiently small so that $v(x_*) = 0$, then $x_*$ solves (1.1). In order to ensure that such a $\rho$ exists, however, we formally consider situations as described in the following theorem. In the theorem, we define a local stability condition known as *calmness* [11, 35]. We then state a result given as Proposition 1 and Theorem 1 in [36], the proofs of which draw from that of Proposition 6.4.3 in [11].

THEOREM 2.1. *Suppose $f$ and $c$ are locally Lipschitz on $\mathbb{R}^n$ and let $x'$ be a local minimum of* (1.1). *Moreover, suppose problem* (1.1) *is* calm *at $x'$ in that there exists $\upsilon > 0$ such that for all sequences $(x_k, s_k) \to (x', 0)$, with each pair $(x_k, s_k)$ satisfying $c(x_k) \leq s_k$, we have $f(x') - f(x_k) \leq \upsilon\|s_k\|$. Then, there exists $\rho' > 0$ and $\epsilon' > 0$ such that for all $x \in \mathbb{B}(x', \epsilon')$*

$$\phi(x;\rho) \geq \phi(x';\rho) \quad \textit{whenever} \quad \rho \geq \rho'. \tag{2.1}$$

*If, in addition, $x'$ is an isolated local minimum, then $\rho' > 0$ and $\epsilon' > 0$ can be chosen so that the inequalities in* (2.1) *are strict.*

Theorem 2.1 leads to the notion of *exact penalization* commonly discussed in smooth optimization; e.g., see §17.2 in [30]. With this concept in mind, it is appropriate to use the minimization of $\phi(x;\rho)$ as an algorithmic tool for solving (1.1). That is, under the assumption that $\rho$ is sufficiently small and (1.1) is sufficiently calm, the focus of our discussion and analysis will be on monotonically reducing $\phi(x;\rho)$. It is worthwhile to note that calmness is a weak constraint qualification in our context as other qualifications, such as the Mangasarian-Fromowitz and Slater conditions, both imply calmness. We also have the following result care of Proposition 6.4.4 in [11], proving the existence of Lagrange multipliers at calm solutions to (1.1).

THEOREM 2.2. *Suppose $f$ and $c$ are locally Lipschitz on $\mathbb{R}^n$ and let $x'$ be a calm local minimum of* (1.1). *Then, there exist Lagrange multipliers $\lambda' \geq 0$ such that*

$$\begin{aligned} &\partial_x L(x', \lambda') \ni 0 \\ &\lambda'^j c^j(x') = 0, \ j = 1, \ldots, m, \end{aligned}$$

*where $L(x, \lambda) := f(x) + \lambda^T c(x)$ is the Lagrangian for* (1.1).

Penalty functions in algorithms for constrained optimization problems have been studied extensively and widely implemented since the analyses by Han [19] and Powell [33]. Indeed, they have also been investigated in the context of nonsmooth applications [4, 3, 23, 25, 26, 36]. In our context, the two main benefits of the penalty function are that it provides a mechanism for judging progress in the algorithm and, when carried through the derivation of the search direction computation, the penalty terms have important regularization effects on the subproblem formulation. For example, for smooth optimization problems, it can be seen that the minimization of a quadratic model of the penalty function $\phi(x;\rho)$ at a given iterate $x_k$ is equivalent to the QP

$$\begin{aligned} &\min_{d,r} \ \rho(f(x_k) + \nabla f(x_k)^T d) + \sum_{j=1}^{m} r^j + \tfrac{1}{2} d^T H_k d \\ &\text{s.t. } c(x_k) + \nabla c(x_k)^T d \leq r, \quad r \geq 0, \end{aligned} \tag{2.2}$$

where $r \in \mathbb{R}^m$ is a vector of auxiliary variables [8]. The solution component $d_k$ of this subproblem is guaranteed to provide descent in the penalty function from the current iterate $x_k$, which means that (2.2) is consistent with the choice of judging progress by reductions in $\phi(x;\rho)$. Moreover, due to the presence of the auxiliary variables, this subproblem is always feasible, meaning that $d_k$ is always well-defined. Comparing this subproblem with (1.2), the only difference is that the constraints have been relaxed (with appropriate objective terms for penalizing violations in these constraints), so one may view (2.2) as a regularized version of subproblem (1.2).

In our algorithm, the search direction calculation is performed by solving a QP of the form (1.4) after appropriate penalty terms have been incorporated. The subproblem may be viewed as a regularized version of subproblem (1.4) or, alternatively, as a robust version of subproblem (2.2). In particular, to incorporate ideas from the GS algorithms described in [7, 27], we define

$$\begin{aligned} \mathcal{B}_k^f &:= \{x_{k0}^f, x_{k1}^f, \ldots, x_{kp}^f\}, \quad \text{where} \quad x_{k0}^f := x_k, \\ \text{and} \quad \mathcal{B}_k^{c^j} &:= \{x_{k0}^{c^j}, x_{k1}^{c^j}, \ldots, x_{kp}^{c^j}\}, \quad \text{where} \quad x_{k0}^{c^j} := x_k, \quad \text{for } j = 1, \ldots, m \end{aligned} \tag{2.3}$$

as sets of independent and identically distributed random points sampled uniformly from $\mathbb{B}(x_k, \epsilon)$ for some sample size $p \geq n+1$. Then, as long as

$$\mathcal{B}_k^f \subset \mathcal{D}^f \quad \text{and} \quad \mathcal{B}_k^{c^j} \subset \mathcal{D}^{c^j} \quad \text{for } j = 1, \ldots, m, \tag{2.4}$$

where $\mathcal{D}^f$ and $\mathcal{D}^{c^j}$, respectively, are the open dense subsets of $\mathbb{R}^n$ over which the functions $f$ and $c^j$, $j = 1, \ldots, m$, are locally Lipschitz and continuously differentiable, the QP subproblem

$$\begin{aligned} &\min_{d,z,r} \ \rho z + \sum_{j=1}^m r^j + \tfrac{1}{2} d^T H_k d \\ &\text{s.t.} \ \begin{cases} f(x_k) + \nabla f(x)^T d \leq z, \ \forall x \in \mathcal{B}_k^f \\ c^j(x_k) + \nabla c^j(x)^T d \leq r^j, \ \forall x \in \mathcal{B}_k^{c^j}, \ r^j \geq 0, \ j = 1, \ldots, m \end{cases} \end{aligned} \tag{2.5}$$

is well-defined and provides a direction of descent for the penalty function $\phi(x;\rho)$ at $x_k$ (see Lemma 3.8 below). Indeed, let us define the local model

$$\begin{aligned} &q(d;\rho, x_k, \mathcal{B}_k, H_k) \\ &:= \rho \max_{x \in \mathcal{B}_k^f} \{f(x_k) + \nabla f(x)^T d\} + \sum_{j=1}^m \max_{x \in \mathcal{B}_k^{c^j}} \{\max\{c^j(x_k) + \nabla c^j(x)^T d, 0\}\} + \tfrac{1}{2} d^T H_k d \end{aligned}$$

of the penalty function $\phi(x;\rho)$ at $x_k$. If $(d_k, z_k, r_k)$ is a solution to (2.5), then $d_k$ also solves

$$\min_d \ q(d;\rho, x_k, \mathcal{B}_k, H_k) \tag{2.6}$$

(e.g., see [8, 14]), and the two subproblems produce equal optimal objective values. Due to this equivalence, we refer to (2.5) and (2.6) interchangeably throughout the rest of our discussion.

We remark that it may be tempting to choose $\mathcal{B}_k^f = \mathcal{B}_k^{c_1} = \cdots = \mathcal{B}_k^{c^m}$ so that only $p$ random points in $\mathbb{B}(x_k, \epsilon)$ need to be generated during iteration $k$. Indeed, this

choice may work well in practice. However, our convergence results in §3 (in particular, Lemma 3.9) rely on the fact that points are generated independently for each problem function. We also remark that, as in other methods that have been proposed for nonsmooth constrained optimization (e.g., see [22]), it may be tempting to *aggregate* all of the inequality constraints into the single constraint $\max_j\{c^j(x)\} \leq 0$ in order to reduce the problem size, especially in terms of the QP (2.5). However, we believe that there are significant risks in this type of reformulation. First, the reformulation is not scale invariant, and so an employed algorithm may perform differently for different scalings of the constraints and poorly for certain choices of scalings. Second, such a reformulation disregards information about constraints that do not currently yield the largest value. This can especially be detrimental to the search if numerous constraints are active or near-active at a solution point. In contrast, subproblem (2.5) maintains information about all constraints during all iterations and is less sensitive to poor scalings of the constraint functions.

**2.2. Algorithm description.** Our complete algorithm is presented as Algorithm 2.1. As in [7, 27], the sampling radius $\epsilon$ is driven to zero throughout the solution process so that in the limit the method locates a stationary point of $\phi(x;\rho)$ (see Definition 3.3). The indicator for reducing $\epsilon$ is the reduction obtained in the model $q(d;\rho,x_k,\mathcal{B}_k,H_k)$. For a given $d_k$, we define this reduction in $q(d;\rho,x_k,\mathcal{B}_k,H_k)$ to be the value

$$\begin{aligned}\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) &:= q(0;\rho,x_k,\mathcal{B}_k,H_k) - q(d_k;\rho,x_k,\mathcal{B}_k,H_k)\\ &= \phi(x_k;\rho) - q(d_k;\rho,x_k,\mathcal{B}_k,H_k) \geq 0.\end{aligned}$$

(The reduction is nonnegative due to the fact that $d = 0$ yields an objective value of $\phi(x_k;\rho)$ in subproblem (2.6).) As illustrated in Lemma 3.6, $\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) = 0$ if $x_k$ is stationary for $\phi(x;\rho)$ (with respect to the current value of $\epsilon$), so a sufficiently small reduction in $q(d;\rho,x_k,\mathcal{B}_k,H_k)$ indicates that a reduction in $\epsilon$ is appropriate.

