New subspace method for unconstrained derivative-free optimization

Morteza Kimiaei
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: kimiaeim83@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~kimiaei`

Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~neum`

Parvaneh Faramarzi
Department of Mathematics, Faculty of Science, Razi University
Kermanshah, Iran
email: p.faramarzi2018@gmail.com

Abstract. This paper defines an efficient subspace method, called `SSDFO`, for unconstrained derivative-free optimization problems where the gradients of the objective function are Lipschitz continuous but only exact function values are available. `SSDFO` employs line searches along directions constructed on the basis of quadratic models. These approximate the objective function in a subspace spanned by some previous search directions. A worst case complexity bound on the number of iterations and function evaluations is derived for a basic algorithm using this technique. Numerical results for a practical variant with additional heuristic features show that, on the unconstrained `CUTEst` test problems, `SSDFO` has superior performance compared to the best solvers from the literature.

Keywords. Unconstrained derivative-free optimization; subspace technique; line search approach; complexity

2000 AMS Subject Classification: primary 90C56.

Acknowledgment The first author acknowledges the financial support of the Doctoral Program *Vienna Graduate School on Computational Optimization* (*VGSCO*) funded by the Austrian Science Foundation under Project No W1260-N35. We thank Christophe Tribes and Sébastien Le Digabel for their helps in running `NOMAD` with various options and two referees for the useful comments.

December 24, 2022

1 Introduction

In this paper, we discuss a new subspace technique for the unconstrained derivative-free optimization (DFO) problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbb{R}^n, \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with Lipschitz continuous gradients, accessible through exact function values. In particular, gradients, Lipschitz constants, or other information about the structure of f are not available. However, a numerical gradient approximation can be estimated by a finite difference method. We denote by $\widetilde{g} := \widetilde{g}(x)$ the estimated gradient at x and by $g_p(x) \approx g(x)^T p$ the approximate directional derivative in the direction p and at x. For DFO problems and the related algorithms for solving these problems see Audet & Hare [4] and Conn et al. [11].

1.1 Related work

There are many different algorithms for solving the problem (1). They can be divided into deterministic and randomized. An excellent reference for the deterministic methods is Conn et al. [11]. Recently, Larson et al. [22] gave a comprehensive discussion of both deterministic and stochastic methods. A comprehensive treatment of DFO solvers can be found in [19, 20, 33]. We list a large number of DFO solvers in Table 1 and compare our new solver with them in Section 5.

kind of algorithm	solvers
model-based	`BCDFO` [15], `UOBYQA` [32], `DFOTR` [6]
direct search	`NOMAD` [2], `BFO` [31], `NMSMAX` [17], `DSPFD` [14], `ADSMAX` [17], `NELDER` [18]
line search	`SDBOX` [26], `VRBBO` [20], `VRDFON` [19], `FMINUNC` [27]
evolution strategy	`LMMAES` [25], `FMAES` [7]

Table 1: A list of DFO solvers from the literature

Model-based algorithms construct quadratic model functions to approximate the nonlinear objective function by fitting or interpolation. These models have restricted trust regions to avoid large steps where the models are no longer reliable. If the agreement between the model function and the objective function is close to one, the trial point is accepted as the new point and the trust region is expanded or remains unchanged; otherwise, the trial point is discarded and the trust region is reduced. These solvers have excellent behavior for problems with small dimensions, but become very slow in large dimension due to the need for a lot of sample points for the model construction.

Direct search and line search algorithms search iteratively along appropriate directions to reduce the function value. If the function value is decreased at the trial point, the evaluated trial point is accepted as the new best point and its corresponding step size is expanded or remains unchanged; otherwise, the trial point is discarded and its corresponding step size is reduced. Line search algorithms, unlike direct search algorithms, use paths along the search directions to find a trial point with lower function value, accepted as the new best point.

Matrix adaptation evolution strategy algorithms iteratively proceed through three phases. The mutation phase searches along a finite number of mutation directions to generate mutation points. Each mutation direction is the product of the direction sampled from a standard normal distribution and a heuristically determined affine scaling matrix. Each mutation point is the sum of the previously accepted point and the mutation direction, scaled by a step size called the mutation step size. Then the selection phase sorts the function values at the mutation points in ascending order and selects the distribution directions and mutation directions based on this sorting. These are needed for the recombination phase, which first calculates recombination directions as a weighted average of the selected mutation directions, then evaluates the recombination point, which is the sum of the previous recombination point and the recombination direction, scaled by a recombination step size based on the information from the selection phase. The recombination phase also updates the mutation step size, recombination step size, and the new affine scaling matrix.

1.2 Complexity and global convergence of DFO methods

It is well known that optimization methods can get stuck in almost flat regions, and that the gradient for a local minimizer in finite precision arithmetic does not vanish. For a given threshold $\varepsilon > 0$, a local DFO algorithm finds, with $N(\varepsilon)$ function evaluations, an ε-approximate stationary point x with

$$f(x) \le \sup\{f(y) \mid y \in \mathbb{R}^n, \quad f(y) \le f(x_0), \quad \text{and} \quad \|g(y)\| \le \varepsilon\} \tag{2}$$

using the Euclidean norm and under the assumptions that
(A1) the level set $L(x_0) := \{x \in \mathbb{R}^n \mid f(x) \le f(x_0), \ x_0 \in \mathbb{R}^n\}$ of f at the starting point x_0 is compact,
(A2) there is a convex neighbourhood Ω of $L(x_0)$ such that in Ω, the objective function f is continuously differentiable and its gradient g is Lipschitz continuous, i.e., there exists a constant $L > 0$ such that

$$\|g(x) - g(y)\| \le L\|x - y\|, \quad \text{for all} \ \ x, y \in \Omega. \tag{3}$$

It is well-known that (A1) and (A2) imply that some global minimizer $\widehat{x}$ exists, and

$$\widehat{f} = f(\widehat{x}) := \inf_{\ell \ge 0} f(x_\ell) \tag{4}$$

is attained there, hence finite. Moreover, (A2) implies

$$\|f(x+s) - f(x) - g(x)^T s\| \le \frac{L}{2}\|s\|^2, \quad \text{for } x, x+s \in \Omega. \tag{5}$$

The criterion $\|g(x)\| \le \varepsilon$ in (2) cannot be verified under our assumptions. Instead, our algorithm tries to find a point whose exact function value is at least as good as the exact function value of an ε-approximate stationary point, for a threshold $\varepsilon = O(\sqrt{\omega})$, where ω is a bound on the error in the function values (the source of this error is that the finite difference approaches are sensitive in the approximation of the gradient due to rounding errors). If f has only one stationary point and ω is sufficiently small, this guarantees that the point found is arbitrarily close to the minimizer. Indeed, if the gradients are small only near a global optimizer, a point near the local optimizer is found. If some iterate passes close to a non-global local optimizer or a saddle point, the algorithm may leave its neighbourhood. In this case, only a variant of the algorithm with restarts would lead to convergence to a point with a small gradient.

1.2.1 DFO methods with exact f

Larson et al. [22, Table 8.1] and Kimiaei & Neumaier [20, Tables 1–3] summarized known complexity results for DFO methods with exact function values. Under the assumptions (A1) and (A2), DFO algorithms (line search, direct search, and trust region) need $\mathcal{O}(\varepsilon^{-2})$ function evaluations for the general case, $\mathcal{O}(\varepsilon^{-1})$ function evaluations for the convex case, and $\mathcal{O}(\log \varepsilon^{-1})$ function evaluations for the strongly convex case to obtain a point x guaranteeing (2).

1.2.2 Finite difference line search methods

Berahas et al. [8] discuss complexity bounds for nonconvex, convex, and strongly convex cases for the accuracy of two different gradient estimates. To estimate the gradient, he discussed three bounds on the number of function evaluations, step sizes, and the exact gradient norm for all existing randomized and deterministic methods, such that the reduction in the error of the gradient norm is asymptotically bounded by the reduction in the exact gradient norm. For global convergence of derivative-free line search methods, see Xie et al. [35] and Shi et al. [34].

1.3 Derivative-free line search methods with exact f

Derivative-free line search methods with exact f can be classified according to whether or not derivative are approximated by finite differences. These approximations may be used not only to compute quasi-Newton directions but also in a decrease condition to check whether or not the function values are sufficiently decreased. In the latter case, only approximates to directional derivatives are needed.

For example, `FMINUNC` [27] approximates gradients by the finite difference method of Conn et al. [10, Section 8.4.3] and performs Wolfe line search methods along quasi-Newton directions. This solver was found efficient and moderately robust for low and medium dimensional problems (cf. [20]) in the noiseless case, but not in the presence of significant noise (cf. [19]), since the traditional difference method leads to misleading information and gives a poor quasi-Newton direction and the approximate Wolfe line search condition cannot be satisfied. Like FMINUNC, VRBBO approximates gradients by the finite difference method of Conn et al. [10, Section 8.4.3].

Unlike FMINUNC, SDBOX [26] and VRBBO [20] do not use the approximate directional derivatives inside the line search condition, because the traditional finite difference method behaves poorly in the presence of significant noise. To check whether or not a reduction of the function value is found, instead of directional derivatives, VRBBO and SDBOX use a forcing function, non-decreasing and positive. Direct search methods also use forcing functions instead of the approximate directional derivatives.

1.4 Finite difference line search methods

Derivative-free line search methods often use finite difference methods to approximate gradients to perform approximate Armijo or Wolfe line searches along quasi-Newton directions. The traditional approximate Wolfe line search methods with exact function values and approximate gradients are efficient only in the noiseless case. More recent methods use the knowledge of the level of the noise and use noisy variants of approximate Wolfe line search methods to be efficient in the presence of significant noise.

Suppose that $\tau_1 \gg 1$, $\tau_2 \in (0,1)$, and let ω_f be the noise level. For a given direction $p \in \mathbb{R}^n$ with $\|p\| = 1$, Moré & Wild [28] try to find an interval for the step size t satisfying

$$|\Delta(t)| \geq \tau_1 \omega_f \quad \text{and} \quad |f(x \pm tp) - f(x)| \leq \tau_2 \max\{|f(x)|, |f(x \pm tp)|\}, \tag{6}$$

but without guarantee. The goal was to produce step sizes t that are hopefully neither too small nor too large. Accordingly, Berahas et al. [9] estimated noise in the same way as Hamming [16] did. This estimate was added to the Armijo line search method and used to estimate the gradient by a finite difference technique. However, the effects of noise in BFGS updating have not been studied.

Recently, Xie et al. [35] showed that the Wolfe conditions can be satisfied with the exact function value and gradient when these conditions are satisfied with the inexact function and gradient and the exact gradient is not small. They modified the BFGS method with the Wolfe line search method without estimating noise and showed that this method converges to a neighborhood of the solution while taking into account the effects of noise in the BFGS updating.

More recently, Shi et al. [34] suggested two finite difference interval algorithms, requiring the knowledge of noise level. These algorithms balance truncation and measurement error in the finite difference approach.

1.5 Wolfe line search in exact arithmetic

The Wolfe line search method of Al-Baali & Fletcher [1] assumes in exact arithmetic. We therefore call the latter WLS-exact. It finds a step size that not only leads to a sufficient reduction in the function value, but is also close to a minimizer of the objective function with respect to the step size. Each iteration of `WLS-exact` is the scheme (S2) in [1, Section 2]. Another variant of the Wolfe line search can be found in Nocedal & Wright [30].

