PDFO: A Cross-Platform Package for Powell's Derivative-Free Optimization Solvers

Tom M. Ragonneau · Zaikun Zhang

To the memory of the late Professor M. J. D. Powell FRS

Abstract The late Professor M. J. D. Powell devised five trust-region methods for derivative-free optimization, namely COBYLA, UOBYQA, NEWUOA, BOBYQA, and LINCOA. He carefully implemented them into publicly available solvers, renowned for their robustness and efficiency. However, the solvers were implemented in Fortran 77 and hence may not be easily accessible to some users. We introduce the PDFO package, which provides user-friendly Python and MATLAB interfaces to Powell's code. With PDFO, users of such languages can call Powell's Fortran solvers easily without dealing with the Fortran code. Moreover, PDFO includes bug fixes and improvements, which are particularly important for handling problems that suffer from ill-conditioning or failures of function evaluations. In addition to the PDFO package, we provide an overview of Powell's methods, sketching them from a uniform perspective, summarizing their main features, and highlighting the similarities and interconnections among them. We also present experiments on PDFO to demonstrate its stability under noise, tolerance of failures in function evaluations, and potential to solve certain hyperparameter optimization problems.

T. M. Ragonneau
Dept. of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong, China
E-mail: tom.ragonneau@gmail.com
ORCID: 0000-0003-2717-2876

Z. Zhang (corresponding author)
Dept. of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong, China
E-mail: zaikunzhang@gmail.com
ORCID: 0000-0001-8934-8190

1 Introduction

Most optimization algorithms rely on classical or generalized derivative information of the objective and constraint functions. However, in many applications, such information is not available. This is the case, for example, if the objective function does not have an explicit formulation but can only be evaluated through complex simulations or experiments. Such problems motivate the development of optimization algorithms that use only function values but not derivatives, also known as derivative-free optimization (DFO) algorithms.

Powell devised five algorithms to tackle unconstrained and constrained problems without using derivatives, namely COBYLA [50], UOBYQA [53], NEWUOA [56], BOBYQA [58], and LINCOA. Not only did he propose these algorithms but he also implemented them into publicly available solvers, paying great attention to the stability and complexity of their numerical linear algebra computations. Renowned for their robustness and efficiency, these solvers are used in a wide spectrum of applications, for instance, aeronautical engineering [25], astronomy [39], computer vision [33], robotics [40], and statistics [5].

However, Powell implemented the solvers in Fortran 77, an old-fashioned language that damps the enthusiasm of many users to exploit these solvers in their projects. There has been a continued demand from both researchers and practitioners for the availability of Powell's solvers in more user-friendly languages such as Python and MATLAB.

Responding to such a demand, this paper presents a package named PDFO, an acronym for "Powell's Derivative-Free Optimization solvers." PDFO interfaces Powell's Fortran solvers with other languages, enabling users of such languages to call Powell's solvers without dealing with the Fortran code. For each supported language, PDFO provides a simple function that can invoke one of Powell's solvers according to the user's request (if any) or according to the type of the problem to solve. The current release (Version 2.2.0) of PDFO supports Python and MATLAB, with more languages to be covered in the future. The signature of the Python function is consistent with the `minimize` function of the SciPy optimization library, and that of the MATLAB function is consistent with the `fmincon` function of the MATLAB Optimization Toolbox. PDFO is cross-platform, available on Linux, macOS, and Windows at

https://www.pdfo.net   and   https://github.com/pdfo/pdfo ,

with the DOI 10.5281/zenodo.3887568. It has been downloaded more than 120,000 times as of June 2024, mirror downloads excluded. Moreover, it is one of the optimization engines in GEMSEO [25],[1] an industrial software package for multidisciplinary design optimization (MDO).

PDFO is not the first attempt to facilitate the usage of Powell's solvers in languages other than Fortran. Various efforts have been made in this direction. Py-BOBYQA [9,10] provides a Python implementation of BOBYQA

---

[1] https://gemseo.readthedocs.io .

(although it is not meant to be a faithful re-implementation of Powell's version); NLopt includes multi-language interfaces for COBYLA, NEWUOA, and BOBYQA;[2] minqa wraps UOBYQA, NEWUOA, and BOBYQA in R;[3] SciPy makes COBYLA available in Python under its optimization library.[4] However, PDFO has several features that distinguish it from others.

1. *Comprehensiveness.* To the best of our knowledge, PDFO is the only package that provides all of COBYLA, UOBYQA, NEWUOA, BOBYQA, and LINCOA with a *uniform interface.*
2. *Solver selection.* PDFO can automatically select a solver for a given problem. The selection takes into account the performance of the solvers on the CUTEst [28] problem set.
3. *Problem preprocessing.* PDFO preprocesses the inputs to simplify the problem and reformulate it to meet the requirements of Powell's solvers.
4. *Code patching.* PDFO patches several bugs in the Fortran code. Such bugs can lead to serious problems such as infinite cycling or memory errors.
5. *Fault tolerance.* PDFO tolerates failures of function evaluations. In case of such failures, PDFO will not exit but try to progress.
6. *Additional options.* PDFO includes options for the user to control the solvers in some manners that are useful in practice. For example, the user can request PDFO to scale the problem according to bound constraints on the variables before solving.

In addition to the PDFO package, this paper also provides an overview of Powell's DFO methods. We will not repeat Powell's description of these methods but summarize them from a uniform viewpoint, aiming at easing the understanding of Powell's methods and paving the way for further development based on them.

The analysis of Powell's DFO methods is not within the scope of this paper. Under some assumptions, adapted versions of Powell's algorithms may be covered by existing theory for trust-region DFO methods [18, Chapter 10]. However, it will still be interesting to pursue a tailored theory for Powell's algorithms.

The remaining part of this paper is organized as follows. Section 2 briefly reviews DFO methods in order to provide the context of PDFO. We then present an overview of Powell's DFO methods in Section 3, including a sketch of the algorithms and a summary of their main features. A detailed exposition of PDFO is given in Section 4, highlighting its solver selection, problem preprocessing, bug fixes, and handling of function evaluation failures. Section 5 presents some experiments on PDFO, demonstrating its stability under noise, tolerance of function evaluation failures, and potential in hyperparameter optimization. We conclude the paper with some remarks in Section 6.

---

[2] https://github.com/stevengj/nlopt .

[3] https://cran.r-project.org/package=minqa .

[4] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-cobyla.html .

2 A brief review of DFO methods

Consider a nonlinear optimization problem

$$\min_{x \in \Omega} \ f(x), \tag{1}$$

where $f \colon \mathbb{R}^n \to \mathbb{R}$ is the objective function and $\Omega \subseteq \mathbb{R}^n$ represents the feasible region. As summarized in [18], two strategies have been developed to tackle problem (1) without using derivatives, which we will introduce in the following.

The first strategy, known as direct search,[5] samples the objective function f and chooses iterates by simple comparisons of function values, examples including the Nelder-Mead algorithm [44], the MADS methods [2, 37], and BFO [46, 47]. See [35], [18, Chapters 7 and 8], [3, Part 3], and [36, § 2.1] for more discussions on this paradigm, and we refer to [29, 30] for recent developments of randomized methods in this category.

The second strategy approximates the original problem (1) by relatively simple models and locates the iterates according to these models. Algorithms applying this strategy are referred to as model-based methods. They often make use of the models within a trust-region framework [17] or a line-search framework [6]. Interpolation and regression are two common ways of establishing the models [4, 7, 15, 16, 52, 65, 74]. Algorithms using finite-difference gradients can also be regarded as model-based methods because such gradients essentially come from linear (for the forward and backward differences) or quadratic (for the central difference) interpolation of the function under consideration over rather special interpolation sets [62, § 1.4.3]. Most model-based DFO methods employ linear or quadratic models, examples including Powell's algorithms [50, 53, 56, 58] in PDFO, MNH [73], DFLS [76], DFO-TR [4], and DFO-LS [9, 32], but there are also methods exploiting radial basis functions (RBFs), such as ORBIT [74], CONORBIT [65], and BOOSTERS [45].

Hybrids between direct search and model-based approaches exist, for instance, Implicit Filtering [34, Algorithm 4.7] and MADS with quadratic models [12]. Theories of global convergence and convergence rate have been established for both direct search [22, 29, 35, 71, 72] and model-based methods [13, 17, 26, 59]. Since the objective and constraint functions in DFO problems are commonly expensive to evaluate, the worst-case complexity in terms of function evaluations is a major theoretical aspect of DFO algorithms. Examples of such complexity analysis can be found in [22, 26, 29, 72]. For more extensive discussions on DFO methods and theory, see the monographs [3, 18], the survey papers [20, 36, 66], the recent thesis [62], and the references therein.

---

[5] In some early papers (e.g., [50, 51]) Powell and many other authors used "direct search" to mean what is known as "derivative-free optimization" today. Powell rarely used the word "derivative-free optimization." The only exceptions known to us are his last paper [61] and his distinguished lecture titled "A parsimonious way of constructing quadratic models from values of the objective function in derivative-free optimization" at the National Center for Mathematics and Interdisciplinary Sciences, Beijing on November 4, 2011 [8].

3 Powell's derivative-free algorithms

Powell published in 1964 his first DFO algorithm based on conjugate directions [48].[6] His code for this algorithm is contained in the HSL Mathematical Software Library as subroutine VA24.[7] It is not included in PDFO because the code is not in the public domain, although open-source implementations are available (see [14, Footnote 4]).

From the 1990s to the final years of his career, Powell developed five model-based DFO algorithms to solve (1), namely COBYLA [50] (for nonlinearly constrained problems), UOBYQA [53] (for unconstrained problems), NEWUOA [56] (for unconstrained problems), BOBYQA [58] (for bound-constrained problems), and LINCOA (for linearly constrained problems). Moreover, Powell implemented these algorithms into Fortran solvers and made the code publicly available. They are the cornerstones of PDFO. This section provides an overview of these five algorithms, starting with a sketch in Section 3.1 and then presenting more details afterward.