The line search is performed by backtracking along the search direction in order to ensure progress in reducing the value of the penalty function $\phi(x;\rho)$ during each iteration. The main condition that we enforce is the sufficient decrease condition

$$\phi(x_{k+1};\rho) \leq \phi(x_k;\rho) - \eta\alpha_k\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k), \tag{2.7}$$

where $x_{k+1} \leftarrow x_k + \alpha_k d_k$ for some steplength $\alpha_k \in (0,1]$ and $\eta \in (0,1)$. Note that this condition is satisfied for sufficiently small $\alpha > 0$ as long as $\phi'(d_k;\rho,x_k) < 0$, where

$$\phi'(d_k;\rho,x_k) := \lim_{\alpha\to 0^+} \frac{\phi(x_k+\alpha d_k;\rho) - \phi(x_k;\rho)}{\alpha} \tag{2.8}$$

is the directional derivative of $\phi(x;\rho)$ at $x_k$ along $d_k$. If $x_{k+1} \in \mathcal{D}$, where

$$\mathcal{D} := \mathcal{D}^f \cap \mathcal{D}^{c^1} \cap \cdots \cap \mathcal{D}^{c^m},$$

then we continue; otherwise, we replace $x_{k+1}$ with any point in $\mathcal{D}$ satisfying (2.7) and

$$\|x_k + \alpha_k d_k - x_{k+1}\| \leq \min\{\alpha_k,\epsilon\}\|d_k\|. \tag{2.9}$$

Since $x_{k+1} \notin \mathcal{D}$ is an unlikely event, it is argued in [7, 27] that we may practically always set $x_{k+1} \leftarrow x_k + \alpha_k d_k$, in which case (2.9) is trivially satisfied. However, this careful consideration that the iterates remain in $\mathcal{D}$ is necessary for our analysis.

**Algorithm 2.1** Sequential Quadratic Programming w/ Gradient Sampling (SQP-GS)

1: (Initialization): Choose a sample radius $\epsilon > 0$, penalty parameter $\rho > 0$, sample size $p \geq n+1$, line search constant $\eta \in (0,1)$, backtracking constant $\gamma \in (0,1)$, reduction factor $\beta \in (0,1)$, and tolerance parameter $\nu > 0$. Choose an initial iterate $x_0 \in \mathcal{D}$ and set $k \leftarrow 0$.
2: (Gradient sampling): Generate $\mathcal{B}_k$ satisfying (2.3) and (2.4).
3: (Search direction calculation): Choose $H_k \succ 0$ and compute $(d_k, z_k, r_k)$ from (2.5).
4: (Parameter reduction): If $\Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) \leq \nu\epsilon^2$, then set $\epsilon \leftarrow \beta\epsilon$, $x_{k+1} \leftarrow x_k$, and $\alpha_k \leftarrow 0$, and go to step 6.
5: (Line search): Set $\alpha_k$ as the largest value in $\{1, \gamma, \gamma^2, \dots\}$ such that $x_{k+1} \leftarrow x_k + \alpha_k d_k$ satisfies (2.7). If $x_{k+1} \notin \mathcal{D}$, then replace $x_{k+1}$ with any point in $\mathcal{D}$ satisfying (2.7) and (2.9).
6: (Iteration increment): Set $k \leftarrow k+1$ and go to step 2.

We remark that the sampling procedure in step 2 and the computation of $x_{k+1} \in \mathcal{D}$ satisfying (2.7) and (2.9) in step 5 can easily be implemented as finite processes. If, for example, the algorithm generates $x_{ki}^f \notin \mathcal{D}^f$ for some $i \in \{1, \dots, p\}$, then this point may be discarded and replaced by another randomly generated point in $\mathbb{B}(x_k, \epsilon)$, which is then also checked for inclusion in $\mathcal{D}^f$. Since the points are sampled independently and uniformly from $\mathbb{B}(x_k, \epsilon)$ and $\mathcal{D}^f$ is dense, this procedure terminates. Similarly, if $x_k + \alpha_k d_k \notin \mathcal{D}$, then we may sample $x_{k+1}$ from a uniform distribution defined on

$$\mathbb{B}(x_k + \alpha_k d_k, \min\{\alpha_k, \epsilon\}\|d_k\|/i),$$

incrementing $i$ by 1 each time until $x_{k+1} \in \mathcal{D}$ and (2.7) holds. As in [27], since $\alpha_k$ is chosen to satisfy (2.7), the continuity of $\phi(x; \rho)$ implies that this procedure terminates.

It is worthwhile to note that in the context of smooth constrained optimization, Algorithm 2.1 (with a sample size of $p = 0$) reduces to a S$\ell_1$QP method [14], and in the context of nonsmooth unconstrained optimization (with $m = 0$), the algorithm reduces to a variation of the gradient sampling algorithms in [7, 27]. Thus, Algorithm 2.1 generalizes each of these methods to nonsmooth constrained problems.

**3. Global convergence.** We make two assumptions throughout our global convergence analysis. The first relates to the properties of the problem functions themselves, though we believe that in practice Algorithm 2.1 is a viable approach even when the problem functions do not satisfy this assumption.

ASSUMPTION 3.1. *The function $f$ ($c^j$ for $j \in \{1, \dots, m\}$) is locally Lipschitz on $\mathbb{R}^n$ and continuously differentiable on the open dense subset $\mathcal{D}^f$ ($\mathcal{D}^{c^j}$) $\subseteq \mathbb{R}^n$.*

The second assumption relates to the iterates generated in the algorithm.

ASSUMPTION 3.2. *The sequences of iterates and sample points generated by Algorithm 2.1 are contained in a convex set over which the functions $f$ and $c$ and their first derivatives are bounded. In addition, there exist constants $\overline{\xi} \geq \underline{\xi} > 0$ such that $\underline{\xi}\|d\|^2 \leq d^T H_k d \leq \overline{\xi}\|d\|^2$ for all $k$ and $d \in \mathbb{R}^n$.*

We remark that restricting $H_k$ to the space of symmetric, positive semidefinite, and bounded matrices is standard for SQP methods since otherwise one cannot be sure that a QP solver is able to compute a global solution to subproblem (2.5). Indeed, we go slightly further and require $H_k \succ 0$ to ensure that the solution to the QP is bounded in norm (see Lemma 3.10).

The main result that we prove in this section states that, with probability one, Algorithm 2.1 will successfully locate a stationary point for the penalty function $\phi(x;\rho)$ (see Theorem 3.7 for the precise statement of our result). Since (1.1) is nonconvex and we do not assume exact second-order information is available or is used, this is the best that one can prove, though the hope is that with $\rho$ sufficiently small and problem (1.1) being sufficiently calm, the results of Theorems 2.1 and 2.2 will apply and a given stationary point will in fact be a local minimum for (1.1).

Our notion of stationarity for $\phi(x;\rho)$ differs slightly from the more common notion of Clarke stationarity found in the literature for nonsmooth optimization [11]. Rather, our definition is more closely related to a notion of stationarity discussed in the constrained optimization community (e.g., see Theorem 3.2 in [8], which draws from the results in [20]) as it relates to the solution of a constrained optimization subproblem. We begin our analysis by resolving the relationship between these two concepts. In addition, we present our definition of $\epsilon$-stationarity for $\phi(x;\rho)$. Again, our definition differs from the notion of Clarke $\epsilon$-stationarity on which the methods in [7, 27] are based, but it is similar, so we prove a second lemma to show that the two approaches are related in spirit. This discussion is also important to illustrate a crucial difference between our approach and that of applying the GS algorithm directly to the penalty function $\phi(x;\rho)$, as we explain following Lemma 3.4.

Consider any $x' \in \mathbb{R}^n$. This point is Clarke stationary for $\phi(x;\rho)$ if

$$0 \in \bar{\partial}\phi(x';\rho).$$

In contrast, let us define the subproblem

$$\begin{aligned} &\min_{d,z,r} \ \rho z + \sum_{j=1}^{m} r^j + \tfrac{1}{2} d^T H' d \\ &\text{s.t.} \begin{cases} f(x') + \nabla f(x)^T d \le z, \ \forall x \in \mathcal{B}^f \\ c^j(x') + \nabla c^j(x)^T d \le r^j, \ \forall x \in \mathcal{B}^{c^j}, \ r^j \ge 0, \ j = 1, \ldots, m, \end{cases} \end{aligned} \tag{3.1}$$

where $H'$ satisfies the same conditions as $H_k$ in Assumption 3.2 and

$$\mathcal{B}' := (\mathcal{B}^f, \mathcal{B}^{c^1}, \ldots, \mathcal{B}^{c^m}) := (\mathbb{B}(x',\epsilon) \cap \mathcal{D}^f, \mathbb{B}(x',\epsilon) \cap \mathcal{D}^{c^1}, \ldots, \mathbb{B}(x',\epsilon) \cap \mathcal{D}^{c^m}). \tag{3.2}$$

The $d$ component of the solution to (3.1) can also be obtained by solving

$$\begin{aligned} &\min_d \ q(d;\rho,x',\mathcal{B}',H') \\ &= \rho \max_{x \in \mathcal{B}^f} \{f(x') + \nabla f(x)^T d\} + \sum_{j=1}^{m} \max_{x \in \mathcal{B}^{c^j}} \{\max\{c^j(x') + \nabla c^j(x)^T d, 0\}\} + \tfrac{1}{2} d^T H' d. \end{aligned} \tag{3.3}$$

We define the following notion of stationarity.

DEFINITION 3.3. *The point $x'$ is* stationary *for $\phi(x;\rho)$ if and only if for all $\epsilon > 0$ the solution to* (3.3) *is $d' = 0$.*

Our first lemma relates this definition and Clarke stationarity for $\phi(x;\rho)$.