Al-Baali & Fletcher [1] assume that the objective function is bounded below, i.e., $f(\alpha) \geq \overline{f}$ for $\alpha \geq 0$. In practice, the unknown $\overline{f}$ is replaced by a threshold computed in line 2 of Algorithm 1 in Section 3 as in the `FMINUNC` [27] solver. When function values and gradients are known exactly, `WLS-exact` enforces the Wolfe conditions

$$f(x + \alpha p) \leq f(x) + \rho\alpha g(x)^T p, \tag{7}$$

$$|g(x + \alpha p)^T p| \leq -\sigma g(x)^T p, \tag{8}$$

where $\sigma > \rho > 0$, and restricts the step sizes to $(0, \mu]$ with

$$\mu := \frac{\overline{f} - f(x)}{\rho g(x)^T p}. \tag{9}$$

It finds an interval with suitable step sizes that satisfy the sufficient decrease condition (7), and then finds a desired step size within that interval that satisfies the curvature condition (8). Interpolation and extrapolation are used to generate step sizes.

Let a_j (for $j = 1, 2, \cdots, \infty$) be a sequence of the step sizes generated by `WLS-exact` satisfying the condition (7) but not the condition (8), let b_j (for $j = 1, 2, \cdots, \infty$) be a sequence of step sizes generated by `WLS-exact` violating either (7) or $f(x + b_j p) \geq f(x + a_j p)$ or both, and let α_j (for $j = 1, 2, \cdots, \infty$) be a sequence of the trial step sizes generated by `WLS-exact`. Then `WLS-exact` finds the interval

$$I(a_j, b_j) := \begin{cases} (a_j, b_j) & \text{if } a_j < b_j, \\ (b_j, a_j) & \text{if } a_j > b_j, \end{cases}$$

containing the acceptable points or the points that

$$f(\alpha_j) \leq \overline{f}. \tag{10}$$

For $j = 1$, $a_1 = 0$, $b_1 = \infty$, and $0 < \alpha_1 \leq \mu$. Let $0 < \tau_1 \leq \tau_2 \leq \frac{1}{2}$, $1 < \tau_3 \leq \tau_4$, $0 < \tau_5 \leq \tau_6 \leq \frac{1}{2}$, and $a_j, b_j > 0$ for $j \geq 1$. In the jth iteration, `WLS-exact` performs the following steps:
Step 1: If the condition (10) holds, `WLS-exact` ends.
Step 2: If α_j satisfies the condition

$$f(x + \alpha_j p) > f(x) + \alpha_j \rho g(x)^T p \quad \text{or} \quad f(x + \alpha_j p) \geq f(x + a_j p), \tag{11}$$

(S2a) the $(j + 1)$th step size

$$\alpha_{j+1} \in T(a_j, \alpha_j) := \begin{cases} [a_j + \tau_1(\alpha_j - a_j), \alpha_j - \tau_2(\alpha_j - a_j)] & \text{if } a_j < \alpha_j, \\ [\alpha_j + \tau_2(a_j - \alpha_j), a_j - \tau_1(a_j - \alpha_j)] & \text{if } \alpha_j < a_j \end{cases} \tag{12}$$

is obtained;
(S2b) the $(j + 1)$th interval $I(a_{j+1}, b_{j+1})$ is updated by setting

$$a_{j+1} = a_j \quad \text{and} \quad b_{j+1} = \alpha_j. \tag{13}$$

Step 3: If α_j does not satisfy (11), then
(S3a) $g(x + \alpha_j p)^T p$ is computed;
(S3b) if the condition (8) holds, `WLS-exact` ends;
(S3c) the $(j + 1)$th step size

$$\alpha_{j+1} \in E(a_j, \alpha_j, b_j) := \begin{cases} [\min(\tau_3\alpha_j, \mu), \min(\tau_4\alpha_j, \mu)] & \text{if } a_j < \alpha_j < b_j = \infty, \\ [\alpha_j + \tau_5(b_j - \alpha_j), b_j - \tau_6(b_j - \alpha_j)] & \text{if } a_j < \alpha_j < b_j \le \mu, \\ [b_j + \tau_6(\alpha_j - b_j), \alpha_j - \tau_5(\alpha_j - b_j)] & \text{if } b_j < \alpha_j < a_j \le \mu \end{cases} \tag{14}$$

is computed;
(S3d) the $(j + 1)$th interval $I(a_{j+1}, b_{j+1})$ is updated by setting

$$a_{j+1} = \alpha_j \quad \text{and} \quad b_{j+1} = b_j. \tag{15}$$

In (12) and (14), α_{j+1} can be found within the defined intervals by minimizing quadratic/cubic polynomial interpolations; for more details see [1, Section 3].

1.1 Proposition. *[1, Theorem 2.2]*
Given $\sigma \ge \rho > 0$ and the finite threshold value $\overline{f}$, one of the following statements is valid:
(i) `WLS-exact` ends with $f(x + \alpha_j p) < \overline{f}$ for some $\alpha_j \in (0, \mu]$. Here μ comes from (9).
(ii) `WLS-exact` ends with α_j satisfying (7) and (8).
(iii) `WLS-exact` fails, each interval $I(a_j, b_j)$, $j = 1, 2, \cdots, \infty$, contains an interval of acceptable points and there exists a limit point $\widehat{\alpha} \in I(a_j, b_j)$, for all j, such that, for sufficiently large j, a_j are monotonically increasing, $\lim\limits_{j\to\infty} a_j = \widehat{\alpha}$ and b_j are monotonically decreasing, $\lim\limits_{j\to\infty} b_j = \widehat{\alpha}$, and $\widehat{\alpha}$ is an acceptable point as in (ii).
In particular, if $\sigma > \rho > 0$, (iii) cannot occur and `WLS-exact` ends.

1.6 Overview of the new method

This paper introduces `SSDFO`, a new deterministic subspace solver for unconstrained noiseless DFO problems. `SSDFO` may be understood as an efficient improvement of a finite-difference variant `DLMBOPT` of the gradient-based solver `LMBOPT` recently proposed by Kimiaei et al. [21].

`SSDFO` computes new directions in two ways by an improved subspace technique, which is new, or an adaptation of the limited memory quasi-Newton technique of Kimiaei et al. [21] for DFO. In each iteration, `SSDFO` determines the step sizes by performing `WLS`, an approximate version of `WLS-exact`, using the exact function values and approximates the gradients $\widetilde{g}$ by the finite difference method of Conn et al. [10, Section 8.4.3] and the directional derivative $\widetilde{g}_p$ in the direction p by a cheap finite difference method, overcoming rounding errors that occur in finite precision arithmetic.

`SSDFO` is available at `https://github.com/GS1400/SSDFO`.

1.6.1 `DLMBOPT`

`LMBOPT` (Kimiaei et al. [21]) is a limited memory method for bound-constrained optimization using exact gradients, with minimal memory requirements. `LMBOPT` is an active set strategy with two phases. In the first phase, `LMBOPT` finds the set of optimal active bound constraints by constructing a face including a stationary point of the problem. In the second phase, `LMBOPT` solves approximately an unconstrained subproblem by a regularized conjugate gradient technique or a limited memory quasi-Newton technique to explore the face of the feasible region. To solve this subproblem, these two techniques use an improved Goldstein line search method to find step sizes. This line search method finds initial step size heuristically and takes exact directional derivative as input without any more exact directional derivatives or exact gradients.

`LMBOPT` becomes a DFO solver if approximate gradients are obtained by the finite difference approach of Conn et al. [10, Section 8.4.3]. We call this DFO solver `DLMBOPT`, for discrete `LMBOPT`. The conjugate gradient directions become poor approximate conjugate gradient directions. Thus `DLMBOPT` itself is a poor solver (see Section 5).

1.6.2 `SSDFO`

`SSDFO` uses the following three techniques from `DLMBOPT`:
Enforcing approximate angle condition: We ensure that our search direction p at a point x satisfies the approximate bounded angle condition

$$\frac{g_p(x)}{\|\widetilde{g}(x)\|\|p\|} \le -\Delta_{\rm a} < 0, \tag{16}$$

where $g_p(x)$ is an approximate of $g(x)^T p$, i.e., the angle between the direction p and the approximate gradient $\widetilde{g}(x)$ at the point x obtained from a finite difference method remains bounded away from 90°. Here $0 < \Delta_{\rm a} < 1$ is a given threshold for the approximate angle condition (16).
Heuristic step size: When `WLS-inexact` fails in finite precision arithmetic, `SSDFO` uses heuristic step sizes, resulting in an implemented version of `WLS-inexact`. Our goal here differs from `DLMBOPT`, which uses these heuristic step sizes as the initial step sizes for an improved Goldstein line search. Although our algorithm may move somewhat away from a minimizer in this case, it may approach a minimizer in the next few attempts. At the very least, this idea helps us avoid failure.
Limited memory quasi-Newton direction: We use the limited memory quasi-Newton direction (22) only when the new direction of `SSDFO` (discussed in Section 1.6.2) cannot be computable, meaning that a decrease in both the model function value and its gradient norm cannot be found.

The enhancements that make `SSDFO` competitive are:
New direction: We compute our new subspace direction by finding a solution to a linear problem in a subspace of the previous points and then solving a simple optimization problem to find, if possible, a decrease in both the model function value and its gradient norm.

New heuristic step size: To compute the approximate directional derivatives $g_p(x) \approx g(x)^T p$ in the direction p and at x, we suggest a new heuristic formula for computing the finite difference step sizes.
Computational advantage: Our implemented version of `WLS-inexact` is not costly even for problems in high dimensions, since one function evaluation is required in each iteration to estimate the directional derivative and n function evaluations are required to estimate the gradient by the forward finite difference method of CONN et al. [10, Section 8.4.3] only in the last iteration.

Important differences between `SSDFO` and `DLMBOPT` are:
• `SSDFO` uses an implemented version of `WLS-inexact`, which differs from the improved Goldstein line search of `DLMBOPT`.
• `SSDFO` uses a new direction, while `DLMBOPT` uses an approximate conjugate gradient direction.

1.6.3 Complexity result

Assuming that discretization and rounding errors occur when the gradient is estimated by a finite difference method and the exact gradient norm is not small, we prove in the same way as XIE et al. [35] that the Wolfe line search conditions can be satisfied with both the exact gradient and the exact function value, while these conditions are numerically satisfied with both the exact function and the inexact gradient. In this case, we prove the complexity result for both the approximate Wolfe line search algorithm and the basic version of `SSDFO` for the general case regardless of the choice of search directions. The order of our complexity bound is consistent with that of BERAHAS et al. [8]. In contrast to XIE et al. [35], however, we enforce that our new limited memory direction enforces the approximate angle condition (16).

1.6.4 Numerical results

For all 506 unconstrained problems from the `CUTEst` [13] collection of test problems for optimization with 2 to 9000 variables, we say that a solver is robust if it has a high number of solved problems, and efficient if it has a low relative cost of function evaluations. We say a solver is competitive if it is robust or efficient. The efficiency and robustness of all solvers are studied by plotting the performance profiles (DOLAN & MORÉ [12]) and data profiles (MORÉ & WILD [29]) of these solvers. It is shown in Section 5 that on the unconstrained `CUTEst` noiseless test problems `SSDFO` is a competitive derivative-free line search-based algorithm using an improved subspace approach in low to high dimensions compared to the well-known DFO solvers listed in Table 1.

2 DLMBOPT

Here we describe only the part of DLMBOPT, needed to understand SSDFO. It is obtained from LMBOPT by simply replacing gradients with approximate gradients.

2.1 Finite differences

The FinDiff algorithm computes the gradient by the forward finite difference method of Conn et al. [10]. It requires n additional function evaluations for the approximate gradient, which can be costly in high dimensions.

FinDiff takes as input the current point x and its function value $f(x)$. It uses the tuning parameter $0 < \varepsilon_h < 1$ for adjusting the finite difference step size and returns the estimated gradient $\widetilde{g}(x)$ at x. As discussed in Subsection 1.1, FinDiff uses the traditional formula

$$h_i := \sqrt{\varepsilon_h}\,\mathrm{sign}\,(x_i)\max\{|x_i|, 1\}$$

to compute the step size and approximates the ith component of the gradient

$$g_{e_i}(x) := \frac{f(x + h_i e_i) - f(x)}{h_i}, \quad \text{for } i = 1, \ldots, n, \tag{17}$$

as in Conn et al. [10, Section 8.4.3], where e_i is the ith coordinate direction. Hence the approximate gradient

$$\widetilde{g}(x) = (\, g_{e_1}(x), g_{e_2}(x), \ldots, g_{e_n}(x)\,)^T$$

is obtained.

