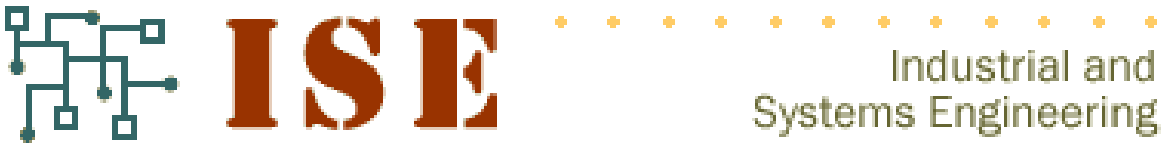

A Quasi-Newton Algorithm for Nonconvex, Nonsmooth Optimization with Global Convergence Guarantees

Frank E. Curtis and Xiaocun Que

Department of Industrial and Systems Engineering, Lehigh University, USA

COR@L Technical Report 14T-002-R1

Noname manuscript No.
(will be inserted by the editor)

A Quasi-Newton Algorithm for Nonconvex, Nonsmooth Optimization with Global Convergence Guarantees

Frank E. Curtis · Xiaocun Que

March 30, 2015

Abstract A line search algorithm for minimizing nonconvex and/or nonsmooth objective functions is presented. The algorithm is a hybrid between a standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) and an adaptive gradient sampling (GS) method. The BFGS strategy is employed because it typically yields fast convergence to the vicinity of a stationary point, and together with the adaptive GS strategy the algorithm ensures that convergence will continue to such a point. Under loose assumptions, it is proved that the algorithm converges globally with probability one. The algorithm has been implemented in C++ and the results of numerical experiments illustrate the efficacy of the proposed approach.

Keywords nonsmooth optimization, nonconvex optimization, unconstrained optimization, quasi-Newton methods, gradient sampling, line search methods

Mathematics Subject Classification (2000) 49M05 · 65K05 · 65K10 · 90C26 · 90C30 · 90C53 · 93B40

1 Introduction

We propose an algorithm for minimizing a locally Lipschitz objective function that is continuously differentiable in an open, dense subset of a real vector space. Such a function may be nonsmooth and/or nonconvex, which precludes well-known techniques for solving convex optimization problems that have been developed in recent decades. Applications in which problems of this type arise include, e.g., robust control [22, 23, 46, 45], robust optimization [2, 19, 47], image restoration [10,

Both authors were supported in part by National Science Foundation grant DMS-1016291.

Frank E. Curtis
Dept. of Industrial & Systems Engineering, Lehigh University, Bethlehem, PA 18018, USA.
E-mail: frank.e.curtis@gmail.com

Xiaocun Que
Dept. of Industrial & Systems Engineering, Lehigh University, Bethlehem, PA 18018, USA.
E-mail: xiq209@lehigh.edu

11], eigenvalue optimization [1], compressed sensing [8, 9, 16], and decomposition methods for large-scale or complex optimization problems [4, 41].

Our algorithm is based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [5, 17, 18, 43]. Since its inception, this approach—arguably the most effective quasi-Newton method [39]—has been extremely popular for solving smooth optimization problems. This popularity stems from the fact that the method only requires first derivatives of the objective function, and yet can achieve a local superlinear rate of convergence. Moreover, many have witnessed good performance of BFGS when solving nonsmooth problems [24, 25], despite the fact that global convergence guarantees for the algorithm in this context are rather limited [35]. In order to overcome this theoretical deficiency, our algorithm enhances BFGS with an adaptive gradient sampling (GS) strategy adopted from the method in [14]. With this enhancement, as well as other practical features, we have designed an algorithm that exhibits good practical behavior, and for which we have established global convergence guarantees under loose assumptions.

A feature critical to the practical performance of our algorithm is that, when it is applied to solve many problem instances, the algorithm reduces to an unadulterated BFGS strategy for the majority of the iterations. This feature is intentional, and is motivated by the encouraging results presented in [35]. Indeed, a straightforward BFGS algorithm applied to solve a nonsmooth, nonconvex optimization problem is often very effective in making progress toward a solution. However, it suffers from two important drawbacks: (*i*) it does not inherently offer termination conditions related to a stationarity measure that can be guaranteed to be satisfied in the limit, meaning that there is no immediate way of determining whether a solution has been reached; and (*ii*) guaranteeing global convergence appears to be difficult in general because the inverse Hessian approximations may tend to singularity in the neighborhood of any solution point at which the objective function is not differentiable. Overall, these deficiencies suggest that while BFGS may be able to converge to a neighborhood of a solution, enhancements—such as our adaptive GS procedure—may be needed to obtain high accuracy and provide the means to guarantee a certificate of stationarity.

The GS algorithm was introduced by Burke, Lewis, and Overton [7]. It employs a strategy of randomly sampling gradients to approximate the ϵ-subdifferential of the objective about each iterate [6]. The algorithm was proposed as a strategy for establishing global convergence guarantees when solving nonconvex, locally Lipschitz optimization problems. Enhancements to the algorithm have also been established over the past few years, both to improve the theoretical and practical behavior of the algorithm [14, 32] and to extend the methodology to broader classes of problems [13, 26, 27, 33]. The main disadvantage of the algorithm, however, is that each iteration is significantly more expensive than a BFGS iteration. Moreover, the algorithm in [7] does not employ variable-metric Hessian approximations, and thus it may fail to fully capture the curvature information that makes the BFGS method so effective. These disadvantages motivated the enhancements proposed in [14], though a drawback of the algorithm in that paper is that each iteration requires sampling gradient information. The subproblems that arise in the GS algorithm and its variants are related to those in the popular class of bundle methods [28, 30], which were initially developed for solving convex minimization problems, but for which there are enhancements for handling nonconvexity [24, 25, 29]. The subproblems are distinct, however, in that bundle methods are

based on a methodology of computing cutting planes—i.e., affine underestimators of the objective function—whereas the GS algorithm and its variants do not involve cutting planes, even when the objective function is convex; see [13] for further discussion on the difference between GS and bundle methods.

In summary, our proposed BFGS-GS algorithm possesses theoretical and practical advantages. It typically behaves as an unadulterated BFGS algorithm, and thus often converges to a neighborhood of a solution with a computational effort on the order of one gradient evaluation and one matrix-vector product per iteration. Throughout, the algorithm dynamically employs an adaptive GS strategy in order to provide a practical stationarity certificate as well as global convergence guarantees. Careful attention has been paid to the design of our line search, sample set update, and inverse Hessian approximation subroutines so that the algorithm attains this desirable behavior. For example, in certain situations, we replace a BFGS inverse Hessian approximation with a carefully constructed limited memory BFGS (i.e., L-BFGS [38]) approximation to ensure positive definiteness and boundedness. We have also implemented the algorithm in C++ and performed a variety of experiments that illustrate the efficacy of the proposed method.

In §2, we present our main algorithm, including its relevant subroutines for the line search, sample set update, and inverse Hessian approximation strategies. We then analyze the well-posedness and global convergence properties of the algorithm in §3, building on results proved during the algorithmic development in §2. An implementation of our algorithmic framework and the results of numerical experiments on a set of test problems is the subject of §4.

Notation and definitions

The sets of n-dimensional real, natural, and positive natural numbers are denoted by $\mathbb{R}^n$, $\mathbb{N}^n$, and $\mathbb{N}_+^n$, respectively, where $\mathbb{N} := \{0, 1, 2, \dots\}$ and $\mathbb{N}_+ := \{1, 2, \dots\}$. The ith element of a vector $x \in \mathbb{R}^n$ is written as x^i. We denote the closure and convex hull of a subset $S \subseteq \mathbb{R}^n$ as $\operatorname{cl} S$ and $\operatorname{conv} S$, respectively. The closed Euclidean ball with radius $\epsilon > 0$ about $x \in \mathbb{R}^n$ is denoted as $\mathbb{B}_\epsilon(x) := \{\overline{x} \in \mathbb{R}^n : \|\overline{x} - x\|_2 \leq \epsilon\}$. The cardinality of a finite subset $S \subset \mathbb{R}^n$ is written as $|S| \in \mathbb{N}$. For a matrix W, we write $W \succ 0$ to indicate that W is real, symmetric, and positive definite. Given $W \succ 0$ and $x \in \mathbb{R}^n$, we define the "W-norm" of x as $\|x\|_W := \|W^{1/2}x\|_2$ so that $\|x\|_W^2 = x^T W x$. Given $W \succ 0$ and nonempty bounded $S \subseteq \mathbb{R}^n$, we define the (oblique) "W-projection" of the origin onto $\operatorname{cl}\operatorname{conv} S$ as $P_W(S)$, which is the unique solution of $\min_x \|x\|_W^2$ subject to $x \in \operatorname{cl}\operatorname{conv} S$. The quantities e and I respectively represent a vector of ones and an identity matrix whose sizes are determined by the context in which each quantity appears. For $\{a, b\} \subset \mathbb{R}^n$, we write $a \perp b$ to indicate that a and b are complementary, i.e., that $a^i b^i = 0$ for all $i \in \{1, \dots, n\}$. We use a subscript for a quantity to denote the iteration number of an algorithm to which it corresponds; e.g., the value for a vector x in the kth iteration of an algorithm is written as x_k. If the limit of a sequence $\{a_k\}$ as k tends to infinity (i.e., $k \to \infty$) exists and equals a, then we write $\{a_k\} \to a$.

For a function $f : \mathbb{R}^n \to \mathbb{R}$, the sublevel set corresponding to a point $x \in \mathbb{R}^n$ is written as $\mathcal{L}_f(x) := \{\overline{x} \in \mathbb{R}^n : f(\overline{x}) \leq f(x)\}$. Such a function is locally Lipschitz over $\mathbb{R}^n$ if for every compact subset $S \subset \mathbb{R}^n$, there exists a constant $L_S \geq 0$ such that $|f(x) - f(y)| \leq L_S \|x - y\|_2$ for any $\{x, y\} \subseteq S$. If f is locally Lipschitz on $\mathbb{R}^n$,

then the Clarke subdifferential [12] of f at x can be written as

$$\partial f(x) := \bigcap_{\epsilon>0} \operatorname{cl}\operatorname{conv} \nabla f(\mathbb{B}_\epsilon(x) \cap \mathcal{D}),$$

and the Clarke ϵ-subdifferential [20] of f at x is $\partial_\epsilon f(x) := \operatorname{cl}\operatorname{conv} \partial f(\mathbb{B}_\epsilon(x))$. For such a function, a point $x \in \mathbb{R}^n$ is Clarke stationary if $0 \in \partial f(x)$, and is Clarke ϵ-stationary if $0 \in \partial_\epsilon f(x)$. For the sake of brevity, hereafter we drop the distinction "Clarke" from all of the terms defined here.

2 Algorithm Description

Consider the unconstrained optimization problem

$$\min_{x\in\mathbb{R}^n} \; f(x), \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ satisfies the following assumption.

Assumption 2.1 *The objective function $f : \mathbb{R}^n \to \mathbb{R}$ of problem (2.1) is locally Lipschitz over $\mathbb{R}^n$ and continuously differentiable in an open, dense subset $\mathcal{D}$ of $\mathbb{R}^n$.*

Given an initial iterate $x_0 \in \mathcal{D}$, our desire is to compute a solution of (2.1). However, since f may be nonconvex and/or nonsmooth, our algorithm is designed merely to locate a stationary point for f in the sublevel set $\mathcal{L}_f(x_0)$. More precisely, it is designed to compute a sequence of (approximately) ϵ_k-stationary points for a sequence $\{\epsilon_k\} \to 0$ that is set dynamically within the algorithm.

We present our algorithm in four subsections. The first subsection describes the main algorithm, at the heart of which is the search direction computation. We then discuss, in turn, the details of our line search, sample set generation scheme, and inverse Hessian approximation strategy. Since f may be nonsmooth, here we use the term "Hessian" loosely as a matrix that approximates changes in ∇f about a given point in $\mathcal{D}$, changes that may be arbitrarily large relative to the distance between the given point and nearby points in $\mathcal{D}$. Each of the latter algorithmic components are carefully constructed so that the main algorithm is well-posed and globally convergent to a stationary point of f under Assumption 2.1.

Our algorithm employs various user-specified parameters, which, for convenience, we enumerate in Table 2.1. Our global convergence theory allows for any choices of these parameters in the given ranges, except for a restriction on the curvature threshold ξ and its relationship to other parameter values. This restriction, which is required due to a technical lemma revealed in the development of our algorithm, is given at the beginning of §3.

