Full-low evaluation methods for bound and linearly constrained derivative-free optimization

C. W. Royer[1], O. Sohab[*2], and L. N. Vicente[2]

[1]LAMSADE, CNRS, Université Paris Dauphine-PSL, Place du Maréchal de Lattre de Tassigny, 75016 Paris, France

[2]Department of Industrial and Systems Engineering, Lehigh University, 200 West Packer Avenue, Bethlehem, PA 18015-1582, USA

June 1, 2024

Abstract

Derivative-free optimization (DFO) consists in finding the best value of an objective function without relying on derivatives. To tackle such problems, one may build approximate derivatives, using for instance finite-difference estimates. One may also design algorithmic strategies that perform space exploration and seek improvement over the current point. The first type of strategy often provides good performance on smooth problems but at the expense of more function evaluations. The second type is cheaper and typically handles non-smoothness or noise in the objective better. Recently, full-low evaluation methods have been proposed as a hybrid class of DFO algorithms that combine both strategies, respectively denoted as Full-Eval and Low-Eval. In the unconstrained case, these methods showed promising numerical performance.

In this paper, we extend the full-low evaluation framework to bound and linearly constrained derivative-free optimization. We derive convergence results for an instance of this framework, that combines finite-difference quasi-Newton steps with probabilistic direct-search steps. The former are projected onto the feasible set, while the latter are defined within tangent cones identified by nearby active constraints. We illustrate the practical performance of our instance on standard linearly constrained problems, that we adapt to introduce noisy evaluations as well as non-smoothness. In all cases, our method performs favorably compared to algorithms that rely solely on Full-eval or Low-eval iterations.

1 Introduction

Derivative-Free Optimization (DFO) [3, 15, 16, 27, 37] methods are developed for the minimization of functions whose corresponding derivatives are unavailable for use or expensive to compute. Particularly useful for complex simulation problems, DFO is often employed when the objective function is derived from numerical simulations, making derivatives inaccessible

[*]Corresponding author. Email: ous219@lehigh.edu
Contributing authors: clement.royer@lamsade.dauphine.fr, lnv@lehigh.edu

for algorithmic purposes. The field of derivative-free optimization now spans a wide range of algorithms and has been applied in numerous engineering and applied science fields [1].

In these settings, evaluating the objective function represents the main computational bottleneck, that must be accounted for while designing DFO algorithms. Besides, simulation codes often enforce hard constraints on their parameters, typically under the form of bounds or linear relationships, that must be satisfied for the simulation to terminate and for function information to be obtained. As a result, a plurality of DFO schemes have been developed to target constrained problems, with a careful classification of the nature of those constraints. For a comprehensive coverage of constraints in DFO, we refer the reader to existing survey papers [27, 28]. We review below algorithmic approaches that bear direct relevance to this paper.

Direct-search methods [25] are a common choice of derivative-free algorithms due to their ease of implementation. These iterative methods sample new function evaluations along suitably chosen directions at every iteration, in order to find a point at which the function value decreases. Direct-search schemes have been endowed with theoretical guarantees even in presence of non-smooth objectives, while being intrinsically robust to the presence of noise in the function evaluations [3]. In presence of linear constraints, direct-search methods generally use directions that conform to the geometry of the feasible set, thereby ensuring feasible descent without relying on derivative information [26]. Recent results have proposed probabilistic variants of direct search, in which the directions are only guaranteed to be feasible descent with a given probability [22]. A probabilistic direct-search iteration can be performed using a significantly smaller number of directions (and, thus, of function evaluations) than its deterministic counterpart.

An alternative to direct-search techniques consists in building an approximate derivative from function evaluations, which then enables the calculation of steps similar to those in the derivative-based setting. Model-based derivative-free techniques obey this logic, and rely on trust-region globalization arguments from nonlinear optimization to guarantee convergence of the methods [15]. As a result, bounds and linear constraints are typically handled in a similar fashion as in the derivative-based setting [13], even though ad hoc strategies have also been considered [21]. Another widely common approach consists in using finite differences to approximate derivatives, so as to leverage existing algorithms from the derivative-based literature [35, Chapter 8]. In particular, recent advances in applying quasi-Newton updates using finite-difference gradients have demonstrated good numerical performance, especially in a smooth setting [8, 9, 39]. This performance is mitigated by the inherent cost of finite-difference estimates, that scales at least linearly with the dimension, and thus may be expensive to perform in a simulation-based environment.

The full-low evaluation (`Full-Low Evaluation`) framework was recently proposed as a principled way of combining derivative-free steps with different costs and properties [9]. In the unconstrained setting, it was proposed to instantiate this framework using a BFGS step computed through finite differences as well as a probabilistic direct-search iteration. This hybrid approach was shown to outperform the individual strategies while being competitive with an established solver on smooth and piecewise smooth problems.

In this paper, we propose an extension of the `Full-Low Evaluation` framework that handles bounds and linear constraints by producing feasible iterates and feasible steps. We analyze an instance of this approach that combines projected steps built on finite-difference gradient estimates with direct-search steps based on probabilistic feasible descent as handled in [22]. The former (`Full-Eval` step) is considered expensive in terms of evaluations but provides good performance and convergence results in the presence of a smooth objective. The latter (`Low-Eval` step) is cheaper in terms of evaluations, while being more robust to noise or non-smoothness in the objective. Similarly to the unconstrained setting, a switching condition determines the nature of the step taken at each iteration.

The rest of this paper is organized as follows. Section 2 states our problem of interest, as well as the key geometrical concepts used to design our algorithm. Section 3 describes our generic **`Full-Low Evaluation`** framework, as well as the two subroutines that define our instance of interest. Section 4 provides convergence results for both smooth and non-smooth objectives. Section 5 details our implementation and our experimental setup, while the output of our tests is analyzed in Section 6. Final remarks are given in Section 7. A list of our test problems is provided in Appendix A.

2 Linearly constrained optimization and tangent cones

The purpose of this section is twofold. First, we describe our problem of interest as well as associated optimality measures in Section 2.1. These concepts will serve as a basis for the **`Full-Eval`** part of our algorithm. Secondly, we discuss the notion of tangent cones and its connection to feasibility in Section 2.2. Those definitions will be instrumental in designing our `Low-Eval` step based on direct search.

2.1 Problem and optimality measure

In this paper, we are interested in solving linearly constrained problems of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & f(x) \\ \text{s.t. } & Ax = b \\ & \ell \leq A_{\mathcal{I}} x \leq u, \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $A_{\mathcal{I}} \in \mathbb{R}^{m_{\mathcal{I}} \times n}$, $b \in \mathbb{R}^m$ and $(\ell, u) \in \bar{\mathbb{R}}^{m_{\mathcal{I}}} \times \bar{\mathbb{R}}^{m_{\mathcal{I}}}$ where $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ with $\ell < u$. To encompass bound constrained problems into the general formulation (2.1), we consider the possibility that the matrix A is empty, in which case m is equal to zero. When $m > 0$, the matrix A is assumed to have full row rank, and we let $W \in \mathbb{R}^{n \times (n-m)}$ be an orthonormal basis for the null space of A. When $m = 0$, we consider W to be the identity matrix in $\mathbb{R}^n$. Finally, we denote the set of feasible points by $\mathcal{F}$.

Assuming that the function f is continuously differentiable, it is possible to define a criticality measure for problem (2.1) that characterizes first-order stationary point. We focus on the quantity $q(\cdot)$ defined by

$$\forall x \in \mathcal{F}, \quad q(x) := \left\| P_{\mathcal{F}} \left[x - \nabla f(x) \right] - x \right\|, \tag{2.2}$$

where $P_{\mathcal{F}}[x] = \operatorname{argmin}_{y \in \mathcal{F}} \|x - y\|$ is the projection of x onto the feasible region $\mathcal{F}$ and $\| \cdot \|$ denotes the Euclidean norm. In derivative-free optimization, the metric (2.2) has been employed for analyzing the convergence of algorithms designed for the linearly constrained setting [29, 30]. Although more recent approaches have focused on another metric bearing a close connection with the direct-search stepsize [22, 25], the measure (2.2) is quite common in projected gradient techniques [10]. Since our theory relies on that of projected gradient techniques in the smooth setting, we naturally focus on the measure (2.2).

When the smoothness of the function f is not guaranteed but f is locally Lipschitz continuous, necessary optimality condition for problem (2.1) can be formulated based on the Clarke-Jahn generalized directional derivative of f [24]. For a given point $x \in \mathcal{F}$, a direction d is called feasible at x there exists $\epsilon > 0$ for which $x + \epsilon d \in \mathcal{F}$. Given $x \in \mathcal{F}$ and a feasible direction d at x, the Clarke-Jahn generalized directional derivative of f at x in direction d is defined as

$$f^\circ(x;d) := \limsup_{\substack{y \to x,\, y \in \mathcal{F} \\ t \downarrow 0,\, y+td \in \mathcal{F}}} \frac{f(y+td)-f(y)}{t}. \tag{2.3}$$

Any $x^* \in \mathcal{F}$ such that $f^\circ(x^*;d) \geq 0$ for any feasible direction d is called a Clarke-Jahn stationary point. Note that such a condition was recently used in the context of non-smooth optimization with linear constraints [7]. In the linearly constrained case, the set of feasible directions at x^* coincides with the tangent cone $T(x^*)$.

2.2 Approximate Tangent Cones

Tangent cones are key concepts to characterize feasibility and optimality in constrained optimization [35]. Although approximate tangent cones have been less studied, they have proven quite useful in the context of derivative-free optimization, as they characterize directions that are feasible for a given step size, by accounting for constraints that are either active or approximately active [25]. We recall below the key definitions related to approximate tangent cones, by following the presentation in Gratton et al. [22].

For convenience, we will define approximate tangent cones based on a parameterization of the feasible set. More precisely, we fix a reference vector $\bar{x} \in \mathbb{R}^{n-m}$ such that $A\bar{x} = b$. Then, any feasible point $x \in \mathcal{F}$ can be written as $x = W\tilde{x} + \bar{x}$, where $\tilde{x} \in \mathbb{R}^{n-m}$ is such that

$$\ell - A_{\mathcal{I}}\bar{x} \;\leq\; A_{\mathcal{I}}W\tilde{x} \;\leq\; u - A_{\mathcal{I}}\bar{x}.$$

Using this decomposition, we define the *approximate active inequality constraints* at $x = W\tilde{x}+\bar{x}$ according to a step size $\xi > 0$ as

$$\begin{cases} I_u(x,\xi) := & \left\{ i :\; |u_i - [A_{\mathcal{I}}\bar{x}]_i - [A_{\mathcal{I}}W\tilde{x}]_i| \leq \xi \left\| W^\top A_{\mathcal{I}}^\top e_i \right\| \right\} \\ I_\ell(x,\xi) := & \left\{ i :\; |\ell_i - [A_{\mathcal{I}}\bar{x}]_i - [A_{\mathcal{I}}W\tilde{x}]_i| \leq \xi \left\| W^\top A_{\mathcal{I}}^\top e_i \right\| \right\}, \end{cases} \tag{2.4}$$

where $e_1, \ldots, e_{m_{\mathcal{I}}}$ denote the coordinate vectors in $\mathbb{R}^{m_{\mathcal{I}}}$. Note that one can assume without loss of generality that $W^\top A_{\mathcal{I}}^\top e_i \neq 0$, otherwise, given that we assume that $\mathcal{F}$ is nonempty, the inequality constraints/bounds $\ell_i \leq [A_{\mathcal{I}}x]_i \leq u_i$ would be redundant. Those indices in turn define the *approximate normal cone* associated with (x,ξ) as

$$N(x,\xi) \;:=\; \operatorname{Cone}\left(\{W^\top A_{\mathcal{I}}^\top e_i\}_{i \in I_u(x,\xi)} \} \cup \{ -W^\top A_{\mathcal{I}}^\top e_i\}_{i \in I_l(x,\xi)} \} \right). \tag{2.5}$$

Rather than using directions from the approximate normal cone to compute steps, we rely on the polar of this cone, called the *approximate tangent cone* and defined by

$$T(x,\xi) \;:=\; \left\{ v \in \mathbb{R}^n \;|v^\mathrm{T} u \leq 0 \;\forall u \in N(x,\xi) \right\}. \tag{2.6}$$

An important property of the approximate tangent cone is that it approximates the feasible region around x, and that moving along all its directions for a distance of ξ from x does not break feasibility [22]. Lemma 2.1 below provides a formal description of this property (see [22, Lemma 2.1] which is based on [26, Proposition 2.2]).