LEMMA 3.4. *Define $\widetilde{\mathcal{B}} := \mathcal{B}^f \cap \mathcal{B}^{c^1} \cap \cdots \cap \mathcal{B}^{c^m}$, the model*

$$\begin{aligned} &\widetilde{q}(d;\rho,x',\widetilde{\mathcal{B}},H') \\ &\quad := \max_{x \in \widetilde{\mathcal{B}}} \left\{ \rho(f(x_k) + \nabla f(x)^T d) + \sum_{j=1}^{m} \max\{c^j(x_k) + \nabla c^j(x)^T d, 0\} \right\} + \tfrac{1}{2} d^T H' d \end{aligned}$$

*of* $\phi(x;\rho)$ *at* $x'$*, and the subproblem*

$$\min_d \ \widetilde{q}(d;\rho,x',\widetilde{\mathcal{B}},H'). \tag{3.4}$$

*If* $d'$ *is the solution to* (3.3) *and* $\widetilde{d}$ *is the solution to* (3.4)*, then*

$$\Delta q(d';\rho,x',\mathcal{B}',H') \leq \phi(x';\rho) - \widetilde{q}(\widetilde{d};\rho,x',\widetilde{\mathcal{B}},H'). \tag{3.5}$$

*Thus, if* $x'$ *is Clarke stationary for* $\phi(x;\rho)$*, then* $x'$ *is stationary for* $\phi(x;\rho)$ *according to Definition 3.3.*

*Proof.* Subproblems (3.3) and (3.4) have a similar structure. The difference, however, is that for given $\epsilon > 0$ and $d \in \mathbb{R}^n$ the function gradients in (3.4) must all be evaluated at the same $x \in \widetilde{\mathcal{B}}$, whereas in (3.3) the points at which the gradients are evaluated may vary. Therefore, for any $\epsilon > 0$ and $d \in \mathbb{R}^n$, the fact that all elements of $\mathcal{B}'$ include $\widetilde{\mathcal{B}}$ implies that $q(d;\rho,x',\mathcal{B}',H') \geq \widetilde{q}(d;\rho,x',\widetilde{\mathcal{B}},H')$. Thus,

$$q(d';\rho,x',\mathcal{B}',H') \geq \widetilde{q}(d';\rho,x',\widetilde{\mathcal{B}},H') \geq \widetilde{q}(\widetilde{d};\rho,x',\widetilde{\mathcal{B}},H'),$$

which implies (3.5).

Now suppose $x'$ is Clarke stationary. Since $H' \succ 0$, this is equivalent to stating that for all $\epsilon > 0$, the solution to (3.4) is $\widetilde{d} = 0$; see [11], Lemma 2.1. Inequality (3.5) then implies that for all $\epsilon > 0$, the solution to (3.3) is $d' = 0$. □

In inequality (3.5) lies a crucial difference between Algorithm 2.1 and that of applying the GS algorithm directly to the penalty function $\phi(x;\rho)$. Applying GS to $\phi(x;\rho)$ means that search directions are computed by subproblems that approximate (3.4), whereas in Algorithm 2.1 search directions are computed by subproblems that approximate (3.3). If $x'$ is Clarke stationary for $\phi(x;\rho)$, then Lemma 3.4 states that $x'$ will also be stationary by our definition. However, at all other points, inequality (3.5) reveals (along with Lemma 3.6 below) that, when compared to a method that applies GS to $\phi(x;\rho)$ directly, Algorithm 2.1 will compute shorter steps. We also expect these steps to be more productive due to the fact that they handle the problem functions as distinct quantities; see our discussion at the end of §2.1.

Our second definition is the following.

DEFINITION 3.5. *If for a given* $\epsilon > 0$ *the solution to* (3.3) *is* $d' = 0$*, then* $x'$ *is* $\epsilon$-stationary *for* $\phi(x;\rho)$.

This definition leads to our next lemma. The result relates Algorithm 2.1 to the methods in [7, 27], both in terms of the algorithmic decision made in step 4 and with respect to the line search conditions (2.7) and (2.9). More precisely, considering non-normalized search directions as in §4.1 of [27], the algorithms in [7, 27] essentially update the sample radius $\epsilon$ based on norms of the search directions, whereas in our case the value is updated based on $\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k)$. The lemma illustrates that these choices are consistent.

LEMMA 3.6. *The solution* $d_k$ *to* (2.6) *with* $H_k \succ 0$ *and* $\mathcal{B}_k$ *satisfying* (2.3) *and* (2.4) *yields*

$$\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) = \tfrac{1}{2} d_k^T H_k d_k. \tag{3.6}$$

*Therefore, if* $\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) = 0$*, then* $x_k$ *is* $\epsilon$*-stationary for* $\phi(x;\rho)$.

*Proof.* Equation (3.6) follows from the optimality conditions of the QP (2.5); see [30], equation (16.37). Thus, under Assumption 3.2, $H_k \succ 0$ implies $d_k = 0$ if and

only if $\Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) = 0$. The second part of the lemma then follows from the fact that if $d_k = 0$ solves (2.6), then it also solves (2.6) with $\mathcal{B}_k$ replaced by $(\mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^f, \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^{c^1}, \ldots, \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^{c^m})$. □

The remainder of our analysis proceeds in the following manner. We illustrate that the line search procedure in step 5 is well-defined so that the method will generate an infinite sequence of iterates $\{x_k\}$. We then show that if the iterates remain in a neighborhood of a point $x'$, the algorithm eventually computes a search direction $d_k$ that sufficiently approximates the solution $d'$ to (3.3). This, along with a lemma illustrating that the computed search directions $\{d_k\}$ are sufficiently bounded in norm, will be used to prove that the algorithm successfully locates (near) $\epsilon$-stationary points for the penalty function $\phi(x; \rho)$. Our main result, the proof of which can be found at the end of this section, is the following.

THEOREM 3.7. *Let $\{x_k\}$ be a sequence of iterates generated by Algorithm 2.1. Then, with probability one, every cluster point of $\{x_k\}$ is stationary for $\phi(x; \rho)$.*

Now to the results in our analysis. At $x_k$ and for a given $d \in \mathbb{R}^n$, we respectively define the sets of linearly active and violated constraints as

$$\begin{aligned} \mathcal{A}(d; x_k) &:= \{j \in \{1, \ldots, m\} \mid c^j(x_k) + \nabla c^j(x_k)^T d = 0\} \\ \text{and} \quad \mathcal{V}(d; x_k) &:= \{j \in \{1, \ldots, m\} \mid c^j(x_k) + \nabla c^j(x_k)^T d > 0\}, \end{aligned}$$

so that for $x_k \in \mathcal{D}$ we have

$$\phi'(d; \rho, x_k) = \rho \nabla f(x_k)^T d + \sum_{\substack{j \in \mathcal{A}(0; x_k) \\ j \in \mathcal{V}(d; x_k)}} \nabla c^j(x_k)^T d + \sum_{j \in \mathcal{V}(0; x_k)} \nabla c^j(x_k)^T d. \tag{3.7}$$

LEMMA 3.8. *If Algorithm 2.1 executes step 5 during iteration $k$, then*

$$\phi'(d_k; \rho, x_k) \leq -d_k^T H_k d_k < 0,$$

*and hence there exists $\alpha_k > 0$ such that condition (2.7) is satisfied.*

*Proof.* Since $x_k \in \mathcal{B}_k^f \cap \mathcal{B}_k^{c^1} \cap \cdots \cap \mathcal{B}_k^{c^m}$ by (2.3), we have from (3.7) that

$$\begin{aligned} & \phi'(d_k; \rho, x_k) \\ \leq\ & \rho \max_{x \in \mathcal{B}_k^f} \nabla f(x)^T d_k + \sum_{\substack{j \in \mathcal{A}(0; x_k) \\ j \in \mathcal{V}(d; x_k)}} \max_{x \in \mathcal{B}_k^{c^j}} \nabla c^j(x)^T d_k + \sum_{j \in \mathcal{V}(0; x_k)} \max_{x \in \mathcal{B}_k^{c^j}} \nabla c^j(x)^T d_k \\ =\ & \rho(z_k - f(x_k)) + \sum_{\substack{j \in \mathcal{A}(0; x_k) \\ j \in \mathcal{V}(d; x_k)}} r_k^j + \sum_{j \in \mathcal{V}(0; x_k)} (r_k^j - c^j(x_k)) \\ \leq\ & -\phi(x_k; \rho) + \rho z_k + \sum_{j=1}^m r_k^j = -\Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) - \tfrac{1}{2} d_k^T H_k d_k. \end{aligned} \tag{3.8}$$

By Assumption 3.2 and Lemma 3.6, we then have

$$\phi'(d_k; \rho, x_k) \leq -\Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) - \tfrac{1}{2} d_k^T H_k d_k = -d_k^T H_k d_k < 0.$$

Thus, $d_k$ is a descent direction for $\phi(x; \rho)$ at $x_k$, so there exists $\alpha_k > 0$ such that (2.7) holds. □

We now turn to the algorithm's ability to approximate the solution to (3.3) when $x_k$ is sufficiently close to $x'$. We remark that, according to Definition 3.3, a measure

of the proximity of any point $x'$ to $\epsilon$-stationarity of the penalty function $\phi(x;\rho)$ is $\Delta q(d';\rho,x',\mathcal{B}',H')$, where $d'$ solves (3.3). For a given $x'$, we define

$$\mathcal{S}(x_k) := \left(\prod_1^p(\mathbb{B}(x_k,\epsilon)\cap\mathcal{D}^f), \prod_1^p(\mathbb{B}(x_k,\epsilon)\cap\mathcal{D}^{c^1}), \ldots, \prod_1^p(\mathbb{B}(x_k,\epsilon)\cap\mathcal{D}^{c^m})\right)$$

and consider the set

$$\mathcal{T}(\rho,x_k,x',\omega) := \{\mathcal{B}_k \in \mathcal{S}(x_k) \mid \Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) \le \Delta q(d';\rho,x',\mathcal{B}',H') + \omega\}.$$

(Here, in the definition of $\mathcal{T}(\rho,x_k,x',\omega)$ we continue with the notation that $d_k$ solves (2.6) and $d'$ solves (3.3).) We show that for $x_k$ sufficiently close to $x'$ the set $\mathcal{T}(\rho,x_k,x',\omega)$ is nonempty.

