PDFO: A Package of Derivative-Free Optimization Based on Powell's Solvers

Tom M. Ragonneau[*] Zaikun Zhang[†]

Abstract

During his phenomenal career, late Prof. M. J. D. Powell spearheaded many computational techniques in optimization and strived to create theoretical and practical upstanding algorithms. In the derivative-free context, he implemented five methods for constrained and unconstrained problems, namely UOBYQA, NEWUOA, BOBYQA, LINCOA, and COBYLA, renowned for their efficiency and robustness. However, the implementation of these algorithms has been originally made in Fortran, damping the potential enthusiasm of most practitioners for their use on most applications in our present-day world.

In this paper, we introduce PDFO, a cross-platform package providing MATLAB® and Python interfaces for using the above-mentioned Powell's solvers. A brief review of these methods is also made, together with a comparison with different Bayesian optimization algorithms, showing the usefulness of the package on different practical examples.

1 Introduction

Most optimization algorithms rely on (classical or generalized) derivative information of the objective function and constraints. 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. Optimization problems of such kind arise from automatic error analysis [32, 31], machine learning [27], analog circuit design [38], aircraft engineering [26], and chemical product design [62], to name but a few. These problems motivate the development of optimization algorithms that use only function values but not derivatives.

[*]Department of Applied Mathematics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong (`tom.ragonneau@connect.polyu.hk`). Support for this author was provided by the Research Grants Committee of Hong Kong under the Hong Kong Ph.D. Fellowship Scheme (ref. no. PF18-24698).

[†]Department of Applied Mathematics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong (`zaikun.zhang@polyu.edu.hk`). Support for this author was partially provided by the Research Grants Committee of Hong Kong (ref. no. PolyU 253012/17P and PolyU 153054/20P) and The Hong Kong Polytechnic University under the project P0009767.

Between 1994 and 2015, Powell developed five solvers to tackle unconstrained and constrained problems without using derivatives, namely COBYLA, UOBYQA, NEWUOA, BOBYQA, and LINCOA. These solvers were implemented by Powell, with particular attention paid to their numerical stability and algebraic complexity. Renowned for their robustness and efficiency, these solvers are extremely appealing to practitioners and widely used in applications, for instance, aeronautical engineering [25], astronomy [8, 39], computer vision [33], robotics [40], and statistics [4].

However, Powell coded in Fortran 77, an old-fashion language that damps the enthusiasm of many users to exploit these solvers in their projects. There have been a considerable demand from both researchers and practitioners for the availability of Powell's solvers in more user-friendly languages such as Python, MATLAB®, Julia, and R. Our aim is to wrap Powell's Fortran code into a package, namely PDFO, which enables users of such languages to call Powell's solvers without any need of dealing with the Fortran code. For each supported language, PDFO provides a simple subroutine 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 1.1) of PDFO supports Python and MATLAB®, with more languages to be covered in the future. The signature of the Python subroutine is consistent with the `minimize` function in `scipy.optimize`; the signature of the MATLAB® subroutine is consistent with the `fmincon` function in the Optimization Toolbox. The package is cross-platform, available on Linux, macOS, and Microsoft Windows at once.

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 in response to the continual demands from both researchers and practitioners: Py-BOBYQA [9] provides a Python implementation of BOBYQA; NLopt [34] includes multi-language interfaces for COBYLA, NEWUOA, and BOBYQA; `minqa` [5] wraps UOBYQA, NEWUOA, and BOBYQA in R; SciPy [64] makes COBYLA available in Python under the `optimize` subpackage. Nevertheless, PDFO has several features that distinguishes itself 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. In addition to homogenizing the usage, such an interface eases the comparison between these solvers in case multiple of them are able to tackle a given problem.
2. Solver selection. When using PDFO, the user can specifically call one of Powell's solvers; nevertheless, if the user does not specify any solver, PDFO will select automatically a solver according to the given problem. The selection takes into consideration the performance of the solvers on the CUTEst [28] problem set. Interestingly, it turns out that the solver with the best performance may not be the most intuitive one. For example, NEWUOA is not always the best choice for solving an unconstrained problem.
3. Code patching. During the development of PDFO, we spotted in the original Fortran code some bugs, which led to infinite cycling or segmentation faults on some ill-conditioned problems. The bugs have been patched in PDFO. Nevertheless, we provide an option (namely, `classical`) that can enforce the package to use the original code of Powell without the patches, which is not recommended except for research. In addition, PDFO provides

COBYLA in double precision, whereas Powell used single precision when he implemented it in the 1990s.

4. Fault tolerance. PDFO takes care of failures in the evaluation of the objective or constraint functions when NaN or infinite values are returned. In case of such failures, PDFO will not exit but try to progress. Moreover, PDFO ensures that the returned x is not a point where the evaluation fails, while the original code of Powell may return an x whose objective function value is numerically NaN.
5. Problem preprocessing. PDFO preprocesses the inputs to simplify the problem and reformulate it to meet the requirements of Powell's solvers. For instance, if the problem has linear constraints $A_{\mathcal{E}}x = b_{\mathcal{E}}$, PDFO can rewrite it into a problem on the null space of $A_{\mathcal{E}}$, eliminating such constraints and reducing the dimension. Another example is that the starting point of a linearly constrained problem is projected to the feasible region, because LINCOA needs a feasible starting point to work properly.
6. Additional options. PDFO includes options for the user to control the solvers in some manners that are useful in practice. For example, by setting the option `scale` to `true`, the user can request PDFO to scale the problem according to the bounds of the variables before solving it.

The organization of this paper is as follows: Section 3 presents a brief description of the five Powell's derivative-free optimization (DFO) algorithms and Section 4 introduces PDFO, a cross-platform package providing MATLAB® and Python interfaces for using the above-mentioned solvers.