3.1 A sketch of the algorithms

Powell's model-based DFO algorithms are trust-region methods. At iteration k, the algorithms construct a linear (for COBYLA) or quadratic (for the other methods) model f_k for the objective function f according to the interpolation condition

$$f_k(y) = f(y), \quad y \in \mathcal{Y}_k, \tag{2}$$

where $\mathcal{Y}_k \subseteq \mathbb{R}^n$ is a finite interpolation set updated along the iterations. COBYLA models the constraints by linear interpolants on $\mathcal{Y}_k$ as well. Instead of repeating Powell's description of these algorithms, we outline them in the sequel, emphasizing the trust-region subproblem, the interpolation problem, and the management of the interpolation set.

3.1.1 The trust-region subproblem

In all five algorithms, iteration k places the trust-region center x_k at the "best" point where the objective function and constraints have been evaluated so far. Such a point is selected according to the objective function or a merit function that takes the constraints into account. After choosing the trust-region

---

[6] According to Google Scholar, this is Powell's second published paper and also the second most cited work. The earliest and most cited one is his paper on the DFP method [23] co-authored with Fletcher and published in 1963. DFP is not a DFO algorithm but the first quasi-Newton method. The least-change property [21] of quasi-Newton methods is a major motivation for Powell to investigate the least Frobenius norm updating [54] of quadratic models in DFO, which is the backbone of NEWUOA, BOBYQA, and LINCOA.

[7] https://www.hsl.rl.ac.uk .

center x_k, with the trust-region model f_k constructed according to (2), a trial point x_k^{t} is then obtained by solving approximately the trust-region subproblem

$$\min_{x \in \Omega_k} \ f_k(x) \quad \text{s.t.} \quad \|x - x_k\| \le \Delta_k, \tag{3}$$

where Δ_k is the trust-region radius, and $\|\cdot\|$ is the ℓ_2-norm in $\mathbb{R}^n$. In this subproblem, the set $\Omega_k \subseteq \mathbb{R}^n$ is a local approximation of the feasible region Ω. COBYLA defines Ω_k by linear interpolants of the constraint functions over the set $\mathcal{Y}_k$, whereas the other four algorithms take $\Omega_k = \Omega$.

3.1.2 The interpolation problem

Fully determined interpolation. The interpolation condition (2) is essentially a linear system. Given a base point $y^{\mathrm{b}} \in \mathbb{R}^n$, which may depend on k, a linear model f_k takes the form of $f_k(x) = f(y^{\mathrm{b}}) + (x - y^{\mathrm{b}})^{\mathsf{T}} \nabla f_k(y^{\mathrm{b}})$, and hence (2) is equivalent to

$$f_k(y^{\mathrm{b}}) + (y - y^{\mathrm{b}})^{\mathsf{T}} \nabla f_k(y^{\mathrm{b}}) = f(y), \ y \in \mathcal{Y}_k, \tag{4}$$

which is a linear system with respect to $f_k(y^{\mathrm{b}}) \in \mathbb{R}$ and $\nabla f_k(y^{\mathrm{b}}) \in \mathbb{R}^n$, the degrees of freedom being $n+1$. COBYLA builds linear models by the system (4), with $\mathcal{Y}_k$ being an interpolation set of $n + 1$ points updated along the iterations. Similarly, if f_k is a quadratic model, then (2) is equivalent to

$$f_k(y^{\mathrm{b}}) + (y - y^{\mathrm{b}})^{\mathsf{T}} \nabla f_k(y^{\mathrm{b}}) + \frac{1}{2}(y - y^{\mathrm{b}})^{\mathsf{T}} \nabla^2 f_k(y^{\mathrm{b}})(y - y^{\mathrm{b}}) = f(y), \ y \in \mathcal{Y}_k, \tag{5}$$

a linear system with unknowns $f_k(y^{\mathrm{b}}) \in \mathbb{R}$, $\nabla f_k(y^{\mathrm{b}}) \in \mathbb{R}^n$, and $\nabla^2 f_k(y^{\mathrm{b}}) \in \mathbb{R}^{n \times n}$, the degrees of freedom being $(n+1)(n+2)/2$ due to the symmetry of $\nabla^2 f_k(y^{\mathrm{b}})$. UOBYQA constructs quadratic models by the system (5). To decide a quadratic model f_k completely by this system alone, UOBYQA requires that $\mathcal{Y}_k$ contains $(n + 1)(n + 2)/2$ points, and f should have been evaluated at all these points before the system can be formed. Even though most of these points will be reused at the subsequent iterations so that the number of function evaluations needed per iteration is tiny (see Section 3.1.3), we must perform $(n + 1)(n + 2)/2$ function evaluations during the very first iteration. This is impracticable unless n is small, which motivates the underdetermined quadratic interpolation.

Underdetermined quadratic interpolation. In this case, models are established according to the interpolation condition (2) with $|\mathcal{Y}_k|$ being less than or equal to $(n + 1)(n + 2)/2$, the remaining degrees of freedom being taken up by minimizing a certain functional $\mathcal{F}_k$ to promote the regularity of the quadratic model. More specifically, this means building f_k by solving

$$\min_{Q \in \mathcal{Q}_n} \ \mathcal{F}_k(Q) \quad \text{s.t.} \quad Q(y) = f(y), \ y \in \mathcal{Y}_k, \tag{6}$$

where $\mathcal{Q}_n$ is the space of polynomials on $\mathbb{R}^n$ of degree at most 2. NEWUOA, BOBYQA, and LINCOA construct quadratic models in this way, with

$$\mathcal{F}_k(Q) = \left\| \nabla^2 Q - \nabla^2 f_{k-1} \right\|_{\mathsf{F}}^2, \tag{7}$$

which is inspired by the least-change property of quasi-Newton updates [21], although other functionals are possible (see, e.g., [4, 19, 60, 75, 78]). The first model f_1 is obtained by setting $f_0 = 0$. Powell [60] referred to his approach as the *symmetric Broyden update* of quadratic models (see also [77, § 3.6] and [62, § 2.4.2]). It can be regarded as a derivative-free version of Powell's symmetric Broyden (PSB) quasi-Newton update [49], which minimizes the functional $\mathcal{F}_k$ among all quadratic polynomials that fulfill $Q(x_k) = f(x_k)$, $\nabla Q(x_k) = \nabla f(x_k)$, and $\nabla Q(x_{k-1}) = \nabla f(x_{k-1})$ (see [21, Theorem 4.2]), with x_k and x_{k-1} being the current and the previous iterates, respectively. The interpolation problem (6)–(7) is a convex quadratic programming problem with respect to the coefficients of the quadratic model.

Solving the interpolation problem. Powell's algorithms do not solve the interpolation problems (4), (5), and (6)–(7) from scratch. COBYLA maintains the inverse of the coefficient matrix for (4) and updates it along the iterations. Since each iteration of COBYLA alters the interpolation set $\mathcal{Y}_k$ by only one point (see Subsection 3.1.3), the coefficient matrix is modified by a rank-1 update, and hence its inverse can be updated according to the Sherman-Morrison-Woodbury formula [31]. UOBYQA does the same for (5), except that [53, § 4] describes the update in terms of the Lagrange functions of the interpolation problem (5), the coefficients of a Lagrange function corresponding precisely to a column of the inverse matrix. For the underdetermined quadratic interpolation (6)–(7), NEWUOA, BOBYQA, and LINCOA maintain and update the inverse of the coefficient matrix for the KKT system of (6)–(7). The update is also done by the Sherman-Morrison-Woodbury formula as detailed in [55, § 2]. In this case, each iteration modifies the coefficient matrix and its inverse by rank-2 updates. In addition, the columns of this inverse matrix readily provide the coefficients of Lagrange functions that make the interpolation problem (6)–(7) easy to solve (see [54, § 3]).

The base point. The choice of the base point y^{b} is also worth mentioning. COBYLA sets y^{b} to the center x_k of the current trust region. In contrast, the other four algorithms initiate y^{b} to the starting point provided by the user and keep it unchanged except for occasionally updating y^{b} to x_k, without which the distance $\|y^{\mathrm{b}} - x_k\|$ may become unfavorably large for the numerical solution of the interpolation problem. See [54, § 5] and [56, § 7] for more elaboration.

3.1.3 The interpolation set

The strategy to update $\mathcal{Y}_k$ is crucial. It should reuse points from previous iterations, at which the objective and constraint functions have been evaluated.

Meanwhile, it needs to maintain the geometry of the interpolation set so that it is well poised, or equivalently, the interpolation problem is well conditioned [18].

At a normal iteration, Powell's methods compute a point $x_k^{\mathrm{t}} \in \mathbb{R}^n$ by solving the trust-region subproblem (3), and update the interpolation set as

$$\mathcal{Y}_{k+1} = \left(\mathcal{Y}_k \cup \{x_k^{\mathrm{t}}\}\right) \setminus \{y_k^{\mathrm{d}}\}, \tag{8}$$

where $y_k^{\mathrm{d}} \in \mathcal{Y}_k$ is selected after obtaining x_k^{t}, aiming to maintain the well-poisedness of $\mathcal{Y}_{k+1}$. As mentioned, Powell's methods update the inverse of the coefficient matrix for either the interpolation system or the corresponding KKT system by the Sherman-Morrison-Woodbury formula. To keep the interpolation problem well-conditioned, y_k^{d} is chosen to enlarge the magnitude of the denominator in this formula, which is also the ratio between the determinants of the old and new coefficient matrices.[8] In the fully determined interpolation, this denominator is $\ell_k^{\mathrm{d}}(x_k^{\mathrm{t}})$, where ℓ_k^{d} is the Lagrange function associated with $\mathcal{Y}_k$ corresponding to y_k^{d} (see equations (10)–(13) and § 2 of [52]). In the underdetermined case, the denominator is lower bounded by $[\ell_k^{\mathrm{d}}(x_k^{\mathrm{t}})]^2$ (see equation (2.12), Lemma 1, and § 2 of [55], where the denominator is denoted by σ, and $\ell_k^{\mathrm{d}}(x_k^{\mathrm{t}})$ by τ). However, Powell's methods do not choose the point y_k^{d} merely according to this denominator, but also take into account its distance to the trust-region center, giving a higher priority to farther points, as we can see in [53, Equation (56)] and [56, Equations (7.4)–(7.5)], for example.