2.1 Main algorithm

We now present our main algorithm, designed to converge to a stationary point of f in the sublevel set $\mathcal{L}_f(x_0)$. Ideally, such a point would be revealed as a cluster point of the iterate sequence $\{x_k\}$ obtained via a standard BFGS method, but since such a method generally has unknown convergence properties when employed to solve (2.1), our algorithm includes enhancements with which we guarantee global

Table 2.1 User-specified constants for the proposed algorithm and subroutines

Parameter(s)	Range	Description
ν	$(0,\infty)$	Stationarity measure tolerance
ψ	$(0,1)$	Sampling radius reduction factor
ξ	$(0,\infty)$	Model curvature threshold
$\underline{\eta} < \overline{\eta}$	$(0,1)$	Armijo–Wolfe line search constants
$\underline{\alpha} \leq \overline{\alpha}$	$(0,\infty)$	Step size thresholds
γ	$(0,1)$	Step size modification factor
$\underline{J} \leq \overline{J}$	$\mathbb{N}$	Iteration thresholds for line search
J	$\mathbb{N}$	Iteration threshold for iterate perturbation
p	$[n+1,\infty)\cap\mathbb{N}$	Sample set size threshold
$\underline{\mu} < 1 < \overline{\mu}$	$(0,\infty)$	(L-)BFGS updating thresholds
$\underline{w} \leq \overline{w}$	$(0,\infty)$	(L-)BFGS updating thresholds
m	$\mathbb{N}$	L-BFGS memory length

convergence with probability one. These enhancements are similar to those developed in the adaptive GS method proposed in [14], though are less expensive in the sense that, in many iterations, gradient sampling is not required.

At an iterate $x_k \in \mathcal{D}$ and with an inverse Hessian approximation of f at x_k, call it $W_k \succ 0$, a standard BFGS method computes a search direction as

$$d_k \leftarrow -W_k \nabla f(x_k). \tag{2.2}$$

However, in our approach, we incorporate gradient information at points in a set $X_k := \{x_{k,0}, \ldots, x_{k,p_k}\}$ that has $x_{k,i} = x_k$ for some $i \in \{0,\ldots,p_k\}$ and includes p_k other points from $B_k := \mathbb{B}_{\epsilon_k}(x_k) \cap \mathcal{D}$. We refer to X_k as the sample set and p_k as the sample set size, though note that $p_k = 0$ corresponds to $|X_k| = 1$. With this information, we desire the search direction d_k that is the minimizer of a local piece-wise quadratic model of f at x_k; i.e., we desire d_k to be the solution of

$$\min_{d\in\mathbb{R}^n} \; q_k(d), \quad \text{where} \quad q_k(d) := \max_{x\in X_k} \left\{ \nabla f(x)^T d \right\} + \tfrac{1}{2}\|d\|^2_{W_k^{-1}}. \tag{2.3}$$

Define the matrix of gradients

$$G_k := \begin{bmatrix} g_{k,0} & \cdots & g_{k,p_k} \end{bmatrix} \quad \text{with} \quad g_{k,i} := \nabla f(x_{k,i}) \quad \text{for all} \quad i \in \{0,\ldots,p_k\}. \tag{2.4}$$

The solution d_k of (2.3) can be obtained by solving the primal-dual pair

$$\left\{ \begin{array}{c} \min\limits_{(z,d)\in\mathbb{R}^{n+1}} \; z + \frac{1}{2}\|d\|^2_{W_k^{-1}} \\ \text{s.t. } G_k^T d \leq ze \end{array} \right\} \quad \text{and} \quad \left\{ \begin{array}{c} \max\limits_{y\in\mathbb{R}^{p_k+1}} \; -\frac{1}{2}\|G_k y\|^2_{W_k} \\ \text{s.t. } e^T y = 1,\; y \geq 0 \end{array} \right\}; \tag{2.5}$$

we denote the primal-dual solution of these problems by (z_k, d_k, y_k).

If the sample set has only one element, i.e., if $p_k = 0$ with $X_k = \{x_{k,0}\} = \{x_k\}$, then it is easily seen that our definitions are consistent in that d_k from (2.5) is that in (2.2). Thus, henceforth we may refer to d_k as the solution of (2.3), knowing that if $p_k = 0$, then it can be obtained directly from (2.2), and otherwise it can be obtained by solving the primal quadratic optimization subproblem (QP) in (2.5). In fact, if instead one solves the latter dual QP in (2.5) to obtain y_k, then the search direction can be obtained as $d_k \leftarrow -W_k G_k y_k$. This is the approach that we take in our implementation described in §4, and so it will be the approach used in the remainder of our discussion and analysis. Note that a benefit of this strategy

is that the Hessian approximation W_k^{-1} need not be computed; i.e., we only need the matrix W_k appearing in (2.2) and (2.5) and all subsequent computations will be written in such a way that only W_k is needed, not W_k^{-1}.

Overall, there are two interpretations of the search direction d_k. First, it can be viewed, as in subproblem (2.3), as the minimizer of a local piece-wise quadratic model of f at x_k with gradient information sampled at the points in X_k. Second, it can be viewed, in terms of the dual QP in (2.5), as W_k times the negation of the oblique W_k-projection of the origin onto the convex hull of the gradients of f at the points in the sample set X_k, i.e., as $d_k = -W_k P_{W_k}(\{\nabla f(x)\}_{x\in X_k})$, which is to say that it is W_k times the negation of the minimum W_k-norm element in $\operatorname{conv}\{\nabla f(x)\}_{x\in X_k}$. Clearly, with $W_k = I$, the search direction reduces to the negation of the minimum Euclidean norm element in $\operatorname{conv}\{\nabla f(x)\}_{x\in X_k}$, which is precisely the "nonnormalized search direction" interpretation described in [32, §4.1]. The former interpretation is perhaps more intuitively appealing as that of a search direction for an optimization algorithm, though we will make more use of the second interpretation in our global convergence analysis.

Once the pair (d_k, y_k) has been computed via (2.5), we either compute a null step size—to produce a null step, which may be necessary in some cases—or a positive step size $\alpha_k > 0$ such that the trial point $x_k + \alpha_k d_k$ yields a sufficiently lower objective value than that offered by x_k. If $x_k + \alpha_k d_k \in \mathcal{D}$, then the next iterate x_{k+1} is set to be this trial point; otherwise a point $x_{k+1} \in \mathcal{D}$ in the vicinity of $x_k + \alpha_k d_k$ is computed such that $f(x_{k+1})$ is sufficiently less than $f(x_k)$. In fact, the step size α_k and new iterate x_{k+1} may be chosen also to satisfy a curvature condition to ensure that an unadulterated BFGS update will produce a positive definite inverse Hessian approximation in the following iteration. All of the details of these procedures are given in §2.2. Overall, with $x_0 \in \mathcal{D}$, we ensure $\{x_k\} \subset \mathcal{D}$.

Once the pair (d_k, y_k), step size $\alpha_k \geq 0$, and next iterate $x_{k+1} \in \mathcal{D}$ have been computed, the remainder of the iteration involves setting the next sampling radius $\epsilon_{k+1} \in (0, \epsilon_k]$, the sample set X_{k+1} and related quantities, and the inverse Hessian approximation W_{k+1}. In particular, the value to which the next sampling radius ϵ_{k+1} is set depends on whether or not the following inequalities hold:

$$\|G_k y_k\|_{W_k} \leq \nu \epsilon_k; \tag{2.6a}$$

$$\|G_k y_k\|_{W_k} \geq \xi \|d_k\|_2; \tag{2.6b}$$

$$\alpha_k > 0. \tag{2.6c}$$

The details pertaining to the updates of sample set and inverse Hessian approximation are the subjects of §2.3 and §2.4, respectively.

We now present our main algorithm, stated as Algorithm 1.

We close this subsection by stating a result that if Algorithm 1 reaches Step 3 during iteration k, then it computes d_k as null or as a direction of strict descent for f from $x_k \in \mathcal{D}$. The proof of this result follows similarly to that of [14, Lemma 4.3]. We state the result here, which also reveals an important relationship between the search direction d_k and the dual QP solution y_k, as it will be used to motivate algorithmic choices made in the following subsections.

Lemma 2.2 *If Algorithm 1 reaches Step 3 during iteration k, then it computes a search direction d_k that is zero or a direction of strict descent for f from $x_k \in \mathcal{D}$. In addition, the primal-dual solution (z_k, d_k, y_k) of (2.5) satisfies $\|G_k y_k\|_{W_k} = \|d_k\|_{W_k^{-1}}$.*

Algorithm 1 BFGS Gradient Sampling Algorithm

1: Choose an initial iterate $x_0 \in \mathcal{D}$, inverse Hessian approximation $W_0 \succ 0$, and sampling radius $\epsilon_0 > 0$. Set the initial sample set $X_0 \leftarrow \{x_0\}$, sample set size $p_0 \leftarrow 0$, matrix of sample gradients G_0 as defined in (2.4), and iteration counter $k \leftarrow 0$.
2: If $\nabla f(x_k) = 0$, then terminate and return the stationary point x_k.
3: Compute a search direction $d_k \leftarrow -W_k G_k y_k$ where y_k solves the dual QP in (2.5).
4: Compute a step size $\alpha_k \geq 0$ via Algorithm 2 in §2.2.
5: Compute a new iterate $x_{k+1} \in \mathcal{D}$ via Algorithm 3 in §2.2.
6: If (2.6) holds, then set the new sampling radius $\epsilon_{k+1} \leftarrow \psi\epsilon_k$; otherwise, set $\epsilon_{k+1} \leftarrow \epsilon_k$.
7: Compute a new sample set X_{k+1} with $p_{k+1} \leftarrow |X_{k+1}| - 1$ via Algorithm 4 in §2.3.
8: Compute the matrix of gradients G_{k+1} as defined in (2.4).
9: Compute a new inverse Hessian approximation $W_{k+1} \succ 0$ via Algorithm 5 in §2.4.
10: Set $k \leftarrow k + 1$ and go to Step 2.

Lemma 2.2 allows us to write the conditions in (2.6) as

$$\xi\|d_k\|_2 \leq \|d_k\|_{W_k^{-1}} \leq \nu\epsilon_k \quad \text{and} \quad \alpha_k > 0, \tag{2.6$'$}$$

from which it is clear that, in Algorithm 1, the sampling radius is decreased if and only if the step size is nonzero and the search direction has a W_k^{-1}-norm that is both relatively large compared to its Euclidean norm and relatively small compared to the current sampling radius.

2.2 Line Search

At an iterate $x_k \in \mathcal{D}$, Algorithm 1 either terminates in Step 2 or, by Lemma 2.2, it continues to Step 3 to produce a null or strict descent direction d_k for f from x_k. If $d_k = 0$, then we simply set α_k to its positive initial value, set $x_{k+1} \leftarrow x_k$, and continue to the next step of the algorithm. If $d_k \neq 0$, then our line search aims to compute a step size $\alpha_k > 0$ such that $x_k + \alpha_k d_k$ yields an objective value that is sufficiently less than that yielded by x_k. In fact, it attempts to compute such a step size so that a curvature condition is also satisfied, as this would guarantee that an unadulterated BFGS update will yield $W_{k+1} \succ 0$; see §2.4. However, to ensure that the line search is well-posed under loose assumptions, this latter requirement is abandoned if such a step size is not computed within a predetermined number of line search iterations. We also terminate the search completely (and simply set $\alpha_k \leftarrow 0$ and $x_{k+1} \leftarrow x_k$) if the sample set X_k is not sufficiently large and, after a predetermined number of line search iterations, a sufficient decrease in f has not been obtained. This choice is motivated by the fact that if the sample set is not sufficiently large and a relatively large step size is not acceptable according to the line search conditions, then the algorithm may benefit by collecting more local gradient information before accepting a positive step size—which it can be seen to do by observing the sample set update in §2.3.