2 A brief review of DFO methods

Consider a nonlinear optimization problem

$$\min_{x\in\Omega} f(x), \tag{2.1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is the objective function and $\Omega \subseteq \mathbb{R}^n$ refers to the feasible region of the problem. Broadly speaking, two strategies have been developed to tackle the problem (2.1) without using derivatives. The first strategy, known as direct search, explores the objective function f and chooses iterates by simple comparisons of function values, examples including the Nelder-Mead algorithm [43], the MADS methods [2, 1, 21], and BFO [46]. See [36], [17, Chapters 7 and 8], [3, Part 3], and [37, Section 2.1] for more discussions on this paradigm, and [29, 30] for recent developments on randomized direct search. The second strategy approximates the original problem (2.1) by simple models and locates the iterates according to such models. Algorithms using this strategy are referred to as model-based methods, and they often make use of the models within a trust-region framework [11, 16, 67], the kth iteration of which locates a trial point by solving approximately

$$\min_{x\in\Omega_k} \quad \widehat{f}_k(x) \tag{2.2a}$$

$$\text{s.t.} \quad \|x - x_k\| \le \Delta_k, \tag{2.2b}$$

where x_k is the current iterate, $\widehat{f}_k$ stands for the model of the objective function f, Δ_k denotes the trust-region radius, $\|\cdot\|$ is a norm in $\mathbb{R}^n$, and $\Omega_k \subseteq \mathbb{R}^n$ represents some model of Ω around x_k. The trial point is then accepted if it satisfies some sufficient decrease condition. The models can also be exploited within a line-search framework [6, 60]. It is worth noting that methods using finite-difference approximation of gradients can also be regarded as model-based methods, because such approximation essentially comes from linear or quadratic models of the function under consideration. Hybrids between direct search and model-based approaches exist, for example [20], [35, Algorithm 4.7], and [12]. For more extensive discussions on DFO methods and theory, see the monographs [17, 3], the survey papers [58, 19, 37], and the references therein.

Powell's DFO algorithms are model based trust-region methods, where the models are obtained using quadratic or linear polynomial interpolations, and the trust regions are defined in the 2-norm.

Given an finite set $\mathcal{Y}_k \subseteq \mathbb{R}^n$ of points lying in a region around the iterate x_k, the model $\widehat{f}_k$ then satisfies the interpolation conditions

$$\widehat{f}_k(y) = f(y), \quad y \in \mathcal{Y}_k. \tag{2.3}$$

A crucial aspect of a derivative-free trust-region method based on interpolation is the mechanism it employs to define the set $\mathcal{Y}_{k+1}$. It should recycle points from previous iterations, at which the objective function has already been evaluated, without damaging some geometrical properties to maintain models with good-enough accuracies and ensure convergence [14, 15, 23]. Most of the existing trust-region methods use polynomial models at most quadratic, although some attempts have been made to employ radial basis functions (RBFs), such as ORBIT [66], CONORBIT [57], and BOOSTERS [44]. We focus however our attention on polynomial models of the form

$$\widehat{f}_k(x) \overset{\mathsf{def}}{=} g_k^{\mathsf{T}}(x - x_k) + (x - x_k)^{\mathsf{T}} B_k (x - x_k), \tag{2.4}$$

with $g_k \in \mathbb{R}^n$ and $B_k \in \mathbb{R}^{n \times n}$ being symmetric and may be zero for linear models, diagonal or general depending on the considered method. Such models admit respectively $n + 1$, $2n + 1$ and $(n+1)(n+2)/2$ coefficients, determined so that they satisfy the interpolation conditions (2.3). Whenever the models are defined by underdetermined interpolations, the freedom bequeathed by the interpolation conditions is usually taken up by a variational problem whose constraints are the conditions (2.3). Examples of such variational problems are the minimization of the Hessian of the models in Frobenius norm [13, 65], or the minimization of a term involving g_k and B_k in (2.4) [18]. Powell employed another example of such variational problem, namely the symmetric Broyden formula [48, 50], which will be studied later in this paper in Section 3.

3 Powell's derivative-free methods

Throughout his career, late Prof. M. J. D. Powell FRS developed five DFO trust-region algorithms. At the dawn of the modern DFO, Powell introduced in 1994 his solver COBYLA [47], named after "constrained optimization by linear approximation". It attempts to solve the programming problem (2.1) when the feasible region Ω is general and expressed as

$$\Omega \overset{\mathsf{def}}{=} \bigcap_{i=1}^{m} \{x \in \mathbb{R}^n : c_i(x) \geq 0\},$$

where $c_i \colon \mathbb{R}^n \to \mathbb{R}$ denotes the ith nonlinear constraint function, with $i \in \{1, 2, \dots, m\}$. At each iteration, it approximates the objective and constraint functions with linear models obtained by fully-determined interpolations on $\mathcal{Y}_k$, whose cardinal is fixed to $n + 1$. In this context, the poisedness [59, §1] of the interpolation set, i.e., the geometrical condition on $\mathcal{Y}_k$ required to ensure the uniqueness of the models, can be seen as the volume of the n-simplex engendered by $\mathcal{Y}_k$ to be nonzero. Once the linear models $\widehat{c}_{i,k}$ of the constraint functions c_i for $i \in \{1, 2, \dots, m\}$ are built, an approximation Ω_k of Ω is obtained as the intersection of the half-spaces generated by the linear approximations of the constraint functions, that is

$$\Omega_k \overset{\mathsf{def}}{=} \bigcap_{i=1}^{m} \{x \in \mathbb{R}^n : \widehat{c}_{i,k}(x) \geq 0\}, \tag{3.1}$$

and the trust-region subproblem (2.2) is then solved. As a linear programming problem, the trust-region subproblem is solved in the following way. Solving sequentially the problems (2.2) by replacing the trust-region radius Δ_k by a constant continuously evolving from 0 to Δ_k would generate a continuous piecewise-linear path from x_k to the solution of the subproblem. The strategy of COBYLA is to compute this path, by updating the active sets of the constraint models. However, the trust-region condition (2.2b) and the region (3.1) may contradict each others, in which case the trial point is chosen to solve approximately

$$\begin{aligned} \max_{x \in \mathbb{R}^n} \quad & \min_{i=1,2,\dots,m} \widehat{c}_{i,k}(x) \\ \text{s.t.} \quad & \|x - x_k\| \leq \Delta_k. \end{aligned}$$

In doing so, the method attempts to reduce some violation of the constraint models while ensuring that the trial point lies in the trust region, as it is needed by most global convergence properties.