An alternative update of the interpolation set takes place when the methods detect that f_k does not represent f well enough, attempting to improve the geometry of the interpolation set. In this case, the methods first select a point $y_k^{\mathrm{d}} \in \mathcal{Y}_k$ to drop from $\mathcal{Y}_k$, and then set

$$\mathcal{Y}_{k+1} = \left(\mathcal{Y}_k \setminus \{y_k^{\mathrm{d}}\}\right) \cup \{x_k^{\mathrm{g}}\}, \tag{9}$$

where $x_k^{\mathrm{g}} \in \mathbb{R}^n$ is chosen to improve the well-poisedness of $\mathcal{Y}_{k+1}$. In COBYLA, the choice of y_k^{d} and x_k^{g} is guided by the fact that the interpolation set forms a simplex in $\mathbb{R}^n$, trying to keep $\mathcal{Y}_{k+1}$ away from falling into an $(n-1)$-dimensional subspace, as is detailed in [50, Equations (15)–(17)]. The other four methods select y_k^{d} from $\mathcal{Y}_k$ by maximizing its distance to the current trust-region center x_k, and then obtain x_k^{g} by solving

$$\max_{x \in \Omega} |\ell_k^{\mathrm{d}}(x)| \quad \text{s.t.} \quad \|x - x_k\| \leq \widetilde{\Delta}_k \tag{10}$$

for some $\widetilde{\Delta}_k \in (0, \Delta_k]$. The motivation for this problem is again to enlarge the magnitude of the aforementioned denominator in the Sherman-Morrison-Woodbury updating formula: for UOBYQA, the denominator is $\ell_k^{\mathrm{d}}(x)$, while

---

[8] Suppose that W is a square matrix and consider $\widetilde{W} = W + UV^{\mathsf{T}}$, where U and V are two matrices of the same size and UV^{T} has the same size as W. Then $\det(\widetilde{W}) = \det(W) \det(I + V^{\mathsf{T}} W^{-1} U)$, and the Sherman-Morrison-Woodbury formula is $\widetilde{W}^{-1} = W^{-1} - W^{-1} U (I + V^{\mathsf{T}} W^{-1} U)^{-1} V^{\mathsf{T}} W^{-1}$, assuming that both W and $I + V^{\mathsf{T}} W^{-1} U$ are nonsingular. The number $\det(I + V^{\mathsf{T}} W^{-1} U)$ is the only denominator involved in the numerical computation of the formula.

for NEWUOA, BOBYQA, and LINCOA, the denominator is lower bounded by $[\ell_k^{\mathrm{d}}(x)]^2$. In addition, NEWUOA maximizes this denominator directly if (10) fails to make its magnitude large enough, which rarely happens [56, § 6].

Given the two possible updates (8) and (9) of the interpolation set, it is clear that the number of interpolation points remains constant. As mentioned earlier, this number is $n+1$ in COBYLA and $(n+1)(n+2)/2$ in UOBYQA. NEWUOA, BOBYQA, and LINCOA set it to an integer in $[n+2, (n+1)(n+2)/2]$, with the default value being $2n+1$, which is proved optimal in terms of the well-poisedness of the initial interpolation set chosen by Powell for NEWUOA [63].

3.2 COBYLA

Published in 1994, COBYLA was the first model-based DFO solver by Powell. The solver is named after "Constrained Optimization BY Linear Approximations." It aims to solve problem (1) with the feasible region

$$\Omega \stackrel{\mathrm{def}}{=} \{x \in \mathbb{R}^n : c_i(x) \geq 0, \ i = 1, \ldots, m\},$$

where $c_i \colon \mathbb{R}^n \to \mathbb{R}$ denotes the ith constraint function for each $i \in \{1, \ldots, m\}$. The same as the objective function, all constraints are assumed to be accessible only through function values.

As mentioned before, iteration k of COBYLA models the objective and the constraint functions with linear interpolants on the interpolation set $\mathcal{Y}_k$ of $n+1$ points. Once the linear models $c_{k,i}$ of c_i are built for $i \in \{1, \ldots, m\}$, the trust-region subproblem (3) is formed with

$$\Omega_k \stackrel{\mathrm{def}}{=} \{x \in \mathbb{R}^n : c_{k,i}(x) \geq 0, \ i = 1, \ldots, m\}. \tag{11}$$

This subproblem may not be feasible, as the trust region and the region (11) may not intersect. COBYLA handles the trust-region subproblem in two stages. In the first stage, it solves

$$\min_{x \in \mathbb{R}^n} \max_{1 \leq i \leq m} \ [c_{k,i}(x)]_- \quad \text{s.t.} \quad \|x - x_k\| \leq \Delta_k,$$

where $[t]_- = \max\{0, -t\}$ for any $t \in \mathbb{R}$. In doing so, the method attempts to reduce the ℓ_∞-violation of the linearized constraints within the trust region. If the first stage finds a point in the interior of the trust region, then the second stage uses the resultant freedom in x to minimize the linearized objective function f_k within the trust region subject to no increase in any greatest violation of the linearized constraints.

COBYLA assesses the quality of points and updates the trust-region radius according to an ℓ_∞-merit function and a reduction ratio based on it (see [50, Equations (5), (9), and (10)]). It never increases the trust-region radius and reduces it if the geometry of $\mathcal{Y}_k$ is acceptable but the trust-region trial point x_k^{t} is too close to x_k or does not render a big enough reduction ratio [50, Equation (11)].

3.3 UOBYQA

In 2002, Powell published UOBYQA [53], named after "Unconstrained Optimization BY Quadratic Approximation." It aims at solving the nonlinear optimization problem (1) in the unconstrained case, i.e., when $\Omega = \mathbb{R}^n$.

At iteration k, UOBYQA constructs the model f_k for the objective function f by the fully determined quadratic interpolation on the interpolation set $\mathcal{Y}_k$ containing $(n + 1)(n + 2)/2$ points. The trust-region subproblem (3) is solved with the Moré-Sorensen algorithm [41]. For the geometry-improving subproblem (10), Powell developed an inexact algorithm that requires only $\mathcal{O}(n^2)$ operations. See [53, § 2] for more details.

UOBYQA updates the trust-region radius Δ_k in a noteworthy way. The update is typical for trust-region methods, except that a lower bound ρ_k is imposed on Δ_k. The value of ρ_k can be regarded as an indicator for the current accuracy of the algorithm. Without imposing $\Delta_k \geq \rho_k$, the trust-region radius Δ_k may be reduced to a value that is too small for the current accuracy, making the interpolation points concentrate too much. The value of ρ_k is never increased and is decreased when the UOBYQA decides that the work for the current value of ρ_k is finished. It decides so if Δ_k reaches its lower bound ρ_k, the current trust-region trial step does not perform well, and the current interpolation set seems adequate for the current accuracy. See [53, § 3] for more information on the updates of Δ_k and ρ_k.

3.4 NEWUOA, BOBYQA, and LINCOA

Later on, based on the underdetermined quadratic interpolation introduced in Subsection 3.1.2, Powell developed his last three DFO solvers, namely NEWUOA [56, 57], BOBYQA [58], and LINCOA. BOBYQA and LINCOA are named respectively after "Bound Optimization BY Quadratic Approximation" and "LINearly Constrained Optimization Algorithm," but Powell [56, 57] did not specify the meaning of NEWUOA, which is likely an acronym for "NEW Unconstrained Optimization Algorithm." It is worth mentioning that Powell *never* published a paper to introduce LINCOA, and [61] discusses only how to solve its trust-region subproblem.

NEWUOA, BOBYQA, and LINCOA aim at solving unconstrained, bound-constrained, and linearly constrained problems, respectively. They all set Ω_k in the trust-region subproblem (3) to be Ω, corresponding to the whole space for NEWUOA, a box for BOBYQA, and a polyhedron for LINCOA.

To solve the trust-region subproblem (3), NEWUOA employs the Steihaug-Toint truncated conjugate gradient (TCG) algorithm [69, 70]; if the boundary of the trust region is reached, then NEWUOA may make further changes to the trust-region step, each one obtained by searching in the two-dimensional space spanned by the current step and the corresponding gradient of the trust-region model [56, § 5]. BOBYQA solves (3) by an active-set variant of the TCG algorithm, and it may also improve the TCG step by two-dimensional

searches if it reaches the trust-region boundary [58, § 3]. LINCOA uses another active-set variant of TCG to solve the trust-region subproblem (3) with linear constraints [61, § 3 and § 5]. An accessible description of the TCG algorithms employed by BOBYQA and LINCOA can be found in [62, §§ 6.2.1–6.2.2]. NEWUOA, BOBYQA, and LINCOA manage the trust-region radius in a way similar to UOBYQA, imposing a lower bound ρ_k on Δ_k when updating Δ_k.

