University of Padova

Department of Mathematics

Master Thesis in Data Science

Improving on Stochastic Bayesian Adaptive Direct Search

Supervisor

Professor Luigi Acerbi

University of Helsinki

Co-supervisor

Professor Francesco Rinaldi

University of Padova

Master Candidate

Gurjeet Singh

Academic Year

2020-2022

Abstract

Black Box Optimization (BBO) has seen a growing interest in the field of optimization, and nowadays it is creating even more interest due to its applications in the area of computational neuroscience, machine learning, materials design, and computational biology.
The objective of BBO is to find a global minimum point under some resource constraint problems (e.g. the maximum number of function evaluations) of an expensive and possibly noisy non-smooth function, with no gradient information.

Thanks to the latest efforts made in this field, the Bayesian Adaptive Direct Search (BADS) algorithm proposed a hybrid Bayesian Optimization (BO) method. This approach combines the Mesh Adaptive Direct Search (MADS) method and a Gaussian Process surrogate to compete and find comparable or better solutions of existing state-of-the-art non-convex derivative-free optimizers.

In this thesis, we provide an extended study on BADS, by presenting PyBADS, a Python version of the existing complex MATLAB library of BADS. We include in this study the implementation of a new approach based on the Stochastic Mesh Adaptive Direct Search (StoMADS), by integrating it into the existing framework of BADS. The integration provides a new method for proving the convergence of the algorithm towards stationary points for non-smooth stochastic functions. The implementation of this method is present in the PyBADS python library.
Moreover, we developed a generic black box optimization benchmark for testing and evaluating different optimizers. In this work, we limited its usage to assess and compare PyBADS and BADS on different black box optimization problems, by designing an evaluation method with high statistical power for assessing the performance of the optimizers.
From the benchmarking results, PyBADS has achieved same and competitive outcomes as BADS, this achievement shows the correctness of the porting of BADS.

Acknowledgments

I am so honoured to have been mentored by Professor Luigi Acerbi who made me accomplish this thesis and gave me the opportunity to work at his Machine and Human Intelligence research lab. I thank him for all the advice and teaching, and I am very grateful for his patience while working with him. He made this experience a continuous learning process in my academic education, especially concerning research in Probabilistic Numeric problems and Bayesian inference. I thank the people in the research lab who inspired me with their interesting work. Being in this environment helped me design my research interest and motivated me to advance my studies and pursue the PhD I am going to start.
I would like to thank Professor Francesco Rinaldi, coordinator of the Data Science Master's course at the University of Padova, who helped me during my Erasmus and thesis. Additionally, I would like to thank both professors for helping me to find the PhD I was looking for.

Apart from the achievements and results I obtained in these years of academic education, a big thanks go to all my closest friends from Padova, Verona, and Helsinki with whom I spent the most joyful time. They have been present when I most needed them. They made this journey one of the greatest experiences, by building profound connections with each other. They played a crucial role in inspiring me and supporting me in good and bad times. I can not describe in words their importance to me.
Last but not least, I thank my family, who has always supported my studies and interests. No matter where I have gone in life, they always have stood by me.

Contents

List of Figures

List of Tables

Listing of acronyms

BADS ..... Bayesian Adaptive Direct Search

BBO ..... Black Box Optimization

BO ..... Bayesian Optimization

GP ..... Gaussian Process

MVN ..... Multivariate Normal Distribution

1 Introduction

Optimization theory plays a big role in many STEM areas, and it has been studied by numerous scholars and research groups. A particular field of optimization, called Black Box Optimization (BBO) has shown great interest in the scientific community as it deals with extremely challenging problems. Nowadays, this field finds a broad application in the areas of energy, materials science, computational engineering and computational neuroscience. Moreover, its usage has been expanded even more because of its applications in the recently raising fields of Machine Learning and Deep Learning, like tuning hyperparameters of Neural Networks or statistical models. The aim of BBO is to find in a feasible region Ω the global minimum of a target function f without knowing its full structure. Most of the Black Box Optimization problems include cases where the function evaluations are very expensive to obtain or when they involve costly constraints evaluations. Moreover, most of the time in these problems we do not have access to the gradient information of the function, or even if we have the access to the information it is often unavaible to guide the search. In addition, the objective function can be possibly noisy when numerical approximations and stochastic simulations are required in order to find an optimal solution. Solving such problems can be very challenging even in low dimensional configurations as it requires fitting a non-smooth function which can present many minimum and saddle points. These optimization problems also need to take in account computational cost and find the solution of the problem within a budget of number of function evaluations allowed.

In this work, we analyze Bayesian Adaptive Direct Search (BADS, Acerbi and Ma 2017), a

hybrid Bayesian Optimization algorithm which combines the Mesh Adaptive Direct Search (MADS, Audet and Dennis 2006) and aims to find the minimum (optimally the global minimum) of possibly noisy functions in the setting of Black Box Optimization problems. BADS is a fairly popular optimizer developed in MATLAB, and it is used by many computational labs all over the world. Its popularity has been achieved because of its performance, indeed it has been shown to perform equally or better than many other popular state-of-the-art MATLAB optimizers, such as *fminsearch*, *fmincon*, and *cmaes* (Acerbi and Ma 2017).