LEMMA 3.9. *For any $\omega > 0$, there exists $\zeta > 0$ and a nonempty set $\mathcal{T}$ such that for all $x_k \in \mathbb{B}(x',\zeta)$ we have $\mathcal{T} \subset \mathcal{T}(\rho,x_k,x',\omega)$.*

*Proof.* We present a proof by illustrating that, for a given $\omega > 0$, a nonempty set $\mathcal{T} \subset \mathcal{T}(\rho,x_k,x',\omega)$ can be constructed. First, let $d$ be any vector yielding $\Delta q(d;\rho,x',\mathcal{B}',H') < \Delta q(d';\rho,x',\mathcal{B}',H') + \omega$ whose existence follows from the continuity of $q(\cdot;\rho,x',\mathcal{B}',H')$ and the fact that $\omega > 0$. By the definition of $q(d;\rho,x',\mathcal{B}',H')$, there exists $(y^f, y^{c^1}, \ldots, y^{c^m})$ such that

$$q(d;\rho,x',\mathcal{B}',H') = \rho(f(x') + \nabla f(y^f)^T d) + \sum_{j=1}^m \max\{c^j(x') + \nabla c^j(y^{c^j})^T d, 0\} + \tfrac{1}{2}d^T H' d.$$

Since the sample size $p$ in Algorithm 2.1 satisfies $p \ge n+1$, Carathéodory's theorem [34] implies that there exists

$$\left\{y_i^f\right\}_{i=1}^p \subset \mathbb{B}(x',\epsilon)\cap\mathcal{D}^f$$

and a set of nonnegative scalars $\{\lambda_i^f\}_{i=1}^p$ such that

$$\sum_{i=1}^p \lambda_i^f = 1 \quad \text{and} \quad \sum_{i=1}^p \lambda_i^f y_i^f = y^f.$$

Similarly, for all $j \in \{1,\ldots,m\}$, the same theorem implies the existence of

$$\left\{y_i^{c^j}\right\}_{i=1}^p \subset \mathbb{B}(x',\epsilon)\cap\mathcal{D}^{c^j}$$

and nonnegative scalars $\{\lambda_i^{c^j}\}_{i=1}^p$ such that

$$\sum_{i=1}^p \lambda_i^{c^j} = 1 \quad \text{and} \quad \sum_{i=1}^p \lambda_i^{c^j} y_i^{c^j} = y^{c^j}.$$

Since $f$ ($c^j$ for $j = 1,\ldots,m$) is continuously differentiable on the open set $\mathcal{D}^f$ ($\mathcal{D}^{c^j}$), there exists $\zeta \in (0,\epsilon)$ such that the set

$$\mathcal{T} := \left(\prod_{i=1}^p \operatorname{int}\mathbb{B}(y_i^f,\zeta), \prod_{i=1}^p \operatorname{int}\mathbb{B}(y_i^{c^1},\zeta), \ldots, \prod_{i=1}^p \operatorname{int}\mathbb{B}(y_i^{c^m},\zeta)\right)$$

satisfies the following three properties:

- The first element of $\mathcal{T}$ lies in $\mathbb{B}(x', \epsilon) \cap \mathcal{D}^f$.
- The $j+1$ element of $\mathcal{T}$ lies in $\mathbb{B}(x', \epsilon) \cap \mathcal{D}^{c^j}$ for $j = 1, \ldots, m$.
- The solution $d_k$ to (2.6) with $\mathcal{B}_k \in \mathcal{T}$ satisfies

$$\Delta q(d_k; \rho, x_k, \mathcal{B}_k) \leq \Delta q(d'; \rho, x', \mathcal{B}', H') + \omega.$$

Thus, for all $x_k \in \mathbb{B}(x', \zeta)$, we have that $\mathbb{B}(x', \epsilon - \zeta) \subset \mathbb{B}(x_k, \epsilon)$ and hence $\mathcal{T} \subset \mathcal{T}(\rho, x_k, x', \omega)$. □

The proof of Lemma 3.9 illustrates the need for sampling gradients of each function independently. In summary, for any vector $d$ satisfying $\Delta q(d; \rho, x', \mathcal{B}', H') < \Delta q'(d'; \rho, x', \mathcal{B}', H') + \omega$, the objective value of subproblem (3.3) is defined by up to $m+1$ distinct points at which the gradients of the functions ($f$ and $c^j$ for $j = 1, \ldots, m$) are to be evaluated. In order to guarantee (by Carathéodory's theorem) that these points are included in the convex hull of the sampled gradients, the sampling of each function must be performed independently.

A technical lemma related to the solution of (2.6) follows next.

LEMMA 3.10. *The solution of* (2.6) *is contained in* $\mathbb{B}(0, \xi_1 + \xi_2 \|\hat{d}_k\|)$ *for some constants* $\xi_1, \xi_2 > 0$, *where* $\hat{d}_k$ *is the minimum norm minimizer of*

$$l(d; x_k, \mathcal{B}_k) := \sum_{j=1}^{m} \max_{x \in \mathcal{B}_k^{c^j}} \max\{c^j(x_k) + \nabla c^j(x)^T d, 0\}.$$

*Proof.* The function $l(\cdot; x_k, \mathcal{B}_k)$ is piecewise linear, convex, and bounded below, so it has a minimum norm minimizer and we call it $\hat{d}_k$. Under Assumption 3.2, the vector $\hat{d}_k$ yields

$$\rho \max_{x \in \mathcal{B}_k^f} \{\nabla f(x)^T \hat{d}_k\} + \tfrac{1}{2} \hat{d}_k^T H_k \hat{d}_k \leq \rho \max_{x \in \mathcal{B}_k^f} \|\nabla f(x)\| \|\hat{d}_k\| + \tfrac{1}{2} \overline{\xi} \|\hat{d}_k\|^2. \tag{3.9}$$

Similarly, for an arbitrary vector $d \in \mathbb{R}^n$ with

$$\begin{aligned} \tfrac{1}{4} \underline{\xi} \|d\| &> \rho \max_{x \in \mathcal{B}_k^f} \|\nabla f(x)\| \\ \text{and} \quad \tfrac{1}{4} \underline{\xi} \|d\|^2 &> \rho \max_{x \in \mathcal{B}_k^f} \|\nabla f(x)\| \|\hat{d}_k\| + \tfrac{1}{2} \overline{\xi} \|\hat{d}_k\|^2, \end{aligned} \tag{3.10}$$

we have that

$$\begin{aligned} \rho \max_{x \in \mathcal{B}_k^f} \{\nabla f(x)^T d\} + \tfrac{1}{2} d^T H_k d &\geq -\rho \max_{x \in \mathcal{B}_k^f} \|\nabla f(x)\| \|d\| + \tfrac{1}{2} \underline{\xi} \|d\|^2 \\ &> \tfrac{1}{4} \underline{\xi} \|d\|^2 \\ &> \rho \max_{x \in \mathcal{B}_k^f} \|\nabla f(x)\| \|\hat{d}_k\| + \tfrac{1}{2} \overline{\xi} \|\hat{d}_k\|^2 \\ &\geq \rho \max_{x \in \mathcal{B}_k^f} \{\nabla f(x)^T \hat{d}_k\} + \tfrac{1}{2} \hat{d}_k^T H_k \hat{d}_k, \end{aligned}$$

where the last inequality follows from (3.9). Thus, for $d$ satisfying (3.10), we have $q(d; \rho, x_k, \mathcal{B}_k, H_k) > q(\hat{d}_k; \rho, x_k, \mathcal{B}_k, H_k)$, which means that $d$ cannot be a minimizer of $q(\cdot; \rho, x_k, \mathcal{B}_k, H_k)$. By Assumption 3.1, this means that we can define constants $\xi_1, \xi_2 > 0$ related to the quantities in (3.10) such that no solution of subproblem (2.6) lies outside $\mathbb{B}(0, \xi_1 + \xi_2 \|\hat{d}_k\|)$. □

We are now ready to prove Theorem 3.7. Our proof closely follows that of Theorem 3.3 in [27].

*Proof.* The update condition (2.9) ensures

$$\|x_{k+1} - x_k\| \leq \min\{\alpha_k, \epsilon\}\|d_k\| + \alpha_k\|d_k\| \leq 2\alpha_k\|d_k\|. \tag{3.11}$$

This inequality holds trivially if the algorithm skips from step 4 to step 6, and by the triangle inequality if step 5 yields $x_{k+1} = x_k + \alpha_k d_k$. By condition (2.7), Lemma 3.6, Assumption 3.2, and (3.11), we have

$$\begin{aligned}\phi(x_k;\rho) - \phi(x_{k+1};\rho) &\geq \eta\alpha_k \Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) \\ &\geq \tfrac{1}{2}\eta\alpha_k\underline{\xi}\|d_k\|^2 \\ &\geq \tfrac{1}{4}\eta\underline{\xi}\|x_{k+1}-x_k\|\|d_k\|. \end{aligned}\tag{3.12}$$

Thus, since (2.7) and (3.12) hold for all $k$, we have by Assumption 3.2 that

$$\sum_{k=0}^{\infty} \alpha_k \Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) < \infty, \quad \text{and} \tag{3.13a}$$

$$\sum_{k=0}^{\infty} \|x_{k+1}-x_k\|\|d_k\| < \infty. \tag{3.13b}$$

We continue by considering two cases.

*Case 1*: Suppose there exists $k' \geq 0$ such that $\epsilon = \epsilon' > 0$ for all $k \geq k'$. According to step 4 of Algorithm 2.1, this can only occur if

$$\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) > \nu(\epsilon')^2 \quad \text{for all } k \geq k'. \tag{3.14}$$

The inequality (3.14) in conjunction with (3.13a) means that $\alpha_k \to 0$. Similarly, by Lemma 3.6, inequality (3.14) implies that $\|d_k\|$ is bounded away from zero, which in conjunction with (3.13b) means that $x_k \to x'$ for some $x'$.