Given an iterate x_k and pair (d_k, y_k) from (2.5) with $d_k \neq 0$, we aim to compute a step size $\alpha_k > 0$ satisfying the following Armijo and curvature conditions, which together compose the well-known weak Wolfe line search conditions [39]:

$$f(x_k) - f(x_k + \alpha_k d_k) > \underline{\eta}\alpha_k \|G_k y_k\|_{W_k}^2; \tag{2.7a}$$

$$v^T d_k \geq \overline{\eta} \nabla f(x_k)^T d_k, \text{ where } v \in \partial f(x_k + \alpha_k d_k). \tag{2.7b}$$

(Technically, we are abusing this terminology as the traditional Armijo condition employs the negative directional derivative $-\nabla f(x_k)^T d_k$ in place of $\|G_k y_k\|^2_{W_k}$ in (2.7a). However, our abuse of this terminology is reasonable since, by Lemma 2.2, the condition (2.7a) also ensures sufficient decrease in f from x_k after the step $\alpha_k d_k$.) If the resulting trial point satisfies $x_k + \alpha_k d_k \in \mathcal{D}$, then x_{k+1} is set to be this trial point; otherwise, we aim to compute $x_{k+1} \in \mathcal{D}$ satisfying

$$f(x_k) - f(x_{k+1}) \geq \underline{\eta}\alpha_k \|G_k y_k\|^2_{W_k}, \tag{2.8a}$$

$$\nabla f(x_{k+1})^T d_k \geq \overline{\eta} \nabla f(x_k)^T d_k, \tag{2.8b}$$

$$\text{and} \quad \|x_k + \alpha_k d_k - x_{k+1}\|_2 \leq \min\{\alpha_k, \epsilon_k\}\|d_k\|_2. \tag{2.8c}$$

Note that these conditions are also satisfied when $x_{k+1} \leftarrow x_k + \alpha_k d_k \in \mathcal{D}$, so we may refer to (2.8) as being satisfied whenever (2.7) holds and $x_{k+1} \leftarrow x_k + \alpha_k d_k$.

There are a variety of situations in which it may not be possible to compute a step size $\alpha_k > 0$ satisfying (2.7), or at least not within a predetermined number of iterations. For example, such a situation occurs when f is unbounded below along the ray $\{x_k + \alpha d_k : \alpha \geq 0\}$. However, even if f is bounded below over this ray, finite termination of a straightforward line search scheme may not be guaranteed without strengthening Assumption 2.1, or at least not without additional assumptions about f at x_k along d_k; see Lemma 2.4 below. Hence, we propose Algorithm 2 that guarantees finite termination by abandoning the curvature condition (2.7b) after a finite number of trial step sizes have been rejected. We also completely abandon the search for a positive step size—and set $\alpha_k \leftarrow 0$, $x_{k+1} \leftarrow x_k$, and eventually $\epsilon_{k+1} \leftarrow \epsilon_k$ due to (2.6c)—if X_k is not sufficiently large and the search has not been successful after a predetermined number of iterations. This truncation of the line search is required to prove our global convergence guarantees as it will result, by the method in §2.3, in additional gradient sampling about x_{k+1}.

Algorithm 2 Armijo-Wolfe Line Search

1: Take as input the quantities $(x_k, G_k, W_k, d_k, y_k)$ from Algorithm 1. Set the initial step size boundaries $l_0 \leftarrow 0$ and $u_0 \leftarrow \overline{\alpha}$, step size $\alpha_k \leftarrow \gamma\overline{\alpha}$, and iteration counter $j \leftarrow 0$.
2: If the step is null, i.e., $d_k = 0$, then terminate and return α_k.
3: If the sample set is not sufficiently large in that $p_k < p$ and the upper iteration threshold has been surpassed in that $j > \overline{J}$, then set $\alpha_k \leftarrow 0$, terminate, and return α_k.
4: If the lower iteration threshold has been surpassed in that $j > \underline{J}$, then reset $l_j \leftarrow 0$.
5: If the Wolfe conditions (2.7) hold, or if the Armijo condition (2.7a) holds and the lower iteration threshold has been surpassed in that $j > \underline{J}$, then terminate and return α_k.
6: If the Armijo condition (2.7a) does not hold, then set $l_{j+1} \leftarrow l_j$ and $u_{j+1} \leftarrow \alpha_k$; otherwise, the curvature condition (2.7b) does not hold, so set $l_{j+1} \leftarrow \alpha_k$ and $u_{j+1} \leftarrow u_j$.
7: Set $\alpha_k \leftarrow (1-\gamma)l_{j+1} + \gamma u_{j+1}$.
8: Set $j \leftarrow j + 1$ and go to Step 3.

After employing Algorithm 2 to compute a step size $\alpha_k \geq 0$, we employ Algorithm 3 to compute a new iterate $x_{k+1} \in \mathcal{D}$. If $\alpha_k = 0$, then $x_{k+1} \leftarrow x_k$, but if $\alpha_k > 0$, then x_{k+1} will satisfy the perturbed line search conditions (2.8), or at least (2.8a) and (2.8c). If $\alpha_k > 0$, but (2.7b) does not hold, then we effectively ignore (2.8b) by immediately setting $j \leftarrow J + 1$ in Step 3 of Algorithm 3.

We present the following lemma to show that our line search and iterate perturbation algorithms are well-posed. The lemma also delineates various situations that may result after employing these two subroutines.

Algorithm 3 Iterate Perturbation

1: Take as input the quantities $(x_k, \epsilon_k, G_k, W_k, d_k, y_k, \alpha_k)$ from Algorithm 1. Set the initial new iterate $x_{k+1} \leftarrow x_k + \alpha_k d_k$ and iteration counter $j \leftarrow 0$.
2: If the step or step size is null, i.e., $d_k = 0$ or $\alpha_k = 0$, then terminate and return x_{k+1}.
3: If the curvature condition (2.7b) does not hold, then set $j \leftarrow J + 1$.
4: If $x_{k+1} \notin \mathcal{D}$, then continue to Step 5. Otherwise, if the (perturbed) Wolfe conditions (2.8) hold, or if the (perturbed) Armijo conditions (2.8a) and (2.8c) hold and the iteration threshold has been surpassed in that $j > J$, then terminate and return x_{k+1}.
5: Sample x_{k+1} from a uniform distribution on $\mathbb{B}_{\min\{\alpha_k, \epsilon_k\}/j}(x_k + \alpha_k d_k)$.
6: Set $j \leftarrow j + 1$ and go to Step 4.

Lemma 2.3 *If Algorithm 1 reaches Step 4 during iteration k, then it either computes a null or positive step size α_k, where α_k is guaranteed to be positive if $p_k \geq p$. Moreover, if $\alpha_k > 0$, then the Wolfe conditions* (2.7)*, or at least the Armijo condition* (2.7a)*, is satisfied. Algorithm 1 then proceeds to Step 5, where with probability one it computes a new iterate $x_{k+1} \in \mathcal{D}$ with which the (perturbed) Wolfe conditions* (2.8)*, or at least the (perturbed) Armijo conditions* (2.8a) *and* (2.8c)*, are satisfied.*

Proof If $d_k = 0$, then (2.7) holds for any value of $\alpha_k \geq 0$, so Algorithm 2 terminates in iteration $j = 0$ and returns $\alpha_k \leftarrow \gamma\overline{\alpha} > 0$. In this case, or if Algorithm 2 sets $\alpha_k \leftarrow 0$, then since $x_{k+1} \leftarrow x_k + \alpha_k d_k = x_k \in \mathcal{D}$ satisfies (2.8), it follows that Algorithm 3 terminates in iteration $j = 0$ and returns $x_{k+1} \leftarrow x_k + \alpha_k d_k$. Now suppose $d_k \neq 0$, from which it follows from Lemma 2.2 that d_k is a direction of strict descent for f from x_k. Without loss of generality, we may assume that Algorithm 2 performs at least $\underline{J}$ iterations, at which point it (re)sets $l_j \leftarrow 0$, and that it never sets $\alpha_k \leftarrow 0$. It then follows from the fact that $x_k \in \mathcal{D}$ and Lemma 2.2 that, after a finite number of additional iterations, $\alpha_k > 0$ will be produced at least satisfying the Armijo condition (2.7a). Turning to Algorithm 3, we may assume without loss of generality that at least J iterations will be performed, after which it follows from the strict inequality in (2.7a), the continuity of f, and Assumption 2.1 that, after a finite number of additional iterations and with probability one, a new iterate x_{k+1} will be produced satisfying (2.8a) and (2.8c). □

With additional assumptions about f and α_k initialized to some positive scalar, one could employ Algorithm 2 with the step size threshold set to $\overline{\alpha} \leftarrow \infty$ and iteration thresholds set to $\underline{J} \leftarrow \infty$ and $\overline{J} \leftarrow \infty$ and still have a well-posed algorithm. To make this claim concrete, we present the following result, the proof of which follows from the results in [35, §4]; see also [34, 48, 37]. Although we do not wish to make the additional assumptions required in this lemma, we present this result to motivate the appeal of our line search strategy.

Lemma 2.4 *Suppose $f_k(\alpha) := f(x_k + \alpha d_k) - f(x_k)$ is bounded below and weakly lower semismooth [37] over $\{\alpha \in \mathbb{R} : \alpha > 0\}$. Then, if Algorithm 1 reaches Step 4 during iteration k and the function f_k is differentiable at all trial step sizes, Algorithm 2 with α_k initialized in $(0, \infty)$, $\overline{\alpha} \leftarrow \infty$, $\underline{J} \leftarrow \infty$, $\overline{J} \leftarrow \infty$, and Step 7 replaced by*

"7: If $u_{j+1} < \infty$, then set $\alpha_k \leftarrow (1 - \gamma) l_{j+1} + \gamma u_{j+1}$; else, set $\alpha_k \leftarrow \alpha_k / \gamma$."

terminates finitely with $\alpha_k > 0$ satisfying (2.7).

2.3 Sample point generation

After the pair (d_k, y_k), step size α_k, and new iterate x_{k+1} have been computed, we are ready in Algorithm 1 to establish the new sample set X_{k+1}. As previously mentioned, we claim that the ideal behavior of the algorithm is that of an unadulterated BFGS method, at least when such a method continues to make sufficient progress in reducing the objective function f. Hence, if the curvature of W_k along $G_k y_k$—i.e., by Lemma 2.2, the curvature of W_k^{-1} along d_k—is bounded below in that (2.6b) holds, and if the computed step size is sufficiently large in that

$$\alpha_k \geq \underline{\alpha}, \tag{2.9}$$

then we set the default value of $X_{k+1} \leftarrow \{x_{k+1}\}$ so that an unadulterated BFGS step will be computed in the following iteration. However, if either of these conditions does not hold, then we augment the sample set with points obtained from the previous sample set and some randomly generated in an ϵ_{k+1}-neighborhood about x_{k+1}. The details of our sample set update are stated in Algorithm 4, and the salient consequences of this strategy are provided in the following lemma.

Algorithm 4 Sample Set Update

1: Take as input the quantities $(x_{k+1}, \epsilon_{k+1}, G_k, W_k, d_k, y_k, \alpha_k)$ from Algorithm 1.
2: If the curvature of the inverse Hessian approximation is bounded below in that (2.6b) holds and the step size is sufficiently large in that (2.9) holds, then set $X_{k+1} \leftarrow \{x_{k+1}\}$ and $p_{k+1} \leftarrow 0$, terminate, and return (X_{k+1}, p_{k+1}).
3: Set $X_{k+1} \leftarrow (X_k \cap B_{k+1}) \cup \{x_{k+1}\}$ and choose $\overline{p}_{k+1} \in \mathbb{N}_+$.
4: Set $\overline{X}_{k+1}$ as a collection of $\overline{p}_{k+1}$ points generated independently from a uniform distribution over $\mathbb{B}_{\epsilon_{k+1}}(x_{k+1})$.
5: If $\overline{X}_{k+1} \not\subset \mathcal{D}$, then go to Step 4.
6: Set $X_{k+1} \leftarrow X_{k+1} \cup \overline{X}_{k+1}$ and $p_{k+1} \leftarrow |X_{k+1}| - 1$.
7: If $p_{k+1} > p$, then remove the $p_{k+1} - p$ eldest members of $X_{k+1} \backslash \{x_{k+1}\}$ and set $p_{k+1} \leftarrow p$.
8: Terminate and return (X_{k+1}, p_{k+1}).

Lemma 2.5 *If Algorithm 1 reaches Step 7 during iteration k, then it either sets $X_{k+1} \leftarrow \{x_{k+1}\}$ and $p_{k+1} \leftarrow 0$, or, with probability one, it produces*

$$X_{k+1} \leftarrow ((X_k \cap B_{k+1}) \cup \{x_{k+1}\} \cup \overline{X}_{k+1}) \subset B_{k+1}$$

and $p_{k+1} \geq \min\{p_k + 1, p\}$.

Proof If (2.6b) and (2.9) hold, then the algorithm sets $X_{k+1} \leftarrow \{x_{k+1}\}$ and $p_{k+1} \leftarrow 0$, which is the first desirable result. Otherwise, the result follows by the construction of Algorithm 4, Assumption 2.1, and the fact that the points in $\overline{X}_{k+1}$ are generated independently and uniformly in $\mathbb{B}_{\epsilon_{k+1}}(x_{k+1})$. □

2.4 Hessian approximation strategy

Upon computing the pair (d_k, y_k), step size α_k, new iterate x_{k+1}, and new sample set X_{k+1}, the final main step of Algorithm 1 is to compute a new inverse Hessian

approximation W_{k+1}. If the curvature along $G_k y_k$ determined by W_k is bounded below in that (2.6b) holds and if the step size is sufficiently large in that (2.9) holds, then we obtain $W_{k+1} \succ 0$ from $W_k \succ 0$ via a standard damped BFGS update. However, if one of these conditions does not hold, then we have reason to believe that a standard BFGS update may lead to an approximation whose ill-conditioning may be detrimental to the performance of the algorithm, or at least to our mechanisms for guaranteeing productive steps and/or verifying stationarity. Hence, in such cases, we set $W_{k+1} \succ 0$ by an L-BFGS strategy in which we monitor the updates so that the resulting matrix has a provably bounded condition number.