When solving the geometry-improving subproblem (10), NEWUOA first takes $x_k \pm \widetilde{\Delta}_k(y_k^{\mathrm{d}} - x_k)/\|y_k^{\mathrm{d}} - x_k\|$, with the sign that provides the larger value of ℓ_k^{d}, and then revises it by a procedure similar to the two-dimensional searches that improve the TCG step for (3) (see [56, § 6]). BOBYQA computes two approximate solutions to (10) and chooses the better one: the first one solves (10) with an additional constraint that x is located on the straight lines through x_k and another point in $\mathcal{Y}_k$, and the second is obtained by a Cauchy step for (10) (see [58, § 3]). The geometry-improving step of LINCOA is more complex, as it is chosen from three approximate solutions to (10):

1. the point that maximizes $|\ell_k^{\mathrm{d}}|$ within the trust region on the lines through x_k and another point in $\mathcal{Y}_k$,
2. a point obtained by a gradient step that maximizes $|\ell_k^{\mathrm{d}}|$ within the trust region, and
3. a point obtained by a projected gradient step that maximizes $|\ell_k^{\mathrm{d}}|$ within the trust region, the projection being made onto the null space of the constraints that are considered active at x_k.

Note that the first two cases disregard the linear constraints (i.e. $x \in \Omega_k = \Omega$), while the third case considers only the active constraints. LINCOA first selects the point among the first two alternatives for a larger value of $|\ell_k^{\mathrm{d}}|$; further, this point is replaced with the third alternative if the latter nearly satisfies the linear constraints $x \in \Omega$ while rendering a value of $|\ell_k^{\mathrm{d}}|$ that is not too small compared with the above one.

BOBYQA respects the bound constraints $x \in \Omega$ when solving the trust-region subproblem (3) and the geometry-improving subproblem (10), even though these problems are solved approximately. It also chooses the initial interpolation set $\mathcal{Y}_1$ within the bounds. Therefore, BOBYQA is a feasible method. In contrast, LINCOA may violate the linear constraints when solving the geometry-improving subproblem and when setting up the initial interpolation set. Consequently, LINCOA is an infeasible method, which requires f to be defined even when the linear constraints are not satisfied.

4 The PDFO package

This section details the main features of PDFO, in particular the signature of the main function, solver selection, problems preprocessing, bug fixes, and handling failures of function evaluations. For more features of PDFO, we refer to its homepage at `https://www.pdfo.net`.

Before starting, we emphasize that PDFO does not re-implement Powell's solvers but rather enables Python and MATLAB to call Powell's Fortran

implementation. At a low level, it uses F2PY[9] to interface Python with Fortran, and MEX to interface MATLAB with Fortran, although users never need such knowledge to employ PDFO.

4.1 Signature of the main function

The philosophy of PDFO is simple: providing a single function named `pdfo` to solve DFO problems with or without constraints, calling Powell's Fortran solvers in the backend. It takes for input an optimization problem of the form

$$\min_{x \in \mathbb{R}^n} \ f(x) \tag{12a}$$

$$\text{s.t. } l \le x \le u, \tag{12b}$$

$$A_{\mathrm{I}}x \le b_{\mathrm{I}}, \ A_{\mathrm{E}}x = b_{\mathrm{E}}, \tag{12c}$$

$$c_{\mathrm{I}}(x) \le 0, \ c_{\mathrm{E}}(x) = 0, \tag{12d}$$

where f a real-valued objective function, while c_{E} and c_{I} are vector-valued constraint functions. The bound constraints are given by n-dimensional vectors l and u, which may take infinite values. The linear constraints are formulated by real matrices A_{E} and A_{I} together with real vectors b_{E} and b_{I} of proper sizes. We allow one or more of the constraints (12b)–(12d) to be absent. Being a specialization of (1), problem (12) is broad enough to cover numerous applications of DFO.

In the Python version of PDFO, the signature of the `pdfo` function is compatible with the `minimize` function available in the `scipy.optimize` module of SciPy. It can be invoked in exactly the same way as `minimize` except that `pdfo` does not accept derivative arguments. The MATLAB version of PDFO designs the `pdfo` function following the signature of the `fmincon` function available in the Optimization Toolbox of MATLAB. In both Python and MATLAB, users can check the detailed syntax of `pdfo` by the standard `help` command.

4.2 Automatic selection of the solver

When invoking the `pdfo` function, the user may specify which solver to call in the backend. In the Python and MATLAB versions, this can be done by setting the argument `method` and the option `solver`, respectively. These names are consistent with those used in `scipy.optimize.minimize` and `fmincon`. However, if the user does not specify a solver or chooses a solver that is incapable of solving the problem (e.g., UOBYQA cannot solve constrained problems), then `pdfo` selects the solver as follows.

1. If the problem is unconstrained, then UOBYQA is selected when $2 \le n \le 8$, and NEWUOA is selected when $n = 1$ or $n > 8$.
2. If the problem is bound-constrained, then BOBYQA is selected.

---

[9] https://numpy.org/doc/stable/f2py .

3. If the problem is linearly constrained, then LINCOA is selected.
4. Otherwise, COBYLA is selected.

The problem type is detected automatically according to the input. In the unconstrained case, we select UOBYQA for small problems because it is more efficient, and the number 8 is set according to our experiments on the CUTEst [28] problems. Note that Powell's implementation of UOBYQA cannot handle univariate unconstrained problems, for which NEWUOA is chosen.

In addition to the `pdfo` function, PDFO provides functions named `cobyla`, `uobyqa`, `newuoa`, `bobyqa`, and
textttlincoa, which invoke the corresponding solvers directly, but it is highly recommended to call the solvers via the `pdfo` function.

4.3 Problem preprocessing

PDFO preprocesses the input of the user in order to fit the data structure expected by Powell's Fortran code.

For example, LINCOA needs a feasible starting point to work properly unless the problem is infeasible. If the starting point is not feasible, then LINCOA would modify the right-hand sides of the linear constraints to make it feasible and then solve the modified problem. Therefore, for linearly constrained problems, PDFO attempts to project the user-provided starting point onto the feasible region before passing the problem to the Fortran code so that a feasible problem will not be modified by LINCOA.

Another noticeable preprocessing of the constraints made by PDFO is the treatment of the linear equality constraints in (12c). As long as these constraints are consistent, we eliminate them and reduce (12) to an $(n - \mathrm{rank}\, A_{\mathrm{E}})$-dimensional problem. This is done using a QR factorization of A_{E}. The main motivation for this reduction comes again from LINCOA, which accepts only linear inequality constraints. An alternative approach is to write a linear equality constraint as two inequalities, but our approach reduces the dimension of the problem, which is beneficial for the efficiency of DFO solvers in general.

4.4 Bug fixes in the Fortran source code

The current version of PDFO patches several bugs in the original Fortran source code, particularly the following ones.

1. The solvers may encounter infinite loops. This happens when the exit conditions of a loop can never be met because variables involved in these conditions become NaN due to floating point exceptions. The user's program will never end if this occurs.
2. The Fortran code may encounter memory errors due to uninitialized indices. This is because some indices are initialized according to conditions that can never be met due to NaN, similar to the previous case. The user's program will crash if this occurs.

In our extensive tests based on the CUTEst problems, these bugs take effect from time to time but not often. They are activated only when the problem is rather ill-conditioned or the inputs are rather extreme. This has been observed, for instance, on the CUTEst problems DANWOODLS, GAUSS1LS, and LAKES with some perturbation and randomization.

Even though these bugs are rarely observed in our tests, it is vital to patch them for two reasons. First, their consequences are severe once they occur. Second, application problems are often more irregular and savage than the testing problems we use, and hence the bugs may be triggered more often than we expect. Nevertheless, PDFO allows the users to call Powell's original Fortran code without these patches by setting the option `classical` to true, which is highly discouraged.

4.5 Handling failures of function evaluations

PDFO tolerates NaN values returned by function evaluations. Such a value can be used to indicate failures of function evaluations, which are common in applications of DFO.

To cope with NaN values, PDFO applies a *moderated extreme barrier*. Suppose that $f(\tilde{x})$ is evaluated to NaN at a certain $\tilde{x} \in \mathbb{R}^n$. PDFO takes the view that $\tilde{x}$ violates a hidden constraint [1, 38]. Hence it replaces NaN with a large but finite number `HUGEFUN` (e.g., 10^{30}) before passing $f(\tilde{x})$ to the Fortran solver, so that the solver can continue to progress while penalizing $\tilde{x}$. Indeed, since Powell's solvers construct trust-region models by interpolation, all points that are close to $\tilde{x}$ will be penalized. Similar things are done when the constraint functions return NaN. A caveat is that setting $f(\tilde{x})$ to `HUGEFUN` may lead to extreme values or even NaN in the coefficients of the interpolation models, but Powell's solvers turn out to be quite tolerant of such values.

The original extreme barrier approach [18, Equation (13.2)] sets `HUGEFUN` to ∞, which is inappropriate for methods based on interpolation. In fact, we also moderate $f(\tilde{x})$ to `HUGEFUN` if it is actually evaluated to ∞. Our approach is clearly naive, but it is better than terminating the solver once the function evaluation fails. In our experiments, this simple approach significantly improves the robustness of PDFO with respect to failures of function evaluation, as will be demonstrated in Subsection 5.2. There do exist other more sophisticated approaches [1], which will be explored in the future.

5 Numerical results

This section presents numerical experiments on PDFO. Since Powell's solvers are widely used as benchmarks in DFO, extensive comparisons with standard DFO solvers are already available in the literature [42, 66]. Instead of repeating such comparisons, the purpose of our experiments is the following.