Later on, in 2002, Powell developed UOBYQA [56], named after “unconstrained optimization by quadratic approximation”. It aims at solving the nonlinear programming problem (2.1) in the unconstrained case, i.e., when $\Omega = \mathbb{R}^n$. To do so, at each iteration, it models the objective function with a quadratic of the form (2.4), obtained by fully-determined interpolation on the set $\mathcal{Y}_k$ containing $N = (n + 1)(n + 2)/2$ points. The set $\mathcal{Y}_{k+1}$ differs from $\mathcal{Y}_k$ of at most one point. During a classical iteration, a trial point, i.e., an approximate solution of the trust-region subproblem (2.2) replaces an interpolation of $\mathcal{Y}_k$. As we already mentioned, it is essential to maintain some geometrical properties of $\mathcal{Y}_k$ to ensure the existence and the uniqueness of the models from a computational viewpoint. Hence, UOBYQA may undertake geometry-improvement steps whenever the models seem not to be accurate, led a remarkable result pointed out in [52]. It states that given a point $\bar{y} \in \mathcal{Y}_k$ to remove from the interpolation set, its most suitable substitute to build $\mathcal{Y}_{k+1}$ solves the subproblem

$$\begin{aligned} \max_{x \in \mathbb{R}^n} \quad & |\ell_{\bar{y}}(x)| \\ \text{s.t.} \quad & \|x - x_k\| \le \Delta_k, \end{aligned} \tag{3.2}$$

where $\ell_{\bar{y}} \colon \mathbb{R}^n \to \mathbb{R}$ denotes the Lagrange function associated with $\bar{y}$, defined by $\ell_{\bar{y}}(y) = \delta_{\bar{y},y}$ for each $y \in \mathcal{Y}_k$, where $\delta_{\bar{y},y}$ is the Kronecker delta. Since UOBYQA requires only a rough solution of the geometry-improvement subproblem (3.2), Powell developed an algorithm that requires only $\mathcal{O}(n^2)$ operations, based on an estimation of $|\ell_{\bar{y}}(\cdot)|$. Once such a point is calculated, the solver continues with a classical trust-region step, and the subproblem (2.2) is solved with the Moré-Sorensen algorithm [41], as it controls the accuracy of the solution.

The major flaw of UOBYQA is the amount of required interpolation points, which can become prohibitively huge when n increases. Therefore, Powell designed a mechanism to define quadratic models of the form (2.4) that requires fewer interpolation points [50]. Given the set $\mathcal{Y}_k$, containing typically $\mathcal{O}(n)$ points (the default value $2n+1$ being recommanded), the model $\widehat{f}_k$ is the quadratic satisfying the interpolation conditions whose Hessian's update is least in Frobenius norm, that is

$$\begin{aligned} \widehat{f}_k \overset{\mathsf{def}}{=} \operatorname*{arg\,min}_{Q \in \mathcal{P}_{n,2}} \quad & \|\nabla^2 Q - \nabla^2 \widehat{f}_{k-1}\|_{\mathsf{F}} \\ \text{s.t.} \quad & Q(y) = f(y), \quad y \in \mathcal{Y}_k, \end{aligned} \tag{3.3}$$

where $\mathcal{P}_{n,d}$ is the space of n-variate polynomials of degree at most d and $\nabla^2 \widehat{f}_{-1}$ is set to be zero. If the number of interpolation points was less than or equal to $n+1$, the models would remain linear, so that Powell requires the number of points in $\mathcal{Y}_k$ to be at least $n+2$ and at most $(n+1)(n+2)/2$. As we mentioned earlier, the choice of the variational problem (3.3) is fostered by the fact that it occurs in the quasi-Newton update for unconstrained optimization when the first derivative of the objective function is available [24, §3.6]. As in the fully-determined case, the geometry of the interpolation set plays a crucial role, as it influences the accuracy of the quadratic models, and the geometry-improvement steps (3.2) should be also entertained.

The last three DFO solvers of Powell are NEWUOA [55, 49], BOBYQA [54], and LINCOA [51]. BOBYQA and LINCOA are named respectively after "bound optimization by quadratic approximation" and "linearly-constrained optimization algorithm", but the meaning of the acronym NEWUOA is not known (even though most people agree on "new unconstrained optimization algorithm"). Powell never published a paper introducing LINCOA, and [51] discusses only the resolution of its trust-region subproblem. As their names suggest, they aim at solving unconstrained, bound-constrained and linearly-constrained problems respectively, using quadratic models of the objective function of the form (2.4). All three use the underdetermined interpolation technique described above to build the quadratic models, so that NEWUOA is more suitable than UOBYQA for solving problems with a relatively high dimension. They all set Ω_k in the trust-region subproblem (2.2) to be Ω, corresponding the whole space for NEWUOA, a box for BOBYQA and a polyhedron for LINCOA. A subtlety of BOBYQA is that the constraints are always respected, for each iterate and each point in $\mathcal{Y}_k$. Therefore, the geometry-improvement subproblem (3.2) should be adapted to incorporate the bound constraints, which makes its resolution more difficult. Moreover, as we already mentioned, at most one point of the interpolation set is altered at each iteration, which leads to an at-most rank-2 update of the Karush-Kuhn-Tucker (KKT) matrix of the variational problem (3.3). This remark suggests that it can be much more efficient to update such KKT matrix instead of computing it from scratch at each iteration. Powell derived an updating formula in [53] that requires only $\mathcal{O}(N^2)$ operations instead of $\mathcal{O}(N^3)$ if the computation was made from scratch with no loss of accuracy, where N denotes the number of interpolation points. When it comes to solving the trust-region subproblems (2.2), NEWUOA employs the Steihaug-Toint truncated conjugate gradient algorithm [61, 63], and BOBYQA and LINCOA use an active-set variation of it. However, the trust-region subproblem solver of BOBYQA always respects the bounds, while the solver of LINCOA may visit point lying outside of the polyhedron of the constraints, and may even return points that are slightly infeasible.