Because of its increasing use, a Python version has been very much requested by several labs. Moreover, releasing a Python version of the library would increase its accessibility to other users, since Python is one of the most popular languages across the scientific community.
To satisfy such demand, we developed PyBADS, the Python version of the existing BADS MATLAB library. We report in this work a detailed description of the new version and its performance analysis against BADS.
Moreover, the work did not only consists in a mere porting of the algorithm, but we also tried to improve the existing algorithm with new approaches and combine the recent work found in Adaptive Direct Search for **stochastic** target functions made by Audet, Dzahini, et al. 2021 to ensure and improve the convergence of the algorithm to a stationary point.

The thesis is divided in four main chapters. In chapter 2, we first cover the background knowledge required to understand BADS and Black Box optimizations, by explaining three essential components of BADS: the Mesh Adaptive Direct Search (MADS), Gaussian Processes and Bayesian Optimization. Chapter 3 contains instead the core work carried out in this thesis, which includes a report on the new Python implementation of BADS, the analysis carried on Sto-MADS and the work involved for integrating Sto-MADS in BADS. Moreover, this chapter comprehends the evaluation method used to set, run and evaluate the experiments. It also contains a description of the benchmark software developed for comparing the algorithms on different problems made of synthetic data. Finally in the last two chapters we report the experimental results obtained by comparing different optimizers and the investigation carried out to assess the if the algorithms developed in this thesis are demonstrating equal or better performance than the existing BADS algorithm.

2

Background

This chapter covers all the main content required to grasp and understand the work carried out in this thesis on the Bayesian Adaptive Direct Search (BADS) algorithm (Acerbi and Ma 2017). The Background chapter is divided into four sections, their sequence order is designed to define a clear path to guide the reader for a complete understanding of BADS. Together all the contents will be later combined and used in BADS, as explained in the last section.
In Section 2.1 we explain the Mesh Adaptive Direct Search framework and introduce some theoretical insights of non-smooth black-box optimization problems. In the next sections (Section 2.2, Section 2.3) we cover Gaussian Processes and later Bayesian Optimization. We first recall the Multivariate Normal (MVN) distribution and extend its structure to introduce Gaussian Processes. Once having understood the theoretical content, we present the Bayesian Optimization framework, a recent method (independent from existing adaptive direct search methods) that has shown a great potential in solving Black Box Optimization problem. Section 2.3 will also show the importance of Gaussian Processes in the Bayesian Optimization framework, and will be the final content to complete our prerequisites before introducing BADS in Section 2.4. In the final Background's section, we present a comprehensive explanation of BADS, by explaining the main steps of the iterative algorithm and the choices made related to the hyperparameters of the surrogate model and the algorithm.

2.1 Mesh Adaptive Direct Search (MADS)

MADS (Audet and Dennis 2006) has been one of the most influential algorithm in the field of BBO, and it proposes a general method (illustrated in Algorithm 1) for minimizing a deterministic nonsmooth function $f : \mathbb{R}^D \to \mathbf{R} \cup \{+\infty\}$ under general constraint $x \in \Omega \neq \emptyset \subseteq \mathbb{R}^D$. To cope with nonsmooth functions it exploits Clarke's generalized directional derivatives (Clarke 1990),

$$f^\circ(\hat{x}; v) := \limsup_{y \to \hat{x},\ y \in \Omega\ \ t \downarrow 0,\ y+tv \in \Omega} \frac{f(y+tv) - f(y)}{t}, \tag{2.1}$$

and the Lipschitz continuity assumption near $\hat{\boldsymbol{x}}$ (*the incumbent*) for building a compact set of directions $\mathbf{D} \subset \mathbb{R}^D$ (called *poll directions*). Under appropriate assumptions, these directions allow to reach a local stationary point ensured by the theoretical convergence analysis made in Theorem 3.17 of MADS (Audet and Dennis 2006). The directions are built such that each Clarke derivative is non-negative for every direction in the hypertangent cone, making them an asymptotically dense set of refining directions. Thanks to the construction of these directions, the method guarantees a stationary point $0 \in \partial f(\hat{x})$, such that:

$$f^\circ(\hat{x}; v) \geq 0 \text{ for all } v \in \mathbb{R}^n \iff 0 \in \partial f(\hat{x}) := \left\{ s \in \mathbb{R}^n : f^\circ(\hat{x}; v) \geq v^T s \text{ for all } v \in \mathbb{R}^n \right\}, \tag{2.2}$$

allowing to state a stronger result than a **finite** number of directions as stated in the Generalized Pattern Search (GPS) algorithm (Booker et al. 1999).