Lemma 2.1 *Let $x \in \mathcal{F}$ and $\xi > 0$. Then, for any vector $\tilde{d} \in T(x, \xi)$ such that $\|\tilde{d}\| \leq \xi$, we have $x + W\tilde{d} \in \mathcal{F}$.*

Direct-search techniques rely on approximate tangent cones to define new feasible points in a way that guarantees convergence to first-order stationarity [26].

3 Full-low evaluation framework with linear constraints

In this section, we describe our main algorithmic framework, that belongs to the class of `Full-Low Evaluation` algorithms. The main idea behind this technique is the combination of two categories of steps. On the one hand, `Full-Eval` steps, that are produced at a significant cost in terms of function evaluations, are used to yield good performance of the method especially in the presence of smoothness. On the other hand, `Low-Eval` steps are cheaper to compute because they require less evaluations, and are often designed to handle the presence of noise and/or non-smoothness in the objective function. We first present our general algorithm that combines both types of steps, then dedicate a section to each category.

The general mechanism of the `Full-Low Evaluation` approach is described in Algorithm 1. In this framework, the iteration type, denoted as t_k, determines whether `Full-Eval` or `Low-Eval` is invoked. The subsequent iteration type t_{k+1} is decided within the invoked function itself, possibly through a user-defined condition. We detail the conditions for switching from one iteration type to another in the next sections.

Algorithm 1 `Full-Low Evaluation` framework

Input: Initial iterate $x_0 \in \mathcal{F}$, low-eval stepsize $\alpha_0 > 0$ and iteration type $t_0 =$ `Full-Eval`.
Output: The final iterate x_∞.

```
1: For $k = 0, 1, 2, \ldots$
2:   If $t_k =$ Full-Eval:
3:     Call $[t_{k+1}, x_{k+1}, \alpha_{k+1}] =$ Full-Eval$(x_k, \alpha_k)$ to compute a Full-Eval step.
4:   Else if $t_k =$ Low-Eval:
5:     Call $[t_{k+1}, x_{k+1}, \alpha_{k+1}] =$ Low-Eval$(x_k, \alpha_k)$ to compute a Low-Eval step.
```

Apart from requiring feasibility of the initial point, note that Algorithm 1 is identical to that of Berahas et al. [9], and that linear constraints are assumed to be handled upon computation of a `Low-Eval` or a `Full-Eval` step. In the next sections, we detail our choices for computing those steps.

3.1 Full-eval step based on projections

`Full-Eval` steps can be implemented by building a model of the objective function around the current point and minimizing it to define the next point. A popular approach that lies within the derivative-free paradigm consists in computing a finite-difference gradient approximation to define a search direction, as well as a stepsize computed via line search based on this approximation [9]. We extend here this approach to the linearly constrained setting by considering projections onto the feasible set, a popular technique for dealing with linear constraints [10].

If the k-th iteration of Algorithm 1 is a `Full-Eval` step, we define a search direction p_k based on an approximate gradient g_k computed through finite differences. A natural choice consists in using $p_k = -g_k$. In Section 5, we detail our practical choices based on quasi-Newton formulas. Once p_k has been computed, we then seek candidate steps by considering the feasible direction

$$\bar{x}_k \; = \; P_{\mathcal{F}}\left[x_k + p_k\right], \tag{3.1}$$

and performing a line search along the direction $\bar{x}_k - x_k$. More precisely, we seek the largest value $\beta \in (0, \bar{\beta}]$ such that

$$f\left(x_k + \beta(\bar{x}_k - x_k)\right) \; \leq \; f(x_k) + c\beta g_k^\top(\bar{x}_k - x_k). \tag{3.2}$$

where $c \in (0, 1)$. We will show in Section 4 that condition (3.2) is satisfied for a sufficiently small value of β.

Algorithm 2 describes the calculation of a `Full-Eval` step for the k-th iteration of Algorithm 1. Similarly to the unconstrained case [9], we introduce a switching condition[*] that controls the norm of the `Full-Eval` step. A value β is accepted if it satisfies the decrease condition (3.2) and

$$\beta \; \geq \; \gamma\alpha_k, \tag{3.3}$$

where $\gamma > 0$ is independent of k. Condition (3.3) guarantees that β does not go below a certain multiple of α_k, which is the stepsize used for computing `Low-Eval` steps (see Section 3.2). When both (3.2) and (3.3) are satisfied, we set $\beta_k = \beta$ and define the new iterate as $x_k + \beta_k(\bar{x}_k - x_k)$. On the other hand, if condition (3.3) is violated, the `Full-Eval` step is skipped.

Algorithm 2 Constrained `Full-Eval` Iteration: Feasible Line Search

Input: Iterate $x_k \in \mathcal{F}$ and α_k. Backtracking global parameters $\bar{\beta} \in (0, 1], \gamma > 0$, $\tau \in (0, 1)$.
Output: t_{k+1}, x_{k+1}, and α_{k+1}.

```
1: Compute the gradient approximation $g_k$ as well as a search direction $p_k$. Compute $\bar{x}_k$
   according to (3.1).
2: Backtracking line-search: Set $\beta = \bar{\beta}$.
3: While True
4:    if (3.2) is true or (3.3) is false, break.
5:    Set $\beta = \tau\beta$.
6: If (3.3) is true, set $\beta_k = \beta$, $x_{k+1} = x_k + \beta_k(\bar{x}_k - x_k)$, and $t_{k+1}$ = Full-Eval. Else, set
   $x_{k+1} = x_k$ and $t_{k+1}$ = Low-Eval.
   (The Low-Eval parameter $\alpha_k$ remains unchanged, $\alpha_{k+1} = \alpha_k$; see Algorithm 3.)
```

The convexity of the set $\mathcal{F}$ guarantees that all iterates remain within the feasible set. This is evident when expressing x_{k+1} in the form $(1 - \beta_k)x_k + \beta_k\bar{x}_k$.

[*]In the unconstrained case [9], we have proposed the switching condition $\beta \geq \gamma\rho(\alpha_k)$. Both work for the convergence theory, in the sense of Lemma 4.2.

3.2 Low-eval step based on feasible descent cones

`Low-Eval` steps are based on the low evaluation paradigm of probabilistic direct search. This approach can be extended to the linearly constrained case as described in Algorithm 3. We suppose that a feasible initial point is provided by the user. At every iteration, the algorithm uses a finite number of polling directions to seek a new feasible iterate x^+ that reduces the objective function value by a sufficient amount

$$f(x^+) \;\leq\; f(x) - \rho(\alpha), \tag{3.4}$$

where ρ is a forcing function classically employed in direct-search methods. The characteristics of ρ are specified in Section 4.2.

Algorithm 3 Constrained `Low-Eval` Iteration: Feasible Direct Search

Input: Iterate $x_k \in \mathcal{F}$ and stepsize α_k. Direct-search global parameters $\lambda \geq 1$ and $\theta \in (0,1)$.
Output: t_{k+1}, x_{k+1}, and α_{k+1}.

```
1: Generate a finite set $D_k$ of non-zero polling directions.
2: If a feasible poll point $x_k + \alpha_k d_k$ is found such that (3.4) is true for some $d_k \in D_k$, set
   $x_{k+1} = x_k + \alpha_k d_k$ and $\alpha_{k+1} = \lambda\alpha_k$.
3: Else, set $x_{k+1} = x_k$ and $\alpha_{k+1} = \theta\alpha_k$.
4: Decide if $t_{k+1} =$ Low-Eval or if $t_{k+1} =$ Full-Eval through a user-defined condition.
```

To ensure the feasibility in Line 2, one can choose directions of the form $W\tilde{d}$, where $\tilde{d} \in T(x_k, \alpha_k)$ with a stepsize less or equal than α_k, as shown by Lemma 2.1.

As Line 4 indicates, the user has the discretion to decide the switching condition from `Low-Eval` to `Full-Eval`. The only theoretical requirement is the eventual return to `Full-Eval`. An example of such condition could involve restricting the sequence to a predetermined maximum of `Low-Eval` iterations. In our actual implementation, this is achieved by limiting the number of unsuccessful `Low-Eval` attempts to equal the count of backtracking steps executed during the preceding `Full-Eval`. This approach ensures a balanced distribution of effort between both types of steps.

4 Convergence Analysis

4.1 Rate of convergence in the smooth case

In this section, we analyze the behavior of the class of `Full-Low Evaluation` methods in the smooth case. We show that if the `Full-Eval` step generates an infinite sequence of iterates, then the norm of $q(x_k)$ converges to zero with a rate of $1/\sqrt{k}$. We now introduce the assumptions needed for the analysis, starting with standard boundedness and smoothness requirements.

Assumption 4.1 *The objective function f is bounded below by $f_{\text{low}} \in \mathbb{R}$, i.e., $f(x) \geq f_{\text{low}}$ for all $x \in \mathbb{R}^n$.*

Assumption 4.2 *The function f is continuously differentiable and its gradient ∇f is Lipschitz continuous with constant $L > 0$.*

The next assumptions are related to our approximate gradient and stationary measure. For any iterate x_k in $\mathcal{F}$ computed by Algorithm 1, we define

$$q_k = P_{\mathcal{F}}\left[x_k - \nabla f(x_k)\right] - x_k, \quad q_k^g = P_{\mathcal{F}}\left[x_k - g_k\right] - x_k \quad \text{and} \quad q_k^p = \bar{x}_k - x_k = P_{\mathcal{F}}\left[x_k + p_k\right] - x_k, \tag{4.1}$$

In our algorithm, we rely on directions defined using g_k. Those should be close to the negative of that approximate gradient, in the sense of Assumption 4.3 below.

Assumption 4.3 *For every iteration k,*

$$\frac{(-g_k)^\top q_k^p}{\|q_k^g\|\|q_k^p\|} \geq \kappa \quad \textit{and} \quad u_p\|q_k^p\| \leq \|q_k^g\| \leq U_p\|q_k^p\|.$$

with $u_p > 0$, $U_p > 0$ and $\kappa \in (0, 1]$.

When $p_k = -g_k$, Assumption 4.3 holds with $\kappa = u_p = U_p = 1$. Indeed, the first inequality can be proved using the property of the projection [10, Proposition 1.1.4(b)] that implies that

$$(x_k - g_k - \bar{x}_k)^\top (x - \bar{x}_k) \leq 0 \qquad \forall x \in \mathcal{F}.$$

Moreover, using $x = x_k$ in the previous inequality as well as $\bar{x}_k - x_k = q_k^g = q_k^p$ gives

$$g_k^\top (\bar{x}_k - x_k) \leq -\|\bar{x}_k - x_k\|^2 \quad \Rightarrow \quad -g_k^\top q_k^p \geq \|q_k^g\|\|q_k^p\|,$$

Finally, in order to relate the control the discrepancy between the true criticality measure and its approximation using g_k, we require the following assumption.

Assumption 4.4 *The approximate gradient g_k computed at x_k satisfies*

$$\|\nabla f(x_k) - g_k\| \leq u_g\|q_k^g\|, \tag{4.2}$$

where $u_g \in (0, \kappa(1 - c))$ is independent of k.

This condition generalizes that in full-low methods for unconstrained optimization [9, Assumption 3.2], albeit with a restriction on the constant u_g that becomes superfluous in the unconstrained setting. Nevertheless, condition (4.2) can be guaranteed in a finite number of steps when the gradient is estimated using finite differences as shown in Section 4.3.

We now start our analysis by establishing a lower bound on the stepsize β_k.