If $x'$ is $\epsilon'$-stationary for $\phi(x;\rho)$, then we have $\Delta q(d';\rho,x',\mathcal{B}',H') = 0$. Thus, with $\omega = \nu(\epsilon')^2/2$ and $(\zeta,\mathcal{T})$ chosen as in Lemma 3.9, there exists $k'' \geq k'$ such that $x_k \in \mathbb{B}(x',\zeta)$ for all $k \geq k''$ and

$$\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) \leq \nu(\epsilon')^2/2 \quad \text{whenever } k \geq k'' \text{ and } \mathcal{B}_k \in \mathcal{T}. \tag{3.15}$$

Together, (3.14) and (3.15) imply that $\mathcal{B}_k \notin \mathcal{T}$ for all $k \geq k''$. However, this is a probability zero event since for all such $k$ the points in $\mathcal{B}_k$ are sampled uniformly from $\mathcal{S}(x_k)$, which includes the nonempty set $\mathcal{T}$.

If $x'$ is not $\epsilon'$-stationary, then for all $k \geq k'$ any $\alpha$ not satisfying (2.7) yields

$$\phi(x_k + \alpha d_k;\rho) - \phi(x_k;\rho) > -\eta\alpha\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k),$$

whereas (3.8) implies

$$\phi(x_k + \alpha d_k;\rho) - \phi(x_k;\rho) \leq -\alpha\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k) + \alpha^2 L_k\|d_k\|^2$$

where $L_k$ is the Lipschitz constant for the gradient of $\phi(x;\rho)$ on $[x_k, x_k + \alpha_k d_k]$. Combining these inequalities yields a lower bound on any $\alpha$ not satisfying (2.7), which, since the line search in Algorithm 2.1 has a backtracking factor of $\gamma$, yields the bound

$$\alpha_k > \gamma(1-\eta)\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k)/(L_k\|d_k\|^2).$$

However, with $\omega = \Delta q(d'; \rho, x', \mathcal{B}', H')$ (which is strictly positive since $x'$ is not $\epsilon'$-stationary) and $(\zeta, \mathcal{T})$ again chosen as in Lemma 3.9, there exists $k'' \geq k'$ such that $x_k \in \mathbb{B}(x', \zeta)$ for all $k \geq k''$ and

$$\Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) \leq 2\Delta q(d'; \rho, x', \mathcal{B}', H') \quad \text{whenever } k \geq k'' \text{ and } \mathcal{B}_k \in \mathcal{T}. \quad (3.16)$$

Under Assumptions 3.1 and 3.2, the fact that $x_k \to x'$, Lemma 3.10, and (3.16) imply that for all $k$ sufficiently large, $L_k \|d_k\|^2 \leq L$ for some constant $L > 0$, meaning that for all $k \geq k''$ such that $\mathcal{B}_k \in \mathcal{T}$, $\alpha_k$ is bounded away from zero. Together, this fact and the fact that $\alpha_k \to 0$ imply that $\mathcal{B}_k \notin \mathcal{T}$ for all $k \geq k''$, which is a probability zero event.

*Case 2*: Suppose that $\epsilon \to 0$ and $\{x_k\}$ has a cluster point $x'$. First, we show that

$$\liminf_{k \to \infty} \max\{\|x_k - x'\|, \|d_k\|\} = 0. \quad (3.17)$$

If $x_k \to x'$, then by construction in the algorithm and Lemma 3.6, $\epsilon \to 0$ if and only if there is an infinite subsequence $K$ of iterations where

$$\tfrac{1}{2}\underline{\xi}\|d_k\|^2 \leq \tfrac{1}{2} d_k^T H_k d_k = \Delta q(d_k; \rho, x_k, \mathcal{B}_k, H_k) \leq \nu\epsilon^2.$$

Thus, since $\epsilon \to 0$,

$$\lim_{k \in K} \|d_k\| = 0$$

and (3.17) follows. On the other hand, if $x_k \not\to x'$, then we proceed by contradiction and suppose that (3.17) does not hold. Since $x'$ is a cluster point of $\{x_k\}$, there is an $\epsilon' > 0$ and an index $k' \geq 0$ such that the set $K' := \{k \mid k \geq k',\ \|x_k - x'\| \leq \epsilon',\ \|d_k\| > \epsilon'\}$ is infinite. By (3.13b), this means

$$\sum_{k \in K'} \|x_{k+1} - x_k\| < \infty. \quad (3.18)$$

Since $x_k \not\to x'$, there exists an $\epsilon > 0$ such that for all $k_1 \in K'$ with $\|x_{k_1} - x'\| \leq \epsilon'/2$ there is $k_2 > k_1$ satisfying $\|x_{k_1} - x_{k_2}\| > \epsilon$ and $\|x_k - x'\| \leq \epsilon'$ for all $k_1 \leq k < k_2$. Thus, by the triangle inequality, we have $\epsilon < \|x_{k_1} - x_{k_2}\| \leq \sum_{k=k_1}^{k_2-1} \|x_{k+1} - x_k\|$. However, for $k_1 \in K'$ sufficiently large, (3.18) implies that the right-hand side of this inequality must be strictly less than $\epsilon$ (a contradiction).

Finally, since for all $k$ we have

$$\mathcal{B}_k \subset (\mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^f, \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^{c^1}, \ldots, \mathbb{B}(x_k, \epsilon) \cap \mathcal{D}^{c^m}),$$

equation (3.17) and the fact that $\epsilon \to 0$ together imply that the cluster point $x'$ is stationary for the penalty function $\phi(x; \rho)$. □

**4. An Implementation.** We implemented Algorithm 2.1 in `MATLAB`.[1] We tested a few QP solvers for subproblem (2.6) and overall found that `MOSEK` produced the best results in terms of reliability and efficiency.[2] In this section we describe some of the remaining details of our implementation. In particular, we discuss three enhancements that are designed to improve the practical performance of the method, and then comment on our choices for the input parameters.

[1] Publicly available at http://coral.ie.lehigh.edu/~frankecurtis

[2] Available at http://www.mosek.com

First, the most significant enhancement addresses the fact that Algorithm 2.1 considers only a fixed value of the penalty parameter $\rho$ throughout the solution process. If this value is too large, then Algorithm 2.1 may converge to stationary points of the penalty function $\phi(x;\rho)$ that are infeasible for the original nonlinear program (1.1). If $\rho$ is too small, then the algorithm may converge slowly as too much emphasis is placed on maintaining feasibility of the iterates. Thus, we have implemented an approach that initializes $\rho$ to a moderate value (see Table 4.2), and then decreases it only when it appears that feasibility is not being attained. We do this by defining a feasibility indicator parameter $\theta$. During the parameter reduction check in step 4 of Algorithm 2.1, if $\Delta q(d_k;\rho,x_k,\mathcal{B}_k,H_k)$ is sufficiently small so that $\epsilon$ is decreased, then our implementation also compares the current value of $\theta$ and the infeasibility measure $v(x_k)$. If $v(x_k) \leq \theta$, then the current point is both $\epsilon$-stationary and sufficiently feasible, so $\theta$ is decreased to tighten the indicator. Conversely, if $v(x_k) > \theta$, then $\rho$ is decreased to place a higher priority on attaining feasibility in the remainder of the run. We use the same reduction factor, $\beta$, for all reductions in $\epsilon$, $\rho$, and $\theta$; see Table 4.1.

We believe that this dynamic update for the penalty parameter strikes a balance between simply fixing the value at a small quantity and a process that attempts to locate stationary points for $\phi(x;\rho)$ before considering a decrease in $\rho$. More sophisticated techniques, such as the steering rules described in [9], may be fruitful, but for our purposes here we determined this basic approach to be sufficient.

Our second enhancement relates to the choice of $H_k$ made during each iteration. In an attempt to achieve a fast rate of convergence, we set this matrix as a damped BFGS approximation of the Hessian of the penalty function $\phi(x;\rho)$; e.g., see [30]. We initialize $H_0 = \mu I$ for $\mu > 0$ and then, for all $k$, employ the update

$$H_{k+1} \leftarrow H_k - \frac{H_k s_k s_k^T H_k^T}{s_k^T H_k s_k} + \frac{w_k w_k^T}{s_k^T w_k}, \tag{4.1}$$

where

$$s_k := x_{k+1} - x_k$$

is the displacement in $x$ computed during iteration $k$,

$$y_k := \nabla\phi(x_{k+1};\rho_{k+1}) - \nabla\phi(x_k;\rho_k)$$

is the corresponding displacement in the gradient of $\phi(x;\rho)$,

$$w_k := \chi_k y_k + (1-\chi_k) H_k s_k$$

and, for some $\kappa \in (0,1)$,

$$\chi_k := \begin{cases} 1 & \text{if } s_k^T y_k \geq (1-\kappa) s_k^T H_k s_k \\ (\kappa s_k^T H_k s_k)/(s_k^T H_k s_k - s_k^T y_k) & \text{otherwise.} \end{cases}$$

A few remarks are necessary here. First, it should be noted that changes in the penalty parameter value will have an effect on this update in a manner that is difficult to quantify. For example, if $\rho = \rho_k$ at the start of iteration $k$ differs from $\rho = \rho_{k+1}$ at the start of iteration $k+1$, then $H_k$ is intended to be an approximation of $\nabla^2\phi(x;\rho_k)$ while $H_{k+1}$ is intended to be an approximation of $\nabla^2\phi(x;\rho_{k+1})$. The update (4.1), however,

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $p$ | 2n | $\beta$ | 5e-1 |
| $\eta$ | 1e-8 | $\gamma$ | 5e-1 |
| $\nu$ | 2e+4 | $\kappa$ | 2e-1 |
| $\underline{\xi}$ | 1e-4 | $\overline{\xi}$ | 1e+4 |

TABLE 4.1
*Input values for the static parameters for Algorithm 2.1*

makes no distinction between this case and that when $\rho_k = \rho_{k+1}$. In our implementation, we simply disregard this occurrence and update the Hessian according to (4.1). A second issue is that, because we skip a differentiability check to ensure that the algorithm maintains $x_k \in \mathcal{D}$ for all $k$, there is a (probability zero) chance that we will compute an iterate at which the penalty function is nondifferentiable, in which case $y_k$ is not well-defined. In our implementation, we also disregard this event and simply compute an arbitrary subgradient of $\phi(x;\rho_{k+1})$ at such points. Finally, the BFGS matrix $H_k$ will likely become ill-conditioned as iterates approach stationary points of the penalty function $\phi(x;\rho)$; see [28]. Thus, in our implementation, we perturb each update so that the resulting matrix $H_{k+1}$ satisfies Assumption 3.2 for predefined $\underline{\xi}$ and $\overline{\xi}$ (see Table 4.1). More precisely, we use MATLAB's built-in `eigs` function to compute the smallest and largest eigenvalues of the right-hand side of (4.1) and, if necessary, replace (4.1) with

$$H_{k+1} \leftarrow (1-\tau)\left(H_k - \frac{H_k s_k s_k^T H_k^T}{s_k^T H_k s_k} + \frac{w_k w_k^T}{s_k^T w_k}\right) + \tau I, \tag{4.2}$$

where $\tau \in [0,1]$ is the smallest positive scalar such that Assumption 3.2 holds.