The algorithm presented in this section makes use of the quantities

$$s_k := x_{k+1} - x_k \quad \text{and} \quad t_k := \nabla f(x_{k+1}) - \nabla f(x_k) \quad \text{for all} \quad k \geq 0. \tag{2.10}$$

It also potentially uses the set of pairs $\{(s_{k-m+1}, t_{k-m+1}), \ldots, (s_{k-1}, t_{k-1})\}$, where each element is defined similarly as in (2.10) for the previous $m-1$ iterations. We did not mention these pairs in our description of Algorithm 1, though it is obvious that these vectors may be stored in Algorithm 1 for use in the algorithm in this subsection without affecting other aspects of Algorithm 1 in any way.

If $s_k = 0$ or $t_k = 0$, then we claim that we have obtained no useful curvature information from the step from x_k to x_{k+1}, so we set $W_{k+1} \leftarrow W_k$. Otherwise, if (2.6b) and (2.9) hold, then we damp the BFGS update by setting

$$r_k \leftarrow \delta_k s_k + (1 - \delta_k) W_k t_k, \tag{2.11}$$

where the scalar δ_k is defined by

$$\delta_k \leftarrow \begin{cases} 1 & \text{if } s_k^T t_k \geq \underline{\mu} t_k^T W_k t_k \\ (1 - \underline{\mu}) t_k^T W_k t_k / (t_k^T W_k t_k - s_k^T t_k) & \text{if } s_k^T t_k < \underline{\mu} t_k^T W_k t_k, \end{cases} \tag{2.12}$$

then employ the standard BFGS formula with (s_k, t_k) replaced by (r_k, t_k) [39]:

$$W_{k+1} \leftarrow \left(I - \frac{r_k t_k^T}{r_k^T t_k}\right) W_k \left(I - \frac{t_k r_k^T}{r_k^T t_k}\right) + \frac{r_k r_k^T}{r_k^T t_k}. \tag{2.13}$$

On the other hand, if $s_k \neq 0$ and $t_k \neq 0$, but at least one of (2.6b) or (2.9) does not hold, then we employ a damped L-BFGS strategy, proceeding in the following manner. First, choosing a scalar $w_k > 0$, we initialize $W_{k+1}^{(k-m)} \leftarrow w_k I$. Then, for increasing j in the ordered set $\{k-m+1, \ldots, k\}$, we set

$$r_j \leftarrow \delta_j s_j + (1 - \delta_j) W_{k+1}^{(j-1)} t_j, \tag{2.14}$$

where

$$\delta_j \leftarrow \begin{cases} 1 & \text{if } s_j^T t_j \geq \underline{\mu} t_j^T W_{k+1}^{(j-1)} t_j \\ (1 - \underline{\mu}) t_j^T W_{k+1}^{(j-1)} t_j / (t_j^T W_{k+1}^{(j-1)} t_j - s_j^T t_j) & \text{if } s_j^T t_j < \underline{\mu} t_j^T W_{k+1}^{(j-1)} t_j, \end{cases} \tag{2.15}$$

and

$$W_{k+1}^{(j)} \leftarrow \left(I - \frac{r_j t_j^T}{r_j^T t_j}\right) W_{k+1}^{(j-1)} \left(I - \frac{t_j r_j^T}{r_j^T t_j}\right) + \frac{r_j r_j^T}{r_j^T t_j}. \tag{2.16}$$

Finally, we set $W_{k+1} \leftarrow W_{k+1}^{(k)}$. In this procedure, in order to guarantee that the resulting inverse Hessian approximation has a bounded condition number, for each j we skip the update in (2.16)—and simply set $W_{k+1}^{(j)} \leftarrow W_{k+1}^{(j-1)}$—unless

$$s_j \neq 0, \quad t_j \neq 0, \quad \text{and} \quad \max\{\|r_j\|_2^2, \|t_j\|_2^2\} \leq \overline{\mu} r_j^T t_j. \tag{2.17}$$

We formalize our inverse Hessian approximation strategy as Algorithm 5. We assume that the vectors in $\{(s_{-m+1}, t_{-m+1}), \ldots, (s_{-1}, t_{-1})\}$ are initialized to zero so that if the L-BFGS strategy is employed in iteration $k < m$, then, as a consequence of the condition (2.17), at most k updates will be performed.

Algorithm 5 Inverse Hessian Approximation Update

Take as input the quantities $(x_k, x_{k+1}, \nabla f(x_k), \nabla f(x_{k+1}), W_k)$ from Algorithm 1 and the previously computed sequence $\{(s_{k-m+1}, t_{k-m+1}), \ldots, (s_{k-1}, t_{k-1})\}$.
Set s_k and t_k by (2.10).
If $s_k = 0$ or $t_k = 0$, then set $W_{k+1} \leftarrow W_k$, terminate, and return W_{k+1}.
If the curvature of the inverse Hessian approximation is bounded below in that (2.6b) holds and the step size is sufficiently large in that (2.9) holds, then set r_k, δ_k, and W_{k+1} by (2.11)–(2.13), terminate, and return W_{k+1}.
Choose $w_k \in [\underline{w}, \overline{w}]$ and initialize $W_{k+1}^{(k-m)} \leftarrow w_k I$.
for increasing $j \in \{k-m+1, \ldots, k\}$ **do**
    Set r_j and δ_j by (2.14)–(2.15).
    if (2.17) holds **then**
        Set $W_{k+1}^{(j)}$ by (2.16).
    else
        Set $W_{k+1}^{(j)} \leftarrow W_{k+1}^{(j-1)}$.
Set $W_{k+1} \leftarrow W_{k+1}^{(k)}$, terminate, and return W_{k+1}.

In the remainder of this section, we prove properties of the inverse Hessian approximation W_{k+1} returned by Algorithm 5 during iteration k of Algorithm 1. First, we state the result that the damped BFGS update (2.13) yields $W_{k+1} \succ 0$. This fact is well known [39], so we state it without proof.

Lemma 2.6 *With $W_k \succ 0$, $s_k \neq 0$, and $t_k \neq 0$, the update (2.13) yields $W_{k+1} \succ 0$.*

Next, we state a result about the update (2.16). The proof of this result is similar to that of [14, Lemma 3.2], so we exclude it here for the sake of brevity.

Lemma 2.7 *Suppose that for $\overline{\theta} \geq \underline{\theta} > 0$ we have*

$$\underline{\theta}\|t\|_2^2 \leq t^T W_{k+1}^{(j-1)} t \leq \overline{\theta}\|t\|_2^2 \quad \text{for all} \quad t \in \mathbb{R}^n. \tag{2.18}$$

Then, with (r_j, s_j, t_j) satisfying (2.17), the update (2.16) yields $W_{k+1}^{(j)}$ such that

$$(\underline{\theta}^{-1} + \overline{\mu})^{-1}\|t\|_2^2 \leq t^T W_{k+1}^{(j)} t \leq (2\overline{\theta}(1 + \overline{\mu}^2) + \overline{\mu})\|t\|^2 \quad \text{for all} \quad t \in \mathbb{R}^n. \tag{2.19}$$

We conclude this section with the following lemma revealing that for any k, the matrix W_{k+1} returned by Algorithm 5 is positive definite, and is also bounded in certain important situations; see the similar result [14, Theorem 3.3].

Lemma 2.8 *Algorithm 5 with input $W_k \succ 0$ yields W_{k+1} satisfying the following.*

(a) If $s_k = 0$ or $t_k = 0$, then $W_{k+1} \leftarrow W_k \succ 0$.
(b) If $s_k \neq 0$, $t_k \neq 0$, and (2.6b) and (2.9) hold, then $W_{k+1} \succ 0$.
(c) If $s_k \neq 0$ and $t_k \neq 0$, but at least one of (2.6b) or (2.9) does not hold, then $W_{k+1} \succ 0$ and for all $t \in \mathbb{R}^n$ we have

$$t^T W_{k+1} t \geq (\underline{w}^{-1} + m\overline{\mu})^{-1} \|t\|_2^2 \tag{2.20a}$$

$$t^T W_{k+1} t \leq \left(2^m \left(1 + \overline{\mu}^2\right)^m \overline{w} + \overline{\mu} \left(\frac{2^m \left(1 + \overline{\mu}^2\right)^m - 1}{2\left(1 + \overline{\mu}^2\right) - 1} \right) \right) \|t\|_2^2. \tag{2.20b}$$

Proof If $s_k = 0$ or $t_k = 0$, then, by Step 3, Algorithm 5 sets $W_{k+1} \leftarrow W_k \succ 0$, as desired. Otherwise, if (2.6b) and (2.9) hold, then, by Step 4, Algorithm 5 sets W_{k+1} by (2.13), which by Lemma 2.6 implies that $W_{k+1} \succ 0$, as desired.

All that remains is to consider the case when $s_k \neq 0$ and $t_k \neq 0$, but at least one of (2.6b) or (2.9) does not hold. In this case, by Steps 5–12, Algorithm 5 sets W_{k+1} by choosing $w_k \in [\underline{w}, \overline{w}]$, initializing $W_{k+1}^{(k-m)} = w_k I \succ 0$, applying (at most) m updates of the form (2.16) with quantities satisfying (2.17), and finally setting $W_{k+1} \leftarrow W_{k+1}^{(k)}$. Since, by Lemma 2.7, each application of (2.16) takes the bounds in (2.18) and produces the wider bounds in (2.19), we may assume without loss of generality that all m updates are performed, i.e., that (2.17) holds for all $j \in \{k - m + 1, \ldots, k\}$. Thus, starting with $\underline{\theta} = \overline{\theta} = w_k$, the result of Lemma 2.7 can be applied repeatedly for increasing $j \in \{k - m + 1, \ldots, k\}$. In particular, as seen in the proof of Lemma 2.7, the upper bound corresponding to the inverse of the approximation increases by the constant factor $\overline{\mu} > 0$ with each update, so after m updates we obtain (2.20a). As for the upper bound (2.20b), by applying Lemma 2.7 for increasing $j \in \{k - m + 1, \ldots, k\}$ we obtain for all $t \in \mathbb{R}^n$ that

$$\begin{aligned} t^T W_{k+1} t \leq\ & (2^m(1+\overline{\mu}^2)^m w_k + 2^{m-1}(1+\overline{\mu}^2)^{m-1}\overline{\mu} + \cdots + 2(1+\overline{\mu}^2)\overline{\mu} + \overline{\mu})\|t\|_2^2 \\ =\ & \left(2^m(1+\overline{\mu}^2)^m w_k + \overline{\mu}\left(\frac{2^m(1+\overline{\mu}^2)^m - 1}{2(1+\overline{\mu}^2) - 1} \right) \right) \|t\|_2^2, \end{aligned}$$

which, since $w_k \in [\underline{w}, \overline{w}]$, implies that (2.20b) holds. □

3 Global Convergence Analysis

In this section, we prove that Algorithm 1 is globally convergent from remote starting points. Specifically, with the restriction that

$$0 < \xi \leq \left(2^m \left(1 + \overline{\mu}^2\right)^m \overline{w} + \overline{\mu} \left(\frac{2^m \left(1 + \overline{\mu}^2\right)^m - 1}{2\left(1 + \overline{\mu}^2\right) - 1} \right) \right)^{-1} \tag{3.1}$$

the result that we prove is the following.

Theorem 3.1 *Algorithm 1 either terminates finitely with a stationary point for f or, with probability one, it produces an infinite sequence of iterates $\{x_k\}$. In the latter case, with probability one, either $\{f(x_k)\} \to -\infty$ or $\{\epsilon_k\} \to 0$ and every cluster point of the iterate sequence $\{x_k\}$ is stationary for f.*

We begin our analysis for proving Theorem 3.1 by summarizing the results of the previous section to prove that Algorithm 1 is well-posed.