1. Demonstrate the fact the PDFO is capable of adapting to noise without fine-tuning according to the noise, in contrast to methods based on finite differences. This is done in Subsection 5.1 by comparing PDFO with finite-difference CG and BFGS on unconstrained CUTEst problems.
2. Verify the effectiveness of the moderated extreme barrier mentioned in Subsection 4.5 for handling failures of function evaluations. This is done in Subsection 5.2 by testing PDFO with and without the barrier on unconstrained CUTEst problems.
3. Illustrate the potential of PDFO in hyperparameter optimization problems from machine learning, echoing the observations made in [27] about trust-region DFO methods for such problems. This is done in Subsection 5.3 by comparing PDFO with two solvers from the Python package `hyperopt`.[10]

Our experiments are carried out in double precision based on the Python version of PDFO 2.2.0. The finite-difference CG and BFGS are provided by SciPy 1.11.3. The version of `hyperopt` is 0.2.7. All these packages are tested with the latest stable version at the time of writing. We conduct the test on a ThinkStation P620 with an AMD Ryzen Threadripper PRO 3975WX CPU and 64 GB of memory, the operating system being Ubuntu 22.04, and the Python version being 3.10.12.

5.1 Stability under noise

We first compare PDFO with finite-difference CG and BFGS on unconstrained problems with multiplicative Gaussian noise. We take the view that multiplicative noise makes more sense if the scale of the objective function changes widely, as is often the case in applications.

SciPy provides both CG and BFGS under the `minimize` function in the `scipy.optimize` module. These methods rely on the gradient of the objective function. If we do not provide such information, SciPy approximates it by finite differences, and we do the same in our experiments. For PDFO, we specify NEWUOA as the solver, while setting all the other options to the default ones. In particular, the initial trust-region radius is 1, the final trust-region radius is 10^{-6}, and the number of interpolation points is $2n + 1$, where n is the dimension of the problem being solved. We perform the comparison on 166 unconstrained problems with $n \leq 50$ from the CUTEst [28] problem set using PyCUTEst 1.5.1 [24]. For each testing problem, the starting point is set to the one provided by CUTEst, and the maximal number of function evaluations is $500n$.

Let $\sigma \geq 0$ be the noise level to test. For a testing problem with the objective function f, we define
$$\tilde{f}_\sigma(x) = [1 + \sigma R(x)]f(x), \tag{13}$$
with $R(x) \sim \mathrm{N}(0, 1)$ being independent and identically distributed when x varies. If $\sigma = 0$, then $\tilde{f}_\sigma = f$, corresponding to the noise-free case. In general, σ

---

[10] https://hyperopt.github.io/hyperopt .

is the standard deviation of the noise. The function $\tilde{f}_\sigma$ is the one received by the optimization solvers. In particular, the CG and BFGS methods of SciPy approximate the components of $\nabla \tilde{f}_\sigma(x)$ by the finite difference

$$\frac{\partial \tilde{f}_\sigma}{\partial [x]_i}(x) \approx \frac{\tilde{f}_\sigma(x + h e_i) - \tilde{f}_\sigma(x)}{h},$$

where $[x]_i$ denotes the ith component of x, $e_i \in \mathbb{R}^n$ is the ith standard coordinate vector, and $h > 0$ is the difference parameter

$$h = \operatorname{sign}([x]_i) \max\{|[x]_i|, 1\} \sqrt{u}, \tag{14}$$

where $u \approx 2.2 \times 10^{-16}$ is the unit roundoff. When there is noise ($\sigma > 0$), we also test another value of h, namely

$$h = \sqrt{\sigma \max\{|\tilde{f}_\sigma(x)|, 1\}}. \tag{15}$$

This adaptive difference parameter is inspired by the optimal choice that depends on the second-order information of f (see, e.g., [43] and [67, Equation (2.2)]). However, it relies on the knowledge of the noise level σ. In contrast, PDFO does not require the knowledge of σ and as we will see, provides better performance.

Given a noise level $\sigma \geq 0$ and a convergence tolerance $\tau \in (0, 1)$, we will plot the performance profiles [42] of the solvers on the testing problems. We run all the solvers on all the problems, every objective function being evaluated by its contaminated version (13). For each solver, the performance profile displays the proportion of problems solved with respect to the normalized cost to solve the problem up to the convergence tolerance τ. For each problem, the cost to solve the problem is the number of function evaluations needed to achieve

$$f(x_0) - f(x_k) \geq (1 - \tau)[f(x_0) - f_*], \tag{16}$$

and the normalized cost is this number divided by the minimum cost of this problem among all solvers; we define the normalized cost as infinity if the solver fails to achieve (16) on this problem. Here, x_0 represents the starting point, and [42, § 2.1] suggests that the value f_* should be the least value of f obtained by all solvers. Note that the convergence test (16) uses the values of f and not those of $\tilde{f}_\sigma$. This means that we assess the solvers according to the true objective function values, even though the objective function fed to the solvers is $\tilde{f}_\sigma$, which is contaminated unless $\sigma = 0$.

To make our results more reliable, when $\sigma > 0$, the final performance profile is obtained by averaging the profiles obtained via the above procedure over ten independent runs. In addition, the value f_* in the convergence test (16) is set to the least value of f obtained by all solvers during all these ten runs plus a run with $\sigma = 0$. Finally, for better scaling of the profiles, we plot the binary logarithm of the normalized cost on the horizontal axis, instead of the normalized cost itself.

Figure 1 shows the performance profiles of the solvers for the noise levels $\sigma = 0$, $\sigma = 10^{-10}$, and $\sigma = 10^{-8}$. Two profiles are included for each noise level, with the convergence tolerance being $\tau = 10^{-2}$ and $\tau = 10^{-4}$ respectively. In the legends, "CG" and "BFGS" denote the finite-difference methods using (14), whereas "CG adaptive" and "BFGS adaptive" are their counterparts with h given by (15). Note that the subfigures corresponding to $\sigma = 0$ do not include "CG adaptive" and "BFGS adaptive" because these methods are designed to tackle noisy problems.

In the noise-free case ($\sigma = 0$), PDFO is more efficient than finite-difference CG and BFGS, although the distinction is less visible when τ is smaller, and BFGS can solve slightly more problems than PDFO. When there is noise ($\sigma > 0$), the advantage of PDFO becomes significant. The performances of CG and BFGS using (14) deteriorate considerably under noise, even though the noise level is not high and the convergence tolerance is not demanding. As expected, the adaptive difference parameter (15) improves the performance of CG and BFGS by a large margin, but they are still inferior to PDFO. This shows the advantage of PDFO in noisy settings, even when compared to methods that explicitly use the magnitude of the noise. We have conducted similar experiments with larger values of σ. The results are similar to those shown in Figure 1, except that the advantage of PDFO over CG and BFGS with (15) is even more visible, and the other two methods barely solve any problem. For conciseness, we do not include these results.

It is not surprising that CG and BFGS perform unfavorably when using (14). As mentioned in [43,67,68], the difference parameter should be adapted according to the noise level, as we do in the adaptive versions. In contrast, PDFO does not need such adaptation when handling noisy problems.

To summarize, the performance of finite-difference CG and BFGS is encouraging when there is no noise, yet much more care is needed when the problems are noisy. In contrast, PDFO adapts to noise automatically in our experiment, demonstrating good stability under noise without requiring knowledge about the noise level. This is because Powell's methods (NEWUOA in this experiment) gradually adjust the geometry of the interpolation set during the iterations, making progress until the interpolation points are too close to distinguish noise from true objective function values. This is not specific to Powell's methods but also applies to other algorithms that sample the objective function on a set of points with adaptively controlled geometry, including finite-difference methods with well-chosen difference parameters [67].