The third enhancement to Algorithm 2.1 that has been implemented in our code is a special handling of functions that are known to be smooth. Indeed, if a function is known to be continuously differentiable everywhere in $\mathbb{R}^n$, then one can argue that for that function it is not necessary to sample gradients at nearby points! (We remark that, with this enhancement, if all functions are smooth, then our approach reduces to a standard type of penalty-SQP method.) Specifically, if the objective ($j$th constraint) is known to be smooth, then this is equivalent to setting $\mathcal{B}_k^f = \{x_k\}$ ($\mathcal{B}_k^{c^j} = \{x_k\}$) for all $k$. This choice can have a significant impact on the practical performance of the algorithm as this effectively eliminates $pn_s$ linear inequality constraints from the quadratic program (2.5), where $n_s$ is the number of smooth functions; otherwise, the subproblems will always have $p(m+1)$ such constraints in addition to the $m$ bounds on the auxiliary variables.

Finally, we use the values for the static input parameters indicated in Table 4.1, as we found them to yield the best results for the experiments in §5. Initial values for the dynamic parameters are provided in Table 4.2. A sample size of $p = n+1$ is all that is required in our analysis, but for the same reasons as described in [7] we instead choose $p = 2n$. The chosen values for $\eta$ and $\gamma$ are standard for line search methods. The remaining values were decided upon during the course of our tests.

**5. Numerical experiments.** In this section we discuss the results of our MATLAB implementation of Algorithm 2.1 applied to a set of test problems. The first problem is somewhat contrived, but is still an interesting illustrative example in that it involves the minimization of a classically difficult type of objective function. All of

| Parameter | Initial Value |
|---|---|
| $\epsilon$ | 1e-1 |
| $\rho$ | 1e-1 |
| $\theta$ | 1e-1 |

TABLE 4.2
*Initial values for the dynamic parameters for Algorithm 2.1*

the remaining problems are derived from real applications. We remark that in a few cases, special purpose solvers designed for the individual applications will most likely provide better results than are presented here, but it should be noted that ours is only a preliminary implementation and, in fact, there are natural ways in which our approach can be tailored to each problem individually (see §6). Thus, for our purposes here, we simply apply our general purpose implementation of Algorithm 2.1 and illustrate that it is effective on a wide range of problems.

For each problem we ran our implementation with 10 different starting points, each sampled from a standard normal distribution. Due to the stochastic nature of the algorithm itself, we could have decided to run the algorithm from the *same* starting point multiple times, expecting to find variations in the performance over each run. However, in the end we found that the algorithm behaved consistently even for different starting points, and so we provide the results for these runs with the idea that they more clearly indicate the robustness of the approach. In all cases we imposed a maximum iteration limit of 500. Our code also includes an optimality check that terminates the solver when a sufficiently feasible $\epsilon$-stationary point has been found (for any small $\epsilon > 0$ provided by the user), but in our tests we turned this feature off and simply let the code run until the iteration limit was reached, allowing $\epsilon$ to decrease indefinitely.

EXAMPLE 5.1. *Find the minimizer of a nonsmooth Rosenbrock function subject to an inequality constraint on the weighted maximum value of the variables:*

$$\begin{aligned} &\min_x \ w|x_1^2 - x_2| + (1 - x_1)^2 \\ &s.t. \ \max\{c_1 x_1, c_2 x_2\} \leq 1. \end{aligned} \tag{5.1}$$

The Rosenbrock function is a standard smooth, nonconvex optimization test problem, and so the nonsmooth variation in problem (5.1) is an interesting one to consider for our method. Defining the objective weight to be $w = 8$ and the coefficients to be $(c_1, c_2) = (\sqrt{2}, 2)$, we illustrate this problem on the left-hand side of Figure 5.1. The curved contours of the objective illustrate that the unconstrained minimizer lies at $(1, 1)$. The feasible region, however, is the rectangle including the lower left-hand corner of the graph, so the solution of the constrained problem is $x_* = (\frac{\sqrt{2}}{2}, \frac{1}{2})$, a point at which both the objective and the constraint functions are nondifferentiable.

In all of our runs, Algorithm 2.1 converged rapidly to $x_*$. On the right-hand side of Figure 5.1, we plot $\|x_k - x_*\|$ with respect to $k$ and note that in all cases this distance was less than 1e-6 within 60 iterations.

EXAMPLE 5.2. *Find an $N \times N$ matrix $X$ with normalized columns so that the*

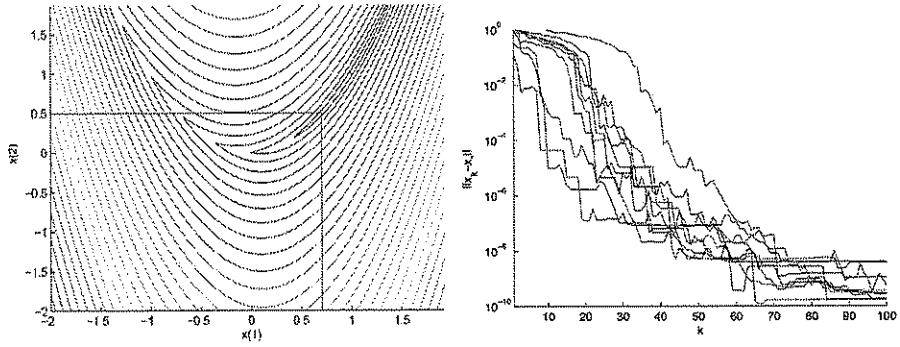

FIG. 5.1. *Illustration of the objective contours and feasible region for an instance of Example 5.1 (left) and a plot of $\|x_k - x_*\|$ with respect to $k$ for 10 runs of Algorithm 2.1 (right).*

*product of the $K$ largest eigenvalues of the matrix $A \circ X^T X$ is minimized:*

$$\begin{aligned} \min_X \ & \ln\left(\prod_{j=1}^{K} \lambda_j(A \circ X^T X)\right) \\ s.t. \ & \|X_j\|^2 = 1, \ j = 1, \ldots, N, \end{aligned} \tag{5.2}$$

*where $\lambda_j(M)$ denotes the $j$th largest eigenvalue of the matrix $M$, $A$ is a real symmetric $N \times N$ matrix, $\circ$ denotes the Hadamard matrix product, and $X_j$ represents the $j$th column of $X$.*

Example 5.2 is a nonconvex relaxation of an entropy minimization problem arising in an environmental application [1]. We remark that this problem is one of the examples in [7], where the variables were defined as the off-diagonal entries in $M = X^T X$ and $M$ was forced to be positive semidefinite through a penalty term in the objective, and in [28], where the constraint is enforced through a redefinition of the objective function. However, in our case, we can simply impose that the columns of $X$ are normalized by defining the constraints in (5.2), with the equalities defined as pairs of inequalities as described in §1.

| $N$ | $K$ | $n$ | $f_*$ | $v_*$ |
|---|---|---|---|---|
| 2 | 1 | 4 | 1.000000e+00 | 2.976561e-10 |
| 4 | 2 | 16 | 7.462929e-01 | 1.975641e-09 |
| 6 | 3 | 36 | 6.336426e-01 | 7.525639e-08 |
| 8 | 4 | 64 | 5.598340e-01 | 2.077412e-07 |
| 10 | 5 | 100 | 2.182082e-01 | 1.946443e-06 |
| 12 | 6 | 144 | 1.225451e-01 | 9.199095e-06 |
| 14 | 7 | 196 | 8.361197e-02 | 1.053643e-05 |
| 16 | 8 | 256 | 5.690842e-02 | 4.976905e-06 |

TABLE 5.1
*Results for Example 5.2 for various values of $N$.*

Table 5.1 shows the results of Algorithm 2.1 applied to problem (5.2) for various values of $N$, where in each case we choose $K = N/2$. We set the matrix $A$ to be the

$N \times N$ leading submatrix of a $63 \times 63$ covariance data matrix.[3] The data we provide are the number of optimization variables ($n$), the best value of the objective value obtained over all the runs ($f_*$), and the value of the infeasibility measure (see $v(x)$ in §2) corresponding to this best objective value ($v_*$). The results are comparable to those obtained in [7].

EXAMPLE 5.3. *Find the minimum $\ell_p$-norm vector that approximates the solution of a linear system insofar as the residual is within a given tolerance:*

$$\begin{aligned} &\min_x \|x\|_p \\ &s.t.\ \|Rx - y\| \leq \delta, \end{aligned} \tag{5.3}$$

*where $p > 0$, $R \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$, and $\delta \geq 0$ is an error tolerance.*

Example 5.3 is a problem of interest in compressed sensing, where the goal is to reconstruct a sparse vector $x_*$ of length $n$ from $m < n$ observations. In our test, we choose $m = 32$ and $n = 256$ and generate $R$ by taking the first $m$ rows of the $n \times n$ discrete cosine transform (DCT) matrix. The vector we aim to recover, $x_*$, is chosen to take entries equal to 0 with probability 0.90 and entries that have a standard normal distribution with probability 0.1; see Figure 5.2. The vector $x_*$ that we generated has 19 nonzero entries.