4 An interface for the Powell's derivative-free solvers

Powell implemented all five methods in Fortran, in a very robust and efficient manner. However, less and less people are using Fortran in our present-day world, and Fortran has quite old standards. The authors developed therefore PDFO[1], named after "Powell's derivative-free optimization solvers". It is cross-platform package providing MATLAB® and Python interfaces for using all five Powell's DFO solvers, available for Linux, macOS and Windows. PDFO does not reimplement Powell's solvers, but rather links the modern languages MATLAB® and Python with the Fortran source code. At a low-level, it uses F2PY [45] to interface Python with Fortran and MEX to interface it with MATLAB®.

4.1 Main structure of the package

The philosophy of PDFO is simple: providing to users a single function to solve a DFO problem. It takes for input an optimization problem of the form

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{4.1a}$$

$$\text{s.t.} \quad l \le x \le u, \tag{4.1b}$$

$$A_{\mathcal{I}} x \le b_{\mathcal{I}}, \quad A_{\mathcal{E}} x = b_{\mathcal{E}}, \tag{4.1c}$$

$$c_{\mathcal{I}}(x) \le 0, \quad c_{\mathcal{E}}(x) = 0, \tag{4.1d}$$

where $l, u \in (\mathbb{R} \cup \{\pm\infty\})^n$, $A_{\mathcal{E}}$ and $A_{\mathcal{I}}$ are real matrices, $b_{\mathcal{E}}$ and $b_{\mathcal{I}}$ are real vectors, and $c_{\mathcal{E}}$ and $c_{\mathcal{I}}$ are multivariate functions. The problem (4.1) covers every possible case, since all matrices, vectors and functions can be set to zero from a mathematical viewpoint and PDFO does not require the constraints (4.1b), (4.1c), and (4.1d) to be provided. A broad example of use in MATLAB® is shown in Listing 1, where variable names have clear correspondances with the

[1]Available at `https://www.pdfo.net/`.

problem (4.1). The value returned by PDFO is the best point calculated, evaluated via a merit function. PDFO for MATLAB® also returns the corresponding optimal value, together with different fields describing the backend calculations and their behaviors.

Listing 1: An elementary example of PDFO in MATLAB®.

```
x = pdfo(@fun, x0, Aineq, bineq, Aeq, beq, lb, ub, @nonlcon);

function fx = fun(x)
...
return
end

function [cineq, ceq] = nonlcon(x)
...
return
end
```

The package PDFO preprocesses the arguments provided by the user, detects the type of the problem and then invokes the Powell's solver that match the best the given problem. The initial preprocessing of the arguments includes a handling of their internal types together with some programming-related procedures to allow as much freedom in the problem definition as possible, to make the use of PDFO as easy as possible. More importantly, PDFO preprocesses the constraints provided, to generate a problem as simple as possible. For instance, all linear constraints in (4.1c) that are satisfied for every point in $\mathbb{R}^n$ are removed and obvious infeasibility in the constraints (4.1b) and (4.1c) are detected. Another noticeable preprocessing of the constraints made by PDFO is the treatment of the linear equality constraints in (4.1c). As long as these constraints are consistent, they define a subspace of $\mathbb{R}^n$ of lower dimension, and PDFO takes into account this property to generate a new $(n - \operatorname{rank} A_{\mathcal{E}})$-dimensional problem that is exactly equivalent to (4.1), using the QR factorization of $A_{\mathcal{E}}$.

A crucial point of BOBYQA and LINCOA is that they require the initial guess to be feasible, so that PDFO attempts to project the provided initial guess onto the feasible set (LINCOA would otherwise increase the coefficients of the right-hand side of the linear constraints to make the initial guess feasible). Another main feature of PDFO is its solver selection mechanism. When a constrained problem is received, the selected solver is the one corresponding to the most general constraint provided. For example, when PDFO receives a problem that admits both bound constraints (4.1b) and linear constraints (4.1c), LINCOA will be chosen. It is possible on some examples that LINCOA gives better results than BOBYQA on bound-constrained problems. This is likely because BOBYQA is a feasible method, while LINCOA may visit infeasible points (but on an engineering problem, these points may be unassessable). At last, when PDFO receives an unconstrained problem, it will attempt to solve it with UOBYQA when its size is reasonable ($2 \leq n \leq 8$, say), and with NEWUOA otherwise. We note that UOBYQA cannot handle problem with univariate objective function.