The MADS algorithm is made of two main phases, a POLL phase and a SEARCH phase, which both depend on the *current mesh* defined as: $M_k = \bigcup_{\boldsymbol{x} \in S_k} \left\{ \boldsymbol{x} + \Delta_k^m \mathbf{D} \boldsymbol{z} : \boldsymbol{z} \in \mathbb{N}^D \right\}$ where $S_k \in \mathbb{R}$ is the set of all points evaluated with the oracle from the start until the k-th

Algorithm 1: General MADS algorithm

```
Init: Let x_0 ∈ Ω, Δ_0^m ≤ Δ_0^p, D, G, τ, w^− and w^+
while Stopping Criteria do
    Optional SEARCH
    LOCAL POLL
    Parameters Update
end
```

iteration, $\Delta_k^m \in \mathbb{R}_+$ is the *mesh size parameter* that scales a finite set of directions $\mathbf{D} \subset \mathbb{R}^{D \times n_D}$. $\mathbf{D}$ must be a positive spanning set and each direction $d_j \in \mathbb{D}$ be the product Gz_j of a fixed non-singular generating matrix $G \in \mathbb{R}^{n \times n}$ and an integer vector $z_j \in \mathbb{Z}^{\times}$. In the use case, the $\mathbf{D}$ spanning set is chosen to be $\mathbf{D} = [\boldsymbol{I}_D, -\boldsymbol{I}_D]$, where $\boldsymbol{I}_D$ is the identity matrix in dimension D as proposed in LTMADS, a practical implementation of MADS that will be introduced later.

2.1.1 Poll phase

The POLL method guarantees theoretical convergence towards the Clarke stationary point. This stage consists of a local exploration of the optimization space that evaluates testing points in the neighbourhood of the current incumbent $\boldsymbol{x}_k$, by exploiting the constructed poll directions to move the incumbent in each direction depending on the iteration dependent mesh. The construction of the poll set of directions $\mathbf{D}_k = \text{span}\{\mathbf{D}\}$ of the poll candidate points $P_k = \{\boldsymbol{x}_k + \Delta_k^m \boldsymbol{v} : \boldsymbol{v} \in \mathbf{D}_k\} \subset M_k$ (called *frame*) is fundamental for the stationary point convergence. The poll set defines which directions have to be tested in order to achieve the Clarke stationary point. Instead, the iteration dependent mesh size controls the step size for such directions: the higher the step size, the higher the step towards the constructed directions.

In addition to the mesh size, the POLL stage introduces the *poll size* parameter Δ_k^p, which defines the maximum length of poll displacement vectors $\Delta_k^m \boldsymbol{v}$ for $\boldsymbol{v} \in \mathbf{D}_k$ from the incumbent $\boldsymbol{x}_k$. The poll size must be $\Delta_k^p \geq \Delta_k^m$ and typically $\Delta_k^p \approx \Delta_k^m ||\boldsymbol{v}||$, in a such manner it allows the number of positive spanning sets $\mathbf{D}_k$ to be non-constant over all the iteration, and makes the poll points in P_k to be chosen from a larger directions set $\mathbf{D}_k$. If the mesh size coincides with the poll size (as it is done in GPS), the algorithm results into having constant number of positive spanning and restricts the possible directions from the $\mathbf{D}_k$ set. A clear example that emphasizes such scenario can be seen in Figure 2.1 and Figure 2.2. The sampled directions are retrieved from the compact set $\mathbf{D}$ defined as $\{(d_1, d_2)^T \neq (0,0)^T : d_1, d_2 \in \{-1, 0, 1\}\} \subset \mathbb{R}^2$, and $\Delta_k^p = n\sqrt{\Delta_k^m}$.
By comparing the two figures, we see that the candidate points in the case of $\Delta_k^p = n\sqrt{\Delta_k^m}$ (Figure 2.1) can be sampled among the mesh points lying in a higher space of directions which is delineated by the dark contour in the figure.
Using such technique, MADS can exploit more combinations and possible directions within $\Delta_k^m ||\boldsymbol{d}|| \leq \Delta_k^p \max\{||\boldsymbol{d'}|| : \boldsymbol{d'} \in \boldsymbol{D}\}$ constraint, and it allows the directions to not be confined to a finite set, as it is done in GPS.

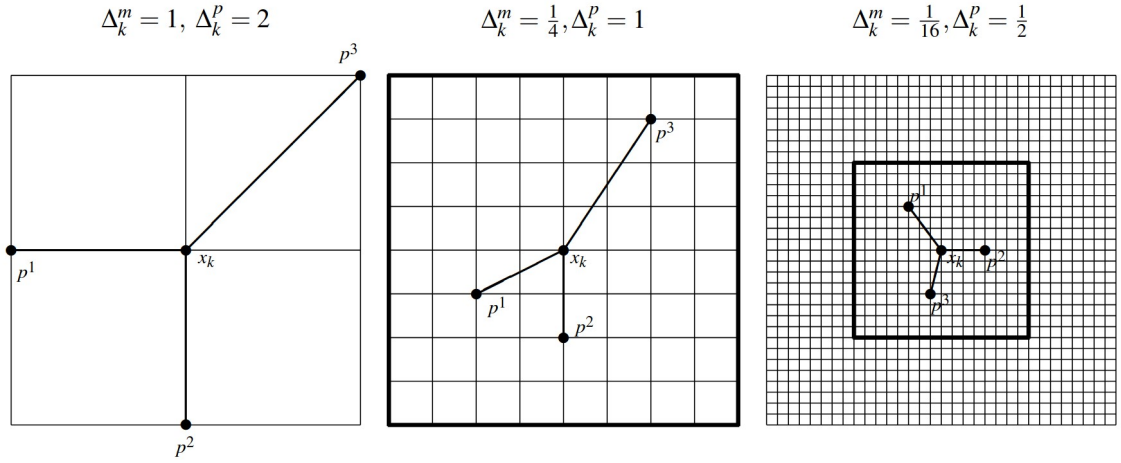


Figure 2.1: Example of MADS frames $P_k = \{\boldsymbol{x}_k + \Delta_k^m \boldsymbol{v} : \boldsymbol{v} \in \mathbf{D}_k\} = p_1, p_2, p_3$ where $\Delta_k^p = n\sqrt{\Delta_k^m}$ (reproduced from Figure 2.2 of MADS, Audet and Dennis 2006).