5.2 Robustness with respect to failures of function evaluations

We now test the robustness of the solvers when function evaluations fail from time to time. We assume that the objective function returns NaN if the evaluation fails, which occurs randomly with a certain probability. As mentioned in Section 4.5, PDFO uses a moderated extreme barrier to handle such failures. To verify the effectiveness of this approach, we compare PDFO
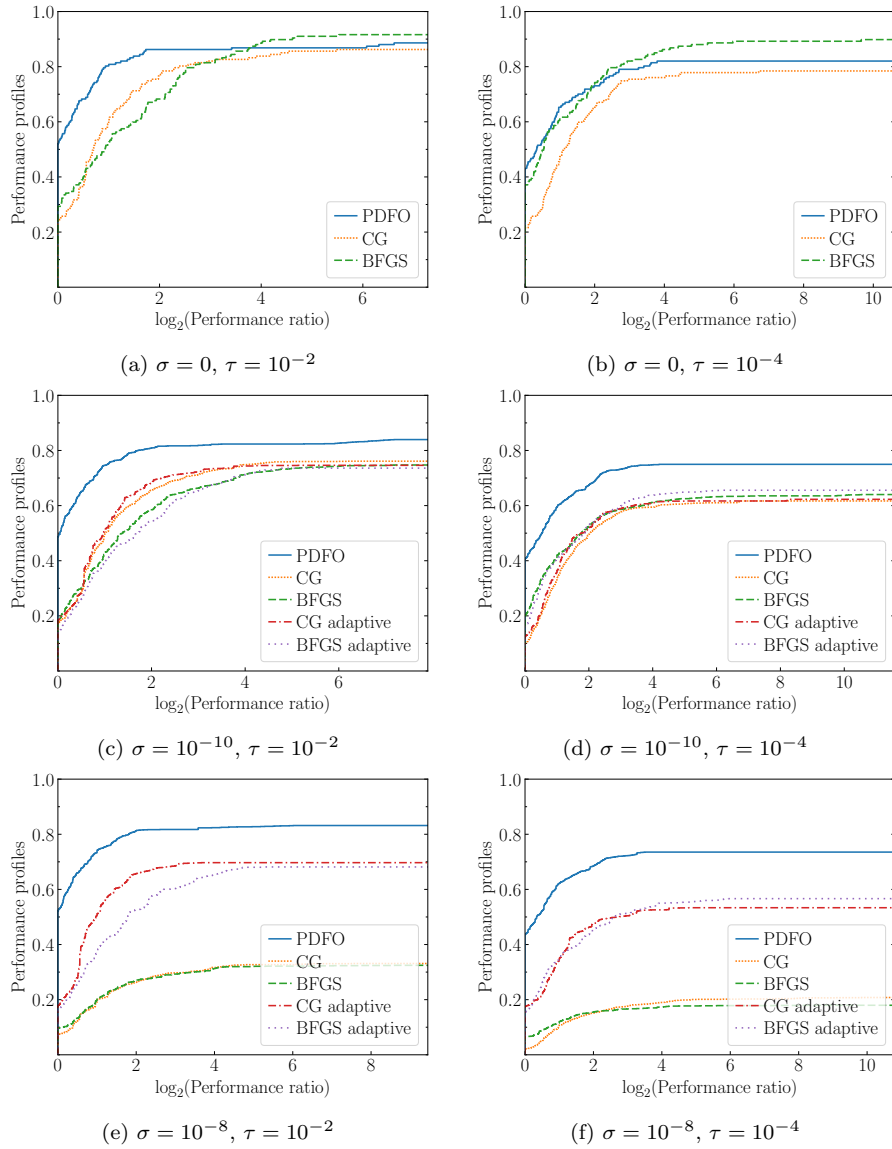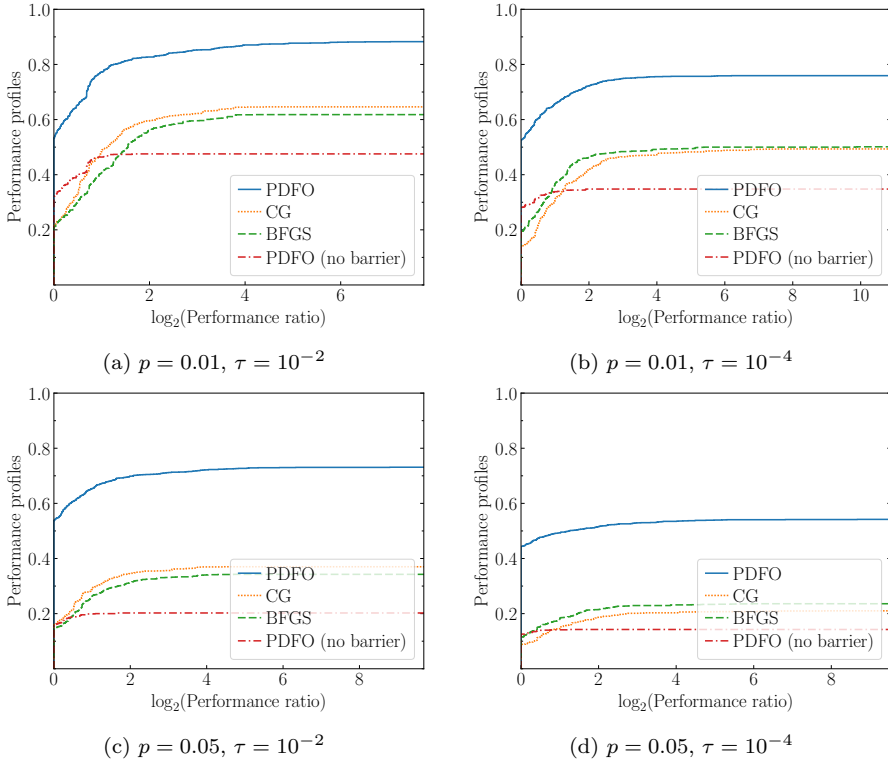
(a) $\sigma = 0$, $\tau = 10^{-2}$

(b) $\sigma = 0$, $\tau = 10^{-4}$

(c) $\sigma = 10^{-10}$, $\tau = 10^{-2}$

(d) $\sigma = 10^{-10}$, $\tau = 10^{-4}$

(e) $\sigma = 10^{-8}$, $\tau = 10^{-2}$

(f) $\sigma = 10^{-8}$, $\tau = 10^{-4}$

Figure 1: Performance profiles of PDFO, CG, and BFGS on unconstrained problems with the objective functions evaluated by $\tilde{f}_\sigma$ in (13)

with its variant that does not apply the barrier. To make the experiment more informative, we also include the finite-difference CG and BFGS tested before, which do not handle evaluation failures particularly. The solvers are set up in the same way as in the previous experiment, and we still employ the 166 unconstrained CUTEst problems used previously.

Let $p \in [0, 1]$ be the failure probability of function evaluations. For a testing problem with the objective function f, we define

$$\hat{f}_p(x) = \begin{cases} f(x) & \text{if } U(x) \geq p, \\ \text{NaN} & \text{otherwise,} \end{cases} \tag{17}$$

where $U(x)$ follows the uniform distribution on $[0, 1]$, being independent and identically distributed when x varies. Note that $\hat{f}_0 = f$. In the experiment, the solvers can evaluate f only via $\hat{f}_p$. We plot the performance profiles of the solvers in a way that is similar to the previous experiment. The profiles are also averaged over ten independent runs. For each problem, the value f_* in the convergence test (16) is set to the least value of f obtained by all solvers during these ten runs plus a run with $p = 0$.

Figure 2 shows the performance profiles of the solvers with $p = 0.01$ and $p = 0.05$. Two profiles are included for each p, with the convergence tolerance being $\tau = 10^{-2}$ and $\tau = 10^{-4}$ respectively.

(a) $p = 0.01$, $\tau = 10^{-2}$

(b) $p = 0.01$, $\tau = 10^{-4}$

(c) $p = 0.05$, $\tau = 10^{-2}$

(d) $p = 0.05$, $\tau = 10^{-4}$

Figure 2: Performance profiles of PDFO, CG, BFGS, and PDFO without barrier on unconstrained problems with the objective functions evaluated by $\hat{f}_p$ in (17)

The contrast is clear. Compared with finite-difference CG and BFGS, PDFO is more efficient and solves significantly more problems given the same convergence tolerance. Moreover, comparing PDFO and its no-barrier counterpart, we see that the moderated extreme barrier improves evidently the robustness of PDFO with respect to failures of function evaluations, even though it is a quite naive approach. When $p = 0.05$, the function evaluation fails roughly once every 20 times, but PDFO can still solve almost 55% of the problems up to the convergence tolerance $\tau = 10^{-4}$ in the sense of (16), whereas all its competitors solve less than 25%. We speculate that the moderated extreme barrier will also benefit other model-based DFO methods, including those based on finite differences. It deserves further investigation in the future.

5.3 An illustration of hyperparameter optimization with PDFO

We now consider a hyperparameter optimization problem from machine learning and illustrate the potential of PDFO for such problems. We compare PDFO with Random Search (RS) and Tree-Structured Parzen Estimator (TPE), two solvers from the Python package `hyperopt` for hyperparameter optimization.

Our experiment is inspired by [27, § 5.3], which investigates the application of trust-region DFO methods to hyperparameter optimization. Similar to [27, § 5.3], we tune the C-SVC model detailed in [11, § 2.1] for binary classifications. This model relies on two hyperparameters: a penalty parameter $C \in (0, \infty)$ and a kernel parameter $\gamma \in (0, \infty)$. As suggested by [11, § 9], we tune C and γ for the performance of the C-SVC. We model this process as solving the problem

$$\max \ P(C, \gamma) \quad \text{s.t.} \quad C > 0, \ \gamma > 0, \tag{18}$$

where $P(C, \gamma)$ measures the performance corresponding to parameters (C, γ). In our experiment, we define P based on the AUC score [27, § 3], which lies in $[0, 1]$ and measures the quality of a classifier on a dataset, the higher the better. More precisely, $P(C, \gamma)$ is set to a five-fold cross-validation AUC score as follows. Split the training dataset $\mathcal{S}$ into five folds, and train the C-SVC five times, each time on a union of four distinct folds. After each training, calculate the AUC score of the resulting classifier on the fold not involved in the training, leading to five scores, the average of which is $P(C, \gamma)$.

Our experiment is based on binary classification problems from LIBSVM,[11] where we adopt three datasets detailed in Table 1. LIBSVM divides each dataset $\mathcal{D}$ into two disjoint subsets, namely a training dataset $\mathcal{S}$ and a testing dataset $\mathcal{T}$. The training in the evaluation of P is done using the `SVC` class of the Python package scikit-learn.[12] We solve (18) by PDFO, RS, and TPE to obtain the tuned parameters $(\bar{C}, \bar{\gamma})$. As in [27, § 5.3], we modify the constraints of (18) to $C \in [10^{-6}, 1]$ and $\gamma \in [1, 10^3]$. For better scaling of the problem, we perform the maximization with respect to $(\log_{10} C, \log_{10} \gamma)$ instead of (C, γ),

---

[11] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets .

[12] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html .

the initial guess being chosen randomly from $[-6, 0] \times [0, 3]$. The solver of PDFO is BOBYQA, for which we set the maximal number of function evaluations to 100. For RS and TPE, we try both 100 and 300 for the maximal number of function evaluations, and they do not terminate until this number is reached.

Table 1: Datasets from LIBSVM

Dataset	Number of features	Size of $\mathcal{S}$	Size of $\mathcal{T}$
splice	60	1,000	2,175
svmguide1	4	3,088	4,000
ijcnn1	22	49,990	91,701

To assess the quality of the tuned parameters $(\bar{C}, \bar{\gamma})$, we train our model on $\mathcal{S}$ with $(C, \gamma) = (\bar{C}, \bar{\gamma})$, and calculate both the AUC score and accuracy of the resulting classifier on $\mathcal{T}$, the latter being the fraction of correctly classified data points. Note that $\mathcal{T}$ is not involved in the tuning process. Table 2 presents the results for this experiment, where $\#P$ denotes the number of evaluations of the function P and "Time" is the computing time for obtaining $(\bar{C}, \bar{\gamma})$.