Lemma 3.2 *Algorithm 1 is well-posed in the sense that it either terminates in Step 2 with a stationary point for f or, with probability one, it produces an infinite sequence of iterates $\{x_k\}$. In either case, for each k, the following hold true:*

(a) The primal-dual solution (z_k, d_k, y_k) of (2.5) satisfies $\|G_k y_k\|_{W_k} = \|d_k\|_{W_k^{-1}}$ where d_k is either zero or is a direction of strict descent for f from $x_k \in \mathcal{D}$.
(b) The step size α_k is nonnegative, and is positive if $p_k \geq p$. If $\alpha_k > 0$, then either the Wolfe conditions (2.7) hold or at least the Armijo condition (2.7a) holds.
(c) With probability one, $x_{k+1} \in \mathcal{D}$ is computed satisfying the (perturbed) Wolfe conditions (2.8) or at least the (perturbed) Armijo conditions (2.8a) and (2.8c).
(d) If Step 6 is reached and (2.6) holds, then $\epsilon_{k+1} \leftarrow \psi\epsilon_k$; otherwise, $\epsilon_{k+1} \leftarrow \epsilon_k$.
(e) If Step 7 is reached and (2.6b) and (2.9) hold, then $X_{k+1} \leftarrow \{x_{k+1}\}$ along with $p_{k+1} \leftarrow 0$; otherwise, with probability one,

$$X_{k+1} \leftarrow ((X_k \cap B_{k+1}) \cup \{x_{k+1}\} \cup \overline{X}_{k+1}) \subset B_{k+1}$$

is generated and $p_{k+1} \geq \min\{p_k + 1, p\}$.
(f) If Step 9 is reached, then $W_{k+1} \succ 0$, where if $s_k \neq 0$, $t_k \neq 0$, and at least one of (2.6b) or (2.9) does not hold, then W_{k+1} satisfies the bounds in (2.20).

Proof The result follows by the construction of Algorithms 1, 2, 3, 4, and 5 along with the results of Lemmas 2.2, 2.3, 2.5, 2.6, and 2.8. □

For simplicity in our analysis until our proof of Theorem 3.1 at the end of this section, we assume without loss of generality that Algorithm 1 produces an infinite iterate sequence $\{x_k\}$. Implicit in this assumption is that the procedures to compute x_{k+1} and X_{k+1} terminate finitely for all k, i.e., that these procedures may be considered deterministic. This is reasonable since, by Lemma 3.2, these procedures terminate finitely with probability one, and since there is nothing else that we aim to prove when they fail to terminate finitely.

In our next result, we prove that there exists an infinite subsequence of iterates in which the algorithm produces a positive step size.

Lemma 3.3 *There exists an infinite subsequence of iterations in which $\alpha_k > 0$.*

Proof To derive a contradiction, suppose that there exists an iteration number k' such that for all $k \geq k'$ we have $\alpha_k = 0$. By Lemma 3.2(b), this must mean that for all $k \geq k'$ we have $p_k \leq p - 1$. However, with $\alpha_k = 0$, we have that (2.9) does not hold, which by Lemma 3.2(e) implies that the algorithm will set $p_{k+1} \geq \min\{p_k + 1, p\}$. This means that for some $k'' \geq k'$ we have $p_{k''} \geq p$, which contradicts the conclusion that $p_k \leq p - 1$ for all $k \geq k'$. □

We now state, in the following lemma, a critical inequality for a subset of iterations. The proof of this lemma is nearly identical to that of [14, Lemma 4.5].

Lemma 3.4 *If (2.6b) holds during iteration k, then*

$$f(x_{k+1}) \leq f(x_k) - \tfrac{1}{2}\underline{\eta}\xi\|x_{k+1} - x_k\|_2\|d_k\|_2.$$

We now prove a useful lemma on approximate least W-norm elements in certain types of sets of interest. For the lemma, recall that for $W \succ 0$ and nonempty bounded $S \subseteq \mathbb{R}^n$, we define $P_W(S)$ as the (oblique) W-projection of the origin onto $\operatorname{cl}\operatorname{conv} S$. The lemma can be seen as a variation of [32, Lemma 3.1].

Lemma 3.5 *Consider $W \succ 0$, a nonempty bounded set $S \subseteq \mathbb{R}^n$, and a constant $\beta \in (0,1)$. If $0 \notin \operatorname{cl}\operatorname{conv} S$, then there exists a constant $\kappa > 0$ such that for any $\{u, v\} \subseteq \operatorname{cl}\operatorname{conv} S$ the inequality $\|u\|_W^2 \leq \|P_W(S)\|_W^2 + \kappa$ implies $v^T W u > \beta \|u\|_W^2$.*

Proof By definition, we have

$$P_W(S) := \underset{x \in \operatorname{cl}\operatorname{conv} S}{\arg\min} \|x\|_W^2,$$

which implies (e.g., see [3, Proposition 1.1.8]) that for all $v \in \operatorname{cl}\operatorname{conv} S$ we have

$$v^T W P_W(S) \geq \|P_W(S)\|_W^2. \tag{3.2}$$

We now prove the result by contradiction. If the result were false, then there exist sequences $\{u_i\} \subseteq \operatorname{cl}\operatorname{conv} S$ and $\{v_i\} \subseteq \operatorname{cl}\operatorname{conv} S$ satisfying $\|u_i\|_W^2 \leq \|P_W(S)\|_W^2 + 1/i$ and $v_i^T W u_i \leq \beta \|u_i\|_W^2$ for all $i \geq 0$. Then, $\{u_i\} \to \overline{u} = P_W(S) \neq 0$, and since S is bounded, it follows that $\operatorname{cl}\operatorname{conv} S$ is compact, meaning that we may assume that $\{v_i\} \to \overline{v} \in \operatorname{cl}\operatorname{conv} S$ such that $\overline{v}^T W \overline{u} \leq \beta \|\overline{u}\|_W^2$. On the other hand, we have from (3.2) that $v^T W \overline{u} \geq \|\overline{u}\|_W^2$ for all $v \in \operatorname{cl}\operatorname{conv} S$, a contradiction. □

Next, we state a technical lemma pertaining to the discrepancy between two related measures of proximity to ϵ-stationarity. Given $x' \in \mathbb{R}^n$, we define

$$\mathcal{G}_k(x') := \operatorname{cl}\operatorname{conv} \nabla f(\mathbb{B}_{\epsilon_k}(x') \cap \mathcal{D}),$$

and, also given a tolerance $\omega > 0$, we define

$$\mathcal{T}_k(x', \omega) := \left\{ X_k \in \prod_0^{p_k} B_k : \|P_{W_k}(\{\nabla f(x)\}_{x \in X_k})\|_{W_k}^2 \leq \|P_{W_k}(\mathcal{G}_k(x'))\|_{W_k}^2 + \omega \right\}.$$

The purpose of the lemma is that it shows that for a sufficiently large sample set size p_k, an iterate x_k sufficiently close to x', and any $\omega > 0$, there exists a nonempty subset of $\mathcal{T}_k(x', \omega)$. The proof of this result is similar to those of [14, Lemma 4.7] and [32, Lemma 3.2(i)], so it is excluded here for the sake of brevity.

Lemma 3.6 *If $p_k \geq n + 1$, then for any $\omega > 0$, there exists $\zeta > 0$ and a nonempty set $\mathcal{T}$ such that for all $x_k \in \mathbb{B}_\zeta(x')$ we have $\mathcal{T} \subseteq \mathcal{T}_k(x', \omega)$.*

We are now prepared to prove our main result.

Proof (Theorem 3.1) If Algorithm 1 terminates finitely with a stationary point for f, or if Algorithm 3 or 4 is called and fails to terminate finitely, then there is nothing left to prove. Otherwise, by Lemma 3.2, Algorithm 1 produces an infinite sequence of iterates $\{x_k\}$. In this case, if $\{f(x_k)\} \to -\infty$, then again there is nothing left to prove, so for the remainder of our analysis we suppose that an infinite iterate sequence $\{x_k\}$ is generated and that

$$\inf_{k \to \infty} f(x_k) > -\infty. \tag{3.3}$$

Our first main goal is to show that $\{\epsilon_k\} \to 0$. To prove this, we consider two cases.

Case 1: Suppose there exists an infinite iteration index set $\mathcal{K}$ such that (2.6b) and (2.9) hold for all $k \in \mathcal{K}$. Then, along with (2.8a), we have

$$f(x_{k+1}) - f(x_k) \leq -\underline{\eta}\alpha_k \|G_k y_k\|_{W_k}^2 \leq -\underline{\eta}\underline{\alpha}\xi^2 \|d_k\|_2^2 \quad \text{for all} \quad k \in \mathcal{K}.$$

Since f is bounded below by (3.3), this implies that

$$\lim_{k \in \mathcal{K}} \|d_k\|_2 = 0,$$

which, by Step 6 of Algorithm 1, implies that $\{\epsilon_k\} \to 0$.

Case 2: Suppose that at least one of (2.6b) or (2.9) does not hold for all sufficiently large k. By the construction of Steps 3–4 of Algorithm 5, it follows that this algorithm will set W_{k+1} satisfying (2.20) for all such k, and hence, with (3.1), it follows that for all sufficiently large k we have

$$\xi^2 \|d\|_2^2 \leq d^T W_{k+1}^{-1} d \quad \text{for all} \quad d \in \mathbb{R}^n, \tag{3.4}$$

or, equivalently,

$$t^T W_{k+1} t \leq \xi^{-2} \|t\|_2^2 \quad \text{for all} \quad t \in \mathbb{R}^n. \tag{3.5}$$

Indeed, in this case, we may assume without loss of generality that these inequalities hold for all k. We now prove that $\{\epsilon_k\} \to 0$ with probability one by showing that the event that $\{\epsilon_k\}$ remains bounded away from zero has probability zero.

Suppose that there exists k' such that $\epsilon_k = \epsilon' > 0$ for all $k \geq k'$. From this fact, it follows that at least one of (2.6a), (2.6b), or (2.6c) does not hold for all $k \geq k'$. In fact, since (3.4) and Lemma 3.2(a) imply that (2.6b) holds for all k, we must have that (2.6a) or (2.6c) does not hold for all $k \geq k'$. However, by Lemma 3.3, we have the existence of the infinite set $\mathcal{K}_\alpha := \{k : k \geq k' \text{ and } \alpha_k > 0\}$. Thus, overall,

$$\|G_k y_k\|_{W_k} > \nu \epsilon' \quad \text{for all} \quad k \in \mathcal{K}_\alpha. \tag{3.6}$$

On the other hand, the fact that $\{f(x_k)\}$ is bounded below by (3.3), the sufficient decrease condition (2.8a), Lemma 3.4 (since (2.6b) holds for all k), and the fact that $\alpha_k = 0$ for all $k \geq k'$ with $k \notin \mathcal{K}_\alpha$ together imply that

$$\sum_{k=k'}^{\infty} \alpha_k \|G_k y_k\|_{W_k}^2 < \infty, \quad \text{and} \tag{3.7a}$$

$$\sum_{k=k'}^{\infty} \|x_{k+1} - x_k\|_2 \|d_k\|_2 < \infty. \tag{3.7b}$$

In conjunction with (3.6), the bound (3.7a) implies $\alpha_k \to 0$. Similarly, (3.7b), (3.6), Lemma 3.2(a), and (3.4) imply that $\{x_k\}$ is a Cauchy sequence, and hence $x_k \to x'$ for some $x' \in \mathbb{R}^n$. We claim that this implies the existence of an infinite iteration index set $\mathcal{K}_p := \{k : k \geq k' \text{ and } p_k = p\}$, for which Lemma 3.2(b) implies $\mathcal{K}_p \subseteq \mathcal{K}_\alpha$. Indeed, if $p_k < p$ for all large k, then, since $\alpha_k \to 0$, Step 3 of Algorithm 2 implies that $\alpha_k = 0$ for all large k. However, as in the proof of Lemma 3.3, this leads to a contradiction as we eventually find $p_k = p$ for some large k. Therefore, we can define $\mathcal{K}_p$ as stated and know $|\mathcal{K}_p| = \infty$. We continue by considering two subcases.

Subcase 2.a: If x' is ϵ'-stationary for f, then $\|P_{W_k}(\mathcal{G}_k(x'))\|_{W_k}^2 = 0$ for any $W_k \succ 0$. Thus, with $\omega = (\nu\epsilon')^2 > 0$ and $(\zeta, \mathcal{T})$ chosen as in Lemma 3.6, there exists $k'' \geq k'$ such that $x_k \in \mathbb{B}_\zeta(x')$ for all $k \geq k''$ and

$$\|G_k y_k\|_{W_k} = \|P_{W_k}(\{\nabla f(x)\}_{x \in X_k})\|_{W_k} \leq \nu \epsilon' \tag{3.8}$$

whenever $k \geq k''$, $k \in \mathcal{K}_p$, and $X_k \in \mathcal{T}$. Together, (3.6) and (3.8) imply that $X_k \notin \mathcal{T}$ for all $k \geq k''$ with $k \in \mathcal{K}_p$. However, this is a probability zero event since

the construction of Algorithm 4 implies that for all such k the set X_k will contain newly generated points from B_k, meaning that with probability one there exists some sufficiently large k such that $X_k \in \mathcal{T}$, yielding (3.8).