Lemma 4.1 *Let Assumptions 4.2, 4.3, and 4.4 hold. If the k-th iteration is a successful* `Full-Eval` *iteration, then*

$$\beta_k \geq \beta_{\min} := \min\left\{\bar{\beta}, \frac{2\tau(\kappa(1 - c) - u_g)u_p}{L}\right\}. \tag{4.3}$$

Proof. If $\beta_k = \bar{\beta}$ satisfies the decrease condition (3.2), then (4.3) holds trivially. Therefore, we consider the case where $\beta_k < \bar{\beta}$ and at least one backtracking step was performed. We consider $\beta_k = \tau\beta_k^f$ where β_k^f represents the final unsuccessful attempt before satisfying the sufficient decrease condition (3.2). This implies that

$$c\beta_k^f g_k^\top (\bar{x}_k - x_k) < f(x_k + \beta_k^f(\bar{x}_k - x_k)) - f(x_k). \tag{4.4}$$

Using a Taylor expansion of f around x_k on the right-hand side of (4.4), we obtain the following inequalities

$$\begin{aligned}
c\beta_k^f g_k^\top(\bar{x}_k - x_k) &\leq \beta_k^f \nabla f(x_k)^\top(\bar{x}_k - x_k) + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2 \\
c\beta_k^f g_k^\top(\bar{x}_k - x_k) &\leq \beta_k^f g_k^\top(\bar{x}_k - x_k) + \beta_k^f[\nabla f(x_k) - g_k]^\top(\bar{x}_k - x_k) \\
&\quad + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2 \\
0 &\leq (1-c)\beta_k^f g_k^\top(\bar{x}_k - x_k) + \beta_k^f[\nabla f(x_k) - g_k]^\top(\bar{x}_k - x_k) \\
&\quad + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2. \qquad (4.5)
\end{aligned}$$

Using Assumption 4.3, we have

$$(g_k)^\top q_k^p \leq -\kappa\|q_k^g\|\|q_k^p\| \quad \Leftrightarrow \quad g_k^\top(\bar{x}_k - x_k) \leq -\kappa\|q_k^g\|\|\bar{x}_k - x_k\|,$$

hence

$$(1-c)\beta_k^f g_k^\top(\bar{x}_k - x_k) \leq -(1-c)\kappa\beta_k^f\|q_k^g\|\|\bar{x}_k - x_k\|. \qquad (4.6)$$

We now turn to the second term in the right-hand side of (4.5). Using Cauchy-Schwarz inequality together with Assumption 4.4, we obtain

$$\begin{aligned}
[\nabla f(x_k) - g_k]^\top(\bar{x}_k - x_k) &\leq \|\nabla f(x_k) - g_k\|\|\bar{x}_k - x_k\| \\
&\leq u_g\|q_k^g\|\|\bar{x}_k - x_k\|.
\end{aligned}$$

Overall, we thus obtain that

$$\beta_k^f[\nabla f(x_k) - g_k]^\top(\bar{x}_k - x_k) \leq u_g\beta_k^f\|q_k^g\|\|\bar{x}_k - x_k\|. \qquad (4.7)$$

Putting (4.6) and (4.7) into (4.5), we obtain

$$\begin{aligned}
0 &\leq -(1-c)\kappa\beta_k^f\|q_k^g\|\|\bar{x}_k - x_k\| + u_g\beta_k^f\|q_k^g\|\|\bar{x}_k - x_k\| \\
&\quad + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2 \\
0 &\leq -(\kappa(1-c) - u_g)\beta_k^f\|q_k^g\|\|\bar{x}_k - x_k\| + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2.
\end{aligned}$$

Using $\kappa(1-c) - u_g \geq 0$ from Assumption 4.4 together with Assumption 4.3, we show

$$0 \leq -(\kappa(1-c) - u_g)u_q\beta_k^f\|\bar{x}_k - x_k\|^2 + \frac{L}{2}(\beta_k^f)^2\|\bar{x}_k - x_k\|^2$$

The latter inequality only holds as long as

$$\beta_k^f \geq \frac{2(\kappa(1-c) - u_g)u_p}{L}.$$

Since $\beta_k^f = \beta_k/\tau$, we can conclude that

$$\beta_k \geq \frac{2\tau(\kappa(1-c) - u_g)u_p}{L}.$$

Combining this result with the case $\beta_k = \bar{\beta}$ gives the desired result. □

We can now establish the main result of the smooth case.

Theorem 4.1 *Let Assumptions 4.1–4.4 hold. Let $K \geq 1$ be the first iteration such that $\|q_{K+1}\| = \|\mathcal{P}_{\mathcal{F}}[x_{K+1} - \nabla f(x_{K+1})] - x_{K+1}\| \leq \epsilon$. Then, to achieve $\|q_{K+1}\| \leq \epsilon$, Algorithm 1 takes at most n_{SF}^K successful* **`Full-Eval`** *iterations, where*

$$n_{SF}^K \;\leq\; \left\lceil L_1(f(x_0) - f_{\text{low}})\epsilon^{-2} \right\rceil, \tag{4.8}$$

with $L_1 = \dfrac{(u_g+1)^2 U_p}{c\kappa\beta_{\min}}$.

Proof. We denote by $\mathcal{I}_{SF}^K$ the set of indices corresponding to successful `Full-Eval` iterations. Let $k \in \mathcal{I}_{SF}^K$. By definition of such an iteration, the sufficient decrease condition (3.2) is satisfied for $x_{k+1} = \bar{x}_k(\beta_k)$, where β_k satisfies (4.3). Moreover, as shown in the proof of Lemma 4.1, we have

$$g_k^\top(\bar{x}_k - x_k) \;\leq\; -\kappa\|q_k^g\|\|\bar{x}_k - x_k\|.$$

Overall, we obtain

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\;\geq\; -c\beta_k g_k^\top(\bar{x}_k - x_k) \\ &\;\geq\; c\kappa\beta_k\|q_k^g\|\|\bar{x}_k - x_k\| \\ &\;\geq\; \frac{c\kappa\beta_{\min}}{U_p}\|q_k^g\|^2. \end{aligned} \tag{4.9}$$

Meanwhile, using Assumption 4.4 gives

$$\|q_k\| \;\leq\; \|q_k - q_k^g\| + \|q_k^g\| \leq (u_g+1)\|q_k^g\|.$$

Therefore, the decrease achieved at iteration k satisfies

$$f(x_k) - f(x_{k+1}) \;\geq\; \frac{c\kappa\beta_{\min}}{U_p(u_g+1)^2}\|q_k\|^2. \tag{4.10}$$

We now consider the changes in function values across all iterations in $\{0,\ldots,K-1\}$. Since the iterate does not change on unsuccessful iterations and the function value decreases on successful `Low-Eval` iterations, we have $f(x_k) - f(x_{k+1}) \geq 0$ for all $k \leq K-1$. Combining this observation with Assumption 4.1 and (4.10) leads to

$$\begin{aligned} f(x_0) - f_{\text{low}} &\;\geq\; f(x_0) - f(x_K) \\ &\;=\; \sum_{k=0}^{K-1} f(x_k) - f(x_{k+1}) \\ &\;\geq\; \sum_{k\in\mathcal{I}_{SF}^K} f(x_k) - f(x_{k+1}) \\ &\;\geq\; \frac{c\kappa\beta_{\min}}{U_p(u_g+1)^2}\sum_{k\in\mathcal{I}_{SF}^K}\|q_k\|^2 \\ &\;>\; \frac{c\kappa\beta_{\min}}{U_p(u_g+1)^2} n_{SF}^K\epsilon^2. \end{aligned}$$

Re-arranging the terms and using the assumption that for $\|q_k\| > \epsilon$ for $k \leq K$, lead to the desired conclusion. □

The rate (4.8) matches existing result for the unconstrained case [9]. This result primarily addresses the number of successful iterations. However, in the context of DFO, the focus shifts more towards the number of function evaluations. Estimating the upper bound on function evaluations needed to achieve $\|q_{K+1}\| \leq \epsilon$ demands careful consideration of various critical aspects. As outlined in Algorithm 1, an iteration could either be a `Full-Eval` iteration, which incurs a cost of up to $n + \log(\beta_{min}/\bar{\beta})/\ \log(\tau) + 1$ function evaluations, or a `Low-Eval` iteration, whose cost is primarily determined by the cardinality of the polling set. While the number of successful `Full-Eval` is established, the dynamics between consecutive `Low-Eval` iterations and unsuccessful `Full-Eval` iterations introduce a layer of complexity that makes it challenging to infer their respective numbers. This difficulty already arises in the unconstrained setting [9], and accurately calculating such a bound falls outside the scope of this convergence analysis.

4.2 Convergence in the non-smooth case

When the smoothness of the function f is not guaranteed, we rely on the properties of the `Low-Eval` steps, and in particular on the sufficient decrease guarantees certified by the forcing function. To this end, we make the following assumptions.

Assumption 4.5 *The function* $\rho : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ *is continuous, positive, non-decreasing, and satisfies* $\lim_{\alpha \to 0^+} \rho(\alpha)/\alpha = 0$.

An example of such function is $\rho(\alpha) = \alpha^p$ with $p > 1$. As in the unconstrained setting [9], we require the following assumption on the failure of `Full-Eval` iterations.

Assumption 4.6 *There exists* $\epsilon_g > 0$ *such that for any* $k \in I_{SF}$*, where* I_{SF} *denotes the set of successful* `Full-Eval` *iterations,* $\|q_k^g\| > \epsilon_g$.

However, we still rely in the analysis on the switching condition (3.3), along with the assumption that the `Low-Eval` iterations generate an infinite subsequence of iterates to prove that the direct-search parameter α_k goes to zero. This result requires the forcing function to satisfy Assumption 4.5 used in the unconstrained regime.

Lemma 4.2 *Let Assumption 4.1, 4.5, and 4.6 hold. Assume that the sequence of iterates* $\{x_k\}$ *is bounded. Then, there exists a point* x_* *and a subsequence* $\mathcal{K} \subset \mathcal{I}_{UL}$ *of unsuccessful* `Low-Eval` *iterates for which*

$$\lim_{k \in \mathcal{K}} x_k \;=\; x_* \quad \text{and} \quad \lim_{k \in \mathcal{K}} \alpha_k \;=\; 0.$$

Proof. First, suppose that the set $\mathcal{I}_{SF} \cup \mathcal{I}_{UF} \cup \mathcal{I}_{SL}$ is of infinite cardinality, where $\mathcal{I}_{UF}$ and $\mathcal{I}_{SL}$ are the sets of unsuccessful `Full-Eval` and successful `Low-Eval` iterations, respectively. Note that this set represents all iterations k for which α_k does not decrease.

For all successful `Full-Eval` iterations $k \in \mathcal{I}_{SF}$, recall that (3.2) holds, i.e.

$$f(x_k) - f(x_{k+1}) \geq -c\beta_k g_k^{\mathrm{T}}(\bar{x}_k - x_k) \geq \frac{c\kappa\beta_k}{U_p}\|q_k^g\|^2.$$

Furthermore, the condition (3.3) is satisfied for $\beta = \beta_k$, leading to

$$f(x_k) - f(x_{k+1}) \geq \frac{c\kappa\gamma}{U_p}\alpha_k\|q_k^g\|^2 \geq \frac{c\kappa\gamma\epsilon_g^2}{U_p}\alpha_k. \tag{4.11}$$

Meanwhile, successful `Low-Eval` iterations $k \in \mathcal{I}_{SL}$ achieve sufficient decrease,

$$f(x_k) - f(x_{k+1}) \geq \rho(\alpha_k). \tag{4.12}$$

Note that in `Full-Eval` unsuccessful iterations $k \in \mathcal{I}_{UF}$ neither x_k nor α_k changes.

Hence, given that for unsuccessful `Low-Eval` iterations ($\mathcal{I}_{UL}$) the function does not decrease, we can sum from 0 to $k \in \mathcal{I}_{SF} \cup \mathcal{I}_{UF} \cup \mathcal{I}_{SL}$ the inequalities (4.11) and (4.12) to obtain

$$\begin{aligned} f(x_0) - f(x_{k+1}) &\geq \sum_{k \in \mathcal{I}_{SF}} (f(x_k) - f(x_{k+1})) + \sum_{k \in \mathcal{I}_{SL}} (f(x_k) - f(x_{k+1})) \\ &\geq \frac{c\kappa\gamma\epsilon_g^2}{U_p} \sum_{k \in \mathcal{I}_{SF}} \alpha_k + \sum_{k \in \mathcal{I}_{SL}} \rho(\alpha_k). \end{aligned}$$

By the boundedness (from below) of f, we conclude that the series are summable, which implies that $\lim_{k \in \mathcal{I}_{SF}} \alpha_k = 0$ or $\lim_{k \in \mathcal{I}_{SL}} \rho(\alpha_k) = 0$ if one of the sets is infinite. Since α remains unchanged during unsuccessful `Full-Eval` steps, and under Assumption 4.5, it follows that $\liminf_{k \in \mathcal{I}_{SF} \cup \mathcal{I}_{UF} \cup \mathcal{I}_{SL}} \alpha_k = 0$ (and thus there must be an infinite subsequence of unsuccessful `Low-Eval` steps driving α_k to zero). If both $\mathcal{I}_{SF}$ and $\mathcal{I}_{SL}$ are finite, this implies that $\mathcal{I}_{UF}$ is infinite. In this case, given the mechanism of the algorithm, there must also exist an infinite subsequence of unsuccessful `Low-Eval` steps driving α_k to zero. Similarly, if $\mathcal{I}_{UL}$ is infinite, there must exist an infinite sequence of unsuccessful `Low-Eval` steps driving α_k to zero.