2.1.2 SEARCH PHASE

During the optimization of the unknown function, MADS alternates between the SEARCH and POLL phases. The former has the aim of exploiting the mesh space by injecting problem specific information into the optimization problem. The framework of the algorithm allows the freedom of using any search strategy with the only restriction that all the points have to lie on the current mesh. By only this restriction, the search method does not require to state formal guarantees for a new minima, limiting the result to be in the mesh grid suffices the convergence of the algorithm to the stationary point when switching back to the POLL phase.
If the search step is not able to find an improved point in the minimization of the objective function, the search fails and calls the POLL before terminating the iteration of the algorithm. Examples of search strategies mostly include surrogate like the Surrogate Management Framework (Booker et al. 1999), the Gaussian Process surrogate proposed by Serafini et al. 1998, and other heuristic strategies can also be used like the randomized evolution strategy with covariance matrix adaptation (CMA-ES, Hansen, Müller, and Koumoutsakos 2003).
In BADS instead, during this phase we perform Bayesian Optimization as a search strategy by using a Gaussian Process as a surrogate model of the unknown function. More detailed information about this method is given in the Section 2.4 of this work.

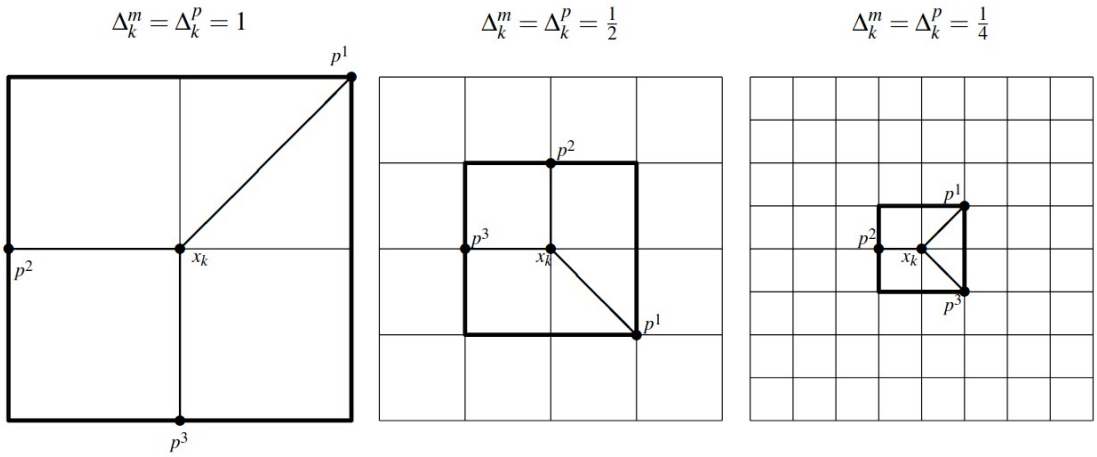


Figure 2.2: Example of GPS frames $P_k = \{\boldsymbol{x}_k + \Delta_k^m \boldsymbol{v} : \boldsymbol{v} \in \mathbf{D}_k\} = p_1, p_2, p_3$ where $\Delta_k^p = \Delta_k^m$ (reproduced from Figure 2.1 of MADS, Audet and Dennis 2006).

2.1.3 Parameters update

As we have seen in the previous sections when exploring and exploiting the current mesh space, the mesh size and the poll size parameters play an important role in the algorithm. They describe the displacement length of the new incumbent applied to the directions retrieved from both search strategies. Moreover, their magnitude tells us how much promising is the retrieved direction at the current stage of the optimization problem. Therefore, defining an update rule is crucial in order to make the algorithm converge to the minimum point.
To define such rule, we need first to evaluate if the new candidate point does describe an improvement in the optimization problem, i.e what a *successful point* is. Since we deal with a deterministic objective function MADS simply defines an *improved mesh point* by a trial mesh point with a lower objective function value than the current incumbent value $f(\boldsymbol{x}_k)$. When it occurs, the iteration is called as a *successful iteration*, thus $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ holds. Accordingly to this event the mesh update rule is defined at the end of the iteration as following:

$$\Delta_{k+1}^m = \begin{cases} \tau^{w_k} \Delta_k^m & \text{if an improved mesh point is found} \\ \tau^{-w_k} \Delta_k^m & \text{otherwise} \end{cases} . \tag{2.3}$$

The τ hyper-parameter is defined as $\tau > 1$ and $w_k \in \mathbb{N}$. On the other hand, when updating the poll size parameter Δ_{k+1}^p the update rule should satisfy the $\Delta_k^p \leq \Delta_k^p$ condition for all k

and

$$\lim_{k \in K} \Delta_k^m = 0 \iff \lim_{k \in K} \Delta_k^p = 0 \text{ for any infinte subset of indices } K. \tag{2.4}$$

When both iterations fail, thus either the search strategy or the poll one fails to find an improved mesh point, we increase the mesh resolution, i.e we reduce the mesh size in order to look for candidate points that are closer to the incumbent. Instead, when a successful point is found the mesh size can be increased or remains the same as described on the previous rule. In addition, when an improved mesh point is found, the poll method can be opportunistic and switch directly to the search phase. The algorithm proceeds until a stopping criterion is met.