Subcase 2.b: Now suppose that x' is not ϵ'-stationary for f. It follows from Lemma 3.2(b) (in particular, the Armijo condition (2.7a)) that for all k we have

$$f(x_k + \gamma^{-1}\alpha_k d_k) - f(x_k) \geq -\underline{\eta}\gamma^{-1}\alpha_k \|G_k y_k\|_{W_k}^2, \tag{3.9}$$

while Lebourg's Mean Value Theorem [12, Theorem 2.3.7] implies the existence of $\tilde{x}_k \in [x_k, x_k + \gamma^{-1}\alpha_k d_k]$ and a corresponding subgradient $v_k \in \partial f(\tilde{x}_k)$ such that

$$f(x_k + \gamma^{-1}\alpha_k d_k) - f(x_k) = \gamma^{-1}\alpha_k v_k^T d_k. \tag{3.10}$$

Together, (3.9), (3.10), and the fact that $d_k = -W_k G_k y_k$ imply

$$v_k^T W_k G_k y_k \leq \underline{\eta}\|G_k y_k\|_{W_k}^2. \tag{3.11}$$

Moreover, with $\omega > 0$ and $(\zeta, \mathcal{T})$ chosen as in Lemma 3.6, there exists $k''' \geq k'$ such that $x_k \in \mathbb{B}_\epsilon(x')$ with $\epsilon = \min\{\zeta, \epsilon'/3\}$ for all $k \geq k'''$ and

$$\|G_k y_k\|_{W_k}^2 = \|P_{W_k}(\{\nabla f(x)\}_{x \in X_k})\|_{W_k}^2 \leq \|P_{W_k}(\mathcal{G}_k(x'))\|_{W_k}^2 + \omega$$

whenever $k \geq k'''$, $k \in \mathcal{K}_p$, and $X_k \in \mathcal{T}$; hence, by Lemma 3.5, for such k we have

$$v^T W_k G_k y_k > \underline{\eta}\|G_k y_k\|_{W_k}^2 \text{ for all } v \in \mathcal{G}_k(x'). \tag{3.12}$$

Together, (3.11) and (3.12) imply that $v_k \notin \mathcal{G}_k(x')$ whenever $k \geq k'''$, $k \in \mathcal{K}_p$, and $X_k \in \mathcal{T}$. However, from the facts that $d_k = -W_k G_k y_k$ and $e^T y_k = 1$ (recall (2.5)), (3.5), Assumption 2.1, and [12, Proposition 2.1.2], we have for all $k \geq k'''$ that

$$\|d_k\|_2 = \|W_k G_k y_k\|_2 \leq \|W_k\|_2 \|G_k y_k\|_2 \leq \xi^{-1} L_{\mathbb{B}_\epsilon(x')},$$

where $L_{\mathbb{B}_\epsilon(x')} \geq 0$ is the Lipschitz constant for f over $\mathbb{B}_\epsilon(x')$; see the similar result [32, Lemma 4.1]. That is, $\{\|d_k\|_2\}$ is bounded for $k \geq k'''$. This, along with the fact that $\alpha_k \to 0$, implies that $\alpha_k \leq \gamma\epsilon'/(3\|d_k\|_2)$ for all sufficiently large k, i.e., $\gamma^{-1}\alpha_k\|d_k\|_2 \leq \epsilon'/3$ for all sufficiently large k. Combining this with the fact that $x_k \in \mathbb{B}_\epsilon(x')$ with $\epsilon = \min\{\zeta, \epsilon'/3\}$ implies $\|x_k - x'\| \leq \epsilon'/3$, we obtain that $\tilde{x}_k \in \mathbb{B}_{2\epsilon/3}(x')$ and hence $v_k \in \mathcal{G}_k(x')$ for all sufficiently large $k \geq k'''$. Overall, since $v_k \notin \mathcal{G}_k(x')$ whenever $k \geq k'''$, $k \in \mathcal{K}_p$, and $X_k \in \mathcal{T}$, yet $v_k \in \mathcal{G}_k(x')$ for all sufficiently large $k \geq k'''$, it follows that $X_k \notin \mathcal{T}$ for all sufficiently large $k \geq k'''$ with $k \in \mathcal{K}_p$. However, since $|\mathcal{K}_p| = \infty$, it follows as in the situation in Subcase 2.a that this is a probability zero event.

We have proved that the situations in Subcases 2.a and 2.b have probability zero, which implies that the event that there exists k' such that $\epsilon_k = \epsilon' > 0$ for all $k \geq k'$ has probability zero. This result and the proof of Case 1 shows that $\{\epsilon_k\} \to 0$ with probability one, as desired.

All that remains is to show that when $\{\epsilon_k\} \to 0$, all cluster points of $\{x_k\}$ are stationary for f. The proof is exactly that of [14, Theorem 4.2, Case 2]. □

4 Implementation and Numerical Experiments

In this section, we describe a C++ implementation of our algorithm along with the results of numerical experiments that we performed to compare our code against other available software for solving problem (2.1). All of our experiments were performed on a machine running Debian 2.6.32 with two 8-Core AMD Opteron 6128 2.0GHz processors and 32GB of RAM.

4.1 An Implementation and Alternative Software

Hereafter, we refer to our implementation of Algorithm 1, along with all the subroutines described as Algorithms 2, 3, 4, and 5, as BFGS-GS. For convenience, BFGS-GS utilizes the linear algebra library ARMADILLO, version 4.300.0 [42]. A critical part of the implementation is the method for solving the QP (2.5), for which we implemented a specialized active set solver adapted from that proposed in [31]; further details for a similar Matlab implementation are discussed in [14, Appendix].

Recalling Table 2.1, the values of the input parameters used in our implementation are given in Table 4.1. The only exception is that we do not set a value for the parameter J since, in BFGS-GS, we do not check whether the iterates or sample points lie in the set $\mathcal{D}$. That is, at all steps in the algorithm and its subroutines where one would normally check for a point's inclusion in $\mathcal{D}$, BFGS-GS determines that the point is indeed included. At such points, BFGS-GS assumes that a (sub)gradient of f is provided. Such an approach was also employed in the gradient sampling algorithms in [7, 13, 14], where it was argued—as we claim here in terms of our experiments—that, due to the presence of a GS strategy, this is a reasonable approach for practically handling nondifferentiability of f at certain points. Our choice of a sample set size threshold of $p = 100$ was based on the fact that this value worked well in our tests, which all involved $n \approx 50$; see §4.2. Also, our model curvature threshold ξ does not satisfy the upper bound in (3.1); instead, we chose a relatively large value that worked well in our experiments.

Table 4.1 Summary of input parameters for algorithm BFGS-GS.

Parameter(s)	Value(s)	Description
ν	1	Stationarity measure tolerance
ψ	0.5	Sampling radius reduction factor
ξ	10^{-4}	Model curvature threshold
$\underline{\eta} < \overline{\eta}$	$10^{-8} < 0.9$	Armijo–Wolfe line search constants
$\underline{\alpha} \leq \overline{\alpha}$	$10^{-4} \leq 1$	Step size thresholds
γ	0.5	Step size modification factor
$\underline{J} \leq \overline{J}$	$5 \leq 10$	Iteration thresholds for line search
p	100	Sample set size threshold
$\underline{\mu} < 1 < \overline{\mu}$	$0.2 < 1 < 100$	(L-)BFGS updating thresholds
$\underline{w} \leq \overline{w}$	$10^{-4} \leq 1$	(L-)BFGS updating thresholds
m	100	L-BFGS memory length

We also use the following input parameters for BFGS-GS. We set the initial sampling radius to $\epsilon_0 \leftarrow 0.1$ as this value generally worked well in our experiments. For the QP solver for subproblem (2.5), we set an optimality tolerance of 10^{-8}

and a maximum iteration limit of 10^3; i.e., the QP solver terminates once the ℓ_∞-norm of the residual of its KKT conditions is less than this tolerance or the iteration counter exceeds this limit. Regardless of the reason for termination of the QP solver, BFGS-GS uses the search direction yielded by the final QP solver iterate; i.e., BFGS-GS does not terminate if the QP solver fails to provide an accurate solution as determined by the KKT error tolerance. For Algorithm 4, we found a good choice to be $\overline{p}_{k+1} \leftarrow 5$ for all k. The initial inverse Hessian approximation corresponding to $k = 0$ is set as $W_k \leftarrow w_k I$, where the scalar w_k is set as

$$w_k \leftarrow \frac{1}{\max\{1, \min\{10^4, \|\nabla f(x_k)\|_2\}\}}.$$

This is also the value for w_k employed in the L-BFGS strategy in Algorithm 5. Finally, BFGS-GS terminates when the iteration counter k exceeds 10^4 or when

$$\epsilon_k \leq \epsilon_f, \tag{4.1a}$$
$$\|G_k y_k\|_{W_k} \leq \epsilon_f, \tag{4.1b}$$
$$\|G_k y_k\|_{W_k} \geq \xi \|d_k\|_2, \tag{4.1c}$$
$$\text{and} \quad \alpha_k > 0, \tag{4.1d}$$

for some constant $\epsilon_f > 0$. In our tests below, we consider $\epsilon_f \in \{10^{-4}, 10^{-6}\}$. Reminiscent of (2.6) and (2.6′), these criteria require—recalling that Lemma 2.2 implies $\|G_k y_k\|_{W_k} = \|d_k\|_{W_k^{-1}}$—that the sampling radius has already been reduced to a sufficiently small value and the current step is sufficiently small while the curvature of the current Hessian approximation is sufficiently positive along d_k.

For comparison purposes, we ran implementations of three other algorithms for our numerical experiments. The first two are variants of the software available at [40], which we refer to as HANSO-BFGS and HANSO-DEFAULT. The former solver—obtained by setting `options.samprad = []`—employs a standard BFGS method with a weak Wolfe line search [35], whereas the latter solver—obtained by leaving `options.samprad` at its default value—runs the same approach followed by the application of a GS method as proposed in [7] to obtain an improved solution. Despite the fact that these solvers are implemented in Matlab while BFGS-GS is implemented in C++, we believe our comparisons are appropriate, at least since we focus on performance measures other than CPU time. In particular, our method represents a technique for incorporating a GS strategy while optimizing with a BFGS-type approach, whereas HANSO-BFGS exhibits the behavior of a BFGS method with no safeguarding for handling nonsmoothness and HANSO-DEFAULT exhibits the behavior of an algorithm that switches from BFGS to a GS method. It is worthwhile to note that by switching to a GS method, HANSO-DEFAULT has theoretical convergence guarantees that are similar to the algorithm proposed in this paper, whereas the BFGS algorithm in HANSO-BFGS only has the convergence guarantees provided in [35], which are limited to only a few types of simple problems.

The third algorithm to which we compare our code is the Fortran 77 solver of Karmitsa [29], for which we used the available mex-driver for Matlab users. This code implements the limited memory bundle method proposed in [24, 25]. We refer to this solver as LMBM, and include it in our experiments to illustrate the performance of our approach compared to an alternative quasi-Newton method for solving nonconvex, nonsmooth optimization problems.

The input parameters for HANSO-BFGS, HANSO-DEFAULT, and LMBM (besides `options.samprad = []` for HANSO-BFGS) are left at their default values, except that we set maximum iteration and CPU time limits on par with that chosen for BFGS-GS. In particular, for HANSO-BFGS, we changed the maximum number of iterations to `options.maxit = 1e+4`. This value was also used for HANSO-DEFAULT, but it should be noted that, once its BFGS method terminates, HANSO-DEFAULT may do as many as 300 GS iterations—meaning that we allowed HANSO-DEFAULT to perform as many as 10^4 (BFGS) + 300 (GS) iterations. For LMBM, we changed the maximum number of iterations to `IPAR(2) = 1e+4` and set the maximum time limit to a large enough number that the solver never terminated due to a time limit in our tests. Overall, none of the solvers that we tested had a CPU time limit that led to termination in any of our experiments.

4.2 Test problems

For our numerical experiments, we measured the performance of all algorithms on 26 nonsmooth minimization problems, some convex and some nonconvex. The name and source of each problem is indicated in Table 4.2. The problems from [25] and [36] were also considered in [24], and all 26 were considered in [44].

Table 4.2 Number, name, and source for each problem in our test set.