The authors wanted to keep the source code of all solvers in its original states, but introduced some minor revisions and corrections. A new parameter has been added to allow each solver to exit whenever a target value has been reached, and a flag of termination has been included in the returned values. A revision has also been made to the source code of COBYLA. In the original version, trial points may be discarded prematurely, before the update of the penalty coefficient of the merit function. This has been revised in PDFO. The authors also detected some minor bugs on failure exits, for which some returned values might not have been updated. Although very rarely, LINCOA might moreover encounter infinite cycling, even on well-conditioned problems. These bugs have been patched. PDFO also handle more carefully ill-conditioned problems, for which NaN values (resulting, e.g., from a division by zero) may occur. These values might cause infinite cycling or segmentation faults in the original code. In the early stage of PDFO, such errors occurred on the CUTEst problems [28] DANWOODLS or GAUSS1LS for example. NaN values detected in the objective or constraint functions are managed using extreme barriers, but NaN values encountered in the variables internal to the Fortran code result in early exit. Besides, when interfacing Fortran with MATLAB® and Python, rounding errors occur in the problem's variables, which led in extreme cases to failures. For example, when using PDFO, the user may provided initial and final trust-region radii Δ_0 and Δ_∞ (respectively set to 1 and 10^{-6} by default). If these values are chosen to be very close, although the condition $\Delta_0 \leq \Delta_\infty$ is satisfied in the MATLAB® or Python code, the Fortran code may receive perturbed values with $\Delta_0 > \Delta_\infty$, leading to failure exit in the original code. Therefore, the conditions required by Powell are ensured in the Fortran code directly.

4.2 Some numerical results

4.2.1 Comparison on the CUTEst library

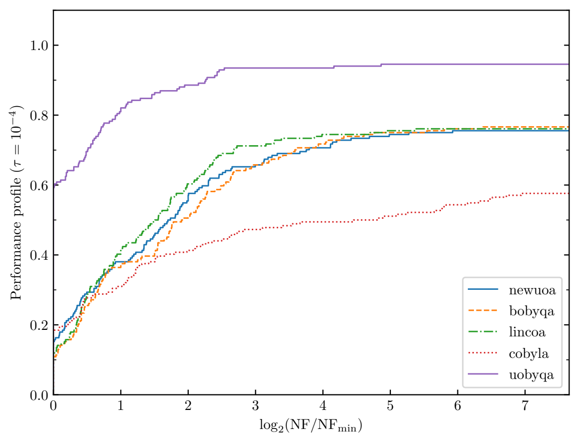
We first make a comparison of the PDFO's solvers on different problems from the CUTEst library [28]. Performance profiles [22, 42] on unconstrained problem of dimensions at most 10 and 50 are provided respectively in Figures 1a and 1b. Broadly speaking, a performance profile plots the proportion of problem solved with respect to the number of function evaluations required to achieve convergence, in logarithmic scale. The optimal value f_* of a given problem is considered to be the least value reached by each solver, and an execution is considered convergent up to a tolerance $\tau \geq 0$ whenever

$$f(x_0) - f(x_k) \geq (1 - \tau)[f(x_0) - f_*]. \tag{4.2}$$

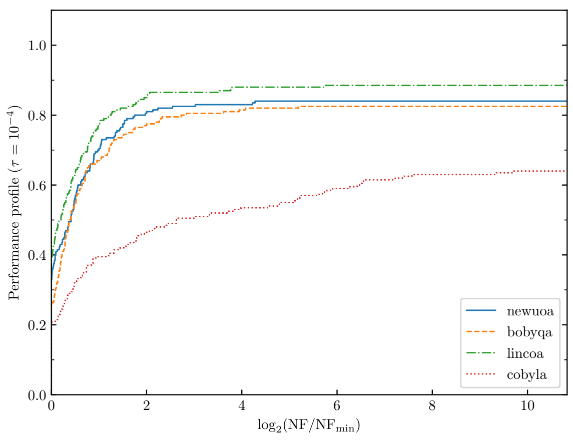
We can observe an expected behavior; UOBYQA performs better on small problems than all the others, as it is based on quadratic models obtained by fully-determined interpolation, and the performances of COBYLA decrease with the dimension rise, as it uses only linear models to approximate locally the problem.

Consider however the following experiment. Given a smooth objective function f, assume that the value received by the solvers is

$$F_\sigma(x) = [1 + \sigma e(x)]f(x),$$



(a) Dimension at most 10.



(b) Dimension at most 10.

Figure 1: Performance profile on unconstrained problems with a precision $\tau = 10^{-4}$.

where $e(x)$ is a random variable that follows a standard normal distribution $\mathcal{N}(0, 1)$, and $\sigma \geq 0$ is a given noise level. Figure 2 presents the performance profiles on the same unconstrained problems of dimension at most 50 from the CUTEst library as the previous experiment by randomizing the objective functions as F_σ with $\sigma = 10^{-2}$. Because of the stochastic behavior of the experiment, the convergence test (4.2) needs to be adapted. Each problem is solved 10 times by each solver, the objective value considered at each iteration is the average value for all runs, and the optimal value f_* of a given problem is decided as follows. It is the least value reached by the solvers for every run on the noised variation of the problem and by all the solvers on the noise-free original problem. In doing so, one should expect a decrease of the proportion of problems solved when compared with the previous experiment.

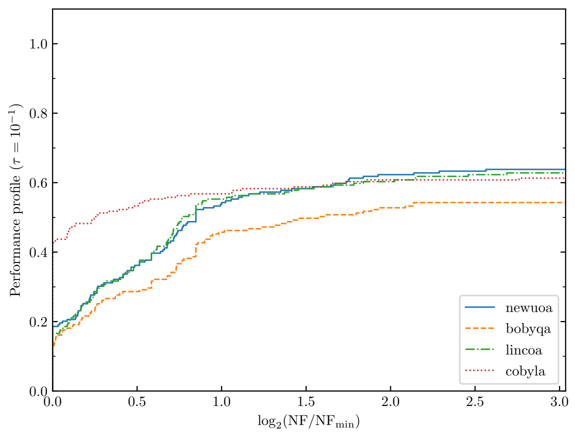


Figure 2: Performance profile on noised variations of unconstrained problems of dimension at most 50 with a precision $\tau = 10^{-1}$.