2.2 Subspace information

Let S_ℓ be the $n \times m$ matrix whose columns are (in the actual implementation a permutation of) the previous m search directions,

$$S_\ell := \{s_{\ell-m+1}, \ldots, s_\ell\} = \{x_{\ell-m+1} - x_{\ell-m}, \ldots, x_\ell - x_{\ell-1}\}, \tag{18}$$

and $Y_\ell \in \mathbb{R}^{n\times m}$ be the corresponding approximate gradient differences,

$$Y_\ell := \{y_{\ell-m+1}, \ldots, y_\ell\} = \{\widetilde{g}_{\ell-m+1} - \widetilde{g}_{\ell-m}, \ldots, \widetilde{g}_\ell - \widetilde{g}_{\ell-1}\}. \tag{19}$$

Here the approximate gradient

$$\widetilde{g}_\ell = \widetilde{g}(x_\ell) = (\, g_{e_1}(x_\ell), g_{e_2}(x_\ell), \ldots, g_{e_n}(x_\ell)\,)^T$$

is defined, where $g_{e_i}(x_\ell)$ is the ith approximate directional derivative in the ith coordinate direction e_i and at x_ℓ computed by (17). The matrices S_ℓ and Y_ℓ satisfy the quasi-Newton condition

$$B_\ell S_\ell = Y_\ell \tag{20}$$

and the matrix

$$H_\ell = S_\ell^T Y_\ell = S_\ell^T B_\ell S_\ell \tag{21}$$

is defined.

2.3 Subspace update

`updateSY` updates the ingredients of the subspace. It takes as input the matrices $S_{\ell-1}$, $Y_{\ell-1}$, and $H_{\ell-1}$ and the subspace dimension m and uses the tuning parameter $m_{\max}$ as an upper bound for m. It returns as output the updated matrices S_ℓ, Y_ℓ, H_ℓ and the subspace dimension m. If m does not exceed $m_{\max}$, `updateSY` replaces m by $m+1$; otherwise, `updateSY` sets $m=1$. `updateSY` updates the mth column of $S_{\ell-1}$ by

$$s_\ell := x_\ell - x_{\ell-1},$$

resulting it S_ℓ, the mth column of $Y_{\ell-1}$ by

$$y_\ell := \widetilde{g}_\ell - \widetilde{g}_{\ell-1},$$

resulting in Y_ℓ, and the mth row of H_ℓ by $y_\ell^T S_{\ell-1}$ and the mth column of H_ℓ by $S_{\ell-1}^T y_\ell$, resulting H_ℓ. In practice, we use S_ℓ, Y_ℓ, and H_ℓ to compute the subspace directions. `updateSY` enforces that H_ℓ becomes symmetric.

2.4 Limited memory quasi-Newton direction

We first describe the raw subspace direction corresponding to the limited memory quasi-Newton direction described in [21, Section 2.2] but with the approximate gradient (17). We denote by `YY` the componentwise squares of Y_ℓ, by `SS` the componentwise squares of S_ℓ , by $//$ the componentwise division, and denote

$$\overline{d} := \sum_{i=1}^{m} \mathtt{YY}_{:i} \quad \text{and} \quad \underline{d} := \sum_{i=1}^{m} \mathtt{SS}_{:i}.$$

Moreover we define the matrices

$$U_\ell := Y_\ell - D_\ell S_\ell, \quad D_\ell := \mathrm{diag}(d_\ell) \quad \text{with} \quad d_\ell := \sqrt{\overline{d}//\underline{d}}, \quad \Sigma_\ell := U_\ell^T S_\ell.$$

As in [21],

$$B_\ell := D_\ell + U_\ell \Sigma_\ell^{-1} U_\ell^T$$

is an approximate Hessian matrix, determined by unique S_ℓ and Y_ℓ and the quasi-Newton condition (20). We define the matrix

$$M_\ell := Y_\ell^T D_\ell^{-1} Y_\ell - H_\ell$$

and obtain the solution z_ℓ of the linear system

$$M_\ell z_\ell = U_\ell^T D_\ell^{-1} \widetilde{g}_\ell.$$

Here $\widetilde{g}_\ell$ defined above is estimated by the finite difference approach, but is exact in [21]. [21, Theorem 1] shows that

$$p_\ell := D_\ell^{-1}(U_\ell z_\ell - \widetilde{g}_\ell), \tag{22}$$

is the solution of $B_\ell p_\ell = -\widetilde{g}_\ell$.

3 A basic version of SSDFO

For the theoretical analysis we describe a simplified algorithm SSDFO-basic, an algorithm for unconstrained noiseless DFO problems. SSDFO-basic has calls to WLS-inexact along directions whose angle with the approximate gradient (17) is bounded away from $90°$. WLS-inexact calls FinDiff for computing the approximate gradients and FinDiffDir for computing the approximate directional derivatives. We prove complexity bounds for WLS-inexact and SSDFO-basic under the assumptions (A1) and (A2).

3.1 FinDiffDir

FinDiffDir computes the directional derivative $g_p(x+\alpha_j p)$ directly with one additional function evaluation. FinDiffDir takes as input the search direction p, the trial point $x+\alpha p$, and the current function value $f(x+\alpha p)$ and uses the tuning parameter $0<\varepsilon_h<1$ for adjusting the step size. It computes the step size

$$h := \varepsilon_h \|x+\alpha p\|/\|p\| \tag{23}$$

and returns as output the approximate directional derivative

$$g_p(x+\alpha p) = \frac{f(x+(\alpha+h)p) - f(x+\alpha p)}{h}.$$

3.2 WLS-inexact

WLS-inexact is an adaptation of the Wolfe line search algorithm described in Section 1.5 to account for the exact functions values and the inexact gradients. Given $\widetilde{\sigma} > \widetilde{\rho} > 0$, WLS-inexact enforces the approximate Wolfe conditions

$$f(x+\alpha p) \le f(x) + \widetilde{\rho}\alpha g_p(x), \tag{24}$$
$$|g_p(x+\alpha p)| \le -\widetilde{\sigma} g_p(x), \tag{25}$$

with the exact function and the estimated gradient. As defined in Section 1.6.1, $g_p(x)$ is the approximation of $g(x)^T p$. The sufficient descent (24) guarantees $f(x+\alpha p) < f(x)$ while the curvature condition (25) guarantees that α is not too far from a minimizer of the function f. WLS-inexact is an inexact version of WLS-exact, but with the approximate Wolfe conditions (24) and (25).

WLS-inexact uses FinDiff for computing the approximate gradients and FinDiffDir for computing the approximate directional derivatives. Therefore, WLS-inexact produces step sizes that are not too far from an approximate minimizer and speeds up the process of reaching an approximate minimizer.

WLS-inexact takes as input the current point x and returns as output the point $x+\alpha p$, its function value $f(x+\alpha p)$, and its estimated gradient $\widetilde{g}(x+\alpha p)$. It uses the tuning parameters $\widetilde{\sigma} > \widetilde{\rho} > 0$ and the finite threshold $\overline{f}$, which will be computed in line 2 of Algorithm 1.

We describe how to work `WLS-inexact` in the jth iteration. It finds the interval $I(a_j, b_j)$, containing the acceptable points or the points that (10) holds. For $j = 1$, $a_1 = 0$, $b_1 = \infty$, and $0 < \alpha_1 \le \widetilde{\mu}$, where

$$\widetilde{\mu} := \frac{\overline{f} - f(x)}{\widetilde{\rho} g_p(x)} \tag{26}$$

with $\widetilde{\rho} > 0$, which is from (25). Let $a_j, b_j > 0$ for $j \ge 1$. In the jth iteration, `WLS-inexact` performs the following steps:
Step 1: If the condition (10) holds, `WLS-inexact` ends.
Step 2: If α_j satisfies the condition

$$f(x + \alpha_j p) > f(x) + \alpha_j \rho g_p(x) \quad \text{or} \quad f(x + \alpha_j p) \ge f(x + a_j p), \tag{27}$$

(S2a) the $(j+1)$th step size $\alpha_{j+1} \in T(a_j, \alpha_j)$ is obtained by (12);
(S2b) for a given tuning parameter $tol > 0$ if the condition

$$|(\alpha_j - a_j) g_p(x + \alpha_j p)| < tol \tag{28}$$

holds, then `WLS-inexact` ends;
(S2c) the $(j+1)$th interval $I(a_{j+1}, b_{j+1})$ is updated by (13).
Step 3: If the condition (27) does not hold
(S3a) $g_p(x + \alpha_{j+1} p)$ is computed by `FinDiffDir`;
(S3b) if the approximate curvature condition (25) holds, `WLS-inexact` ends;
(S3c) $a_{j+1} = \alpha_j$ is chosen;
(S3d) if the condition

$$(b_j - a_j) g_p(x + \alpha_j p) < 0 \tag{29}$$

holds, the $(j+1)$th step size $\alpha_{j+1} \in E(a_j, \alpha_j, b_j)$ is computed by (14) and $b_{j+1} = b_j$ is chosen;
(S3e) if the condition (29) does not hold, the $(j+1)$th step size α_{j+1} is computed by (12) and $b_{j+1} = a_j$ is chosen.

The purpose of (28) is to eliminate the rounding error effect that occurs in finite precision arithmetic and may not allow the Wolfe conditions (24) and (25) to be satisfied.

3.3 Robustness of `WLS-inexact`

DFO solvers described and analyzed for exact function values are in practice always used with function values computed by finite precision arithmetic. Therefore, the computed function values $\widetilde{f}(x)$ at x are not exact but contaminated with rounding errors. Since finite difference approximations of the gradients are sensitive to rounding errors, we need to study their influence on `WLS-inexact`. The following result is a variant of results discussed in Berahas et al. [8, Section 2.1] for forward finite difference methods for estimating the gradient.

3.1 Theorem. *Assume that (A1) and (A2) hold, and let p be the search direction, and suppose that*

$$h\|p\| = \Theta(\sqrt{\omega}) \text{ with } \omega > 0. \tag{30}$$

Then the directional derivative approximation defined by

$$g_p(x) := h^{-1}(f(x+hp) - f(x)),$$

satisfies

$$|g_p(x) - g(x)^T p| = \mathcal{O}(\sqrt{\omega}\|p\|). \tag{31}$$

In particular, the approximate gradient estimated by `FinDiff` *satisfies* (31)*. Moreover,* (31) *even holds for*

$$\widetilde{g}_p(x) := h^{-1}\Big(\widetilde{f}(x+hp) - \widetilde{f}(x)\Big)$$

in place of $g_p(x)$ provided that

$$|\widetilde{f}(x) - f(x)| \le \omega, \quad \textit{for all } x \in \mathbb{R}^n. \tag{32}$$

Proof. By (5) and (32), we have

$$\begin{aligned} h|\widetilde{g}_p(x) - g(x)^T p| &= |\widetilde{f}(x+hp) - \widetilde{f}(x) - hg(x)^T p| \le 2\omega + |f(x+hp) - f(x) - hg(x)^T p| \\ &\le 2\omega + \frac{L}{2}h^2\|p\|^2; \end{aligned}$$

hence by (30)

$$|\widetilde{g}_p(x) - g(x)^T p| \le \frac{2\omega}{h} + \frac{L}{2}h\|p\|^2 = O(\sqrt{\omega}\|p\|)$$

and so (31) holds for $\widetilde{g}_p(x)$ in place of $g_p(x)$. With the choice $\widetilde{f} = f$, (31) also holds for $g_p(x)$.