#	Name	Source	#	Name	Source	#	Name	Source
1	MAXQ	[25]	11	TEST29_2	[36]	21	TILTED_NORM_COND	[35]
2	MXHILB	[25]	12	TEST29_5	[36]	22	CPSF	[35]
3	CHAINED_LQ	[25]	13	TEST29_6	[36]	23	NCPSF	[35]
4	CHAINED_CB3_I	[25]	14	TEST29_11	[36]	24	EIG_PROD	[35]
5	CHAINED_CB3_II	[25]	15	TEST29_13	[36]	25	GREIF_FUN	[21]
6	ACTIVE_FACES	[25]	16	TEST29_17	[36]	26	NUC_NORM	[44]
7	BROWN_FUNCTION_2	[25]	17	TEST29_19	[36]			
8	CHAINED_MIFFLIN_2	[25]	18	TEST29_20	[36]			
9	CHAINED_CRESCENT_I	[25]	19	TEST29_22	[36]			
10	CHAINED_CRESCENT_II	[25]	20	TEST29_24	[36]			

All problems in this test set are scalable in the sense that they can be defined to have different numbers of variables. We chose $n = 50$ for all problems, except for the case of EIG_PROD that requires the number of variables to be the square of an integer, for which we choose $n = 64$. We ran each problem ten times with ten different starting points. For the first 20 problems, the first run was performed with the initial point x_0 stated in [24] while for the remaining nine runs we used a starting point that was randomly generated from a Euclidean ball about x_0 with radius $\|x_0\|_2$. The initial points in [24] satisfy $x_0 \neq 0$ and the initial points for each run were unique. For the remaining six problems, we chose the initial point as a randomly generated point from a Euclidean ball about e with radius $\|e\|_2$.

The last six problems in our test set require certain input parameters. Problems TILTED_NORM_COND, CPSF, and NCPSF require symmetric positive definite matrices with a specified condition number. To generate these, we used Matlab's built-in `sprandsym` function. Similarly, problem NUC_NORM requires an input matrix and vector, which we generated using Matlab's built-in `randn` function. For the matrix

required in `EIG_PROD`, we used the leading 8×8 submatrix of A from [1]; see also the experiments in [7, 13]. For `GREIF_FUN`, we multiplied the transpose of a 10×10 matrix randomly generated by `randn` with the matrix itself to create a symmetric positive definite matrix A so that the $n = 50$ variables composing the 10×5 variable matrix X has the well-defined sum $A + XX^T$.

We implemented all of the test problems in C++ for use by BFGS-GS. For the remaining solvers, we implemented the first 20 test problems in Matlab and obtained the code for the remaining six from the website of [44].

4.3 Numerical results

The purpose of presenting the results of our numerical experiments is to illustrate the efficiency and reliability of our BFGS-GS solver in comparison to HANSO-BFGS, HANSO-DEFAULT, and LMBM with their default parameter settings when run on the $26 \times 10 = 260$ problems in our test set. That is, we tested our 26 problem formulations, each run with ten different starting points. Since the codes are written in various languages and were run in different environments—i.e., compiled C++ versus interpreted Matlab code—we ignore CPU time and focus on the performance measures of iterations, function evaluations, and gradient evaluations required until termination. Despite the fact that we ignore CPU time, we claim that the per-iteration costs of the algorithms underlying BFGS-GS, HANSO-DEFAULT, and LMBM are all relatively similar, especially when averaged over all iterations that may be performed. Thus, by being successful in terms of the performance measures that we consider, we claim that one should expect success in terms of CPU time if all codes were implemented in the same language and run in the same environment. By contrast, the average per-iteration cost of HANSO-BFGS is typically less than all other solvers, at least when ignoring computations performed to evaluate a stationarity measure. However, due to the fact that it is based on an algorithm that lacks theoretical convergence guarantees, one would expect HANSO-BFGS to be less reliable. Indeed, this is evident in our numerical results.

When running our experiments with HANSO-BFGS, HANSO-DEFAULT, and LMBM, we observed that the default settings of these codes resulted in markedly different performance. In particular, the default settings of LMBM led to runs that terminated after many fewer iterations, function evaluations, and gradient evaluations as compared to HANSO-BFGS and HANSO-DEFAULT. However, when comparing solvers for nonsmooth optimization problems, one should not necessarily rely upon the termination conditions employed in a given implementation to have a sense of the quality of the provided solutions. As opposed to smooth optimization where one can simply observe the magnitude of the objective function gradient at the final iterate, stationarity measures for nonsmooth problems require information about the subdifferential or ϵ-subdifferential of the objective at the final iterate, which often can only be approximated. Hence, rather than focus solely on the performance measures mentioned above, we investigated further and found that the performance of LMBM as compared to HANSO-BFGS and HANSO-DEFAULT was not as good when considering a measure of quality of the provided solutions. (We define our quality measure later in this section.) Based on these observations, we could have adjusted the input parameters for all of the codes in order to ensure that solutions of similar quality were found before a given code was allowed to

terminate. However, we found this to be difficult due to the numerous termination conditions employed in the codes; some are based on stationarity measures, but others are based on changes in the function values, failure to compute a direction of strict descent, etc. Hence, instead, we decided to leave the default inputs for these solvers, but present results for two separate runs of our code: one with the stationarity tolerance of $\epsilon_f \leftarrow 10^{-4}$ in (4.1) and one with $\epsilon_f \leftarrow 10^{-6}$. We show that with the former setting, our code—BFGS-GS(10^{-4})—was able to obtain solutions of similar quality as those obtained by LMBM, and could generally do so with fewer iterations, function evaluations, and gradient evaluations. On the other hand, with the latter setting, our code—BFGS-GS(10^{-6})—continued on to obtain solutions that often had similar quality as those obtained by HANSO-BFGS and HANSO-DEFAULT. Overall, our goal in presenting two sets of results for our code is to demonstrate the versatility of our software; it can quickly obtain solutions of reasonable quality, and, when desired, it can be forced to continue to obtain higher solution accuracy.

Table 4.3, below, summarizes the termination flags returned by all of the codes for all of the problems in our tests. We group the flags into three types:

(1) a stationarity measure tolerance was satisfied;
(2) the maximum iteration limit was reached;
(3) other.

Termination flags of the last type indicate termination based on various occurrences such as small changes in the objective, a failure to compute a direction of strict descent, etc. Overall, Table 4.3 reveals that with both termination tolerances our code was very successful in satisfying our termination criteria (4.1), whereas the other codes often terminated due to other reasons.

Flag	BFGS-GS(10^{-4})	BFGS-GS(10^{-6})	HANSO-BFGS	HANSO-DEFAULT	LMBM
(1)	253	229	68	68	20
(2)	7	31	31	19	0
(3)	0	0	161	173	240

Table 4.3 Counts of termination flag types

Next, we illustrate the performance of the algorithms in terms of iterations, function evaluations, and gradient evaluations via profiles in the style of Dolan and Moré [15]. Typically, when preparing such profiles, it is incumbent upon the user to decide when a particular run should be considered successful or unsuccessful. In our experiments, making this distinction was a difficult task due to the various termination flags returned by HANSO-BFGS, HANSO-DEFAULT, and LMBM. Indeed, if we only considered a termination flag of type (1) to be the indicator for a successful run, then the profiles would be skewed in favor of the codes that yielded such a flag most often, even though we often found that other runs also yielded good quality solutions. Hence, having presented the counts for the termination types in Table 4.3 above, we present performance profiles considering all runs by all codes to be successful. Despite the fact that this means, e.g., that a termination flag of type (2) is not considered a failure, we believe that the profiles are still meaningful since, for one thing, our iteration limit of 10^4 was quite large; this means that if a code performed the maximum number of iterations, then this had an adverse affect for the code in the profile, as it would if such a run were considered a failure.

Figures 4.1 and 4.2 have the performance profiles we obtained in terms of iterations, function evaluations, and gradient evaluations, respectively. Based on these profiles, we have a few observations, all of which should be considered along with the results in Table 4.3 and the solution quality measures that we present later in Table 4.4 to obtain a complete picture of the results of our experiments. First, the profiles reveal the observation that we made earlier about LMBM typically terminating after performing fewer iterations, function evaluations, and gradient evaluations as compared to HANSO-BFGS and HANSO-DEFAULT. Second, the profiles reveal that BFGS-GS(10^{-4}) often outperforms LMBM in terms of all three measures; this is most interesting when one observes that these methods often obtained solutions of similar quality, as we show later. Third, the profiles reveal that BFGS-GS(10^{-6}) is more similar to HANSO-BFGS and HANSO-DEFAULT in terms of all three measures than is BFGS-GS(10^{-4}), so—in terms of our code and the solution quality results shown later—it is reasonable to compare the results of BFGS-GS(10^{-6}) to HANSO-BFGS and HANSO-DEFAULT.

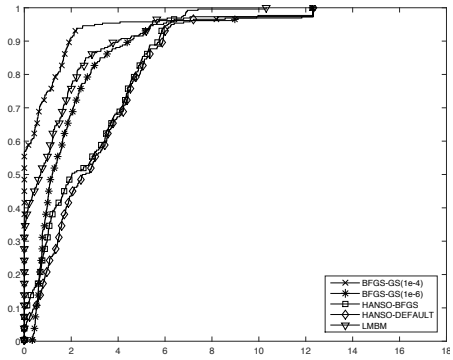

Fig. 4.1 Performance profile for iterations

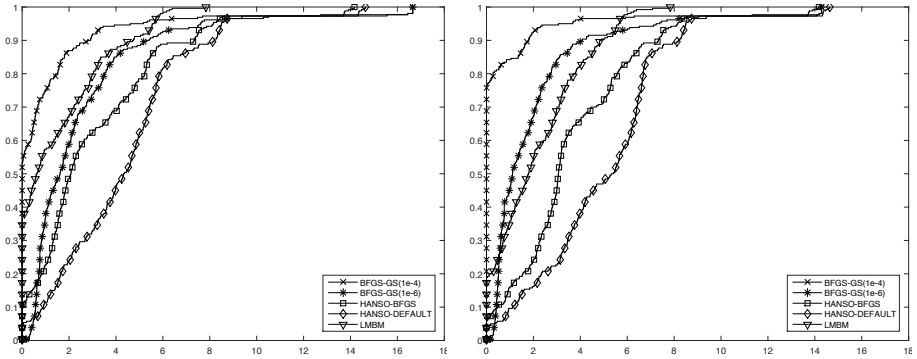

Fig. 4.2 Performance profiles for function (left) and gradient (right) evaluations

We are now prepared to consider the results of our experiments in terms of the quality of the provided solutions. For this purpose, we collected the final iter-

ates provided by all of the codes on all of the problems in our test set. For each final iterate, say x_f, we randomly generated 10^3 points from a uniform distribution defined in a Euclidean ball with radius 10^{-2} about x_f. Then, using a Matlab implementation of our QP solver, we computed the minimum Euclidean norm element of the convex hull of the gradients of the objective evaluated at these points. The norm of this minimum-norm vector represents a reasonable approximation of ϵ-stationarity (with $\epsilon = 10^{-2}$) of x_f with respect to f. This type of measure was employed in [35] as a certificate of stationarity, except that, in that article, the authors employed iterates generated in the algorithm as opposed to randomly generated ones. We could have used iterates in this way as well, but we believe that by randomly generating the points—independent of the algorithm iterates—we obtain a fairer measure for comparing solution quality for the different codes.

For each solver and each of the 26 problems in our original test set, Table 4.4 provides the geometric means of the norms of the minimum Euclidean norm vectors—as described in the previous paragraph—for the ten runs for each problem. We use geometric means as opposed to arithmetic means so that each mean is not skewed by one or a few large terms. Overall, one can see that, for all codes, results can vary in terms of this measurement of solution quality. All of the solvers are competitive, though, broadly speaking, the quality of the solutions provided by BFGS-GS(10^{-4}) and LMBM are not as good as those provided by BFGS-GS(10^{-6}), HANSO-BFGS, and HANSO-DEFAULT. In terms of HANSO-BFGS and HANSO-DEFAULT, we believe that the improved solution quality is due to the termination criteria employed in the software. In particular, these algorithms check for stationarity in a similar way that we measure it here: they compute the minimum Euclidean norm element in the convex hull of gradients evaluated around a given iterate. By contrast, BFGS-GS and LMBM have termination criteria that are influenced by the employed quasi-Newton Hessian approximations. Due to this fact, we could include in BFGS-GS an extra step to measure stationarity using a Euclidean norm measure, but we chose not to do this in order to avoid the extra computational expense of generating additional sample points and solving a large QP in our software. We feel that this is appropriate since, with its tightened stationarity tolerance, BFGS-GS(10^{-6}) is able to obtain solutions of similar quality as those yielded by HANSO-BFGS and HANSO-DEFAULT. That being said, there are cases where BFGS-GS(10^{-6}) yields better or worse solutions. For example, for a few starting points, our solver performs poorly on problem 20.