2.1.4 LT-MADS

During the POLL phase, MADS requires to construct a set of poll directions that has to be in a tangent cone, in order to match the theoretical properties and ensure the convergence to a Clarke's stationary point. MADS proposes a specific implementation of the algorithm, called LTMADS (Audet and Dennis 2006), that includes a stochastic method to construct such set of directions. The strategy is based on a random lower triangular matrix and ensures with probability 1 that the set of refining generated directions is dense in the whole space, more precisely in the hypertangent cone. This method is also exploited by BADS (Section 2.4) during the poll phase. The algorithm fixes $\boldsymbol{D} = [\boldsymbol{I}_D - \boldsymbol{I}_D]$ as the viable search directions in dimension D with $\tau = 4, ;\ w^- = -1$ and $w^+ = 1$. It initializes the mesh and the poll size parameters to 1 ($\Delta_0^m = 1,\ \Delta_0^p = 1$), and the update rule using these parameters never exceeds 1. After setting such configuration the algorithm generates randomly uniform directions, with the constraint that one of the direction has to be dependent on the mesh size parameter. We report in this work the procedure used also in BADS for generating the poll directions with $2n$ basis. A more general rule instead can be found in the section 4 of the MADS paper (Audet and Dennis 2006). In our case, the $2n$ poll directions are constructed as described in the Algorithm 2.

The first method generates some random uniform $2n$ poll directions, and later it combines the directions with a diagonal mesh dependent vector. The resulting matrix is permuted and finally extended to a positive basis by mirroring it, leading to a basis in $\mathbb{R}^D$. Thanks to the mesh dependent diagonal, the resulting matrix has a positive basis $\boldsymbol{D}_k$ and it ensures the convergence to the stationary point, indeed the union of all directions is dense in R^D with probability 1. These two properties are proved in the Proposition 4.2 and Theorem 4.4 of MADS (Audet and Dennis 2006).

Algorithm 2: MADS 2n Poll basis vectors

Input: Let $x_0 \in \Omega, \Delta_k^m \leq \Delta_k^p$

$n_{\max} = \min\left(1, \lceil \frac{\Delta_k^m}{\Delta_k^p} \rceil\right)$

$\boldsymbol{D} \in \mathbb{Z}^{D \times D} \sim \mathcal{U}\{0, 1\}$

$\boldsymbol{D} = 2n_{\max}\, \boldsymbol{D} - n_{\max}$

$\boldsymbol{D} = \text{tril}(\boldsymbol{D}, -1)$

$\boldsymbol{d} = n_{\max}\, 2\,(\mathcal{U}\{1, 3\} - 1.5)$

$\boldsymbol{D} = \boldsymbol{D} + \mathbb{1}_D \odot \boldsymbol{d}$

$\boldsymbol{D} = \text{permute}(\boldsymbol{D})^T$

$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{D} \\ -\boldsymbol{D} \end{bmatrix}$

return $\boldsymbol{B}$

2.2 Gaussian Processes

Gaussian Processes (GPs) are statistical models for approximating an unknown function f in supervised regression and classification problems. They incorporate prior knowledge of the unknown function and exploit Multivariate Normal distribution (MVN) properties for building uncertainty-aware models. Indeed, they allow to quantify the presence of uncertainty for each prediction point by giving its estimated variance. In this paper, we will cover the regression problem since it is related to our work, but their applications can be extended to classification and clustering tasks as shown in (Kapoor et al. 2010) and (Kim and J. Lee 2007).

From a mathematical and a probabilistic point of view, Gaussian Processes are stochastic processes, i.e a collection of random variables indexed by time or space, such that every finite collection of random variables has a MVN distribution. In addition, they can be seen as infinite-dimensional generalization of a Multivariate Normal distribution, since each random variable is described by a Normal distribution. Gaussian Processes are analytically very convenient as they inherit and exploit properties of normal distributions. For this reason, we will first cover MVN distributions and their properties to understand their crucial role in GPs.