Overall, there must be an infinite sequence of unsuccessful `Low-Eval` steps driving α_k to zero. From the boundedness of the sequence of iterates, one can extract a subsequence $\mathcal{K}$ of that subsequence satisfying the statement of the lemma. □

We note that this proof also shows that α_k goes to zero for all k. As in the unconstrained case, convergence results are established using the notion of generalized Clarke-Jahn derivative [12] at x along a direction d. In Theorem 4.2, we show that there exists a limit point which is Clarke-Jahn stationary, provided the so-called refining directions are dense in the tangent cone.

Theorem 4.2 *Let Assumption 4.1, 4.5, and 4.6 hold. Assume that the sequence of iterates $\{x_k\}$ is bounded. Let the function f be Lipschitz continuous around the point x_* defined in Lemma 4.2. Let the set of limit points of*

$$\left\{ \frac{d_k}{\|d_k\|},\ d_k \in D_k, k \in \mathcal{K} \right\} \tag{4.13}$$

be dense in the tangent cone $T(x_*)$, where $\mathcal{K} \subset \mathcal{I}_{UL}$ is given in Lemma 4.2.

Then, x_* is a Clarke-Jahn stationary point, i.e., $f^\circ(x_*; d) \geq 0$ for all normalized d in $T(x^*)$.

Proof. The proof follows standard arguments in [4, 5, 41]. Let $\bar{d}$ be a limit point of (4.13), identified for a certain subsequence $\mathcal{L} \subseteq \mathcal{K}$. Then, from the basic properties of the generalized

Clarke-Jahn derivative, and $k \in \mathcal{L}$,

$$\begin{aligned}
f^\circ(x_*; \bar{d}) &= \limsup_{\substack{x_k \to x_*, x_k \in \mathcal{F} \\ \alpha_k \downarrow 0, x_k + \alpha_k \bar{d} \in \mathcal{F}}} \frac{f(x_k + \alpha_k \bar{d}) - f(x_k)}{\alpha_k} \\
&\geq \limsup_{\substack{x_k \to x_*, x_k \in \mathcal{F} \\ \alpha_k \downarrow 0, x_k + \alpha_k d_k \in \mathcal{F}}} \left\{ \frac{f(x_k + \alpha_k d_k) - f(x_k)}{\alpha_k} - L_f^* \|d_k - \bar{d}\| \right\} \\
&= \limsup_{\substack{x_k \to x_*, x_k \in \mathcal{F} \\ \alpha_k \downarrow 0, x_k + \alpha_k d_k \in \mathcal{F}}} \left\{ \frac{f(x_k + \alpha_k d_k) - f(x_k)}{\alpha_k} + \frac{\rho(\alpha_k)}{\alpha_k} \right\},
\end{aligned}$$

where L_f^* is the Lipschitz constant of f around x_*. Since $k \in \mathcal{L}$ are unsuccessful `Low-Eval` iterations, it follows that $f(x_k + \alpha_k d_k) > f(x_k) - \rho(\alpha_k)$ which implies that

$$\limsup_{\substack{x_k \to x_*, x_k \in \mathcal{F} \\ \alpha_k \downarrow 0, x_k + \alpha_k d_k \in \mathcal{F}}} \frac{f(x_k + \alpha_k d_k) - f(x_k) + \rho(\alpha_k)}{\alpha_k} \geq 0.$$

From this and Assumption 4.5, we obtain $f^\circ(x_*; \bar{d}) \geq 0$. Given the continuity of $f^\circ(x_*; \cdot)$, one has for any $d \in T(x_*)$ such that $\|d\| = 1$, $f^\circ(x_*; d) = \lim_{\bar{d} \to d} f^\circ(x_*; \bar{d}) \geq 0$. $\square$

4.3 More on the smooth case (use of finite difference gradients)

Let us return to the smooth case to clarify the imposition of Assumption 4.4. Such an assumption is related to the satisfaction of the so-called criticality step in DFO trust-region methods [14, 15] based on fully linear models. In the context of Algorithm 2, those models correspond to an approximate gradient g_k built from finite differences.

The i-th component of the forward finite-differences (FD) approximation of the gradient at x_k is defined as

$$[\nabla_{h_k} f(x_k)]_i \;=\; \frac{f(x_k + h_k e_i) - f(x_k)}{h_k}, \quad i = 1, \ldots, n, \tag{4.14}$$

where h_k is the finite difference parameter and $e_i \in \mathbb{R}^n$ is the i-th canonical vector. Computing such a gradient approximation costs n function evaluations per iteration, and it is implicitly assumed that such evaluations can be made. By using a Taylor expansion, the error in the finite-differences gradient (in the smooth and noiseless setting) can be shown [15] to satisfy

$$\|\nabla f(x_k) - \nabla_{h_k} f(x_k)\| \;\leq\; \frac{1}{2}\sqrt{n}\, L\, h_k. \tag{4.15}$$

It becomes then clear that one way to ensure Assumption 4.4 in practice, when $g_k = \nabla_{h_k} f(x_k)$, is to enforce $h_k \leq u_g' \|q_k^{h_k}\|$, where $q_k^{h_k} = P_{\mathcal{F}}[x_k - \nabla_{h_k} f(x_k)] - x_k$ and $u_g' = 2u_g/(\sqrt{n}L)$. Enforcing such a condition is expensive but can be rigorously done through a criticality-step type argument (see Algorithm 4).

Algorithm 4 Criticality step: Performed if $h_k > u'_g \left\| q_k^{h_k} \right\|$

```
Input: $h_k$, $q_k^{h_k(0)} = q_k^{h_k}$, and $\omega \in (0,1)$. Let $j = 0$.
Output: $q_k^{h_k} = q_k^{h_k(j)}$ and $h_k$.
1: While $h_k > u'_g \left\| q_k^{h_k(j)} \right\|$ Do
2:     Set $j = j+1$ and $h_k = \omega^j u'_g \left\| q_k^{h_k(0)} \right\|$.
3:     Compute $\nabla_{h_k} f(x_k)$ using (4.14) and set $q_k^{h_k(j)} = P_{\mathcal{F}}\left[x_k - \nabla_{h_k} f(x_k)\right] - x_k$
```

Proposition 4.1 shows that Algorithm 4 terminates in a finite number of steps.

Proposition 4.1 *Let Assumption 4.2 hold. If $\|q_k\| > 0$, then Algorithm 4 terminates in finitely many iterations by computing h_k such that the condition $h_k \le u'_g \|q_k^{h_k}\|$ is satisfied.*

Proof. Let us suppose that the algorithm loops infinitely. Then, for all $j \ge 1$, using Step 2 and the satisfaction of the while–condition in Step 1,

$$\|q_k^{h_k(j)}\| \;\le\; \omega^j \|q_k^{h_k(0)}\|. \tag{4.16}$$

On the other hand, for all $j \ge 1$, the FD bound (4.15), followed by Step 2, gives us

$$\|\nabla f(x_k) - \nabla_{h_k} f(x_k)^{(j)}\| \;\le\; \frac{1}{2}\sqrt{n} L\, \omega^j u'_g \|q_k^{h_k(0)}\|. \tag{4.17}$$

Hence, using (4.16)–(4.17), we have

$$\begin{aligned}
\|q_k\| \le \|q_k - q_k^{h_k(j)}\| + \|q_k^{h_k(j)}\| \;&\le\; \|\nabla f(x_k) - \nabla_{h_k} f(x_k)^{(j)}\| + \|q_k^{h_k(j)}\| \\
&\le\; \|\nabla f(x_k) - \nabla_{h_k} f(x_k)^{(j)}\| + \omega^j \|q_k^{h_k(0)}\| \\
&\le\; \left(\frac{\sqrt{n} L u'_g}{2} + 1\right) \omega^j \|q_k^{h_k(0)}\|,
\end{aligned}$$

where the second inequality on the first line comes from the [non-expansiveness of orthogonal projection. By taking limits (and noting that $\omega \in (0,1)$), we conclude that $q_k = 0$, which yields a contradiction. $\square$

5 Numerical setup

In this section, we will first present our implementation choices for the `Full-Low Evaluation` linearly constrained method. The complete MATLAB implementation is available on GitHub[†]. The repository includes all the necessary algorithms and testing scripts. The numerical environment of our experiments is also introduced (other methods/solvers tested, test problems chosen, and performance profiles). The tests were run using MATLAB R2019b on an Asus Zenbook with 16GB of RAM and an Intel Core i7-8565U processor running at 1.80GHz.

[†]https://github.com/sohaboumaima/FLE

5.1 Practical `Full-Eval` implementation

In this section, we present a detailed discussion of the implementation of the `Full-Low Evaluation` algorithm in the linearly constrained case. Building upon the principles used in the unconstrained case, we introduce a direction p_k that leverages second-order information for faster convergence. Specifically, we define $p_k = -WH_kW^\top g_k$, where H_k represents an approximation of the inverse Hessian using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton update [11, 18, 20, 38], as described in Algorithm 5. Here, $W \in \mathbb{R}^{n\times(n-m)}$ denotes an orthonormal basis for the null space of matrix A. Notably, due to the positive definiteness of H_k, it follows that $WH_kW^\top$ is also positive definite.

Using $WH_kW^\top g_k$ instead of H_kg_k offers two significant advantages. Firstly, the resulting value of $x_k + p_k$ automatically satisfies the equality constraints, since

$$A(x_k - WH_kW^\top g_k) \;=\; b - AW(H_kW^\top g_k) \;=\; b.$$

Secondly, using this direction allows us to compute $W^\top g_k$ rather than directly calculating g_k, thus reducing the computational cost of finite differences from n to $n-m$ function evaluations. Indeed, the forward finite-differences approximation can be reduced to the null space of the linear equality constraints:

$$[W^\top g_k]_i \;=\; \frac{f(x_k + h_kw_i) - f(x_k)}{h_k}, \quad \text{for} \quad i = 1, \ldots, n-m, \tag{5.1}$$

where h_k is the finite difference parameter, and $w_i \in \mathbb{R}^n$ is the i-th column vector of W. In the numerical experiments, the parameter h_k is set to the square root of Matlab's machine precision.

Our `Full-Eval` line-search iteration is described in Algorithm 5, which includes BFGS updates for the inverse Hessian approximation H_k using (5.4). Here, j_k refers to the previous `Full-Eval` iteration, and s_k and y_k are given in (5.3). Notably, in the non-convex case, the inner product $s_k^\top y_k$ cannot be ensured to be positive. To maintain the positive definiteness of the matrix H_k, we skip the BFGS update if $s_k^\top y_k < \epsilon_c\|s_k\|\|y_k\|$, with $\epsilon_c \in (0,1)$ being independent of k. In our implementation, we use $\epsilon_c = 10^{-10}$.

The line search follows the backtracking scheme described in Algorithm 2, using standard values $\bar\beta = 1$ and $\tau = 0.5$. A key feature of our `Full-Low Evaluation` methodology that led to rigorous results (see the proof of Lemma 4.2) is to stop the line search once condition (3.3) is violated. In our implementation, we use:

$$\gamma \;=\; 1, \quad \rho(\alpha_k) \;=\; \min(\gamma_1, \gamma_2\alpha_k^2), \quad \text{with} \quad \gamma_1 \;=\; \gamma_2 \;=\; 10^{-5}. \tag{5.2}$$

For $k = 0$, we perform a backtracking line search using $p_0 = -WW^\top g_0$ (and update t_1 and x_1) as in Algorithm 2 (with constants as in Algorithm 5). The initialization of H_0 is done as follows: If $t_1 =$ `Full-Eval`, then we set $H_0 = (y_0^\top s_0)/(y_0^\top y_0)I$, in an attempt to make the size of H_0 similar to that of $\nabla^2 f(x_0)^{-1}$ [35]. However, if $t_1 =$ `Low-Eval`, we set $H_0 = I$.