Table 2: Hyperparameter tuning using PDFO, RS, and TPE

Dataset	Solver	AUC Score	Accuracy	$\#P$	Time (s)
splice	PDFO	0.927	0.737	33	6.56
	RS	0.500	0.520	100	9.16
		0.500	0.520	300	2.74×10^1
	TPE	0.500	0.520	100	8.89
		0.500	0.520	300	2.64×10^1
svmguide1	PDFO	0.995	0.966	51	3.45
	RS	0.994	0.961	100	1.13×10^1
		0.995	0.968	300	3.40×10^1
	TPE	0.995	0.965	100	9.53
		0.995	0.968	300	2.53×10^1
ijcnn1	PDFO	0.997	0.980	44	2.44×10^3
	RS	0.997	0.982	100	3.94×10^3
		0.997	0.976	300	1.14×10^4
	TPE	0.998	0.979	100	3.39×10^3
		0.998	0.979	300	8.36×10^3

In terms of the AUC score and accuracy, PDFO achieves a clearly better result than RS and TPE on the "splice" dataset, and they all attain comparable results on the other datasets. However, PDFO always uses much fewer function evaluations, and hence, much less computing time. The difference in the

computing time is particularly visible on the "ijcnn1" dataset, as each evaluation of P takes much time due to the large data size.

Note that our intention is not to manifest that PDFO outperforms existing approaches for hyperparameter tuning in general, which is unlikely the case. Indeed, PDFO has limitations in handling hyperparameter tuning problems, as many such problems contain discrete variables and cannot be handled by PDFO directly. Our objective is rather to provide an example that shows the possibility of applying Powell's methods to hyperparameter optimization, which is not well studied up to now. In doing so, we also hope to call for more investigation on DFO methods for machine learning problems in general, as is suggested in [27].

6 Concluding remarks

We have presented the PDFO package, which aims at simplifying the use of Powell's DFO solvers by providing user-friendly interfaces. More information about the package can be found on the homepage of the package at `https://www.pdfo.net`, including the detailed syntax of the interfaces, extensive documentation of the options, and several examples to illustrate the usage.

In addition, we have provided an overview of Powell's methods behind PDFO. The overview does not intend to repeat Powell's description of the methods, but rather to provide a summary of the main features and structures of the methods, highlighting the intrinsic connections and similarities among them. We hope that the overview will ease the understanding of Powell's methods, in the same way as the PDFO package eases the use of these methods.

Besides Powell's solvers, PDFO also provides a unified interface for DFO solvers. Such an interface can facilitate the development and comparison of different DFO solvers. The interface can readily accommodate solvers other than those by Powell, for example, the COBYQA (Constrained Optimization BY Quadratic Approximations) solver for general nonlinearly constrained DFO problems (see [62, Chapters 5–7] and [64]).

Finally, we stress that PDFO does not implement Powell's DFO solvers in Python or MATLAB, but only interfaces Powell's implementation with such languages. The implementation of these solvers in Python, MATLAB, and other languages is a project in progress under the name of PRIMA (Reference Implementation for Powell's methods with Modernization and Amelioration) [79].

References

1. Audet, C., Caporossi, G., Jacquet, S.: Binary, unrelaxable and hidden constraints in black-box optimization. Oper. Res. Lett. **48**, 467–471 (2020). DOI 10.1016/j.orl.2020.05.011
2. Audet, C., Dennis Jr., J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**, 188–217 (2006). DOI 10.1137/040603371
3. Audet, C., Hare, W.: Derivative-Free and Blackbox Optimization. Springer Ser. Oper. Res. Financ. Eng. Springer, Cham, Switzerland (2017). DOI 10.1007/978-3-319-68913-5
4. Bandeira, A.S., Scheinberg, K., Vicente, L.N.: Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. Math. Program. **134**, 223–257 (2012). DOI 10.1007/s10107-012-0578-z
5. Bates, D., Mächler, M., Bolker, B.M., Walker, S.C.: Fitting linear mixed-effects models using lme4. J. Stat. Softw. **67**, 1–48 (2015). DOI 10.18637/jss.v067.i01
6. Berahas, A.S., Byrd, R.H., Nocedal, J.: Derivative-free optimization of noisy functions via quasi-Newton methods. SIAM J. Optim. **29**, 965–993 (2019). DOI 10.1137/18M1177718
7. Billups, S.C., Larson, J., Graf, P.: Derivative-free optimization of expensive functions with computational error using weighted regression. SIAM J. Optim. **23**, 27–53 (2013). DOI 10.1137/100814688
8. Buhmann, M.D., Fletcher, R., Iserles, A., Toint, P.: Michael J. D. Powell. 29 July 1936–19 April 2015. Biogr. Mems Fell. R. Soc. **64**, 341–366 (2018). DOI 10.1098/rsbm.2017.0023
9. Cartis, C., Fiala, J., Marteau, B., Roberts, L.: Improving the flexibility and robustness of model-based derivative-free optimization solvers. ACM Trans. Math. Software **45**, 32 (2019). DOI 10.1145/3338517
10. Cartis, C., Roberts, L., Sheridan-Methven, O.: Escaping local minima with local derivative-free methods: a numerical investigation. Optimization **71**, 2343–2373 (2022). DOI 10.1080/02331934.2021.1883015
11. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**, 27 (2011). DOI 10.1145/1961189.1961199
12. Conn, A.R., Le Digabel, S.: Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. Optim. Methods Softw. **28**, 139–158 (2013). DOI 10.1080/10556788.2011.623162
13. Conn, A.R., Scheinberg, K., Toint, Ph.L.: On the convergence of derivative-free methods for unconstrained optimization. In: M.D. Buhmann, A. Iserles (eds.) Approximation Theory and Optimization: Tributes to M. J. D. Powell, pp. 83–108. Cambridge University Press, Cambridge, UK (1997)
14. Conn, A.R., Scheinberg, K., Toint, Ph.L.: Recent progress in unconstrained nonlinear optimization without derivatives. Math. Program. **79**, 397–414 (1997). DOI 10.1007/BF02614326
15. Conn, A.R., Scheinberg, K., Vicente, L.N.: Geometry of interpolation sets in derivative free optimization. Math. Program. **111**, 141–172 (2008). DOI 10.1007/s10107-006-0073-5
16. Conn, A.R., Scheinberg, K., Vicente, L.N.: Geometry of sample sets in derivative-free optimization: polynomial regression and underdetermined interpolation. IMA J. Numer. Anal. **28**, 721–748 (2008). DOI 10.1093/imanum/drn046
17. Conn, A.R., Scheinberg, K., Vicente, L.N.: Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points. SIAM J. Optim. **20**, 387–415 (2009). DOI 10.1137/060673424
18. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization. MPS-SIAM Ser. Optim. SIAM, Philadelphia, PA, USA (2009). DOI 10.1137/1.9780898718768
19. Conn, A.R., Toint, Ph.L.: An algorithm using quadratic interpolation for unconstrained derivative free optimization. In: G. Di Pillo, F. Giannessi (eds.) Nonlinear Optimization and Applications, pp. 27–47. Springer, Boston, MA, USA (1996). DOI 10.1007/978-1-4899-0289-4_3
20. Custódio, A.L., Scheinberg, K., Vicente, L.N.: Methodologies and software for derivative-free optimization. In: T. Terlaky, M.F. Anjos, S. Ahmed (eds.) Advances and Trends in Optimization with Engineering Applications, pp. 495–506. SIAM, Philadelphia, PA, USA (2017). DOI 10.1137/1.9781611974683.ch37

21. Dennis Jr., J.E., Schnabel, R.B.: Least change secant updates for quasi-Newton methods. SIAM Rev. **21**, 443–459 (1979). DOI 10.1137/1021091

22. Dodangeh, M., Vicente, L.N.: Worst case complexity of direct search under convexity. Math. Program. **155**, 307–332 (2016). DOI 10.1007/s10107-014-0847-0

23. Fletcher, R., Powell, M.J.D.: A rapidly convergent descent method for minimization. Comput. J. **6**, 163–168 (1963). DOI 10.1093/comjnl/6.2.163

24. Fowkes, J., Roberts, L., Bűrmen, Á.: PyCUTEst: an open source Python package of optimization test problems. J. Open Source Softw. **7**, 4377 (2022). DOI 10.21105/joss.04377

25. Gallard, F., Vanaret, C., Guénot, D., Gachelin, V., Lafage, R., Pauwels, B., Barjhoux, P.J., Gazaix, A.: GEMS: a Python library for automation of multidisciplinary design optimization process generation. In: 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. AIAA, Kissimmee, FL, USA (2018). DOI 10.2514/6.2018-0657

26. Garmanjani, R., Júdice, D., Vicente, L.N.: Trust-region methods without using derivatives: worst case complexity and the nonsmooth case. SIAM J. Optim. **26**, 1987–2011 (2016). DOI 10.1137/151005683

27. Ghanbari, H., Scheinberg, K.: Black-box optimization in machine learning with trust region based derivative free algorithm. Tech. Rep. 17T-005, COR@L, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA (2017)

28. Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput. Optim. Appl. **60**, 545–557 (2015). DOI 10.1007/s10589-014-9687-3

29. Gratton, S., Royer, C.W., Vicente, L.N., Zhang, Z.: Direct search based on probabilistic descent. SIAM J. Optim. **25**, 1515–1541 (2015). DOI 10.1137/140961602