By choosing $p = e_i$ (coordinate direction) for $i = 1, \cdots, n$, we have $|\widetilde{g}_{e_i}(x) - g_{e_i}(x)| = \mathcal{O}(\sqrt{\omega})$ for $i = 1, \cdots, n$, resulting in $\|\widetilde{g}(x) - g(x)\| = \mathcal{O}(\sqrt{\omega})$. □

We deduce from Proposition 1.1 and Theorem 3.1 a complexity bound for `WLS-inexact`.

3.2 Proposition. *Assume that (A1) and (A2) hold, the gradient of f is estimated by* `FinDiff`*, the direction p satisfies the approximate bounded angle condition* (16)*, and the step size α satisfies the strong Wolfe conditions* (24) *and* (25)*. Denote by $\widetilde{\theta}$ the angle between $\widetilde{g}(x)$ and p and by θ the angle between $g(x)$ and p. Then if h satisfies* (30)*:*
(i) For all $x,\ y \in \mathbb{R}^n$ we have

$$\|\widetilde{g}(x) - \widetilde{g}(y)\| = \mathcal{O}(\sqrt{\omega}) + L\|x - y\|.$$

(ii) p satisfies the angle condition for the exact gradient

$$\frac{g(x)^T p}{\|g(x)\|\|p\|} < -\frac{\Delta_{\mathrm{a}}}{2}. \tag{33}$$

(iii) Under the assumptions

$$\|\widetilde{g}(x)\| > \frac{2\sqrt{\omega}}{\Delta_{\mathrm{a}}(1-\sigma)}, \tag{34}$$

$$\|g(x)\| \geq \max\Big\{\frac{4\widetilde{\rho}\sqrt{\omega}}{\rho\Delta_{\rm a}}, \frac{(1+\widetilde{\sigma})\sqrt{\omega}}{\sigma\Delta_{\rm a}}\Big\}, \tag{35}$$

and

$$\|\widetilde{g}(x)\|\|g(x)\| \geq \frac{16L\omega}{\rho(1-\widetilde{\sigma})\Delta_{\rm a}^2}, \tag{36}$$

α satisfies the strong Wolfe conditions

$$f(x+\alpha p) \leq f(x) + \rho_1\alpha g(x)^T p, \tag{37}$$

$$|g(x+\alpha p)^T p| \leq -\sigma_1 g(x)^T p, \tag{38}$$

where $0 < \rho_1 := \widetilde{\rho} - \rho < \widetilde{\rho} < \widetilde{\sigma} < \sigma_1 := \widetilde{\sigma} + \sigma$.
(iv) Given $0 < \rho_1 < \sigma_1$ and the minimal threshold $\alpha_{\min} \in (0,1)$ for the step size, we define

$$\overline{\mu} := \overline{\mu}(x) = \frac{\overline{f} - f(x)}{\rho_1 g(x)^T p}. \tag{39}$$

and choose the step size $\alpha \in (0, \overline{\mu}]$ and $\tau \in (0,1)$. Then `WLS-inexact` needs at most

$$K := K(x) = \Big\lceil \frac{\log \alpha_{\min}/\overline{\mu}(x)}{\log \tau} \Big\rceil \tag{40}$$

iterations and at most

$$N^f := N^f(x) = 2K(x) + n \tag{41}$$

function evaluations to satisfy (37) *and* (38).

Proof. (i) From (A2) and Theorem 3.1, we have

$$\|\widetilde{g}(x) - \widetilde{g}(y)\| = \|\widetilde{g}(x) - g(x) + g(x) - g(y) + g(y) - \widetilde{g}(y)\| = \mathcal{O}(\sqrt{\omega}) + L\|x-y\|.$$

(ii) By [35, Lemma 3.11], since $\cos\widetilde{\theta} \geq \Delta_{\rm a} > 0$ by (16) and (31) holds, $\cos\theta \geq \Delta_{\rm a}/2$ and so (33) is obtained.
(iii) By (i), the Cauchy–Schwarz inequality, and (25), we obtain

$$\begin{aligned}(\mathcal{O}(\sqrt{\omega}) + \alpha L\|p\|)\|p\| &= \|\widetilde{g}(x+\alpha p) - \widetilde{g}(x)\|\|p\| \\ &\geq (\widetilde{g}(x+\alpha p) - \widetilde{g}(x))^T p \geq (\widetilde{\sigma} - 1) g_p(x),\end{aligned}$$

resulting in

$$\alpha \geq \frac{1-\widetilde{\sigma}}{L}\Big(\frac{-g_p(x)}{\|p\|^2}\Big) - \frac{\mathcal{O}(\sqrt{\omega})}{L\|p\|} \geq \frac{(1-\widetilde{\sigma})\Delta_{\rm a}\|\widetilde{g}(x)\| - \sqrt{\omega}}{L\|p\|} \geq \frac{(1-\widetilde{\sigma})\Delta_{\rm a}\|\widetilde{g}(x)\|}{2L\|p\|}.$$

(iii) Under the condition (35), the statements (ii) and (iii) result in

$$\begin{aligned}-\rho\alpha g(x)^T p &\geq \frac{\rho(1-\widetilde{\sigma})\Delta_{\rm a}\|\widetilde{g}(x)\|}{2L\|p\|} g(x)^T p \geq \frac{\rho(1-\widetilde{\sigma})\Delta_{\rm a}\|\widetilde{g}(x)\|}{2L\|p\|}\|g(x)\|\|p\|\cos\theta \\ &\geq \frac{\rho(1-\widetilde{\sigma})\Delta_{\rm a}^2}{4L}\|\widetilde{g}(x)\|\|g(x)\| \geq 4\omega. \end{aligned} \tag{42}$$

On the other hand, (35) gives

$$-\rho\alpha g(x)^T p \geq \frac{\rho\Delta_{\mathrm{a}}}{2}\alpha\|p\|\|g(x)\| \geq \frac{\rho\Delta_{\mathrm{a}}}{2}\alpha\|p\|\Big(\frac{4\widetilde{\rho}\sqrt{\omega}}{\rho\Delta_{\mathrm{a}}}\Big) = 2\widetilde{\rho}\alpha\|p\|\sqrt{\omega}. \tag{43}$$

By summing both sides of (42) with (43), it results in

$$-\rho\alpha g(x)^T p \geq 2\omega + \widetilde{\rho}\alpha\|p\|\sqrt{\omega}.$$

From (24), we obtain

$$\begin{aligned} f(x+\alpha p) &\leq f(x) + \widetilde{\rho}\alpha g_p(x) \leq f(x) + \widetilde{\rho}\alpha\Big(g(x)^T p + \|p\|\sqrt{\omega}\Big) \\ &= f(x) + \widetilde{\rho}\alpha\|p\|\sqrt{\omega} + \widetilde{\rho}\alpha g(x)^T p \\ &\leq f(x) + (\widetilde{\rho} - \rho)\alpha g(x)^T p = f(x) + \rho_1 \alpha g(x)^T p; \end{aligned}$$

hence (37) holds. From (25) we conclude that

$$g(x+\alpha p)^T p + \|p\|\sqrt{\omega} \geq g_p(x+\alpha p) \geq \widetilde{\sigma} g_p(x) \geq \widetilde{\sigma}\Big(g(x)^T p - \|p\|\sqrt{\omega}\Big),$$

so that

$$g(x+\alpha p)^T p \geq \widetilde{\sigma} g(x)^T p - (1+\widetilde{\sigma})\|p\|\sqrt{\omega} \geq (\widetilde{\sigma}+\sigma) g(x)^T p = \sigma_1 g(x)^T p$$

by (35); hence (38) is obtained.
(iv) In the worst case, neither Proposition 1.1(i) nor Proposition 1.1(ii) is satisfied. In this case, Theorem 1.1(iii) holds such that the limit point $\widehat{\alpha}$ of $I(a_j, b_j)$ enforces the Wolfe condition (7) and (8) for sufficiently large j. To obtain a bound on the iterations, we use the condition given by interpolation

$$|b_j - a_j| \leq (1-\tau_1)|b_{j-1} - a_{j-1}| \quad \text{with } \tau_1 \in (0,1) \tag{44}$$

and the condition given by the extrapolation

$$|b_j - a_j| \leq (1-\tau_2)|b_{j-1} - a_{j-1}| \quad \text{with } \tau_2 \in (0,1). \tag{45}$$

The condition (44) is [1, (16)] obtained from (12) and the condition (45) is [1, (17)] obtained from (14). Denoting $\tau := \min\{1-\tau_1, 1-\tau_2\} \in (0,1)$, we have

$$|b_j - a_j| \leq \tau |b_{j-1} - a_{j-1}|. \tag{46}$$

`WLS-inexact` is performed in a finite number of iterations, since one of the statements in Proposition 1.1 holds. Hence if Proposition 1.1(i) does not hold, $b_1 \leq \overline{\mu} < \infty$. Otherwise, if $b_1 = \infty$, no bracket is found and extrapolation is performed until α_j reaches $\overline{\mu}$. Then $f(\overline{\mu}) \leq \overline{f}$. Therefore in any case `WLS-inexact` ends in a finite number K of iterations. Applying the inequality (46) recursively, we get

$$\alpha_{\min} \leq \widehat{\alpha} = |b_K - a_K| \leq \tau^{K-1}|b_1 - a_1| \leq \tau^{K-1}|\overline{\mu} - a_1| \leq \tau^{K-1}\overline{\mu},$$

so that the number is bounded by (40) and hence the number of function evaluations is bounded by (41), since $2K + n$ function evaluations are required. One of the stopping tests (10) and (25) of `WLS-inexact` requires n function evaluations in the last iteration to estimate the gradient with `FinDiff`, and two function evaluations to compute the function value and the directional derivative with `FinDiffDir` in each iteration. □

3.4 `SSDFO-basic`

In each iteration, `SSDFO-basic` performs `WLS-inexact` along directions whose angle with an approximate gradient estimated by the forward finite differences of Conn et al. [10] is bounded away from 90°, resulting in the best point and its function value. Once the approximate gradient norm at the best point is below a given threshold ε_g, `SSDFO-basic` ends and the best point is arbitrarily close to the minimizer.

Algorithm 1 `SSDFO-basic`, *a basic method for unconstrained DFO*

```
    Input: The initial point $x_0$
    Requirements: $\Delta_a \in (0,1)$ (tiny parameter for the approximate angle condition (16)),
    $\widetilde{\sigma} > \widetilde{\rho} > 0$ (parameters for line search), $0 < \varepsilon_h < 1$ (parameter for adjusting the finite
    difference step size $h$), $0 < \varepsilon_g < 1$ (minimum threshold for the stopping test).
    Output: The best point $x_{\rm best}$ and its function value $f_{\rm best}$
1: Compute the initial function value $f_0 := f(x_0)$.
2: Compute $\overline{f} = f_0 - 10^8(1 + |f_0|)$.        ▷ suggested by FMINUNC discussed in Section 1.5
3: Estimate the initial gradient $\widetilde{g}_0 = \widetilde{g}(x_0)$ by FinDiff.
4: for $\ell = 1, 2, 3, \cdots$ do
5:     if $\|\widetilde{g}_\ell\| \le \varepsilon_g$, then set $x_{\rm best} := x_\ell$, $f_{\rm best} := f_\ell$, and stop; end if
6:     Choose $p_\ell$ such that the approximate bounded angle condition (16) holds.
7:     Perform WLS-inexact with $x_{\ell+1} = x_\ell + \alpha_\ell p_\ell$, $f_{\ell+1} = f(x_{\ell+1})$, and $\widetilde{g}_{\ell+1} = \widetilde{g}(x_{\ell+1})$.
8: end for
```

3.5 Complexity of `SSDFO-basic`

Proposition 3.2 leads to the following complexity bound for `SSDFO-basic`.