5.2 `Low-Eval` implementation

We now elaborate on our implementation of Algorithm 3, and more precisely on the calculation of the polling sets. Our algorithm uses positive generators of the approximate tangent cones described in Section 2.2. By describing an approximate tangent cone as a conic hull of a finite set of vectors, we can then use those vectors as (feasible) directions.

Algorithm 5 `Full-Eval` Iteration: BFGS with FD Gradients

Input: Iterate x_k with $k \geq 1$. Information $(x_{j_k}, g_{j_k}, H_{j_k})$ from the previous `Full-Eval` iteration j_k (if $k > 0$). Backtracking parameters $\bar{\beta} > 0$ and $\tau \in (0,1)$. Other parameters $\epsilon_c, \gamma, \gamma_1 > 0$.
Output: t_{k+1} and (x_{k+1}, H_k, g_k). Return the number nb_k of backtrack attempts.

```
1: Compute the FD gradient $W^\top g_k = W^\top \nabla_{h_k} f(x_k)$ using (5.1).
2: Set
       $s_k = x_k - x_{j_k} \quad \text{and} \quad y_k = g_k - g_{j_k}.$    (5.3)
3: If $s_k^\top y_k \geq \epsilon_c \|s_k\| \|y_k\|$, set
       $H_k = \left(I - \frac{s_k y_k^\top}{y_k^\top s_k}\right) H_{j_k} \left(I - \frac{y_k s_k^\top}{y_k^\top s_k}\right) + \frac{s_k s_k^\top}{y_k^\top s_k}.$    (5.4)
4: Else, set $H_k = H_{j_k}$.
5: Compute the direction $-W H_k W^\top g_k$.
6: Perform a backtracking line-search and update $t_{k+1}$ and $x_{k+1}$ as in Algorithm 2.
```

The problem of finding such positive generators from a description of the cone through linear inequalities has attracted significant research in computational geometry, and is sometimes referred to as the representation conversion problem [33]. Recent advances in linearly constrained optimization have featured off-the-shelf softwares to compute those generators [7]. We follow here a popular approach in the direct-search community [31], that splits the problem of computing positive generators in two cases. In the first case, we are able to leverage the description of the approximate normal cone through positive generators given by (2.5) to directly define that of the approximate tangent cone. In the second case, we compute positive generators for a subset of the cone, and positive generators of the tangent cone are then obtained by considering the union of all these sets for all possible subsets of columns that yield a full row rank matrix [36]. One drawback of this strategy is that it leads to combinatorial explosion in the subsets of columns that must be considered and the number of positive generators that are obtained. For this reason, several implementations [25, 31] have relied on the double description method from computational geometry [19]. This technique can significantly reduce the number of generators that are used to describe the approximate tangent cone, in the minority of cases where it is needed on standard test problems [31].

Our implementation is that of a probabilistic variant of the aforementioned approach proposed by Gratton et al. [22], in which the approximate tangent cone is decomposed into a subspace part and a pointed cone part (i.e. a cone that does not contain a straight line). Given a set of generators for the approximate tangent cone, we can then replace the subset related to the subspace by a direction drawn uniformly at random within that subspace and its negative, while we can randomly sample a fraction of the other generators corresponding to the pointed cone part. Such an approach reduces the number of polling directions even further, while being endowed with almost-sure convergence guarantees [22, Proposition 7.1]. Our implementation follows that of the `dspfd` MATLAB code [22], that uses its own implementation of the double description method.

5.3 Other solvers

We compared the numerical performance of our implementation of `Full-Low Evaluation` (denoted `constFLE`) to four other approaches: (i) a line-search BFGS method based on FD gradients (as if there were only `Full-Eval` iterations), referred to as `constBFGS`; (ii) probabilistic direct search (as if there were only `Low-Eval` iterations), referred to as `dspfd`; (iii) a mesh adaptive direct search solver, `NOMAD`; (iv) a direct search solver, referred to as `patternsearch`.

Given the detailed description of `constFLE`, `constBFGS`, and `dspfd` in previous sections, we only elaborate on `NOMAD` and `patternsearch` below. `NOMAD` [6] is a solver that implements Mesh Adaptive Direct Search (MADS) [5] under general nonlinear constraints. The polling directions belong to positive spanning sets that asymptotically cover the unit sphere densely. In the case of inequality constraints, the user is allowed to choose to handle them via extreme-barrier, progressive-barrier [2] or filter approaches. In our experiments, we choose the default option which is progressive-barrier, but note that an extreme-barrier approach would provide similar conclusions. The `patternsearch` function is a MATLAB's built-in function that comes as a part of the global optimization TOOLBOX [23]. This is a directional direct search method that progresses by accepting a point as the new iterate if it satisfies a *simple decrease* condition. For bounds and linear constraints, `patternsearch` modifies poll points to be feasible at every iteration, meaning to satisfy all bounds and linear constraints. We adopted the default settings in the choice of polling set which uses the Generalized Pattern Search strategy [40].

5.4 Classes of problems tested

Evaluating optimization methods crucially involves assessing their performance across diverse scenarios. In pursuit of this, we perform experiments on smooth, noisy, and non-smooth problems. For each category, the test set is classified into three distinct classes, namely bound constrained problems, general linearly constrained problems, and problems with at least one linear inequality constraint. Detailed dimensions and inequality counts for each problem are provided in the Appendix for reference.

For smooth bound constrained problems, we selected 41 instances from the CUTEst library. The dimensions of these instances range from 2 to 20, and the number of bounds varies between 1 and 40. The relevant details are summarized in Table 1. In the context of smooth general linearly constrained problems, we consider a comprehensive set of 76 CUTEst problems. Each of these problems involves at least one linear constraint, which is not a bound on the variable. The dimensions vary from 2 to 24, and in cases where linear inequalities are present, their count ranges from 1 to 2000. A detailed overview of these general constrained problems can be found in Tables 2 to 3.

To investigate the behavior of the optimization solvers on noisy functions, we conduct experiments using perturbed versions of the aforementioned problems. Following the approach of [34], the perturbed functions are formulated as $f(x) = \phi(x)(1 + \xi(x))$, where ϕ represents the original smooth function. In this case, $\xi(x)$ is a realization of a uniform random variable $U(-\epsilon_f, \epsilon_f)$. These noisy functions provide valuable insights into the robustness of optimization algorithms in practical scenarios.

To perform a comparison on nonsmooth problems, we considered two different test sets. The first test set is built from our smooth benchmark, and is meant to illustrate the behavior of our method in presence of mild nonsmoothness. To create such problems, we considered problems with both bounds and general linear constraints, and moved either the general linear constraints or the bounds into the objective function. As a result of this transformation, we generated a total of 52 bound constrained problems and 107 problems with general linear constraints, out of which 52 included at least one inequality constraint. Comprehensive details about these problems are presented in Tables 4 to 6. In generating general linearly constrained optimization problems, we adopt a method where we penalize only the first portion of the bound constraints in certain cases. This prevents the outcome from being dominated solely by linear equality or inequality constraints. We denote this category as "$1/2B$" in the tables for ease of reference. As an illustrative example, let us consider the transformation of problem LSQFIT. The original problem is formulated as follows:

$$\begin{aligned} \min_{x,y} \quad & \sum_{i=1}^{5}(a_i x + y - b_i)^2 \\ \text{s.t.} \quad & x + y \leq 0.85 \\ & x \geq 0, \end{aligned} \tag{5.5}$$

where $a = [0.1, 0.3, 0.5, 0.7, 0.9]$ and $b = [0.25, 0.3, 0.625, 0.701, 1.0]$. After the transformation, the problem becomes:

$$\begin{aligned} \min_{x,y} \quad & \sum_{i=1}^{5}(a_i x + y - b_i)^2 + \lambda|x + y - 0.85| \\ \text{s.t.} \quad & x \geq 0, \end{aligned} \tag{5.6}$$

where λ represents the penalty parameter. Our second test set of nonsmooth problems consists in 14 linearly constrained minimax problems as introduced in [32]. In this set, the non-smoothness is introduced by the max operator, resulting in less structured non-smoothness compared to the previous set. Detailed information about these problems is provided in Table 7.

6 Numerical Results

We present numerical results under the form of performance profiles in order to gauge optimization solvers' effectiveness. As outlined in [17], these profiles provide a mean of assessing the performance of a designated set of solvers $\mathcal{S}$ across a given set of problems $\mathcal{P}$. They are a visual tool where the highest curve corresponds to the solver with the best overall performance. Let $t_{p,s} > 0$ be a performance measure of the solver $s \in \mathcal{S}$ on the problem $p \in \mathcal{P}$, which in our case was set to the number of function evaluations. The curve for a solver s is defined as the fraction of problems where the performance ratio is at most α,

$$\rho_s(\alpha) \;=\; \frac{1}{|\mathcal{P}|}\text{size}\left\{p \in \mathcal{P} : r_{p,s} \leq \alpha\right\},$$

where the performance ratio $r_{p,s}$ is defined as

$$r_{p,s} \;=\; \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}.$$

The convention $r_{p,s} = +\infty$ is used when a solver s fails to satisfy the convergence test for problem p. The convergence test used is

$$f(x_0) - f(x) \;\geq\; (1-\tau)(f(x_0) - f_L), \tag{6.1}$$

where $\tau > 0$ is a tolerance, x_0 is the starting point for the problem, and f_L is computed for each problem $p \in \mathcal{P}$ as the smallest value of f obtained by any solver within a given number of function evaluations.

In our experiments, we use $100(n+1)$ as a maximum number of function evaluations which is what is need for 100 simplex gradients. Solvers with the highest values of $\rho_s(1)$ are the most efficient, and those with the highest values of $\rho_s(\alpha)$, for large α, are the most robust.

6.1 Smooth problems

Bound constrained problems

Analyzing those results given in Figure 1, one observes that `Full-Low Evaluation` (blue curve) demonstrates the best performance in terms of efficiency (as indicated by the highest curve at a ratio of 1). It is closely followed by pure `Full-Eval` (red curve), with `NOMAD` (magenta curve) ranking third. When considering robustness, `Full-Low Evaluation` outperforms the others, while `NOMAD` ranks second. `patternsearch` ranks last in both efficiency and robustness.

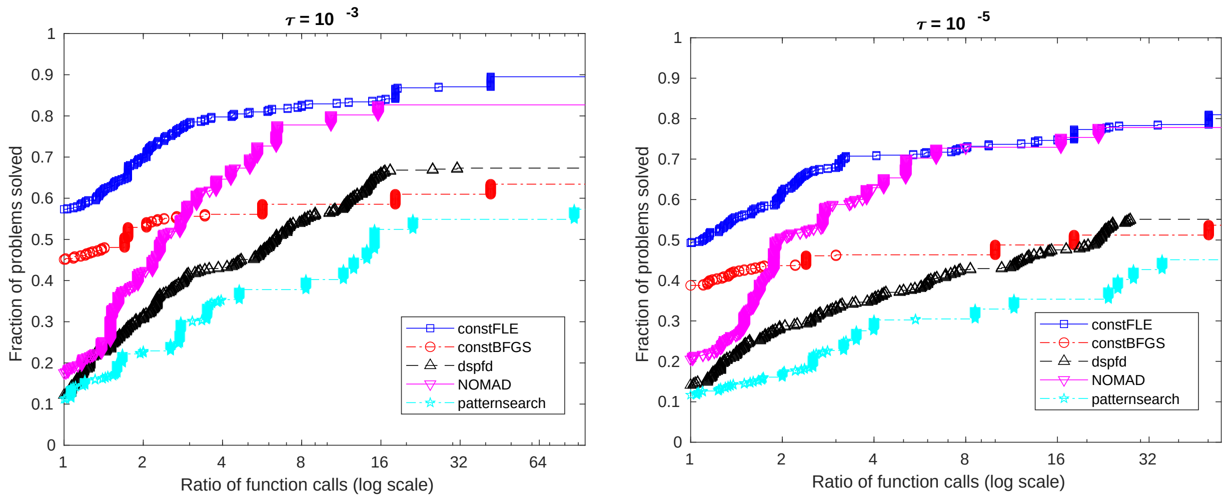


Figure 1: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, NOMAD, and `patternsearch`. The test set contains 41 smooth bound constrained problems from the CUTEst *library.*

Linearly constrained problems

On general linear equality problems, Figure 2 illustrates that our method outperforms the four other solvers in terms of both efficiency and robustness. Pure `Full-Eval` comes second in terms of efficiency, while pure `Low-Eval` performing exceptionally well in terms of robustness and ranks second for that metric. On the other hand, `patternsearch` ranks fourth both in terms of efficiency and robustnes, while `NOMAD` exhibits lower performance due to its limited handling of linear equality constraints, which are present in some of the problems.