30. Gratton, S., Royer, C.W., Vicente, L.N., Zhang, Z.: Direct search based on probabilistic feasible descent for bound and linearly constrained problems. Comput. Optim. Appl. **72**, 525–559 (2019). DOI 10.1007/s10589-019-00062-4

31. Hager, W.W.: Updating the inverse of a matrix. SIAM Rev. **31**, 221–239 (1989). DOI 10.1137/1031049

32. Hough, M., Roberts, L.: Model-based derivative-free methods for convex-constrained optimization. SIAM J. Optim. **32**, 2552–2579 (2022). DOI 10.1137/21M1460971

33. Izadinia, H., Shan, Q., Seitz, S.M.: IM2CAD. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2422–2431. IEEE, Honolulu, HI, USA (2017). DOI 10.1109/CVPR.2017.260

34. Kelley, C.T.: Implicit Filtering. Software Environ. Tools. SIAM, Philadelphia, PA, USA (2011). DOI 10.1137/1.9781611971903

35. Kolda, T.G., Lewis, R.M., Torczon, V.: Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev. **45**, 385–482 (2003). DOI 10.1137/S003614450242889

36. Larson, J., Menickelly, M., Wild, S.M.: Derivative-free optimization methods. Acta Numer. **28**, 287–404 (2019). DOI 10.1017/S0962492919000060

37. Le Digabel, S.: Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm. ACM Trans. Math. Software **37**, 44 (2011). DOI 10.1145/1916461.1916468

38. Le Digabel, S., Wild, S.M.: A taxonomy of constraints in black-box simulation-based optimization. Optim. Eng. (2023). DOI 10.1007/s11081-023-09839-3

39. Mamon, G.A., Biviano, A., Boué, G.: MAMPOSSt: modelling anisotropy and mass profiles of observed spherical systems I. Gaussian 3D velocities. Mon. Not. R. Astron. Soc. **429**, 3079–3098 (2013). DOI 10.1093/mnras/sts565

40. Mombaur, K., Truong, A., Laumond, J.P.: From human to humanoid locomotion—an inverse optimal control approach. Auton. Robot. **28**, 369–383 (2010). DOI 10.1007/s10514-009-9170-7

41. Moré, J.J., Sorensen, D.C.: Computing a trust region step. SIAM J. Sci. Stat. Comp. **4**, 553–572 (1983). DOI 10.1137/0904038

42. Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**, 172–191 (2009). DOI 10.1137/080724083

43. Moré, J.J., Wild, S.M.: Estimating derivatives of noisy simulations. ACM Trans. Math. Software **38**, 19 (2012). DOI 10.1145/2168773.2168777

44. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**, 308–313 (1965). DOI 10.1093/comjnl/7.4.308

45. Oeuvray, R., Bierlaire, M.: BOOSTERS: a derivative-free algorithm based on radial basis functions. Int. J. Model. Simul. **29**, 29–36 (2009). DOI 10.1080/02286203.2009.11442507

46. Porcelli, M., Toint, Ph.L.: BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables. ACM Trans. Math. Software **44**, 6 (2017). DOI 10.1145/3085592

47. Porcelli, M., Toint, Ph.L.: Exploiting problem structure in derivative free optimization. ACM Trans. Math. Software **48**, 6 (2022). DOI 10.1145/3474054

48. Powell, M.J.D.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. Comput. J. **7**, 155–162 (1964). DOI 10.1093/comjnl/7.2.155

49. Powell, M.J.D.: A new algorithm for unconstrained optimization. In: J.B. Rosen, O.L. Mangasarian, K. Ritter (eds.) Nonlinear Programming, pp. 31–65. Academic Press, Madison, WI, USA (1970). DOI 10.1016/B978-0-12-597050-1.50006-3

50. Powell, M.J.D.: A direct search optimization method that models the objective and constraint functions by linear interpolation. In: S. Gomez, J.P. Hennart (eds.) Advances in Optimization and Numerical Analysis, *Math. Appl.*, vol. 275, pp. 51–67. Springer, Dordrecht, Netherlands (1994). DOI 10.1007/978-94-015-8330-5_4

51. Powell, M.J.D.: Direct search algorithms for optimization calculations. Acta Numer. **7**, 287–336 (1998). DOI 10.1017/S0962492900002841

52. Powell, M.J.D.: On the Lagrange functions of quadratic models that are defined by interpolation. Optim. Methods Softw. **16**, 289–309 (2001). DOI 10.1080/10556780108805839

53. Powell, M.J.D.: UOBYQA: unconstrained optimization by quadratic approximation. Math. Program. **92**, 555–582 (2002). DOI 10.1007/s101070100290

54. Powell, M.J.D.: Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. Math. Program. **100**, 183–215 (2004). DOI 10.1007/s10107-003-0490-7

55. Powell, M.J.D.: On updating the inverse of a KKT matrix. In: Y. Yuan (ed.) Numerical Linear Algebra and Optimization, pp. 56–78. Science Press, Beijing, China (2004)

56. Powell, M.J.D.: The NEWUOA software for unconstrained optimization without derivatives. In: G. Di Pillo, M. Roma (eds.) Large-Scale Nonlinear Optimization, *Nonconvex Optim. Appl.*, vol. 83, pp. 255–297. Springer, Boston, MA, USA (2006). DOI 10.1007/0-387-30065-1_16

57. Powell, M.J.D.: Developments of NEWUOA for minimization without derivatives. IMA J. Numer. Anal. **28**, 649–664 (2008). DOI 10.1093/imanum/drm047

58. Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. Rep. DAMTP 2009/NA06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK (2009)

59. Powell, M.J.D.: On the convergence of trust region algorithms for unconstrained minimization without derivatives. Comput. Optim. Appl. **53**, 527–555 (2012). DOI 10.1007/s10589-012-9483-x

60. Powell, M.J.D.: Beyond symmetric Broyden for updating quadratic models in minimization without derivatives. Math. Program. **138**, 475–500 (2013). DOI 10.1007/s10107-011-0510-y

61. Powell, M.J.D.: On fast trust region methods for quadratic models with linear constraints. Math. Program. Comput. **7**, 237–267 (2015). DOI 10.1007/s12532-015-0084-4

62. Ragonneau, T.M.: Model-based derivative-free optimization methods and software. Ph.D. thesis, Department of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong, China (2022). URL https://theses.lib.polyu.edu.hk/handle/200/12294

63. Ragonneau, T.M., Zhang, Z.: An optimal interpolation set for model-based derivative-free optimization methods (2023). DOI 10.48550/arXiv.2302.09992

64. Ragonneau, T.M., Zhang, Z.: COBYQA: Constrained Optimization BY Quadratic Approximations (2024). URL https://www.cobyqa.com

65. Regis, R.G., Wild, S.M.: CONORBIT: constrained optimization by radial basis function interpolation in trust regions. Optim. Methods Softw. **32**, 552–580 (2017). DOI 10.1080/10556788.2016.1226305

66. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. J. Global Optim. **56**, 1247–1293 (2013). DOI 10.1007/s10898-012-9951-y

67. Shi, H.J.M., Xie, Y., Xuan, M.Q., Nocedal, J.: Adaptive finite-difference interval estimation for noisy derivative-free optimization. SIAM J. Sci. Comput. **44**, A2302–A2321 (2022). DOI 10.1137/21M1452470

68. Shi, H.J.M., Xuan, M.Q., Oztoprak, F., Nocedal, J.: On the numerical performance of derivative-free optimization methods based on finite-difference approximations. Optim. Methods Softw. **38**, 289–311 (2023). DOI 10.1080/10556788.2022.2121832

69. Steihaug, T.: The conjugate gradient method and trust regions in large scale optimization. IMA J. Numer. Anal. **20**, 626–637 (1983). DOI 10.1137/0720042

70. Toint, Ph.L.: Towards an efficient sparsity exploiting Newton method for minimization. In: I.S. Duff (ed.) Sparse Matrices and Their Uses, pp. 57–88. Academic Press, New York, NY, USA (1981)

71. Torczon, V.: On the convergence of pattern search algorithms. SIAM J. Optim. **7**, 1–25 (1997). DOI 10.1137/S1052623493250780

72. Vicente, L.N.: Worst case complexity of direct search. EURO J. Comput. **1**, 143–153 (2013). DOI 10.1007/s13675-012-0003-7

73. Wild, S.M.: MNH: a derivative-free optimization algorithm using minimal norm Hessians. In: The Tenth Copper Mountain Conference on Iterative Methods (2008)

74. Wild, S.M., Regis, R.G., Shoemaker, C.A.: ORBIT: optimization by radial basis function interpolation in trust-regions. SIAM J. Sci. Comput. **30**, 3197–3219 (2008). DOI 10.1137/070691814

75. Xie, P., Yuan, Y.: Least H^2 norm updating quadratic interpolation model function for derivative-free trust-region algorithms (2023). DOI 10.48550/arXiv.2302.12017

76. Zhang, H., Conn, A.R., Scheinberg, K.: A derivative-free algorithm for least-squares minimization. SIAM J. Optim. **20**, 3555–3576 (2010). DOI 10.1137/09075531X

77. Zhang, Z.: On derivative-free optimization methods (in Chinese). Ph.D. thesis, Chinese Academy of Sciences, Beijing, China (2012)

78. Zhang, Z.: Sobolev seminorm of quadratic functions with applications to derivative-free optimization. Math. Program. **146**, 77–96 (2014). DOI 10.1007/s10107-013-0679-3

79. Zhang, Z.: PRIMA: Reference Implementation for Powell's methods with Modernization and Amelioration (2024). URL http://www.libprima.net

Statements and declarations

Competing interests. The authors have no relevant financial or non-financial interests to disclose.

Data availability. The code and documentation of PDFO are available at https://www.pdfo.net . The source code of the numerical experiments is available at https://www.github.com/pdfo/paper/blob/main/experiments .