3.3 Theorem. *Suppose that (A1) and (A2) hold, h satisfies* (30) *and let f_0 be the initial function value of f. Given $0 < \mu_{\rm min} < \infty$, define*

$$\overline{\mu}_{\rm min} := \min_{\ell \ge 0}(\mu_{\rm min}, \overline{\mu}_\ell) \quad \text{with } \overline{\mu}_\ell := \overline{\mu}(x_\ell) = \frac{\overline{f} - f(x_\ell)}{\rho_1 g(x_\ell)^T p_\ell}.$$

Here ρ_1 comes from Proposition 3.2(iii) and $\overline{f}$ comes from line 2 of Algorithm 1. Then `SSDFO-basic` needs at most $\mathcal{O}(\omega^{-1})$ iterations and $\mathcal{O}((2\overline{K} + n)\omega^{-1})$ function evaluations to find a point x with $\|g(x)\| = \mathcal{O}(\sqrt{\omega})$. Here

$$K_\ell := K(x_\ell) \le \overline{K} := \left\lceil \frac{\log \alpha_{\rm min}/\overline{\mu}_{\rm min}}{\log \tau} \right\rceil$$

with $\alpha_{\rm min}$ and τ from Proposition 3.2.

Proof. Let $\mathcal{S}$ be the set of iterations generated by `WLS-inexact` such that

$$\|g(x_\ell)\| > \max\left\{\frac{4\widetilde{\rho}}{\rho\Delta_a}, \frac{(1+\widetilde{\sigma})}{\sigma\Delta_a}\right\}\sqrt{\omega} = \mathcal{O}(\sqrt{\omega}) \quad \text{for all } \ell \in \mathcal{S};$$

$\mathcal{S}$ is not empty by Proposition 1.1. In other words, all iterations of `SSDFO-basic` decrease the function value; hence, they are successful. Based on Proposition 1.1, we consider the following two cases:
CASE 1. Suppose that Proposition 1.1(i) holds. Then, we get

$$f_{\ell+1} := f(x_\ell + \alpha_\ell p_\ell) \le \overline{f}, \quad \text{for some } \alpha_\ell \in (0, \overline{\mu}_\ell];$$

hence

$$f_\ell - f_{\ell+1} \ge f_\ell - \overline{f} \ge 0. \tag{47}$$

CASE 2. Suppose that Proposition 1.1(ii) holds. Then, by the angle condition (16), we obtain from Proposition 3.2

$$f_\ell - f_{\ell+1} \ge -\rho_1 \alpha_\ell g_\ell^T p_\ell \ge \frac{4\rho_1\omega}{\rho}. \tag{48}$$

Then (47) and (48) result in

$$f_\ell - f_{\ell+1} \ge \frac{4\rho_1\omega}{\rho}. \tag{49}$$

Summing both sides of (49) and using (4) in it give

$$f_0 - \widehat{f} \ge \sum_{\ell\in\mathcal{S}} (f_\ell - f_{\ell+1}) \ge |\mathcal{S}| \frac{4\rho_1\omega}{\rho},$$

leading to $|\mathcal{S}| \le \dfrac{\rho(f_0 - \widehat{f})}{4\rho_1\omega}$. Denote by $N_\ell^f = N^f(x_\ell)$ the number of function evaluations in each call to `WLS-inexact` for $\ell \ge 1$. By Proposition 3.2(iii), the total number of function evaluations by `SSDFO-basic` is

$$N(\omega) = \sum_{\ell\in\mathcal{S}} N_\ell^f = \sum_{\ell\in\mathcal{S}} (2K_\ell + n) \le (2\overline{K} + n)|\mathcal{S}| = \mathcal{O}\Big((2\overline{K} + n)\omega^{-1}\Big).$$

□

The order of our bound is the same as that found by BERAHAS et al. [8].

4 Enhancements

In this section we discuss how new improvements make `SSDFO-basic` very competitive. A new subspace step size that reduces, if possible, both the value of the model function and its gradient norm, is introduced. This can be done by solving a linear problem and a universal bound-constrained problem in a cheap way. In fact, our new subspace direction is the product of the new subspace step and the traditional subspace direction comes from `DLMBOPT`. If the new subspace step cannot be found, a limited memory quasi-Newton direction is computed as in `DLMBOPT`. We then our new direction if the angle between this direction and approximate gradient estimated by forward finite differences is near zero. Next, we discuss an improved Wolfe line search algorithm enriched with a heuristic formula that helps the algorithm to avoid failure. Finally, the `SSDFO` algorithm is explained.

4.1 Decreasing model function value and its gradient norm

In the ℓth iteration, `subspaceDir` computes our new subspace direction. It takes as input the matrices S_ℓ, Y_ℓ, H_ℓ and the estimated gradient $\widetilde{g}_\ell$ and returns as output the modified direction p_ℓ satisfying the approximate angle condition (16). It uses the tuning parameter $0 < \Delta_\mathrm{a} < 1$ for the approximate bounded angle condition. `subspaceDir` proceeds thorough the four steps, described below.

4.1.1 Step 1, minimizing the quadratic model

In a new way, we discuss how to minimize the maximal norm of the gradient model in the subspace spanned by the columns of the matrix $S_\ell \in \mathbb{R}^{n\times m}$, where typically $m \ll n$. We denote by $f_\ell := f(x_\ell)$ the function value at the current point x_ℓ, by $\widetilde{g}_\ell := \widetilde{g}(x_\ell)$ the estimated gradient function at x_ℓ, and with

$$c_\ell := S_\ell^T \widetilde{g}_\ell \tag{50}$$

the estimated gradient function restricted to S_ℓ at x_ℓ. Then the quadratic model

$$f(x_\ell + S_\ell z) - f_\ell \approx q(z) := c_\ell^T z + \frac{1}{2} z^T H_\ell z$$

is constructed whose gradient is

$$q'(z) := \partial q(z)/\partial z := c_\ell + H_\ell z.$$

Here H_ℓ is from (21) and c_ℓ is from (50). The stationary point

$$\widehat{z} := -H_\ell^{-1} c_\ell \tag{51}$$

is the solution of the linear system $c_\ell + H_\ell z = 0$.

4.1.2 Step 2, computation of the new subspace step size

To find a point that reduces not only the model function value q but also, if possible, its gradient norm $\|q\|$, we perform an exact line search on the model function along $\widehat{z}$ by solving the simpler univariate problem

$$\begin{array}{ll} \min\limits_{\beta} & \|q'(\beta\widehat{z})\|_2 = |1-\beta|\|c_\ell\|_2 \\ \text{s.t.} & q(\beta) = \gamma_1\beta + \gamma_2\beta^2 \le 0, \end{array} \tag{52}$$

where

$$\gamma_1 := c_\ell^T \widehat{z}, \quad \gamma_2 := \frac{1}{2}\widehat{z}^T H_\ell \widehat{z}. \tag{53}$$

In three situations, we do not want to improve the model gradient norm:
• If γ_1 or γ_2 is contaminated by NaN or $\pm\infty$, the constraint is not satisfied.

• If $\gamma_2 \leq 0$, $q(\beta)$ is flat or unbounded below.
• If $\gamma_2 > 0$ but $\gamma_1 \geq 0$, the minimal q is attained at $\beta = 0$.
In these cases, the problem (52) cannot be solved and p_ℓ is computed by (22). In the remaining case, $\gamma_1 < 0 < \gamma_2$ and $\beta^* := -\gamma_1/(2\gamma_2) > 0$. Since $q(\beta) \leq 0$ iff $0 \leq \beta \leq 2\beta^*$, the constraint in (52) is equivalent to $0 \leq \beta \leq 2\beta^*$. This case is acceptable only if β^* is a real value (otherwise p_ℓ is computed by (22)). Consequently, (52) reduces to

$$\begin{array}{ll} \min & |1-\beta| \\ \text{s.t.} & 0 \leq \beta \leq 2\beta^* \end{array}$$

with the solution $\widehat{\beta} := \min(1, 2\beta^*)$. We denote by $\widehat{q} := q(\widehat{\beta})$ the optimal model function value. Indeed, $\widehat{\beta}$ does not necessarily reduce both the function value q and the gradient norm q' of the model in finite precision arithmetic.

4.1.3 Step 3, computation of our new direction

At the ℓth iteration, if the condition $\widehat{q} \leq -\mathtt{df}_\ell$ holds, we have a good decrease in both the model function value and its gradient norm in the current subspace. Here $\mathtt{df}_\ell$ is a computational measure, which is an input for `subspaceDir`. It will be updated outside `subspaceDir` in lines 10 and 11 of Algorithm 2 below, depending on whether or not a decrease in the objective function is found. Then we can compute the new subspace direction

$$p_\ell := \widehat{\beta} S_\ell \widehat{z}. \tag{54}$$

Otherwise, $\widehat{q} > -\mathtt{df}_\ell$ and we compute the direction by (22).

4.1.4 Step 4, enforcing the approximate bounded angle condition

`WLS-inexact` must bound the angle between the approximate gradient computed by `FinDiff` and the search direction computed by `subspaceDir` away from 90°. Inspired by [21, Proposition 1] but using an approximate gradient, we modify the search direction $\overline{p}_\ell = p_\ell$, which can be obtained from either (22) or (54), if this direction does not satisfy the approximate angle condition (16). In this case, `subspaceDir` computes $\eta_1 = \widetilde{g}_\ell^T \widetilde{g}_\ell$, $\eta_2 = \overline{p}_\ell^T \overline{p}_\ell$, and $\eta = \widetilde{g}_\ell^T \overline{p}_\ell$. Then it finds t satisfying

$$\frac{\eta - t\eta_1}{\sqrt{\eta_1(\eta_2 - 2t\eta + t^2\eta_1)}} \leq -\Delta_{\mathrm{a}} \in (0,1), \tag{55}$$

and recomputes the search direction by $p_\ell = \overline{p}_\ell - t\widetilde{g}_\ell$ such that the approximate bounded angle condition (16) holds; but sometimes it is not possible due to rounding errors. In this case, `subspaceDir` recomputes the search direction by a diagonal preconditioned steepest descent direction $p_\ell = -D_\ell^{-1}\widetilde{g}_\ell$.

4.2 WLS, an improved version of WLS-inexact

The statements (i) and (ii) in Proposition 1.1 may not be numerically satisfied due to rounding errors. In this case, WLS-inexact must be improved. To prevent the failure of WLS-inexact, we use the heuristic formulas proposed in [21, Section 3.1], but with the difference that the gradient is estimated by FinDiff. Although the algorithm moves somewhat away from a minimizer in this case, it may move closer to a minimizer on the next few trials. At least this idea helps us avoid failures.

An improved version of WLS-inexact by Al-Baali & Fletcher [1] is called WLS. It uses the tuning parameters, $\widetilde{\sigma} > \widetilde{\rho} > 0$ (parameters for line search), $0 < \Delta_\alpha < 1$ (tiny parameter for adjusting the heuristic step size), and $0 < \varepsilon_h < 1$ (parameter for adjusting the finite difference step size h). It takes as input the direction p, the point x, its function value $f(x)$, and the approximate directional derivative $g_p(x)$ in the direction p and at x.

WLS first calls WLS-inexact to find a step size satisfying the approximate Wolfe line search conditions (24) and (25). If WLS does not enforce the approximate Wolfe conditions, it computes the heuristic step size by the one of following three formulas:
• $\|x\| = 0$ and $\|p\| \neq 0$: The heuristic step size $\alpha := \Delta_\alpha |f(x)/g_p(x)|$ is computed.
• $\|x\| \neq 0$ and $\|p\| \neq 0$: The heuristic step size

$$\alpha := \Delta_\alpha \max(|f(x)/g_p(x)|, \min\{|x_i/p_i| \mid p_i \neq 0\})$$

is computed.
• $\|x\| = \|p\| = 0$: $\alpha := 1$ is chosen.
Then, WLS computes the new point $x + \alpha p$, its function value $f(x + \alpha p)$, and estimates the gradient $\widetilde{g}(x + \alpha p)$, while returning them as output.