2.2.1 The Multivariate Normal Distribution and its properties

The Gaussian distribution occurs in nature because of the Central Limit Theorem (CLT) as either the sum of a large number of random variables or their mean lead to a Normal distribution, making it very attractive for modelling data as they occur often. Moreover, they hold analytical

convenient properties like the *infinite divisibility property* - linear combinations of Gaussian random variables are still normally distributed - and the joint distribution of Normal random variables remains a Gaussian. Because of the latter properties, extending from the univariate case to the multivariate case can be described in a straightforward manner by a joint Gaussian distribution $\boldsymbol{X} \in \mathbb{R}^d$ made by a mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$, thus $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The mean vector describes the expected value of each Normal random variable, instead the covariance matrix defines the variance and the pairwise similarity for each dimension. The diagonal of the square $\boldsymbol{\Sigma}$ matrix describes the variance of each variable, and the off-diagonal elements outline the correlation between the i-th and the j-th random variables. Moreover, because each component of the MVN is normally distributed, if two variables are uncorrelated then they are also independent. The covariance matrix expresses also the shapes of the normal distribution and it is always symmetric and has to be positive semi-definite. Assuming that the random variables are normally distributed, a useful result is the that the predictive distribution is accessible in a closed form solution, and marginalization and the conditional distribution of Gaussians results to be again a Gaussian distribution. Both results can be achieved by exploiting the analytical structure of this family of distribution. Indeed, having the multivariate joint distribution allows us to obtain easily the marginal distribution by dropping out the needed variable from the mean vector and from the diagonal of the covariance matrix. On the other hand, conditioning the multivariate normal distribution on a new observed variable $\boldsymbol{Y} \in \mathbb{R}^d$ lead to a new MVN distribution in the following form:

$$\boldsymbol{X}|\boldsymbol{Y} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{X}} + \boldsymbol{\Sigma}_{\boldsymbol{XY}}\boldsymbol{\Sigma}_{\boldsymbol{YY}}^{-1}\left(\boldsymbol{Y} - \boldsymbol{\mu}_{\boldsymbol{Y}}\right), \boldsymbol{\Sigma}_{\boldsymbol{XX}} - \boldsymbol{\Sigma}_{\boldsymbol{XY}}\boldsymbol{\Sigma}_{\boldsymbol{YY}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{YX}}\right). \tag{2.5}$$

2.2.2 Gaussian Process: a generalization of MVN

We can now extend the concept of Multivariate Normal distribution to define Gaussian Processes, which are stochastic processes of infinite dimensional domain. Thanks to the *Kolmogorov extension theorem*, we can construct them by only using the distribution of arbitrary finite sets of functions values (Øksendal 2003). Therefore, we can benefit of the finite dimensional multivariate Gaussian distributions to inherit their properties and generate a stochastic process, which in our case is a Gaussian Process. GPs can be seen as a generalization of the MVN distribution, to clearly distinguish the two objects we have to keep in mind that Multivariate Normal distributions use **scalars** or **vectors** random variables as domain, instead the GP is a stochastic process that governs properties of **functions** which span an infinite dimensional space (Gar-

nett 2022). Examples of sampled functions from the Gaussian Process can be seen in the Figure 2.3.
Having an unknown function $f : \mathcal{X} \to \mathbb{R}$ to approximate, and some observations $\boldsymbol{y} \in \mathbb{R}$, building a Gaussian Process that fits f just requires to replace the parameters of the MVN by a mean vector, where each value is retrieved from a mean function $\mu : \mathcal{X} \to \mathbb{R}$, and by a positive semi-definite covariance function matrix, where the values are described by a similarity kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We can summarize the GP using the following notation: $p(f) = \mathcal{GP}(f; \mu, K)$. In addition, we have that:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \text{where} \quad \begin{aligned} \boldsymbol{\mu} &= \mathbb{E}[\boldsymbol{y}|\boldsymbol{x}] = \mu(\boldsymbol{x}); \\ \boldsymbol{\Sigma} &= \operatorname{cov}[\boldsymbol{y}|\boldsymbol{x}] = K(\boldsymbol{x}, \boldsymbol{x}). \end{aligned} \tag{2.6}$$

Often the mean function μ is set to be a constant zero function, since the data can be centered, and for this reason it is often uninteresting. Indeed, most of the effort when designing the Gaussian Process is spent on finding the correct covariance function. But having a zero or constant mean function implies the model to predict test points to such a constant value that can be far away from the training set to such a constant value. Instead, in other cases we may want the model to not converge to a constant but to have a different asymptotic behaviour. However, integrating a mean function $\mu \neq 0$ is a straight forward task, it only requires add the mean term to the resulting function values after the prediction step.
The covariance matrix $\boldsymbol{\Sigma}$ is the main component that defines the process's behaviour, each pair of points of the covariance matrix is made by evaluating the kernel function, i.e $\Sigma_{ij} = K(x_i, x_j)$. In addition, the covariance function does not solely describe a similarity measure into a higher space but, it specifies a prior distribution over functions in the GP and it determines which of function is more probable than others. Indeed, just by changing the kernel the samples of functions retrieved from the Gaussian Process change drastically as shown in the Figure 2.3.