We observe however a peculiar behavior of COBYLA on this experiment, as it defeats all other solvers on unconstrained problems even though it is not defined for such kind of problem, and uses the simplest models. It seems that the linear models of COBYLA are, in some sense, less sensitive to Gaussian noise, but the authors did not derive a theory for this behavior as of today.

4.2.2 An example of hyperparameter tuning problem

We consider now the more practical problem of the hyperparameter tuning of a support vector machine (SVM). The model we consider is a C-SVM [10] for binary classification problems with an RBF kernel, admitting two hyperparameters: a regularization parameter $C > 0$ and a kernel parameter $\gamma > 0$. We want to compare the performance of PDFO with a prominent Bayesian optimization method and random search (RS). To this end, we use the Python package `hyperopt` [7] for solving the optimization problems, which provides both tree of Parzen estimators (TPE) and RS methods. Our experiments are based on binary classifications problems from the LIBSVM datasets[2]. A description of the datasets employed is provided in Table 1.

Table 1: Considered LIBSVM datasets description

Dataset $\mathcal{P}$	Attribute characteristic	Dimension d	Dataset size
splice	$[-1, 1]$, scaled	60	1000
svmguide1	$[-1, 1]$, scaled	4	3088
svmguide3	$[-1, 1]$, scaled	21	1242
ijcnn1	$[-1, 1]$	22	49 990

The problem we consider is as follows. A dataset $\mathcal{P} \subseteq [-1, 1]^d$ from Table 1 is randomly split into a training dataset $\mathcal{L}$, admitting approximately 70% of the data, and a testing dataset $\mathcal{T}$, with $\mathcal{P} = \mathcal{L} \cup \mathcal{T}$. We want to maximize the 5-fold area under the curve (AUC) validation score of the SVM trained on $\mathcal{L}$ with respect to the hyperparameters C and γ. The AUC score, a real number in $[0, 1]$, measures the area underneath the receiver operating characteristic (ROC) curve, a graph representing the performance of a binary classification model. This curve plots the true positive classification rate with respect to the false positive classification rate at different classification thresholds. The 5-fold AUC validation score corresponds to the following. The set $\mathcal{L}$ is split into 5 folds, and the model is trained 5 times, on each union of 4 distinct folds. After each training, the AUC score is calculated on the last fold, which was not involved in the training process, giving rise to 5 AUC scores, the average of which corresponds to the 5-fold AUC validation score. It is then clear that such an experiment lies in the DFO context.

The numerical results for this experiment are provided in Appendix A. The AUC scores and accuracies presented in the tables correspond to the ones computed on $\mathcal{T}$ with an SVM trained on $\mathcal{L}$, with the tuned parameters C and γ. In a nutshell, it globally shows that the numerical performances of PDFO against the two classical approaches are very similar, but the computations requires much fewer AUC evaluations, and hence, much less computation time.

[2]Available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

This behavior is particularly visible on the dataset “ijcnn1” in Table 5, as the size of this dataset is much larger than the others. Thus, we can conclude that PDFO performed better than the package `hyperopt` on these problems, even though the final numerical results are mostly similar.

5 Conclusions

We have presented the package PDFO for MATLAB® and Python, which aims at simplifying the use of the Powell’s DFO solvers. A more complete presentation of the package itself can be found on the PDFO website, together with different examples of use. The scope of PDFO in the future is not limited only to the Powell’s DFO solvers. The authors are currently working on a new solver for nonlinearly-constrained optimization, which is aimed to be added to PDFO, and other DFO solvers may be included in the future to PDFO.

References

[1] M. A. Abramson and C. Audet. “Convergence of mesh adaptive direct search to second-order stationary points.” In: *SIAM J. Optim.* 17 (2006), pp. 606–619.

[2] C. Audet and J. E. Dennis, Jr. “Mesh adaptive direct search algorithms for constrained optimization.” In: *SIAM J. Optim.* 17 (2006), pp. 188–217.

[3] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization.* Springer Ser. Oper. Res. Financ. Eng. Cham, CH: Springer, 2017.

[4] D. M. Bates et al. “Fitting linear mixed-effects models using `lme4`.” In: *J. Stat. Softw.* 67 (2015), pp. 1–48.

[5] D. M. Bates et al. *`minqa`: derivative-free optimization algorithms by quadratic approximation.* `http://cran.r-project.org/package=minqa`. R package version 1.2.4. 2014.

[6] A. S. Berahas, R. H. Byrd, and J. Nocedal. “Derivative-free optimization of noisy functions via quasi-Newton methods.” In: *SIAM J. Optim.* 29 (2019), pp. 965–993.

[7] J. Bergstra, D. Yamins, and D. D. Cox. “Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures.” In: *Proceedings of the 30th International Conference on Machine Learning.* Ed. by S. Dasgupta and D. McAllester. Atlanta, GA, US: PMLR, 2013, pp. 115–123.

[8] A. Biviano et al. “CLASH-VLT: the mass, velocity-anisotropy, and pseudo-phase-space density profiles of the $z = 0.44$ galaxy cluster MACS J1206.2-0847.” In: *A&A* 558 (2013), A1:1–A1:22.

[9] C. Cartis et al. “Improving the flexibility and robustness of model-based derivative-free optimization solvers.” In: *ACM Trans. Math. Software* 45 (2019), 32:1–32:41.

[10] C. C. Chang and C. J. Lin. “LIBSVM: a library for support vector machines.” In: *ACM TIST* 2 (2011), 27:1–27:27.

[11] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods.* MPS-SIAM Ser. Optim. Philadelphia, PA, US: SIAM, 2000.