4.3 The new subspace algorithm

Our new algorithm is called SSDFO, a subspace method for unconstrained noiseless DFO problems. SSDFO takes advantage of our new subspace technique, which attempts to significantly reduce not only the value of the model function, but also its gradient norm. As long as SSDFO cannot reach an ε-approximate stationary point, subspaceDir is used to compute the direction along which WLS is tried. Then the ingredients of the subspace are updated.

Algorithm 2 SSDFO, *subspace method for DFO*

```
Input: The initial point $x_0$ and maximal function evaluations (nfmax)
Requirements: $\varepsilon_1 > 0$ (parameter for adjusting the initial df), $\gamma_f^2 > 1$ (parameter for expanding df), $\gamma_f^1, \gamma_f^3 \in (0,1)$ (parameters for reducing df), $m > 0$ (memory for the subspace), $\Delta_{\rm a} \in (0,1)$ (tiny parameter for the approximate angle condition (16)), $0 < \Delta_\alpha < 1$ (tiny parameter for adjusting the heuristic step size), $\widetilde{\sigma} > \widetilde{\rho} > 0$ (parameters for line search), $0 < \varepsilon_h < 1$ (parameter for adjusting the finite difference step size $h$).
Output: The best point $x_{\rm best}$ and its function value $f_{\rm best}$
1: Compute the initial function value $f_0$.
2: Compute $\overline{f} = f_0 - 10^8(1 + |f_0|)$.          ▷ suggested by FMINUNC
3: Estimate the gradient $\widetilde{g}_0$ by FinDiff.
4: Initialize $\mathtt{df}_0 = \varepsilon_1 |f_0|$.
5: Set $x_{\rm best} := x_0$ and $f_{\rm best} := f_0$.
6: for $\ell = 1, 2, 3, \cdots$ do
7:     Compute the subspace direction $p_\ell$ by subspaceDir.
8:     Perform WLS to find $\alpha_\ell$ resulting in $x_{\ell+1}$, $f_{\ell+1}$, and $\widetilde{g}_{\ell+1}$.
9:     if nfmax is reached then Set $x_{\rm best} := x_{\ell+1}$ and $f_{\rm best} := f_{\ell+1}$; stop. end if
10:    if $f_{\ell+1} < f_\ell - \mathtt{df}_\ell$ then, $\mathtt{df}_{\ell+1} = \gamma_f^1 (f_{\ell+1} - f_\ell)$.
11:    else, $\mathtt{df}_{\ell+1} = \max\Big(\gamma_f^2 \mathtt{df}_\ell, \gamma_f^3 (|f_{\ell+1}| + |f_\ell|)\Big)$.
12:    end if
13:    Set $s_\ell := x_{\ell+1} - x_\ell$ and $y_\ell := \widetilde{g}_{\ell+1} - \widetilde{g}_\ell$ and update the subspace by updateSY.
14: end for
```

5 Numerical results

In this section, we compare SSDFO with many known DFO solvers on 506 unconstrained noiseless problems from the CUTEst [13] collection of test problems with 2 to 9000 variables. For all problems, we use the standard initial points from the CUTEst collection.

5.1 Codes compared

We compared SSDFO with the solvers listed in Table 1 plus with two new solvers FMINUNCL and DLMBOPT for unconstrained noiseless DFO problems.

SSDFO uses the tuning parameters:

$$\boxed{\begin{array}{l}\varepsilon_h = \varepsilon_m, \quad \Delta_{\rm a} = 10^{-8}, \quad \widetilde{\sigma} = 10^{-4}, \quad \widetilde{\rho} = 0.9, \quad \Delta_b = \varepsilon_m, \quad \Delta_\alpha = \varepsilon_m, \quad \gamma_f^1 = 0.5, \\ \gamma_f^2 = 2, \quad \gamma_f^3 = 10^{-12}, \quad \varepsilon_1 = 10^{-8}, \quad m = \min(10, n).\end{array}}$$

5.1.1 Model-based

We here give the details of the model-based solvers with the default tuning parameters:

• `UOBYQA`, available at

`http://mat.uc.pt/~zhang/software.html`,

is an algorithm that forms quadratic models by interpolation by Powell [32].

• `BCDFO`, obtained from Anke Troeltzsch (personal communication), is a deterministic model-based trust-region algorithm for derivative-free bound-constrained minimization by Gratton et al. [15].

• `DFOTR` by Bandeira et al. [6], available at

`https://coral.ise.lehigh.edu/lnv/dfo-tr/`

is a trust region interpolation-based method.

5.1.2 Direct search

The details of the direct search solvers are here given:

• `NOMAD` (version 3.9.1) [2, 23], obtained from

`https://www.gerad.ca/nomad`

is a Mesh Adaptive Direct Search algorithm (MADS) [3]. `NOMAD` (model-based) uses the following option set

`opts` = `nomadset`(‘`max_eval`’, `nfmax`,‘max_iterations’, 2*`nfmax`),

while `NOMAD1` (model-free and Nelder–Mead free) uses the following option set

`opts` = `nomadset`(‘`max_eval`’,`nfmax`,‘max_iterations’,2*`nfmax`,
‘model_search’,0,‘model_eval_sort’,0,‘model_search_optimistic’,0,
‘model_eval_sort_cautious’,0, ‘direction_type’,‘ortho n+1 neg’,
‘model_quad_use_wp’,0,‘param_file’,‘param.txt’),

where ‘param.txt’ contains ‘NM_SEARCH no’, ignoring the Nelder–Mead algorithm. `NOMAD2` uses the same option as `NOMAD1` but with the Nelder–Mead algorithm. We here use `NOMAD` only to solve problems in low dimensions, while using `NOMAD1` and `NOMAD2` to solve medium scale problems. Other tuning parameters are default. For more details of how to choose the tuning parameters of `NOMAD`, see [5].

• `BFO` by Porcelli & Toint [31], available at

`https://github.com/m01marpor/BFO`,

is a trainable stochastic derivative-free solver for bound-constrained mixed integer optimization problems.

• Three versions derivative-free direct search algorithms are `NELDER` by Kelley [18]

`https://ctk.math.ncsu.edu/matlab_darts.html`

and `NMSMAX` and `ADSMAX` by Higham, obtained from

`http://www.ma.man.ac.uk/~higham/mctoolbox/`

• `DSPFD` by Gratton et al. [14] uses a direct search algorithm code for DFO problems, available at

`pages.discovery.wisc.edu/%7Ecroyer/codes/dspfd_sources.zip`.

Here `BFO`, `NELDER`, `NMSMAX`, `ADSMAX`, and `DSPFD` used the default tuning parameters.

5.1.3 Line search

The details of the line search solvers with the default tuning parameters are here given:

• `VRBBO` by Kimiaei & Neumaier [20] uses a randomized line search algorithm, available at

`https://www.mat.univie.ac.at/~neum/software/VRBBO/`.

• `VRDFON` by Kimiaei [19] uses a randomized line search algorithm, available at

`https://www.mat.univie.ac.at/~kimiaei/software/VRDFON`.

• `FMINUNC`, available at the Matlab Optimization Toolbox at

`https://ch.mathworks.com/help/optim/ug/fminunc.html`,

is a deterministic quasi-Newton or trust-region algorithm. `FMINUNC` is used with the following options set by `optimoptions`:

```
opts = optimoptions(@fminunc),‘Algorithm’,‘quasi-newton’
‘Display’, ‘Iter’,‘MaxIter’,Inf,‘MaxFunEvals’, limits.nfmax
‘TolX’, 0,‘TolFun’,0,‘ObjectiveLimit’,-1e-50).
```

• `FMINUNCL` is `FMINUNC` using the limited memory BFGS suggested by Liu & Nocedal [24] with the memory $m = 10$. Other tuning parameters were default.

• `SDBOX` by Lucidi & Sciandrone [26] is a derivative-free algorithm for bound-constrained optimization problems discussed, downloaded from

`http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`.

• `LMBOPT` by Kimiaei et al. [21], available at

`https://doi.org/10.5281/zenodo.5607521`,

is a limited memory method for bound-constrained optimization with exact function values and gradients. As defined earlier, `DLMBOPT` is the DFO version of `LMBOPT` with approximate gradients with a finite difference method.

5.1.4 Evolution strategy

The details of the two evolution strategy solvers with the default parameters are:

• `LMMAES` by Loshchilov et al. [25], obtained from

`https://homepages.fhv.at/hgb/downloads.html`,

is a limited memory matrix adaptation evolution strategy.

• `fMAES` by Beyer [7], obtained from

`https://homepages.fhv.at/hgb/downloads.html`,

is a fast matrix adaptation evolution strategy.

5.2 Stopping tests

We denote by `nf` the number of function evaluations and by `sec` time in seconds (excluding the setup time for the objective function). The limits

$$\texttt{nfmax} = 1000n$$

for `nf` and

$$\texttt{secmax} = \begin{cases} 180 \text{ sec} & \text{if } 2 \le n \le 30, \\ 420 \text{ sec} & \text{if } 31 \le n \le 100, \\ 1200 \text{ sec} & \text{if } 101 \le n \le 9000 \end{cases}$$

for `sec` are chosen such that <6% of the test problems cannot be solved by any of the compared solvers (see Tables 2-4). Some solvers may solve these difficult problems by increasing `nfmax` significantly to more than $1000n$, but the changes to the robustness and efficiency of the solvers are minor. For small and medium scale problems no solver stops due to reaching `secmax`, except for the model-based solvers, which require huge `secmax` to solve most medium scale problems. Model-free solvers require more function evaluations compared to model-based solvers, but can solve medium scale problems quickly. They are much cheaper than the model-based solvers since there is no overhead for model building. For large scale problems, `secmax` = 1200 sec was chosen to be large enough so that the most robust solver could solve all but 4% of the test problems (cf. Table 4).

5.3 Comparison criteria

5.3.1 Quality measure

As a measure of quality of a solver s to reach a minimum of the smooth exact function f, we define the quotients

$$q_s := (f_s - f_{\mathrm{opt}})/(f_0 - f_{\mathrm{opt}}) \quad \text{for } s \in \mathcal{S}.$$

Here, f_s denotes the best function value found by the solver s, f_0 denotes the function value at the starting point (common to all solvers), and f_{opt} denotes the function value at the best point known to the algorithm (in most cases, a global minimizer or at least a better local minimizer) found by performing a sequence of gradient-based and local/global gradient-free solvers; see [20]. Although such quotients are not available in real applications, they form a convenient quality measure.

We consider a problem solved by the solver s if $q_s \leq 10^{-4}$. Otherwise, the problem is unsolved because either `nfmax` or `secmax` is exceeded.

5.3.2 Cost measures for data and performance profiles

Denote by $\mathcal{S}$ the list of compared solvers and by $\mathcal{P}$ the list of problems. The efficiency $e_{p,s}$ of the solver s to solve the problem p is the inverse of the **performance ratio**

$$pr_{p,s} := \frac{c_{p,s}}{\min(c_{p,\overline{s}} \mid \overline{s} \in S)},$$

where $c_{p,s}$ is the cost measure of the solver s. Efficiency measures the ability of a solver $s \in \mathcal{S}$ compared to an ideal solver, rounded to integer, which is the inverse of the performance ratio. Indeed, the efficiency of a solver indicates how much this solver is cheaper in terms of `nf` and `sec`, compared to the other solvers compared. To compare `SSDFO` with known DFO solvers, two cost measures `nf` and `sec` are used, and the efficiency with respect to these measures are called the `nf` efficiency and the `sec` efficiency (denoted by `eff` in tables below), respectively.