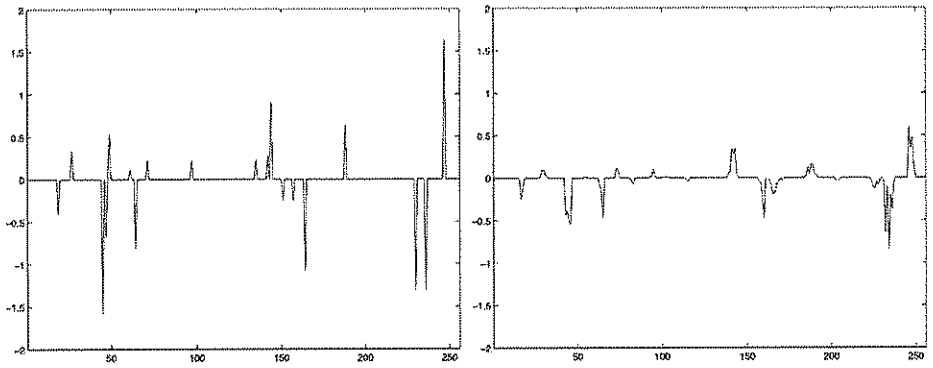

FIG. 5.2. *$x_*$ (left) and $x_{500}$ (right) for Example 5.3 with $p = 1$ and exact data.*

We consider four instances of problem (5.3), each with the same $R$ and $x_*$. In the first two instances, $y = Rx_*$ and $\delta = 0$. We then choose two values of $p$. First, for $p = 1$, we have the common $l_1$-norm minimization problem; e.g., see [10, 12]. On the right-hand side of Figure 5.2 we plot the best solution obtained by Algorithm 2.1. (Note that for this and all other instances of problem (5.3), we define the best solution as that yielding the lowest objective value). It can be seen in Figure 5.2 that the solution vector $x_*$ is not recovered exactly with this choice of $p$. Indeed, defining a "non-zero" entry as any having an absolute value greater than `1e-2`, the illustrated solution has 70 non-zero entries. Thus, in an attempt to steer the algorithm toward sparser solutions, our second set of runs was performed with $p = 0.5$; again, see [10, 12]. In this case, the best solution obtained by Algorithm 2.1 is plotted along with $x_*$ in Figure 5.3. This instance is nonconvex, with multiple local minimizers, and non-Lipschitz. However, we do consistently find that a solution with few non-zero entries is found; e.g., the solution in Figure 5.3 has only 26.

[3] Publicly available at http://www.cs.nyu.edu/faculty/overton/papers/gradsamp/probs/

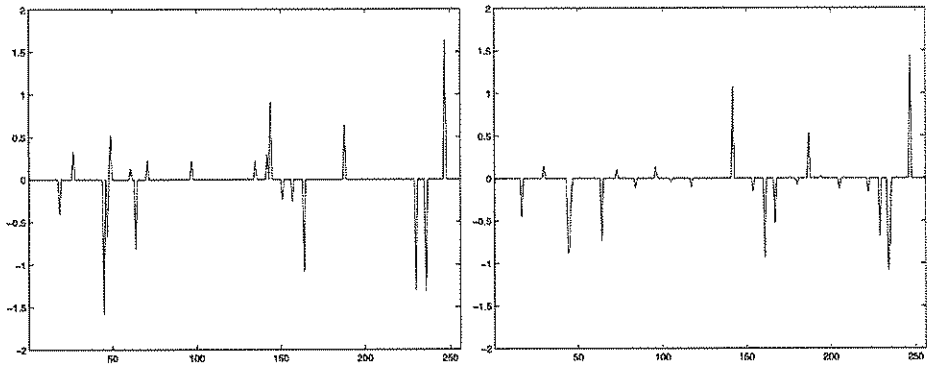

FIG. 5.3. $x_*$ *(left) and* $x_{500}$ *(right) for Example 5.3 with* $p = 0.5$ *and exact data.*

In our other two instances of problem (5.3), $y = Rx_* + e$, where $e$ is a noise vector with element $e_j$ distributed uniformly in $[R_j x_* - 0.005|R_j x_*|, R_j x_* + 0.005|R_j x_*|]$; i.e., there is up to a 0.5% difference (component-wise) between $y$ and $Rx_*$. We assume that this error level is overestimated and so choose $\delta = 2\|Rx_* - y\|$. For $p = 1$ and $p = 0.5$, respectively, the best solutions obtained are plotted in Figure 5.4. For $p = 1$, the illustrated solution has 66 non-zero entries, while for $p = 0.5$, the illustrated solution has only 25.

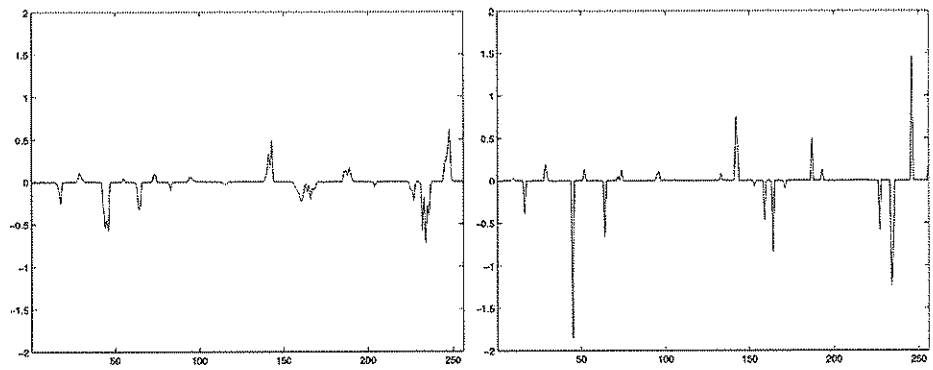

FIG. 5.4. $x_{500}$ *when* $p = 1.0$ *(left) and* $x_{500}$ *when* $p = 0.5$ *(right) for Example 5.3 with noisy data.*

EXAMPLE 5.4. *Find the robust minimizer of a linear objective function subject to an uncertain convex quadratic constraint:*

$$\begin{aligned} &\min_x \ f^T x \\ &s.t. \ x^T A x + b^T x + c \leq 0, \ \forall (A, b, c) \in \mathcal{U}, \end{aligned} \tag{5.4}$$

*where* $f \in \mathbb{R}^n$ *and for each* $(A, b, c)$ *in the uncertainty set* $\mathcal{U}$, $A$ *is an* $n \times n$ *positive semidefinite matrix,* $b$ *is a vector in* $\mathbb{R}^n$, *and* $c$ *is a real scalar value.*

Example 5.4 is a problem of interest in robust optimization. In this modeling framework, the primary concern is that the parameters defining an optimization problem are, in practice, estimated, and are therefore subject to measurement and

statistical errors. Since the solution is typically sensitive to perturbations in these parameters, it is important to take data uncertainty into account in the optimization process. For instance, in problem (5.4), we require that any solution vector $x_*$ satisfy the convex quadratic constraint no matter which instance in the uncertainty set $\mathcal{U}$ is realized. Problems of this type have applications in, for example, robust mean-variance portfolio selection [16], least-squares problems [2], and data-fitting [38].

We generated random data for an instance of problem (5.4) with $n = 100$ unknowns. The uncertainty set was defined to be

$$\mathcal{U} := \left\{ (A,b,c) : (A,b,c) = (A^{(0)},b^{(0)},c^{(0)}) + \sum_{i=1}^{10} u^i (A^{(i)},b^{(i)},c^{(i)}),\ u^T u \le 1 \right\},$$

where $A^{(0)}$, $i = 0,\ldots,10$, were generated randomly using `MATLAB`'s built-in `sprandsym` function. In particular, $A^{(0)}$ was a symmetric $100 \times 100$ matrix with approximately $0.1n^2$ nonzero entries, generated by a shifted sum of outer products so that the condition number was approximately 10 (i.e., $A^{(0)}$ was set by `sprandsym(100,0.1,0.1,2)`), and $A^{(i)}$ for $i = 1,\ldots,10$ was generated in the same way, but multiplied by the scalar factor 0.01. The elements of all of the $b^{(i)}$ vectors had a standard normal distribution. Finally, $c^{(0)} = -10$ while $c^{(i)}$ for $i = 1,\ldots,10$ were uniformly distributed on $[-1,1]$. The feasible region was verified to be nonempty as it included the origin. At a given $x_k$, the corresponding $u_k$ in $\mathcal{U}$ was computed by minimizing a linear function over the closed unit ball, with which the constraint value and gradient was obtained. We remark that, as described in [2], this instance can be reformulated and solved as an equivalent semidefinite program. However, our algorithm can also be applied to more general robust nonlinear programs that do not necessarily have convenient reformulations.

Algorithm 2.1 generally converges rapidly on this instance of problem (5.4). In Figure 5.5, we plot the objective function value $f(x_k)$ with respect to $k$. In most instances, the function values converge to the optimal value within 40 to 60 iterations. However, due to the stochastic nature of the algorithm and the different starting points, in three runs the algorithm required more iterations to find a solution of similar quality. (The visible kinks in these three plotted lines indicate when the algorithm finally sampled a set of points resulting in a solution of the QP subproblem that invoked an initial decrease in $\epsilon$, after which the convergence was more rapid.) Despite the slight inconsistency in the convergence behavior exhibited here, the results appear to indicate that (perhaps with a fine-tuning of the input parameters) Algorithm 2.1 can successfully solve robust optimization problems along the lines of Example 5.4.

**6. Conclusion.** We have proposed and analyzed an algorithm for nonconvex, nonsmooth constrained optimization. The method is based on a sequential quadratic programming framework where a gradient sampling technique is used to create robust linear models of the objective and constraint functions around the current iterate. We have shown that if the problem functions are locally Lipschitz in $\mathbb{R}^n$ and continuously differentiable almost everywhere, then the algorithm successfully locates stationary points of a penalty function. Preliminary numerical experiments illustrate that the algorithm is robust and applicable to numerous types of problems.