Therefore, the choice of the kernel introduces domain knowledge making the GP flexible and capture the trends of the data. Moreover, they enhance their flexibility by composing different kernels resulting a more complex and rich function. These functions can be combined together by concatenating or composing them. Covariance functions can be divided into stationary and non-stationary kernels. The former ones are functions which are invariant to translation and points only depend on their relative position . Instead non-stationary kernels do not have invariant translation and depend on their absolute location (Görtler, Kehlbeck, and

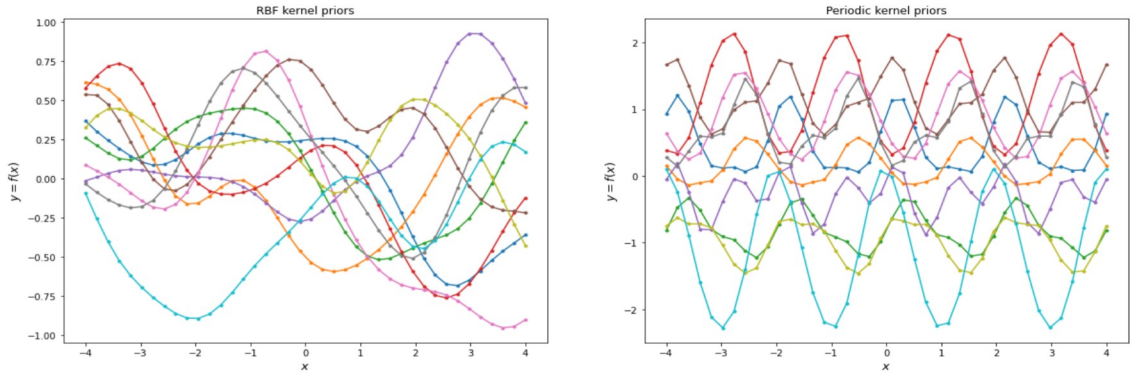


Figure 2.3: Example of twenty sampled functions from the RBF kernel $\sigma^2 \exp\left(-\frac{\|t-t'\|^2}{2l^2}\right)$ and the Periodic kernel $\sigma^2 \exp\left(-\frac{2\sin^2(\pi|t-t'|/p)}{l^2}\right)$, corresponding to the prior samples of the GP.

Deussen 2019). A full overview on different kernels and their properties as well their usage (including composite kernels) can be reviewed in the thesis's work of Duvenaud (Duvenaud 2015).

When dealing with a stochastic function f_Θ, where the noise can be caused by some measurement errors or generally by some uncertainty in the unknown function, the GP can model the noise by adding an error term in the model. To formalize this problem we can describe the observations by $\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is commonly assumed to be $\boldsymbol{\epsilon} \sim \mathcal{N}(\epsilon; \mathbf{0}, \boldsymbol{\sigma^2})$ and identically independent distributed. Therefore, to fully define the Gaussian Process we need to specify the mean and the kernel function and finally the noise distribution. Once having defined all these elements, we can retrieve the joint distribution by exploiting normality assumptions, independent noise, and we also assume zero mean for simplifying the derivations.

$$p(f(x^*), \boldsymbol{x}) = \mathcal{GP}\left(\begin{bmatrix} f(x^*) \\ y(x_1) \\ \dots \\ y(x_n) \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}, \begin{bmatrix} k(x^*, x^*) + \sigma^2 & k(x^*, \boldsymbol{x})^T \\ k(x^*, \boldsymbol{x}) & K_{\boldsymbol{xx}} + \sigma^2 \mathbb{1} \end{bmatrix}\right) \tag{2.7}$$

Furthermore, since we inherit all the MVN properties computing the posterior distribution

$f(x^*)|\boldsymbol{y}$ can be done using the following closed form solution:

$$f\left(x^*\right) \mid \boldsymbol{y} \sim \mathcal{N}\left(k\left(x^*, \mathbf{x}\right)^T \left(K_{\mathbf{xx}} + \sigma^2 \mathbb{1}\right)^{-1} y(\mathbf{x}), \right. \\ \left. k\left(x^*, x^*\right) + \sigma^2 + k\left(x^*, \mathbf{x}\right)^T \left(K_{\mathbf{xx}} + \sigma^2 \mathbb{1}\right)^{-1} k\left(x^*, \mathbf{x}\right)\right) \tag{2.8}$$

From the previous distribution we can obtain a prediction for an input x^* value by sampling from the posterior, otherwise we can extract the the marginal distribution of the mean function and the standard deviation by dropping out the needed variable from the mean vector and the covariance matrix of the posterior distribution and finally construct the confidence interval for each prediction.

In order to fit the Gaussian Process to the data we commonly want to find the model parameters $\boldsymbol{\theta}$ such that maximize the observed data and take in consideration the prior knowledge of the parameters. A common practice for solving this task is to estimate the parameters of the Gaussian Processes by using the maximum a priori (MAP) estimate method. The parameters $\boldsymbol{\theta}$ of the Gaussian Process include the mean function μ, the error parameter σ, and the parameters of the kernel function. We consider some prior knowledge of the parameters and we estimate them by maximizing the maximum a posteriori probability, more formally we solve the following problem:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left[\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\right] \tag{2.9}$$

Other approaches instead consider a fully Bayesian Inference framework by approximating the full posterior distribution using for example variants of Markov Chain Monte Carlo (MCMC) methods or distributional approximation techniques like Variational approximation or Laplace approximation. In general, these methods result to estimate better the parameters of the model, but they do require more computational costs than MAP estimation method.