Figure 3 provides a more specific comparison of the four solvers on the subset of problems that contain at least one inequality constraint. Even in this context, `Full-Low Evaluation` demonstrates the best performance, followed by `Low-Eval`, `Full-Eval`, then `patternsearch`. It is worth noting that `NOMAD` shows improved performance compared to the previous experiment given the lack of equality constraints.

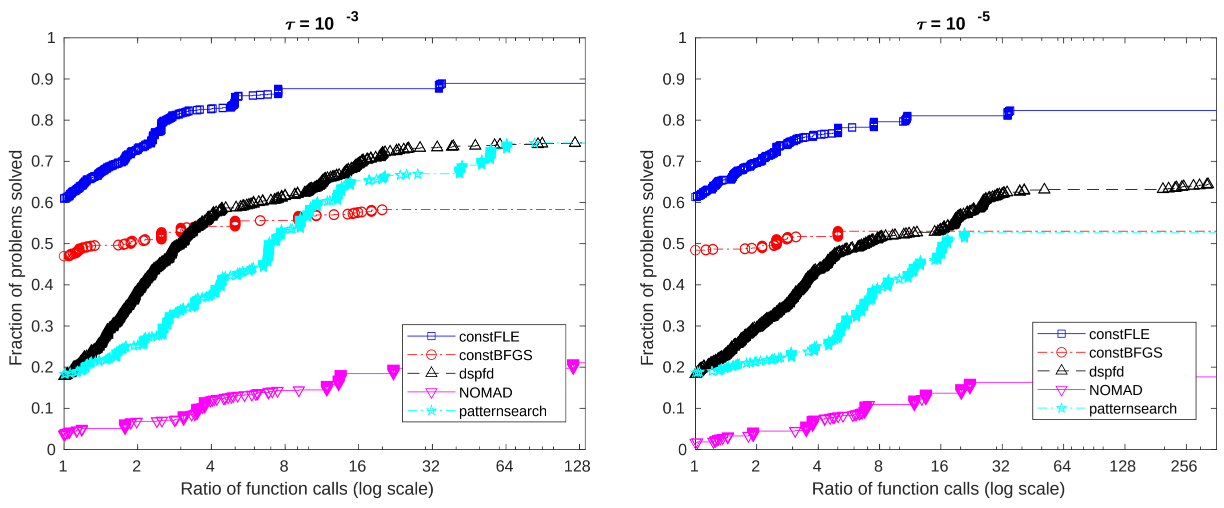


Figure 2: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`, and `patternsearch`. The test set contains 76 smooth problems with general linear constraints from the CUTEst *library.*

6.2 Non-smooth problems

6.2.1 ℓ_1 norm problems

Bound constrained problems: Figure 4 displays the results obtained from testing non-smooth bound constrained problems. `Full-Low Evaluation` is here the most efficient solver, while `Full-Eval` takes the second spot for both low and high accuracy. Meanwhile, `NOMAD` showcases the best robustness. This observed ranking of solvers in the bound constrained setting remains more or less consistent even with the introduction of the non-smooth regularization.

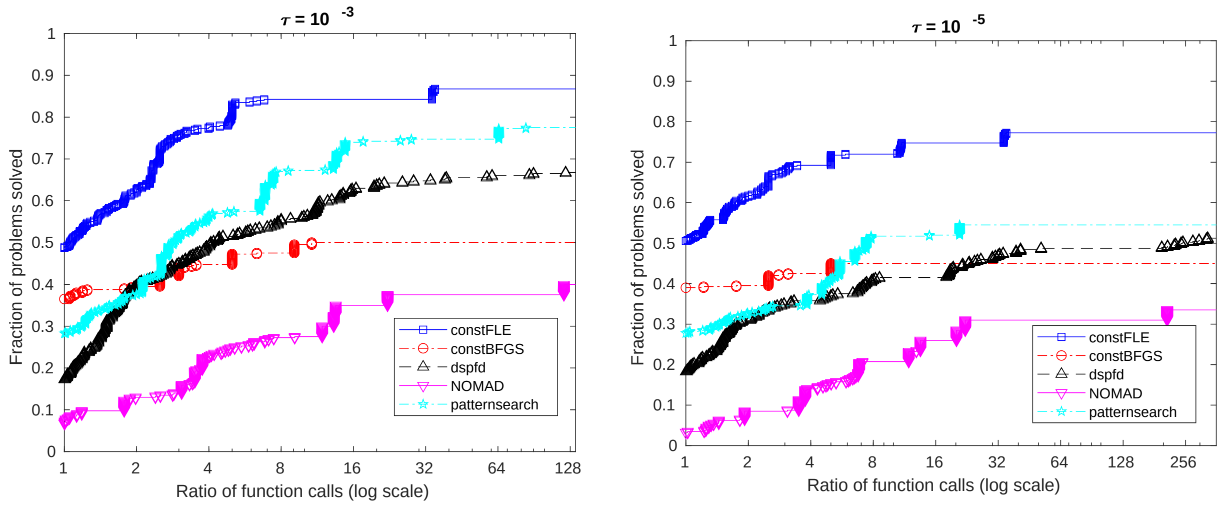


Figure 3: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`, and `patternsearch`. The test set contains 40 smooth problems with at least one inequality constraint from the CUTEst *library.*

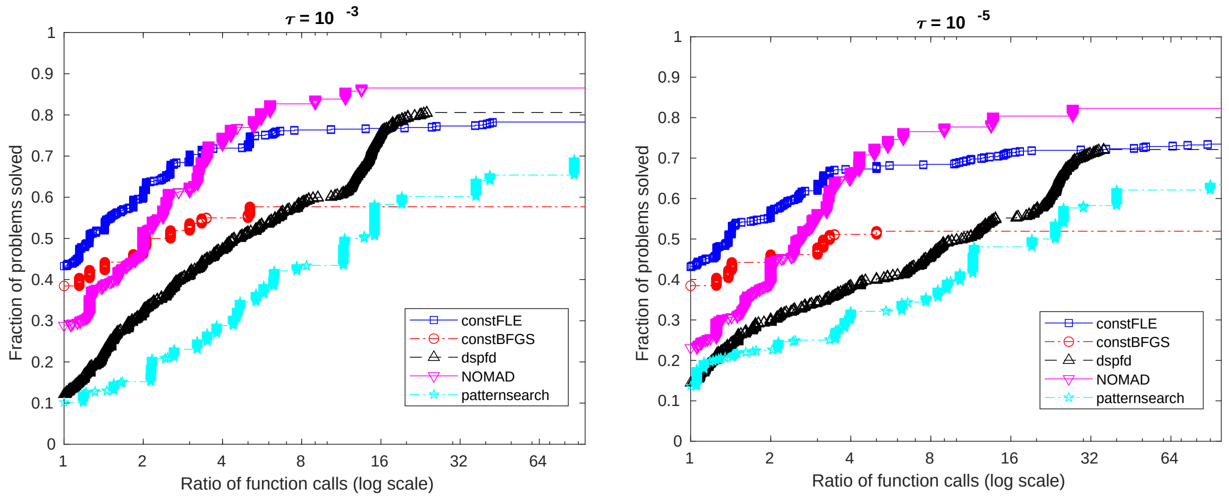


Figure 4: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`, and `patternsearch`. The test set contains 52 non-smooth bound constrained problems.

Linearly constrained problems: In Figure 5, we present the results on general non-smooth problems. One can see that the `Full-Low Evaluation` curve is above all, followed by `Low-Eval` and `patternsearch` which exhibit similar performance, `Full-Eval`, then `NOMAD`. As with the smooth case, employing `Full-Low Evaluation` yields better results than using individual steps alone, providing further confirmation of the effectiveness of our approach.

Furthermore, even within this context, `NOMAD` faces challenges posed by equality constraints. However, upon their removal as shown in Figure 6, `NOMAD` demonstrates improved robustness compared to `Full-Eval`.

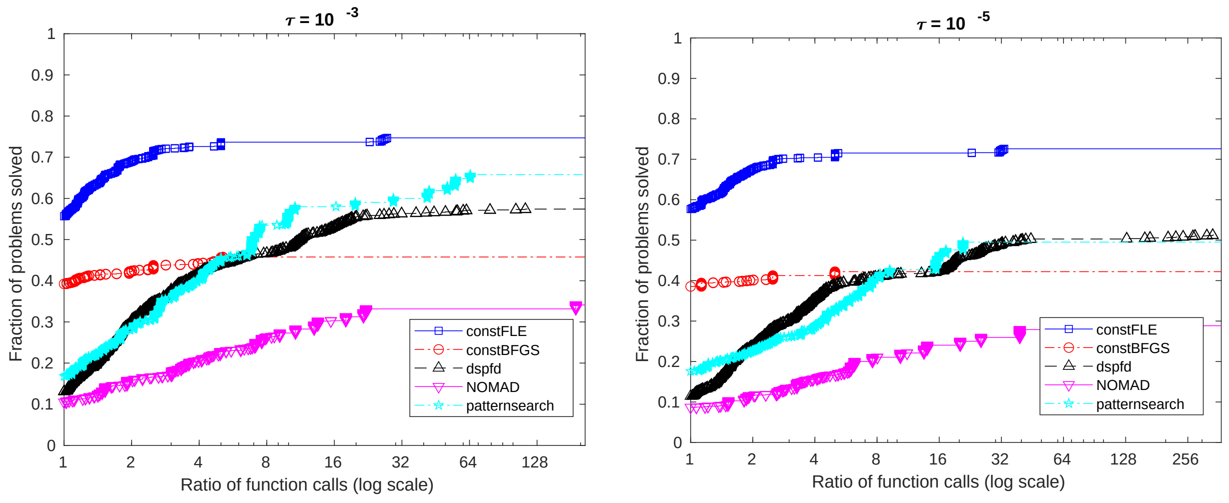


Figure 5: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, NOMAD, and `patternsearch`*. The test set contains 107 non-smooth general linear equality constraints.*

6.2.2 Minimax problems

When the non-smoothness is less structured, methods that estimate the gradient are significantly impacted. Figure 7 illustrates that the relative performance of these methods is notably different from the earlier observations. On this test set, `NOMAD` outperforms the other solvers in terms of both efficiency and robustness, followed by `Low-Eval`. `Full-Eval` comes in third, while `patternsearch` takes the fourth spot. Among all the methods, `Full-Eval` ranks as the least efficient and robust for the reasons aforementioned.

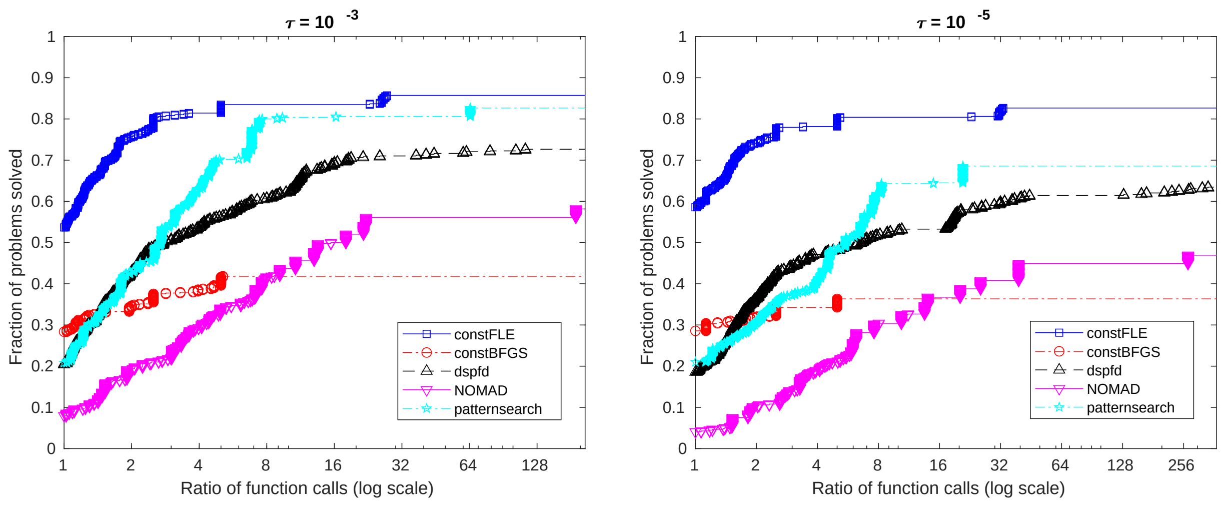


Figure 6: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, NOMAD, and `patternsearch`. The test set contains 52 non-smooth problems with at least one inequality constraint.