We believe that our approach can be fine-tuned for individual applications. For example, for Example 5.3 with $\delta = 0$ and $\rho$ sufficiently small, at any feasible iterate the search directions will subsequently always be computed in the null space of $R$. Thus, one may consider sampling gradients at points only in this space, and may

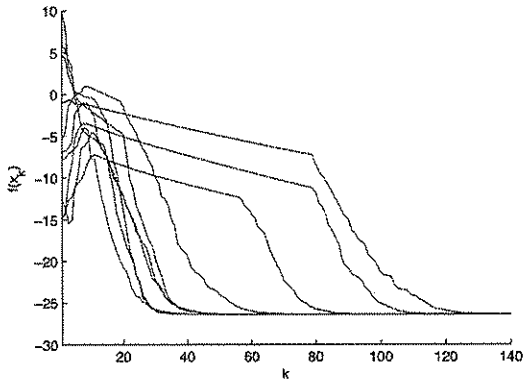


FIG. 5.5. *$f(x_k)$ with respect to $k$ for 10 runs of Algorithm 2.1 on an instance of Example* (5.4).

consider fewer points overall; e.g., $n - m + 1$ as opposed to $n + 1$. More generally, if the given problem is convex, then subproblem (2.5) may be replaced by

$$\begin{aligned} &\min_{d,z} \; z + \tfrac{1}{2} d^T H_k d \\ &\text{s.t.} \; \begin{cases} f(x_k) + \nabla f(x)^T d \le z, \; \forall x \in \mathcal{B}_k^f \\ c^j(x_k) + \nabla c^j(x)^T d \le 0, \; \forall x \in \mathcal{B}_k^{c^j}, \; j = 1, \ldots, m, \end{cases} \end{aligned} \tag{6.1}$$

(i.e., the auxiliary variables $r$ may be fixed to zero) after which the penalty parameter can be decreased, if necessary, to ensure that the computed search direction is one of descent for the penalty function. The main idea here is to reduce the size of the QP subproblem, as its solution is the main computational expense of the approach.

Finally, we remark that the SQP-GS algorithm discussed in this paper can easily be altered into a sequential linear programming algorithm with gradient sampling (SLP-GS). In particular, the quadratic term in (2.5) can be replaced by a trust region constraint on the search direction, yielding the linear programming subproblem

$$\begin{aligned} &\min_{d,z,r} \; \rho z + \sum_{j=1}^{m} r^j \\ &\text{s.t.} \; \begin{cases} f(x_k) + \nabla f(x)^T d \le z, \; \forall x \in \mathcal{B}_k^f \\ c^j(x_k) + \nabla c^j(x)^T d \le r^j, \; \forall x \in \mathcal{B}_k^{c^j}, \; r^j \ge 0, \; j = 1, \ldots, m \\ \qquad\qquad \|d\|_\infty \le \Delta_k \end{cases} \end{aligned} \tag{6.2}$$

where $\Delta_k$ is a trust region radius. Our MATLAB implementation contains an option for running SLP-GS instead of SQP-GS. However, although the computation time per iteration will generally be less for SLP-GS, the rate of convergence is typically much slower when compared to SQP-GS.

*Acknowledgements.* The authors would like to thank Adrian S. Lewis for many useful comments and suggestions that significantly improved the paper.

## REFERENCES


[1] K. M. Anstreicher and J. Lee, *A Masked Spectral Bound for Maximum-Entropy Sampling*, in MODA 7 – Advances in Model-Oriented Design and Analysis, A. Di Bucchianico, H. Läuter, and H. P. Wynn, eds., Berlin, 2004, Springer-Verlag, pp. 1–10.

[2] A. Ben-Tal and A. Nemirovski, *Robust Optimization - Methodology and Applications*, Mathematical Programming, Series B, 92 (2002), pp. 453–480.

[3] J. V. Burke, *An Exact Penalization Viewpoint of Constrained Optimization*, SIAM Journal on Control and Optimization, 29 (1991), pp. 968–998.

[4] ———, *Calmness and Exact Penalization*, SIAM Journal on Control and Optimization, 29 (1991), pp. 493–497.

[5] J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton, *HIFOO - A MATLAB Package for Fixed-order Controller Design and H-infinity Optimization*, in Proceedings of the IFAC Symposium on Robust Control Design, Haifa, 2006.

[6] J. V. Burke, A. S. Lewis, and M. L. Overton, *Approximating Subdifferentials by Random Sampling of Gradients*, Mathematics of Operations Research, 27 (2002), pp. 567–584.

[7] ———, *A Robust Gradient Sampling Algorithm for Nonsmooth, Nonconvex Optimization*, SIAM Journal on Optimization, 15 (2005), pp. 751–779.

[8] R. H. Byrd, G. López-Calva, and J. Nocedal, *A Line Search Penalty Method for Nonlinear Optimization*, Technical Report 08/05, Optimization Technology Center, Evanston, IL, USA, 2008.

[9] R. H. Byrd, J. Nocedal, and R. A. Waltz, *Steering Exact Penalty Methods for Nonlinear Programming*, Optimization Methods and Software, 23 (2008), pp. 197–213.

[10] E. J. Candès and T. Tao, *Near Optimal Signal Recovery from Random Projections: Universal Encoding Strategies*, IEEE Transactions on Information Theory, 52 (2006), pp. 5406–5425.

[11] F. H. Clarke, *Optimization and Nonsmooth Analysis*, Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley & Sons, New York, NY, 1983. Reprinted by SIAM, Philadelphia, 1990.

[12] D. L. Donoho, *Compressed Sensing*, IEEE Transactions on Information Theory, 52 (2006), pp. 1289–1306.

[13] D. Dotta, A. S. E Silva, and I. C. Decker, *Design of Power System Controllers by Nonsmooth, Nonconvex Optimization*, in Proceedings of the IEEE Power and Energy Society General Meeting, Calgary, 2009.

[14] R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, New York, NY, 1987.

[15] M. Fukushima, *A Successive Quadratic Programming Method for a Class of Constrained Nonsmooth Optimization Problems*, Mathematical Programming, 49 (1990), pp. 231–251.

[16] D. Goldfarb and G. Iyengar, *Robust Portfolio Selection Problems*, Mathematics of Operations Research, 28 (2003), pp. 1–38.

[17] S. Guenter, M. Sempf, P. Merkel, E. Strumberger, and C. Tichmann, *Robust control of resistive wall modes using pseudospectra*, New Journal of Physics, 11 (2009), pp. 1–40.

[18] S. Gumussoy, D. Henrion, M. Millstone, and M. L. Overton, *Multiobjective Robust Control with HIFOO 2.0*, in Proceedings of the IFAC Symposium on Robust Control Design, Haifa, 2009.

[19] S. P. Han, *A Globally Convergent Method for Nonlinear Programming*, Journal of Optimization Theory and Applications, 22 (1977), pp. 297–309.

[20] S. P. Han and O. L. Mangasarian, *Exact Penalty Functions in Nonlinear Programming*, Mathematical Programming, 17 (1979), pp. 251–269.

[21] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms II*, A Series of Comprehensive Studies in Mathematics, Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[22] E. W. Karas, A. Ribeiro, C. A. Sagastizábal, and M. V. Solodov, *A Bundle-Filter Method for Nonsmooth Convex Constrained Optimization*, Mathematical Programming, 116 (2007), pp. 297–320.

[23] K. C. Kiwiel, *An Exact Penalty Function Algorithm for Non-smooth Convex Constrained Minimization Problems*, IMA Journal of Numerical Analysis, 5 (1985), pp. 111–119.

[24] ———, *Methods of Descent for Nondifferentiable Optimization*, Lecture Notes in Mathematics, Springer-Verlag, Berlin, New York, 1985.

[25] ———, *A Constraint Linearization Method for Nondifferentiable Convex Minimization*, Numerische Mathematik, 51 (1987), pp. 395–414.

[26] ———, *Exact Penalty Functions in Proximal Bundle Methods for Constrained Convex Nondifferentiable Minimization*, Mathematical Programming, 52 (1991), pp. 285–302.

[27] ———, *Convergence of the Gradient Sampling Algorithm for Nonsmooth Nonconvex Optimiza-*

*tion*, SIAM Journal on Optimization, 18 (2007), pp. 379–388.

[28] A. S. Lewis and M. L. Overton, *Nonsmooth Optimization via BFGS*, SIAM Journal on Optimization, submitted for publication, (2009).

[29] R. Mifflin, *An Algorithm for Constrained Optimization with Semismooth Functions*, Mathematics of Operations Research, 2 (1977), pp. 191–207.

[30] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, New York, NY, second ed., 2006.

[31] E. Polak, D. Q. Mayne, and Y. Wardi, *On the Extension of Constrained Optimization Algorithms from Differentiable to Nondifferentiable Problems*, SIAM Journal on Control and Optimization, 21 (1983), pp. 179–203.

[32] M. J. D. Powell, *A Fast Algorithm for Nonlinearly Constrained Optimization Calculations*, Lecture Notes in Mathematics, Springer, Berlin, Heidelberg, 1978, pp. 144–157.

[33] ———, *Variable Metric Methods for Constrained Optimization*, Springer-Verlag, Berlin, Heidelberg, New York, Toronto, 1983, pp. 288–311.

[34] R. T. Rockafellar, *Convex Analysis*, Princeton Landmarks in Mathematics and Physics, Princeton University Press, Princeton, NJ, 1970.

[35] ———, *Lagrange Multipliers and Subderivatives of Optimal Value Functions in Nonlinear Programming*, no. 17 in Mathematical Programming Studies, North-Holland Publishing Company, Amsterdam, 1982, ch. 3, pp. 28–66.

[36] E. Rosenberg, *Exact Penalty Functions and Stability in Locally Lipschitz Programming*, Mathematical programming, 30 (1984), pp. 340–356.

[37] J. Vanbiervliet, K. Verheyden, W. Michiels, and S. Vandewalle, *A Nonsmooth Optimisation Approach for the Stabilisation of Time-delay Systems*, ESAIM: Control, Optimisation and Calculus of Variations, 14 (2008), pp. 478–493.

[38] G. A. Watson, *Data Fitting Problems with Bounded Uncertainties in the Data*, SIAM Journal on Matrix Analysis and Applications, 22 (2001), pp. 1274–1293.