5 Conclusion

We have proposed an algorithm for solving nonconvex, nonsmooth optimization problems. The main features of the algorithm are that it typically behaves as an unadulterated BFGS method—and, hence, it often has very low per-iteration computational costs—but dynamically incorporates gradient sampling to ensure progress toward stationarity. We have proved that the algorithm has global convergence guarantees with probability one, and, on a set of test problems, we have shown that an implementation of it is competitive with—and in some ways outperforms—other available software for solving such problems.

While the theoretical convergence guarantees of our algorithm in some cases rely on an L-BFGS strategy that ensures sufficiently positive definite and bounded

Problem	BFGS-GS(10^{-4})	BFGS-GS(10^{-6})	HANSO-BFGS	HANSO-DEFAULT	LMBM
1	5.5115e-02	3.2624e-03	1.0931e-14	1.0931e-14	2.9769e-14
2	2.6008e-06	4.0027e-12	2.0981e-14	2.0981e-14	7.4413e-11
3	1.0032e-01	7.9324e-03	1.9953e-01	1.9953e-01	3.6743e-01
4	6.6657e-15	8.0674e-15	6.5293e-15	6.7015e-15	1.1089e-04
5	5.1371e-02	1.4784e-11	1.4116e-11	1.4116e-11	1.5361e-09
6	1.5343e-01	1.5343e-01	2.5912e-16	2.5912e-16	0.0000e+00
7	2.7203e-15	2.3324e-15	2.3766e-15	2.3766e-15	3.9072e-02
8	4.4343e+00	8.8031e-01	5.6539e+00	5.6539e+00	3.5372e+00
9	7.2550e-03	4.7572e-11	9.7894e-12	9.7894e-12	5.0344e-10
10	2.1219e+00	2.3921e+00	2.4665e+00	2.3562e+00	2.2678e+00
11	0.0000e+00	0.0000e+00	0.0000e+00	0.0000e+00	5.3619e-01
12	1.2268e-09	1.4630e-06	1.4011e-06	1.4011e-06	4.9237e-08
13	1.2418e-06	3.1166e-07	3.3843e-03	2.8826e-03	1.4463e-02
14	2.2987e+01	2.5143e+01	2.5657e+01	2.2958e+01	2.0717e+01
15	1.3441e+01	1.2830e+01	2.1905e+02	1.9625e+02	9.4820e-01
16	3.4085e-16	2.4520e-16	9.5007e-15	3.0778e-16	1.4537e-11
17	1.7747e-01	4.2699e-03	2.4028e-03	1.1057e-03	5.3772e-01
18	3.8821e-09	3.1459e-10	1.0375e-06	6.3327e-07	2.1108e-01
19	1.9198e-11	9.5458e-13	1.5087e-13	1.5087e-13	1.7270e-02
20	1.6003e+07	1.1290e+07	1.4061e+00	4.4723e-01	4.4147e+09
21	6.2899e-03	1.4594e-06	1.6764e-06	1.6764e-06	5.4393e-06
22	7.4602e-03	7.2946e-04	1.7976e-06	1.7976e-06	1.5031e-02
23	1.2722e-03	8.8772e-05	6.6831e-07	6.6831e-07	4.9783e-02
24	2.9884e-01	1.1171e-02	4.7271e-09	4.7271e-09	3.5170e-02
25	2.5222e-02	3.6621e-05	4.6504e-07	4.6504e-07	1.0049e-05
26	6.6878e-03	9.3793e-06	1.1201e-06	1.1201e-06	1.4454e-05

Table 4.4 For each solver and each test problem, the geometric means of stationarity measures

Hessian approximations, one can consider a variant of our algorithm that allows these matrices to approach singularity and tend to unboundedness as $\{\epsilon_k\} \to 0$ while preserving our convergence guarantees. In particular, our convergence guarantees rely on the fact that for a given sampling radius, the method eventually satisfies our conditions for reducing this radius with probability one. Hence, one could allow our model curvature threshold and lower (L-)BFGS updating threshold, namely ξ and $\underline{\mu}$, to decrease to zero along with the sampling radius and the upper (L-)BFGS updating threshold, namely $\overline{\mu}$, to correspondly increase to ∞. Our theoretical convergence guarantees hold as long as these parameters remain fixed until the sampling radius is reduced. However, we decided not to propose this variant in the paper since it would require more complicated conditions and a slightly more complicated analysis, and we did not see any benefits of such a strategy in any numerical experiments that we performed.

References

1. Anstreicher KM, Lee J (2004) A Masked Spectral Bound for Maximum-Entropy Sampling. In: MODA 7 - Advances in Model-Oriented Design and Analysis, Berlin, Germany, pp 1–10
2. Ben-Tal A, Nemirovski A (2002) Robust Optimization - Methodology and Applications. Mathematical Programming, Series B 92(3):453–480
3. Bertsekas DP (2009) Convex Optimization Theory. Athena Scientific, Nashua, NH, USA
4. Bonnans JF, Gilbert JC, Lemaréchal C, Sagastizábal CA (2006) Numerical Optimization: Theoretical and Practical Aspects. Springer-Verlag, Berlin Heidelberg New York

5. Broyden CG (1970) The Convergence of a Class of Double-Rank Minimization Algorithms. Journal of the Institute of Mathematics and Its Applications 6(1):76–90
6. Burke JV, Lewis AS, Overton ML (2002) Approximating Subdifferentials by Random Sampling of Gradients. Mathematics of Operations Research 27(3):567–584
7. Burke JV, Lewis AS, Overton ML (2005) A Robust Gradient Sampling Algorithm for Nonsmooth, Nonconvex Optimization. SIAM Journal on Optimization 15(3):751–779
8. Candès EJ, Tao T (2006) Near Optimal Signal Recovery from Random Projections: Universal Encoding Strategies. IEEE Transactions on Information Theory 52(12):5406–5425
9. Chartrand R (2009) Fast Algorithms for Nonconvex compressive Sensing: MRI Reconstruction from Very Few Data. In: ISBI '09: IEEE International Symposium on Biomedical Imaging, pp 262–265
10. Chen X (2012) Smoothing Methods for Nonsmooth, Nonconvex Optimization. Mathematical Programming, Series B 134(1):71–99
11. Chen X, Niu L, Yuan Y (2013) Optimality Conditions and a Smoothing Trust Region Newton Method for Non-Lipschitz Optimization. SIAM Journal on Optimization 23(3):1528–1552
12. Clarke FH (1983) Optimization and Nonsmooth Analysis. Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley & Sons, New York, NY, USA
13. Curtis FE, Overton ML (2012) A Sequential Quadratic Programming Algorithm for Nonconvex, Nonsmooth Constrained Optimization. SIAM Journal on Optimization 22(2):474–500
14. Curtis FE, Que X (2012) An Adaptive Gradient Sampling Algorithm for Nonsmooth Optimization. Optimization Methods and Software DOI 10.1080/10556788.2012.714781
15. Dolan E, Moré J (2002) Benchmarking Optimization Software with Performance Profiles. Mathematical Programming 91:201–213
16. Donoho DL (2006) Compressed Sensing. IEEE Transactions on Information Theory 52(4):1289–1306
17. Fletcher R (1970) A New Approach to Variable Metric Algorithms. Computer Journal 13(3):317–322
18. Goldfarb D (1970) A Family of Variable Metric Updates Derived by Variational Means. Mathematics of Computation 24(109):23–26
19. Goldfarb D, Iyengar G (2003) Robust Portfolio Selection Problems. Mathematics of Operations Research 28(1):1–38
20. Goldstein AA (1977) Optimization of Lipschitz Continuous Functions. Mathematical Programming 13(1):14–22
21. Greif C, Varah J (2006) Minimizing the Condition Number for Small Rank Modifications. SIAM Journal on Matrix Analysis and Applications 29(1):82–97
22. Guenter S, Sempf M, Merkel P, Strumberger E, Tichmann C (2009) Robust Control of Resistive Wall Modes Using Pseudospectra. New Journal of Physics 11(053015):1–40
23. Gumussoy S, Henrion D, Millstone M, Overton ML (2009) Multiobjective Robust Control with HIFOO 2.0. In: Proceedings of the IFAC Symposium on

Robust Control Design, Haifa, Israel
24. Haarala M (2004) Large-Scale Nonsmooth Optimization: Variable Metric Bundle Method with Limited Memory. PhD thesis, University of Jyväskylä
25. Haarala M, Miettinen K, Mäkelä MM (2004) New Limited Memory Bundle Method for Large-Scale Nonsmooth Optimization. Optimization Methods and Software 19(6):673–692
26. Hare W, Macklem M (2013) Derivative-Free Optimization Methods for Finite Minimax Problems. Optimization Methods and Software 28(2):300–312
27. Hare W, Nutini J (2013) A Derivative-Free Approximate Gradient Sampling Algorithm for Finite Minimax Problems. Computational Optimization and Applications 56(1):1–38
28. Hiriart-Urruty JB, Lemaréchal C (1993) Convex Analysis and Minimization Algorithms II. A Series of Comprehensive Studies in Mathematics, Springer-Verlag, New York, NY, USA
29. Karmitsa N (2014) Limited Memory Bundle Method. `http://napsu.karmitsa.fi/lmbm/`, accessed: 2014-05-12
30. Kiwiel KC (1985) Methods of Descent for Nondifferentiable Optimization. Lecture Notes in Mathematics, Springer-Verlag, New York, NY, USA
31. Kiwiel KC (1986) A Method for Solving Certain Quadratic Programming Problems Arising in Nonsmooth Optimization. IMA Journal of Numerical Analysis 6(2):137–152
32. Kiwiel KC (2007) Convergence of the Gradient Sampling Algorithm for Nonsmooth Nonconvex Optimization. SIAM Journal on Optimization 18(2):379–388
33. Kiwiel KC (2010) A Nonderivative Version of the Gradient Sampling Algorithm for Nonsmooth Nonconvex Optimization. SIAM Journal on Optimization 20(4):1983–1994
34. Lemaréchal C (1981) A View of Line-Searches. In: Optimization and Optimal Control: Proceedings of a Conference Held at Oberwolfach, March 16–22, 1980, Berlin, Germany, pp 59–78
35. Lewis AS, Overton ML (2013) Nonsmooth Optimization via Quasi-Newton Methods. Mathematical Programming 141(1–2):135–163
36. Lukšan L, Tůma M, Šiška M, Vlček J, Ramešová N (2002) UFO 2002: Interactive System for Universal Functional Optimization. Tech. Rep. 883, Institute of Computer Science, Academy of Sciences of the Czech Republic
37. Mifflin R (1977) An Algorithm for Constrained Optimization with Semismooth Functions. Mathematics of Operations Research 2(2):191–207
38. Nocedal J (1980) Updating Quasi-Newton Matrices With Limited Storage. Mathematics of Computation 35(151):773–782
39. Nocedal J, Wright SJ (2006) Numerical Optimization, Second edn. Springer
40. Overton ML (2014) HANSO: Hybrid Algorithm for Non-Smooth Optimization. `http://www.cs.nyu.edu/faculty/overton/software/hanso/`, accessed: 2014-05-12
41. Rockafellar RT (1994) Nonsmooth Optimization. In: Birge JR, Murty KG (eds) Mathematical Programming: The State of the Art 1994, University of Michigan Press, Ann Arbor, MI, USA, pp 248–258
42. Sanderson C, Curtin R (2014) Armadillo C++ Linear Algebra Library. `http://arma.sourceforge.net/`, accessed: 2014-05-12

43. Shanno DF (1970) Conditioning of Quasi-Newton Methods for Function Minimization. Mathematics of Computation 24(111):647–656
44. Skajaa A (2010) Limited Memory BFGS for Nonsmooth Optimization. Master's thesis, New York University
45. Vanbiervliet J, Verheyden K, Michiels W, Vandewalle S (2008) A Nonsmooth Optimisation Approach for the Stabilisation of Time-delay Systems. ESAIM: Control, Optimisation and Calculus of Variations 14(3):478–493
46. Vanbiervliet J, Vandereycken B, Michiels W, Vandewalle S, Diehl M (2009) The Smoothed Spectral Abscissa for Robust Stability Optimization. SIAM Journal on Optimization 20(1):156–171
47. Watson GA (2001) Data Fitting Problems with Bounded Uncertainties in the Data. SIAM Journal on Matrix Analysis and Applications 22(4):1274–1293
48. Wolfe P (1975) A Method of Conjugate Subgradients for Minimizing Nondifferentiable Functions. Mathematical Programming Studies 3:145–173