6.3 Noisy functions

Bound constrained problems

In this context, `NOMAD` demonstrates the best performance in terms of efficiency and robustness. Referring to Figure 8, we can see that the curve corresponding to `Full-Low Evaluation` is between the `Full-Eval` and `Low-Eval` curves. Such results are conform to observations made in the unconstrained case [9], especially for low accuracy. This correspondence arises from `Full-Eval` performing poorly when h is equal to the square root of machine precision. Note that `Full-Low Evaluation` is able to outperform `Low-Eval` for high accuracy in term of robustness. On the other hand, at lower accuracies, `patternsearch` ranks fourth, followed by `Full-Eval`. However, its performance improves significantly at higher accuracies, where it ranks second in both efficiency and robustness. This demonstrates its effectiveness in noisy settings.

Linearly constrained problems

When tested on general linear equality constrained problems, `patternsearch` stands out as the most efficient and robust solver. Pure `Low-Eval` (probabilistic direct search) shows a comparable efficiency, especially for higher accuracy, followed by `Full-Low Evaluation` which is more robust than `Low-Eval`. Conversely as observed in Figure 9, `NOMAD` experiences a performance decline similar to observations in both smooth and non-smooth cases. Figure 10 sheds light on problems featuring linear inequalities. Notably, in this context, `NOMAD`'s performance stands on par with `Full-Eval`, and it even surpasses it, especially under conditions demanding higher accuracy. The relative order of performance among the other solvers remained consistent, with `patternsearch`

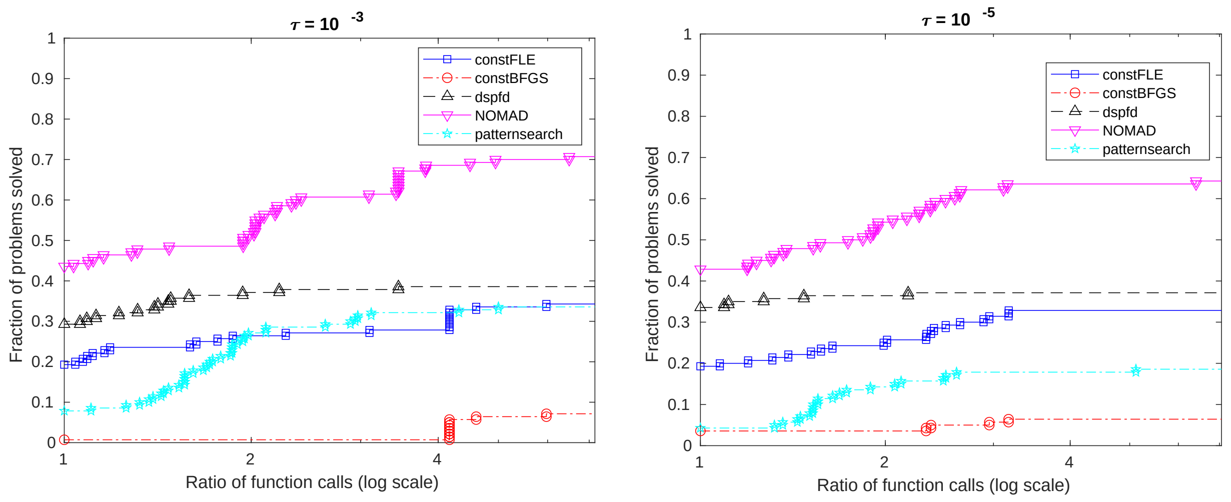


Figure 7: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`*, and* `patternsearch`*. The test set contains 14 bound and linearly constrained problems non-smooth problems.*

demonstrating superior performance.

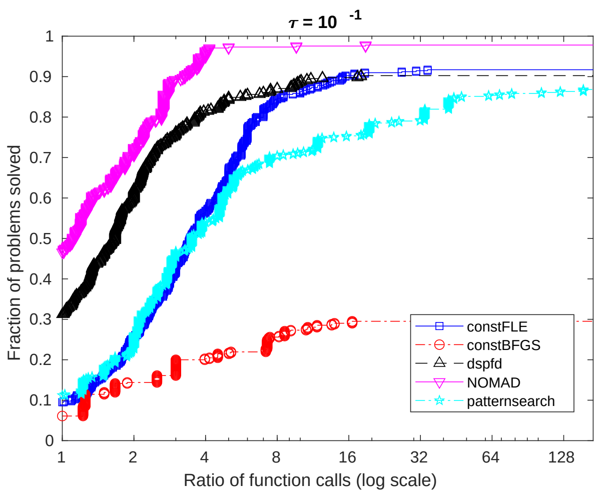
7 Conclusions

We have proposed an instance of the `Full-Low Evaluation` framework tailored to the presence of bound and linear constraints, by combining projected BFGS steps with probabilistic direct-search steps within approximate tangent cones. The result method is equipped with similar guarantees than in the unconstrained case. In addition, its performance has been validated in linearly constrained problems with smooth, non-smooth, and noisy objectives. Those experiments overall suggest that our algorithm is able to get the best of both worlds, and improve over existing algorithms that do not combine `Full-Eval` and `Low-Eval` steps.

Other variants of the `Full-Low Evaluation` framework may be able to improve on our current implementation. In particular, one could rely on trust-region steps as `Full-Eval`, while one- or two-point feedback feasible approaches that have been proposed more generally in the convexly constrained setting. In fact, extending the `Full-Low Evaluation` framework to nonlinear, convex constraints is a natural continuation of our work, which may benefit from existing results in feedback methods as well as mature theory regarding projected gradient techniques.

Acknowledgments

This work is partially supported by the U.S. Air Force Office of Scientific Research (AFOSR) award FA9550-23-1-0217, and by Agence Nationale de la Recherche through program ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).



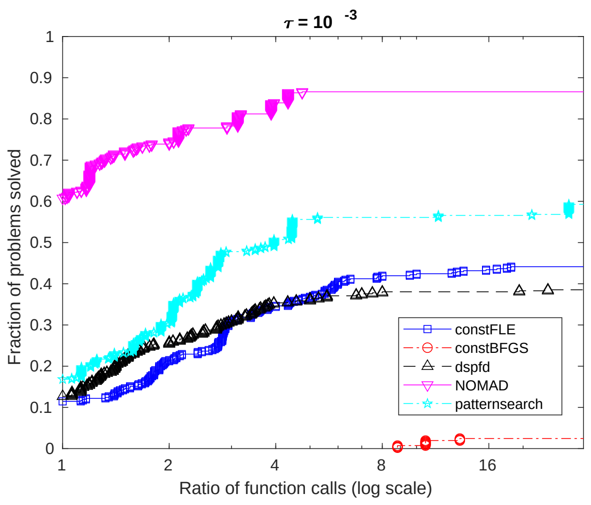


Figure 8: Performance profiles with $\tau = 10^{-1}, 10^{-3}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`, and `patternsearch`. The test set contains 41 noisy bound constrained problems.

Declarations

Conflict of interest: All authors declare that they have no conflict of interest.
Data Availability: The data used to support the findings is publicly available.

References

[1] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel. Two decades of blackbox optimization applications. *EURO J. Comput. Optim.*, 9:100011, 2024.

[2] C. Audet and J. E. Dennis Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM J. Optim.*, 20:445–472, 2009.

[3] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, Cham, Switzerland, 2017.

[4] C. Audet and J. E. Dennis Jr. Analysis of generalized pattern searches. *SIAM J. Optim.*, 13:889–903, 2002.

[5] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.*, 17:188–217, 2006.

[6] C. Audet, S. L. Digabel, V. R. Montplaisir, and C. Tribes. Nomad version 4: Nonlinear optimization with the mads algorithm. *arXiv preprint arXiv:2104.11627*, 2021.

[7] A. Beck and N. Hallak. On the convergence to stationary points of deterministic and randomized feasible descent directions methods. *SIAM J. Optim.*, 30:56–79, 2020.

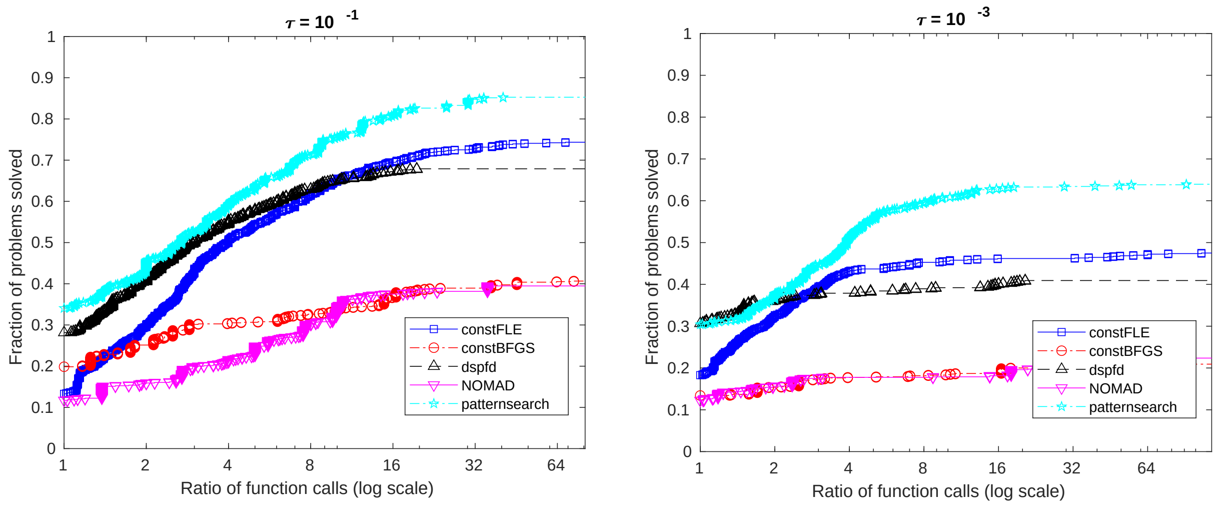


Figure 9: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`, and `patternsearch`. The test set contains 76 noisy problems with general linear constraints.

[8] A. S. Berahas, R. H. Byrd, and J. Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.*, 29:965–993, 2019.

[9] A. S. Berahas, O. Sohab, and L. N. Vicente. Full-low evaluation methods for derivative-free optimization. *Optim. Methods Softw.*, 38:386–411, 2022.

[10] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, third edition, 2016.

[11] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. General considerations. *IMA J. Appl. Math.*, 6:76–90, 1970.

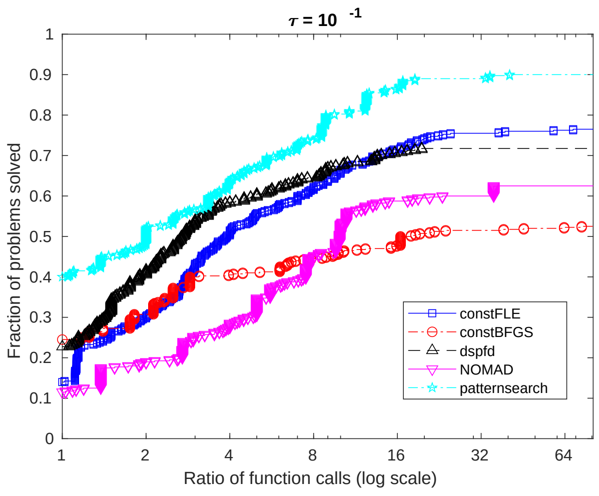
[12] F. H. Clarke. *Optimization and Nonsmooth Analysis.* John Wiley & Sons, New York, 1983. Reissued by SIAM, Philadelphia, 1990.

[13] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods.* MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[14] A. R. Conn, K. Scheinberg, and L. N. Vicente. Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points. *SIAM J. Optim.*, 20: 387–415, 2009.

[15] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization.* MPS-SIAM Series on Optimization. SIAM, Philadelphia, 2009.