[12] A. R. Conn and S. Le Digabel. "Use of quadratic models with mesh-adaptive direct search for constrained black box optimization." In: *Optim. Methods Softw.* 28 (2013), pp. 139–158.

[13] A. R. Conn, K. Scheinberg, and Ph. L. Toint. "A derivative free optimization algorithm in practice." In: *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO, US: AIAA, 1998, pp. 129–139.

[14] A. R. Conn, K. Scheinberg, and L. N. Vicente. "Geometry of interpolation sets in derivative free optimization." In: *Math. Program.* 111 (2008), pp. 141–172.

[15] A. R. Conn, K. Scheinberg, and L. N. Vicente. "Geometry of sample sets in derivative-free optimization: polynomial regression and underdetermined interpolation." In: *IMA J. Numer. Anal.* 28 (2008), pp. 721–748.

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

[17] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization.* MPS-SIAM Ser. Optim. Philadelphia, PA, US: SIAM, 2009.

[18] A. R. Conn and Ph. L. Toint. "An algorithm using quadratic interpolation for unconstrained derivative free optimization." In: *Nonlinear Optimization and Applications.* Ed. by G. Di Pillo and F. Giannessi. Boston, MA, US: Springer, 1996, pp. 27–47.

[19] A. L. Custódio, K. Scheinberg, and L. N. Vicente. "Methodologies and software for derivative-free optimization." In: *Advances and Trends in Optimization with Engineering Applications.* Ed. by T. Terlaky, M. F. Anjos, and S. Ahmed. Philadelphia, PA, US: SIAM, 2017, pp. 495–506.

[20] A. L. Custódio and L. N. Vicente. "Using sampling and simplex derivatives in pattern search methods." In: *SIAM J. Optim.* 18 (2007), pp. 537–555.

[21] S. Le Digabel. "Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm." In: *ACM Trans. Math. Software* 37 (2011), 44:1–44:15.

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

[23] G. Fasano, J. L. Morales, and J. Nocedal. "On the geometry phase in model-based algorithms for derivative-free optimization." In: *Optim. Methods Softw.* 24 (2009), pp. 145–154.

[24] R. Fletcher. *Practical Methods of Optimization.* Second. New York, NY, US: Wiley, 1987.

[25] F. Gallard et al. *The GEMSEO software.* Available at `https://gemseo.readthedocs.io/` (retrieved on August 31, 2021). 2018.

[26] A. Gazaix et al. "Industrial application of an advanced bi-level MDO formulation to aircraft engine pylon optimization." In: *AIAA Aviation Forum.* Dallas, TX, US: AIAA, 2019.

[27] H. Ghanbari and K. Scheinberg. *Black-box optimization in machine learning with trust region based derivative free algorithm.* Tech. rep. 17T-005. Bethlehem, PA, US: COR@L, 2017.

[28] N. I. M. Gould, D. Orban, and Ph. L. Toint. “CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization.” In: *Comput. Optim. Appl.* 60 (2015), pp. 545–557.

[29] S. Gratton et al. “Direct search based on probabilistic descent.” In: *SIAM J. Optim.* 25 (2015), pp. 1515–1541.

[30] S. Gratton et al. “Direct search based on probabilistic feasible descent for bound and linearly constrained problems.” In: *Comput. Optim. Appl.* 72 (2019), pp. 525–559.

[31] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* Second. Philadelphia, PA, US: SIAM, 2002.

[32] N. J. Higham. “Optimization by direct search in matrix computations.” In: *SIAM J. Matrix Anal. Appl.* 14 (1993), pp. 317–333.

[33] H. Izadinia, Q. Shan, and S. M. Seitz. “IM2CAD.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* San Juan, PR, US: IEEE, 2017, pp. 5134–5143.

[34] S. G. Johnson. *The NLopt nonlinear-optimization package.* `http://github.com/stevengj/nlopt`.

[35] C. T. Kelley. *Implicit Filtering.* Software Environ. Tools. Philadelphia, PA, US: SIAM, 2011.

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

[37] J. Larson, M. Menickelly, and S. M. Wild. “Derivative-free optimization methods.” In: *Acta Numer.* 28 (2019), pp. 287–404.

[38] V. Latorre et al. “Derivative free methodologies for circuit worst case analysis.” In: *Optim. Lett.* 13 (2019), pp. 1557–1571.

[39] G. A. Mamon, A. Biviano, and G. Boué. “MAMPOSSt: modelling anisotropy and mass profiles of observed spherical systems – I. Gaussian 3D velocities.” In: *Mon. Not. R. Astron. Soc.* 429 (2013), pp. 3079–3098.

[40] K. Mombaur, A. Truong, and J. P. Laumond. “From human to humanoid locomotion—an inverse optimal control approach.” In: *Auton. Robot.* 28 (2010), pp. 369–383.

[41] J. J. Moré and D. C. Sorensen. “Computing a trust region step.” In: *SIAM J. Sci. Stat. Comp.* 4 (1983), pp. 553–572.

[42] J. J. Moré and S. M. Wild. “Benchmarking derivative-free optimization algorithms.” In: *SIAM J. Optim.* 20 (2009), pp. 172–191.

[43] J. A. Nelder and R. Mead. “A simplex method for function minimization.” In: *Comput. J.* 7 (1965), pp. 308–313.

[44] R. Oeuvray and M. Bierlaire. “BOOSTERS: a derivative-free algorithm based on radial basis functions.” In: *Int. J. Model. Simul.* 29 (2009), pp. 29–36.

[45] P. Peterson. “F2PY: a tool for connecting Fortran and Python programs.” In: *Int. J. Comput. Sci. Eng.* 4 (2009), pp. 296–305.

[46] M. Porcelli and Ph. L. Toint. “BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables.” In: *ACM Trans. Math. Software* 44 (2017), 6:1–6:25.