5.3.3 Data profile

We use the data profile of MORÉ & WILD [29] and the performance profile of DOLAN & MORÉ [12] to determine the robustness and efficiency of all compared solvers. The data profile

$$\delta_s(\kappa) := \frac{1}{|\mathcal{P}|}\left|\left\{p \in \mathcal{P} \;\middle|\; cr_{p,s} := \frac{c_{p,s}}{n_p + 1} \leq \kappa\right\}\right| \tag{56}$$

of the solver s is the fraction of problems that the solver s can solve with $\kappa(n_p + 1)$ function evaluations. Here n_p is the dimension of the problem p, $c_{p,s}$ is a cost measure of the solver s to solve the problem p and $cr_{p,s}$ is the cost ratio of the solver s to solve the problem p.

5.3.4 Performance profile

The performance profile

$$\rho_s(\tau) := \frac{1}{|\mathcal{P}|}\Big|\Big\{p \in \mathcal{P} \;\Big|\; pr_{p,s} \le \tau\Big\}\Big| \tag{57}$$

of the solver s is the fraction of problems that the performance ratio $pr_{p,s}$ is at most τ. Note that $\rho_s(1)$ is the fraction of problems that the solver s wins compared to the other solvers, while $\rho_s(\tau)$ is the fraction of problems for sufficiently large τ that the solver s can solve.

5.3.5 Efficiency and robustness table

A solver is said to be **more robust** than other solvers if it has a higher number of solved problems, and **more efficient** if it has a lower relative cost for function evaluations. A solver is **competitive** if it is more robust or efficient than most other solvers tested.

We summarize our results in Table 2–4, identifying robustness and efficiency of all solvers compared, while plotting data and performance profiles, only for five robust and efficient solvers. In each table, the first row contains our stopping tests. The names of the compared solvers are given in the first column of the tables. The second through sixth columns of the table contain the number of solved problems by the solvers, the number #n of reaching `nfmax`, the number # of reaching `secmax`, the number # of fails, the `nf` efficiency, and the `sec` efficiency.

5.4 Results for noiseless DFO problems

5.4.1 Small scale $2 \le n \le 30$

From Table 2 we conclude that:

(Robustness) `SSDFO` solves 150 problems out of 171 problems, while `NOMAD` solves one problem less than `SSDFO`. Therefore, `SSDFO` and `NOMAD` are more robust than others.

(Efficiency) In terms of `nf`, `DFOTR` and `SSDFO` are cheaper than others on 48% and 44% of the test problems, respectively. In terms of `sec`, `NMSMAX` and `SSDFO` are faster than others on 49% and 41% of the test problems, respectively.

Figures 1 contains data and performance profiles for the five most robust solvers. They show that `SSDFO` is not only more robust but also more efficient than others.

Therefore, `SSDFO` is competitive on the unconstrained small scale noiseless test problems form the `CUTEst` collection compared to the nineteen DFO solvers.

Since model-based solvers are too slow for solving medium scale problems, we ignore them in our comparison in the next section and only compare ten more robust and fast line search and direct solvers for medium scale problems.

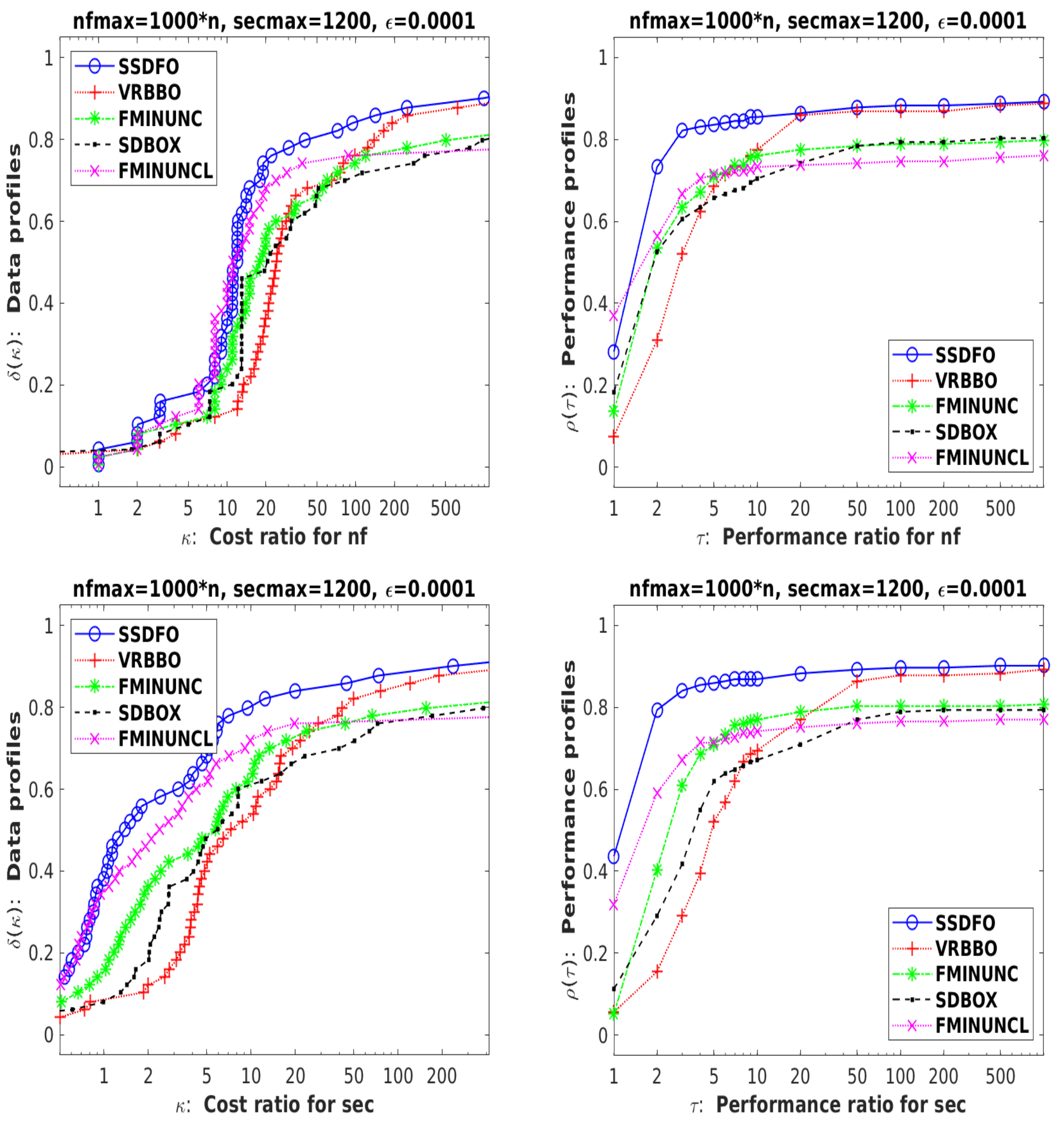

Figure 1: Results for small scale problems ($2 \le n \le 30$): data profiles $\delta(\kappa)$ (left) in dependence of a bound κ on the cost ratio in terms of `nf` and `sec`, see (56), and performance profiles $\rho(\tau)$ (right) in terms of `nf` in dependence of a bound τ on the performance ratio, see (57). Problems solved by no solver are ignored.

stopping test: $q_s \leq 0.0001,\ \texttt{sec} \leq 180,\ \texttt{nf} \leq 1000 * n$						
163 of 171 problems solved						
dim∈[2,30]		# of anomalies			`eff`%	
solver	solved	#n	#t	#f	`nf`	`sec`
`SSDFO`	150	21	0	0	44	41
`NOMAD`	149	0	1	21	24	12
`UOBYQA`	147	0	0	24	43	36
`VRBBO`	147	24	0	0	27	28
`NOMAD2`	147	0	0	24	21	34
`VRDFON`	144	27	0	0	31	15
`DFOTR`	143	21	0	7	48	12
`BCDFO`	141	9	7	14	37	11
`FMAES`	138	8	0	25	10	21
`NMSMAX`	138	33	0	0	22	49
`FMINUNC`	135	1	0	35	34	40
`NELDER`	134	6	0	31	14	24
`NOMAD1`	130	0	0	41	15	28
`BFO`	126	34	0	11	14	14
`SDBOX`	122	49	0	0	32	36
`FMINUNCL`	117	0	0	54	33	29
`ADSMAX`	112	31	0	28	7	9
`DSPFD`	108	43	0	20	13	10
`DLMBOPT`	33	138	0	0	1	3
`LMMAES`	17	5	0	149	5	4

Table 2: Results for `CUTEst` for dimensions 2-30

5.4.2 Medium scale $31 \leq n \leq 100$

From Table 3 we conclude that:

(Robustness) SSDFO solves 109 problems out of 122 problems, while VRBBO solves three problems less than SSDFO. Therefore, SSDFO and VRBBO are more robust than others.

(Efficiency) In terms of nf, SSDFO and VRDFON are cheaper than others on 55% and 44% of the test problems, respectively. In terms of sec, SSDFO and FMINUNC are faster than others on 71% and 53% of the test problems, respectively.

Figures 2 contains data and performance profiles for the five most robust solvers. They show that SSDFO is not only more robust but also more efficient than others.

Therefore, SSDFO is competitive on the unconstrained medium scale noiseless test problems form the CUTEst collection compared to the ten line search and direct search DFO solvers.

stopping test: $q_s \leq 0.0001$, sec ≤ 420, nf $\leq 1000 * n$						
115 of 122 problems solved						
dim∈[31,100]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
SSDFO	109	13	0	0	55	71
VRBBO	106	16	0	0	28	21
VRDFON	98	24	0	0	44	14
SDBOX	97	25	0	0	33	30
FMINUNC	93	4	0	25	43	53
FMINUNCL	84	0	0	38	38	44
DSPFD	77	29	0	16	13	3
BFO	70	47	2	3	3	2
NOMAD1	61	0	0	61	19	8
NOMAD2	56	0	65	1	14	7
NELDER	35	54	0	33	1	0

Table 3: Results for CUTEst for dimensions 31–100.