[16] A. L. Custódio, K. Scheinberg, and L. N. Vicente. Methodologies and software for derivative-free optimization. In T. Terlaky, M. F. Anjos, and S. Ahmed, editors, *Chapter 37 of*



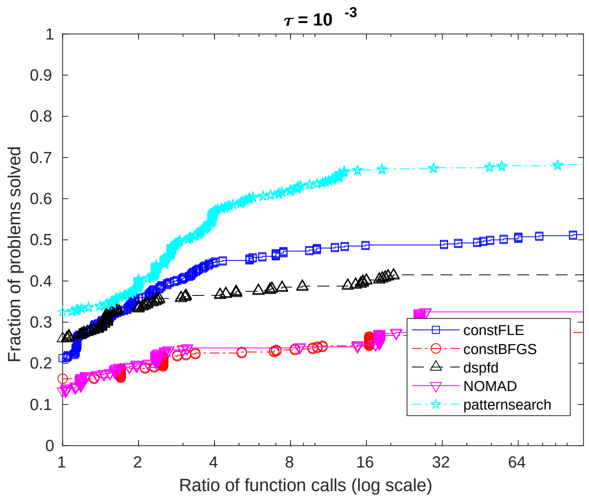


Figure 10: Performance profiles with $\tau = 10^{-3}, 10^{-5}$ of the 5 solvers: constFLE, constBFGS, dspfd, `NOMAD`*, and* `patternsearch`*. The test set contains 40 noisy problems with at least one inequality constraint.*

Advances and Trends in Optimization with Engineering Applications, MOS-SIAM Book Series on Optimization. SIAM, Philadelphia, 2017.

[17] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213, 2002.

[18] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13: 317–322, 1970.

[19] K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and I. Manoussakis, editors, *Combinatorics and Computer Science: 8th Franco-Japanese and 4th Franco-Chinese Conference, Brest, France, July 3–5, 1995 Selected Papers*, pages 91–111. Springer, 1996.

[20] D. Goldfarb. A family of variable-metric methods derived by variational means. *Math. Comp.*, 24:23–26, 1970.

[21] S. Gratton, P. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.*, 21:873–894, 2011.

[22] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic feasible descent for bound and linearly constrained problems. *Comput. Optim. Appl.*, 72: 525–559, 2019.

[23] T. M. Inc. Global optimization toolbox, user's guide. *Version 3.3*, Oct 2014.

[24] J. Jahn. *Introduction to the Theory of Nonlinear Optimization.* Springer Nature, 1994.

[25] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.*, 45:385–482, 2003.

[26] T. G. Kolda, R. M. Lewis, and V. Torczon. Stationarity results for generating set search for linearly constrained optimization. *SIAM J. Optim.*, 17:943–968, 2007.

[27] J. Larson, M. Menickelly, and S. Wild. Derivative-free optimization methods. *Acta Numer.*, 28:287–404, 2019.

[28] S. Le Digabel and S. M. Wild. A taxonomy of constraints in black-box simulation-based optimization. *Optim. Eng.*, 25:1125–1143, 2024.

[29] R. M. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM J. Optim.*, 9:1082–1099, 1999.

[30] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM J. Optim.*, 10:917–941, 2000.

[31] R. M. Lewis, A. Shepherd, and V. Torczon. Implementing generating set search methods for linearly constrained minimization. *SIAM J. Sci. Comput.*, 29:2507–2530, 2007.

[32] L. Lukšan and J. Vlcek. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report 798, Institut of Computer Science, Academy of Sciences of the Czech Republic, 2000.

[33] T. Matheiss and D. S. Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Math. Oper. Res.*, 5:167–185, 1980.

[34] J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20:172–191, 2009.

[35] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, Berlin, second edition, 2006.

[36] C. J. Price and I. D. Coope. Frames and grids in unconstrained and linearly constrained optimization: a nonsmooth approach. *SIAM J. Optim.*, 14:415–438, 2003.

[37] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *J. Global Optim.*, 56:1247–1293, 2013.

[38] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, 24:647–656, 1970.

[39] H.-J. M. Shi, M. Q. Xuan, F. Oztoprak, and J. Nocedal. On the numerical performance of finite-difference-based methods for derivative-free optimization. *Optim. Methods Softw.*, 38:289–311, 2023.

[40] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optim.*, 7:1–25, 1997.

[41] L. N. Vicente and A. L. Custódio. Analysis of direct searches for discontinuous functions. *Math. Program.*, 133:299–325, 2012.

Appendix A List of Problems

The complete list of test problems is presented in Tables 1 through 7. The columns represent various parameters of the problems: Size refers to the dimension of the problem, Bounds indicates the number of bound constraints, LE stands for the number of equality constraints, LI represents the number of inequality constraints, and Func. denotes the number of partial functions in the minimax problem.

Name	Size	Bounds
chenhark	10	10
explin	12	24
harkerp2	10	10
hatfldb	4	5
hs3	2	1
hs4	2	2
maxlika	8	16
ncvxbqp1	10	20
oslbqp	8	11
pspdoc	4	1
weeds	3	4
camel6	2	4
eg1	3	4
cvxbqp1	10	20
explin2	12	24
hart6	6	12
himmelp1	2	4
hs2	2	1
hs3mod	2	1
hs5	2	4
mccormck	10	20
ncvxbqp2	10	20
palmer1a	6	2
palmer4a	6	2
qrtquad	12	12
simbqp	2	2
yfit	3	1
expquad	12	12
hatflda	4	4
hs1	2	1
hs38	4	8
hs45	5	10
hs110	10	20
logros	2	2
mdhole	2	1
ncvxbqp3	10	20
palmer2b	4	2
palmer5b	9	2
probpenl	10	20
s368	8	16
sineali	20	40

Table 1: Bound constrained problems.

Name	Size	Bounds	LE	Name	Size	Bounds	LE
aug2d	24	0	9	genhs28	10	0	8
bt3	5	0	3	hs9	2	0	1
hs28	3	0	1	hs48	5	0	2
hs49	5	0	2	hs50	5	0	3
hs51	5	0	3	hs52	5	0	3
cvxqp2	10	20	2	cvxqp1	10	20	5
fccu	19	19	8	degenlpa	20	40	15
hs41	4	8	1	hong	4	8	1
hs54	6	12	1	hs53	5	10	3
hs62	3	6	1	hs55	6	8	6
ncvxqp1	10	20	5	hs112	10	10	3
ncvxqp3	10	20	5	ncvxqp2	10	20	5
ncvxqp5	10	20	2	ncvxqp4	10	20	2
fits	10	10	6	ncvxqp6	10	20	2
portfl2	12	24	1	portfl1	12	24	1
portfl4	12	24	1	portfl3	12	24	1
reading2	9	14	4	portfl6	12	24	1
sosqp2	20	40	11	sosqp1	20	40	11

Table 2: Linear equality constrained problems.

Name	Size	Bounds	LE	LI
avgasa	8	16	0	10
biggsc4	4	8	0	7
dualc2	7	14	1	228
expfitb	5	0	0	102
hatfldh	4	8	0	7
hs118	15	30	0	17
hs21mod	7	8	0	1
hs268	5	0	0	5
hs35mod	3	4	0	1
hs36	3	6	0	1
hs44	4	4	0	6
hs76	4	4	0	3
hs86	5	5	0	10
lsqfit	2	1	0	1
oet3	4	0	0	1002
simpllpa	2	2	0	2
sipow1	2	0	0	2000
sipow2	2	0	0	2000
sipow3	4	0	0	2000
stancmin	3	3	0	2
tfi2	3	0	0	101
avgasb	8	16	0	10
dualc1	9	18	1	214
dualc5	8	16	1	277
expfita	5	0	0	22
expfitc	5	0	0	502
hs105	8	16	0	1
hs21	2	4	0	1
hs24	2	2	0	3
hs35	3	3	0	1
hs37	3	6	0	2
hs44new	4	4	0	6
hubfit	2	1	0	1
oet1	3	0	0	1002
pentagon	6	0	0	15
simpllpb	2	2	0	3
sipow1m	2	0	0	2000
sipow2m	2	0	0	2000
sipow4	4	0	0	2000
zecevic2	2	4	0	2

Table 3: Linear inequality constrained problems.

Name	Pen. Const
avgasa	LI
biggsc4	LI
dualc2	LE & LI
hatfldh	LI
hs118	LI
hs21mod	LI
hs35mod	LI
hs36	LI
hs44	LI
hs76	LI
hs86	LI
lsqfit	LI
simpllpa	LI
stancmin	LI
avgasb	LI
dualc1	LE & LI
dualc5	LE & LI
hs105	LI
hs21	LI
hs24	LI
hs35	LI
hs37	LI
hs44new	LI
hubfit	LI
simpllpb	LI
zecevic2	LI
cvxqp2	LE
fccu	LE
hs41	LE
hs54	LE
hs62	LE
ncvxqp1	LE
ncvxqp3	LE
ncvxqp5	LE
odfits	LE
portfl2	LE
portfl4	LE
reading2	LE
sosqp2	LE
cvxqp1	LE
degenlpa	LE
hong	LE
hs53	LE
hs55	LE
hs112	LE
ncvxqp2	LE
ncvxqp4	LE
ncvxqp6	LE
portfl1	LE
portfl3	LE
portfl6	LE
sosqp1	LE

Table 4: Non-smooth bound constrained problems.

Name	Pen. Const
dualc2	LI
dualc1	LI
dualc5	LI
cvxqp2	B
cvxqp2	1/2 B
fccu	B
fccu	1/2 B
hs41	B
hs41	1/2 B
hs54	B
hs54	1/2 B
hs62	B
hs62	1/2 B
ncvxqp1	B
ncvxqp1	1/2 B
ncvxqp3	B
ncvxqp3	1/2 B
ncvxqp5	B
ncvxqp5	1/2 B
odfits	B
odfits	1/2 B
portfl2	B
portfl2	1/2 B
portfl4	B
portfl4	1/2 B
reading2	B
reading2	1/2 B
sosqp2	B
sosqp2	1/2 B
cvxqp1	B
cvxqp1	1/2 B
degenlpa	B
degenlpa	1/2 B
hong	B
hong	1/2 B
hs53	B
hs53	1/2 B
hs55	B
hs55	1/2 B
hs112	B
hs112	1/2 B
ncvxqp2	B
ncvxqp2	1/2 B
ncvxqp4	B
ncvxqp4	1/2 B
ncvxqp6	B
ncvxqp6	1/2 B
portfl1	B
portfl1	1/2 B
portfl3	B
portfl3	1/2 B
portfl6	B
portfl6	1/2 B
sosqp1	B
sosqp1	1/2 B

Table 5: Non-smooth linear equality constrained problems.

Name	Pen. Const
avgasa	B
avgasa	1/2 B
biggsc4	B
biggsc4	1/2 B
dualc2	B
dualc2	LE
hatfldh	B
hatfldh	1/2 B
hs118	B
hs118	1/2 B
hs21mod	B
hs21mod	1/2 B
hs35mod	B
hs35mod	1/2 B
hs36	B
hs36	1/2 B
hs44	B

Name	Pen. Const
hs44	1/2 B
hs76	B
hs76	1/2 B
hs86	B
hs86	1/2 B
lsqfit	B
lsqfit	1/2 B
simpllpa	B
simpllpa	LE
stancmin	B
stancmin	1/2 B
avgasb	B
avgasb	1/2 B
dualc1	B
dualc1	LE
dualc5	B
dualc5	LE
hs105	B

Name	Pen. Const
hs105	1/2 B
hs21	B
hs21	1/2 B
hs24	B
hs24	1/2 B
hs35	B
hs35	1/2 B
hs37	B
hs37	1/2 B
hs44new	B
hs44new	1/2 B
hubfit	B
hubfit	1/2 B
simpllpb	B
simpllpb	1/2 B
zecevic2	B
zecevic2	1/2 B

Table 6: Non-smooth linear inequality constrained problems.

Name	Size	Func.	Bounds	LE	LI
MAD1	2	3	0	0	1
MAD2	2	3	0	0	1
MAD4	2	3	0	0	1
MAD5	2	3	0	0	1
PENTAGON	6	3	0	0	15
MAD6	7	163	1	1	7
Wong 2	10	6	0	0	3
Wong 3	20	14	0	0	4
MAD8	20	38	10	0	0
BP filter	9	124	0	0	4
HS114	10	9	20	1	4
Dembo 3	7	13	14	0	2
Dembo 5	8	4	16	0	3
Dembo 7	16	19	32	0	1

Table 7: Non-smooth minimax linearly constrained problems.