[47] M. J. D. Powell. “A direct search optimization method that models the objective and constraint functions by linear interpolation.” In: *Advances in Optimization and Numerical Analysis*. Ed. by S. Gomez and J. P. Hennart. Dordrecht, NL: Springer, 1994, pp. 51–67.

[48] M. J. D. Powell. “A new algorithm for unconstrained optimization.” In: *Nonlinear Programming*. Ed. by J. B. Rosen, O. L. Mangasarian, and K. Ritter. London, UK: Academic Press, 1970, pp. 31–65.

[49] M. J. D. Powell. “Developments of NEWUOA for minimization without derivatives.” In: *IMA J. Numer. Anal.* 28 (2008), pp. 649–664.

[50] M. J. D. Powell. “Least Frobenius norm updating of quadratic models that satisfy interpolation conditions.” In: *Math. Program.* 100 (2004), pp. 183–215.

[51] M. J. D. Powell. “On fast trust region methods for quadratic models with linear constraints.” In: *Math. Program. Comput.* 7 (2015), pp. 237–267.

[52] M. J. D. Powell. “On the Lagrange functions of quadratic models that are defined by interpolation.” In: *Optim. Methods Softw.* 16 (2001), pp. 289–309.

[53] M. J. D. Powell. “On updating the inverse of a KKT matrix.” In: *Numerical Linear Algebra and Optimization*. Ed. by Y. Yuan. Beijing, CN: Science Press, 2004, pp. 56–78.

[54] M. J. D. Powell. *The BOBYQA algorithm for bound constrained optimization without derivatives*. Tech. rep. DAMTP 2009/NA06. Cambridge, UK: Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2009.

[55] M. J. D. Powell. “The NEWUOA software for unconstrained optimization without derivatives.” In: *Large-Scale Nonlinear Optimization*. Ed. by G. Di Pillo and M. Roma. New York, NY, US: Springer, 2006, pp. 255–297.

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

[57] R. G. Regis and S. M. Wild. “CONORBIT: constrained optimization by radial basis function interpolation in trust regions.” In: *Optim. Methods Softw.* 32 (2017), pp. 552–580.

[58] L. M. Rios and N. V. Sahinidis. “Derivative-free optimization: a review of algorithms and comparison of software implementations.” In: *J. Global Optim.* 56 (2013), pp. 1247–1293.

[59] T. Sauer and Y. Xu. “On Multivariate Lagrange Interpolation.” In: *Math. Comp.* 64 (1995), pp. 1147–1170.

[60] H. J. M. Shi et al. *On the numerical performance of derivative-free optimization methods based on finite-difference approximations.* Tech. rep. arXix:2102.09762. 2021.

[61] T. Steihaug. “The conjugate gradient method and trust regions in large scale optimization.” In: *IMA J. Numer. Anal.* 20 (1983), pp. 626–637.

[62] Y. Sun et al. “Derivative-free optimization for chemical product design.” In: *Curr. Opin. Chem. Eng.* 27 (2020), pp. 98–106.

[63] Ph. L. Toint. “Towards an efficient sparsity exploiting Newton method for minimization.” In: *Sparse Matrices and Their Uses.* Ed. by I. S. Duff. London, UK: Academic Press, 1981, pp. 57–88.

[64] P. Virtanen et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python.” In: *Nat. Methods* 17 (2020), pp. 261–272.

[65] S. M. Wild. “MNH: a derivative-free optimization algorithm using minimal norm Hessians.” In: *The Tenth Copper Mountain Conference on Iterative Methods.* 2008.

[66] S. M. Wild, R. G. Regis, and C. A. Shoemaker. “ORBIT: optimization by radial basis function interpolation in trust-regions.” In: *SIAM J. Sci. Comput.* 30 (2008), pp. 3197–3219.

[67] Y. Yuan. “Recent advances in trust region algorithms.” In: *Math. Program.* 151 (2015), pp. 249–281.

A Hyperparameter tuning experiment results

Table 2: Hyperparameter tuning problem on the dataset “splice”.

Solver	No. evaluations	AUC Score (10^{-1})	Accuracy (10^{-1})	Execution time (s)
PDFO	65	9.568	9.933	3.697
RS	100	6.409	5.300	4.635
RS	200	7.880	5.300	9.244
RS	300	7.880	5.300	13.763
TPE	100	5.000	5.033	4.889
TPE	300	7.736	5.300	15.726

Table 3: Hyperparameter tuning problem on the dataset “svmguide1”.

Solver	No. evaluations	AUC Score (10^{-1})	Accuracy (10^{-1})	Execution time (s)
PDFO	68	9.966	9.730	4.906
RS	100	9.966	9.676	16.178
RS	200	9.966	9.676	32.914
RS	300	9.966	9.676	48.404
TPE	100	9.966	9.720	13.057
TPE	300	9.966	9.720	33.392

Table 4: Hyperparameter tuning problem on the dataset “svmguide3”.

Solver	No. evaluations	AUC Score (10^{-1})	Accuracy (10^{-1})	Execution time (s)
PDFO	68	8.241	8.016	2.793
RS	100	8.025	7.882	4.233
RS	200	8.141	7.775	8.308
RS	300	8.141	7.775	12.414
TPE	100	7.774	7.453	4.197
TPE	300	8.106	7.989	12.912

Table 5: Hyperparameter tuning problem on the dataset “ijcnn1”.

Solver	No. evaluations	AUC Score (10^{-1})	Accuracy (10^{-1})	Execution time (10^3s)
PDFO	59	9.940	9.819	1.892
RS	100	9.886	9.773	4.435
RS	200	9.886	9.773	9.146
RS	300	9.886	9.773	13.251
TPE	100	9.891	9.791	4.426
TPE	300	9.896	9.786	12.552