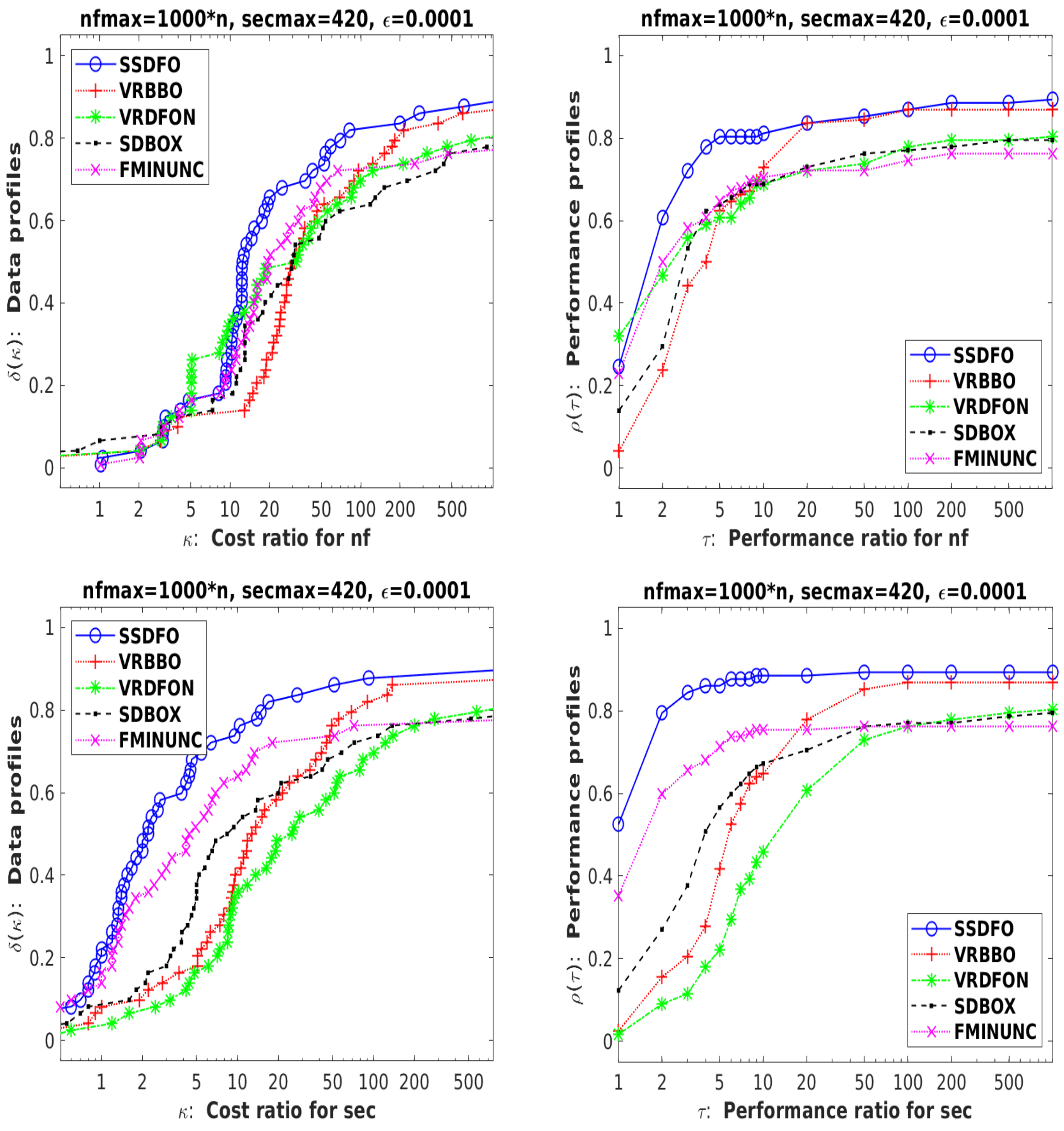

Figure 2: Results for medium scale problems ($31 \leq n \leq 100$). Other details as in Figure 1.

5.4.3 Large scale $101 \leq n \leq 9000$

From Table 4 we conclude that:

(Robustness) SSDFO solves 192 problems out of 213 problems, while VRBBO solves two problems less than SSDFO. Therefore, SSDFO and VRBBO are more robust than others.

(Efficiency) In terms of nf, SSDFO and FMINUNCL are cheaper than others on 59% and 48% of the test problems, respectively. In terms of sec, SSDFO and FMINUNL are faster than others on 75% and 56% of the test problems, respectively.

Therefore, SSDFO is competitive on the unconstrained large scale noiseless test problems form the CUTEst collection compared to the five line search-based DFO solvers.

stopping test: $q_s \leq 0.0001$, $\texttt{sec} \leq 1200$, $\texttt{nf} \leq 1000 * n$						
201 of 213 problems solved						
dim∈[101,9000]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
SSDFO	192	14	7	0	59	75
VRBBO	190	13	9	1	31	26
FMINUNC	172	8	6	27	42	40
SDBOX	171	23	18	1	38	34
FMINUNCL	164	0	2	47	48	56
VRDFON	163	30	19	1	54	25

Table 4: Results for CUTEst for problems with dimensions 101-9000.

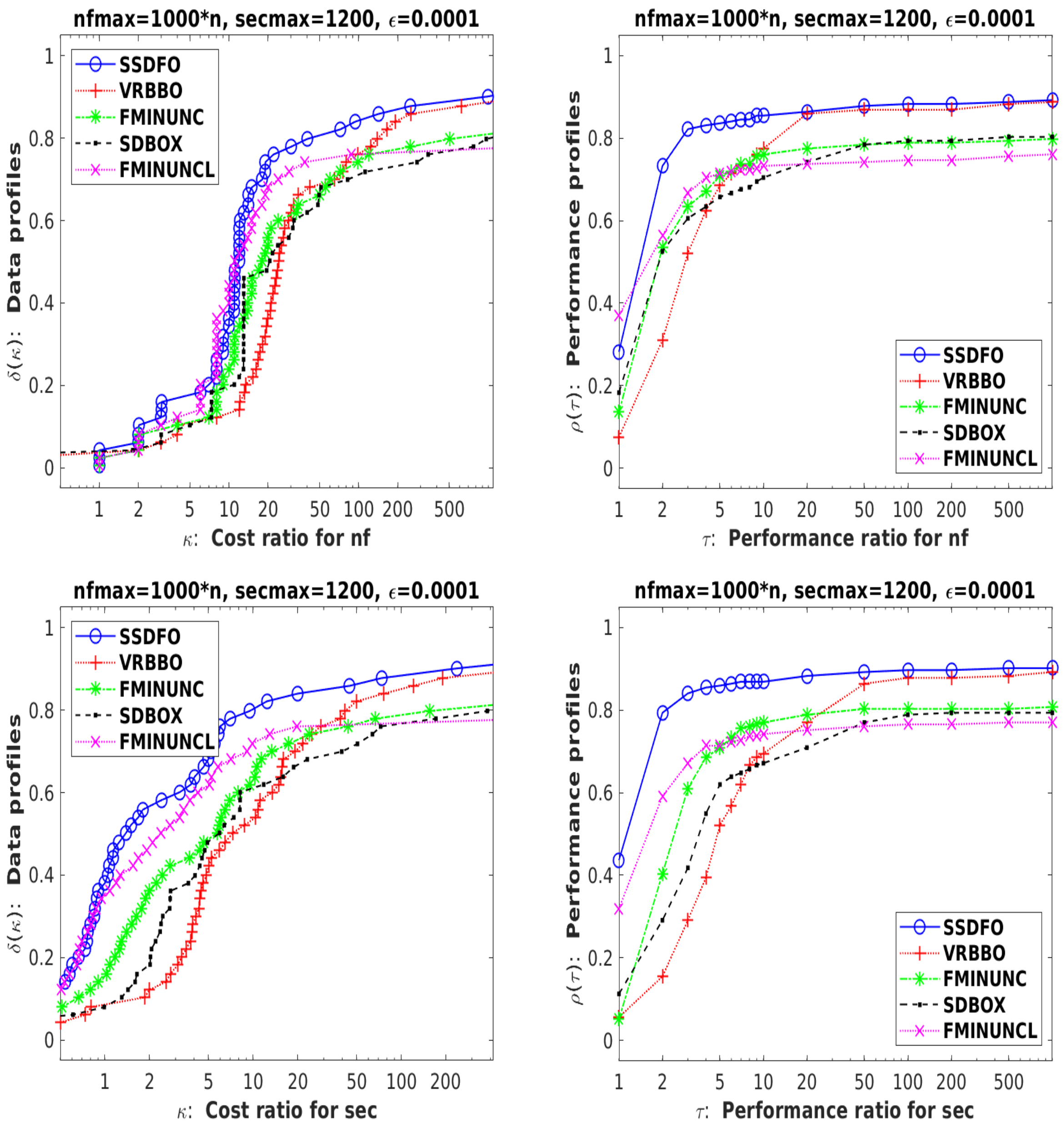

Figure 3: Results for large scale problems ($101 \leq n \leq 9000$). Other details as in Figure 1.

6 Conclusion

SSDFO is an efficient and robust subspace algorithm for unconstrained noiseless DFO problems in low to high dimensions. It is able to achieve a significant reduction in the function value by constructing an efficient direction with limited memory along which a successful prediction of a reduction in the model function and its gradient norm could be achieved. In each iteration, it has calls to an approximate Wolfe line search, while the gradients and directional derivatives are approximated by the finite difference approaches.

In the presence of the exact function values and inexact gradients, a worst-case complexity bound on the number of iterations and function evaluations is found for our algorithm, independent of the choice of search directions, which is consistent with that of Berahas et al. [8].

Our numerical results establish the following:

• Because of working in subspaces, SSDFO is quite fast compared to other DFO solvers for problems in low to high dimensions.

• SSDFO is the first DFO solver that can maintain its efficiency and robustness in low to high dimensions, unlike known DFO solvers that use either the standard BFGS directions or their limited-memory version, e.g., FMINUNC and FMINUNCL, and in low dimensions than known model-based DFO solvers, e.g., UOBYQA, BCDFO, and DFOTR. This is a new feature that has not been found in any DFO solver before, because for small scale problems, solvers using the standard BFGS directions are more robust and efficient than solvers using the limited memory BFGS directions, but for large scale problems the opposite is true.

• Another advantage of SSDFO is that it uses the same strategy for all dimensions, unlike some DFO solvers that have to ignore the model-based and standard BFGS approximations for medium and large dimensions.

Recommendation. Since SSDFO maintains its robustness and efficiency in low to high dimensions compared to the other DFO solvers, it is recommended for solving noiseless DFO problems in low to high dimensions. We do not recommend SSDFO for solving noisy problems because it estimates the gradient through the standard finite differences and uses the approximate directional derivative in the line search condition. In future work, SSDFO may be updated by variants of the methods discussed in Section 1.4 to solve noisy DFO problems.

References

[1] M. Al-Baali and R. Fletcher. An efficient line search for nonlinear least squares. *J. Optim. Theory Appl.* **48** (1986), 359–377.

[2] M. A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at `https://www.gerad.ca/nomad/`.

[3] C. Audet and J. E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization* **17** (2006), 188–217.

[4] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization.* Springer International Publishing (2017).

[5] C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD (2009).

[6] A. S. Bandeira, K. Scheinberg, and L.N. Vicente. Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. *Math. Program.* **134** (2012), 223–257.

[7] H. G. Beyer. Design principles for matrix adaptation evolution strategies. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* (2020).

[8] A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Found. Comut. Math.* (2021) https://doi.org/10.1007/s10208-021-09513-z.

[9] Albert S. Berahas, Richard H. Byrd, and Jorge Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.* **29** (2019), 965–993.

[10] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust region methods.* Society for Industrial and Applied Mathematics (2000).

[11] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization.* Society for Industrial and Applied Mathematics (2009).

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

[13] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* **60** (2014), 545–557.

[14] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim* **25** (2015), 1515–1541.

[15] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.* **26** (2011), 873–894.

[16] R. W. Hamming. *Introduction to Applied Numerical Analysis.* Taylor & Francis/Hemisphere, USA (1989).

[17] N. J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.* **14** (1993), 317–333.

[18] C. T. Kelley. *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics (1999).

[19] M. Kimiaei. Line search in noisy unconstrained derivative-free optimization. `http://www.optimization-online.org/DB_HTML/2020/09/8007.html` (2020).

[20] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput* **14**(2) (2022) 365–414.

[21] M. Kimiaei, A. Neumaier, and B. Azmi. LMBOPT – a limited memory method for bound-constrained optimization. *Math. Program. Comput* **14**(2) (2022), 271–318.

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

[23] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software* **37** (2011), 1–15.

[24] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.* **45** (1989), 503–528.

[25] I. Loshchilov, T. Glasmachers, and H. G. Beyer. Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Trans. Evol. Comput.* **23** (2019), 353–358.

[26] S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.* **21** (2002), 119–142.

[27] MATLAB Optimization Toolbox. The MathWorks, Natick, MA, USA.

[28] J. J. Moré and S. M. Wild. Estimating Derivatives of Noisy Simulations. *ACM Trans. Math. Softw.* **38** (2012), 1–21.

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

[30] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer New York (2006).

[31] M. Porcelli and P. Toint. Global and local information in structured derivative-free optimization with BFO. *arXiv: Optimization and Control* (2020).

[32] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* **92** (2002), 555–582.

[33] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Glob. Optim.* **56** (2012), 1247–1293.

[34] H. J. M. Shi, Y. Xie, M. Q. Xuan, and J. Nocedal. Adaptive Finite-Difference Interval Estimation for Noisy Derivative-Free Optimization. *SIAM J Sci Comput.* **4** (2022). `10.1137/21M1452470`.

[35] Y. Xie, R. H. Byrd, and J. Nocedal. Analysis of the BFGS method with errors. *SIAM J. Optim.* **30** (2020), 182–209.