Unfortunately, Gaussian Processes come also with some known disadvantages like intractability and high computational costs when considering high dimensional problems - in general they require $\mathcal{O}(N^3)$ time complexity and $\mathcal{O}(N^2)$ memory complexity, where N describes the training size - (Belyaev, Burnaev, and Kapushev 2014) moreover they can not best perform when the likelihood is not Gaussian.
However, different prominent works have been proposed for addressing this issue by using different approaches which exploit for example faster methods for decomposing covariance matrices and fast inference in multidimensional problems, reducing the $\mathcal{O}(N^3)$ complexity of GP

learning and inference to $\mathcal{O}(N)$ and the standard $\mathcal{O}(N^2)$ complexity per test point prediction to $\mathcal{O}(1)$ (Wilson, Dann, and Nickisch 2015). Moreover, we would like to share that many other works in the recent years have been published to scale the GP to a lower complexity problem (Gardner et al. 2018; Tran, Ranganath, and Blei 2015; Rossi et al. 2020), making this field a very active area of research.

2.3 Bayesian Optimization

Solving an optimization problem can be seen as a sequence of decisions to reach the minimum or maximum point of an objective function. These decisions are taken by an iterative algorithm that decides where to retrieve the observations by exploiting some prior knowledge of the problem and the available data. In the case of Black Box Optimization (BBO) we want to find the global minimum of an unknown expensive function for which we do not have access to the analytical form and its gradient information. In this configuration, designing a sample efficient algorithm is much needed because of the budget constraint on the maximum number of function evaluations. One of the leading strategies, in the field of BBO that fully adopts Bayesian decision theory is Bayesian Optimization (BO). Its usage has been shown to solve a wide range of problems (Mockus 2012), such in Machine Learning and Deep Learning models for hyper-parameter tuning (Bergstra et al. 2015; Snoek, Larochelle, and Adams 2012), computational neuroscience (Acerbi and Ma 2017), experimental particle physics (Cisbani et al. 2020), material design (P. I. Frazier and Wang 2015) and many others (Lizotte et al. 2007; Brochu, N. d. Freitas, and Ghosh 2007; P. Frazier 2018).

Bayesian Optimization uses active learning methods to guide the optimization algorithm by reasoning on uncertainty measures and the predicted mean given by a surrogate model. The main core of Bayesian Optimization is to define optimization policies based on uncertainty in order to reach the global minimum of the function. The policy in BO corresponds to the so-called acquisition function $\alpha : \mathcal{X} \rightarrow \mathbb{R}$, a function that provides a score to each potential observation location with the aim of finding the next point to evaluate so that minimize the number of function evaluations, and aid the optimization task while maximizing the model accuracy (Agnihotri and Batra 2020). Contrary to the main objective function of the black box, this heuristic function has to be tractable. Indeed, they are typically cheap to evaluate and analytically differentiable. They define a rule that balances between exploring uncertain regions and exploiting certain regions. Several acquisition functions have been proposed for Bayesian Optimization, but we will focus on the most popular ones and those related to our work, we

will describe their functionalities, properties and analytical closed form solution when assuming the surrogate being a Gaussian Process.

Expected Improvement (EI)

One of the most used acquisition is Expected Improvement. It retrieves the point x_{new} that in expectation most improves the objective from the current point observation f'. This corresponds to:

$$\begin{aligned} u(x) &= \max(0, f(x) - f') \\ x_{\text{new}} &= \arg\max_x \alpha_{EI} = \arg\max_x \mathbb{E}\left[u(x)|x, \mathcal{D}\right]. \end{aligned} \tag{2.10}$$

When considering the Gaussian Process as the surrogate model, the 2.10 expression corresponds to the following closed form solution:

$$\begin{aligned} EI(x) &= \begin{cases} (\mu(x) - f' - \epsilon)\,\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \\ Z &= \frac{\mu(x) - f' - \epsilon}{\sigma(x)}. \end{aligned} \tag{2.11}$$

The function Φ describes the cumulative distribution function (CDF) of the standard normal random variable Z and ϕ its probability density function (PDF), instead $\epsilon \in R^+$ balances the rate of exploitation and exploration, the higher it is the more it explores. We can see that the Expected Improvement will be high when $\mu(x) - f'$ is high or when the uncertainty $\sigma(x)$ around a point is high.

Probability of Improvement (PI)

Another acquisition function is the Probability of Improvement. It retrieves the point with the highest probability of improvement from the current minimum f'. It uses the following utility function

$$u(x) = \begin{cases} 1 & \text{if } f(x) \geq f' + \epsilon \\ 0 & \text{else} \end{cases}, \tag{2.12}$$

and maximizes the expected utility as:

$$x_{\text{new}} = \arg\max_x \alpha_{PI}(x) = \arg\max_x \mathbb{E}\left[u(x)|x, \mathcal{D}\right]. \tag{2.13}$$

When considering the Gaussian Process as surrogate model, the closed form solution of the above problem corresponds to the following CDF of the GP:

$$x_{\text{new}} = \arg\max_x \alpha_{PI}(x) = \arg\max_x \Phi\left(\frac{\mu(x) - f' - \epsilon}{\sigma(x)}\right), \tag{2.14}$$

where the ϵ regulates the gained improvement. Moreover, increasing ϵ possibly results in finding candidate point with larger variance, as their PDF is spread (Agnihotri and Batra 2020).

Lower Confidence Bound (LCB)

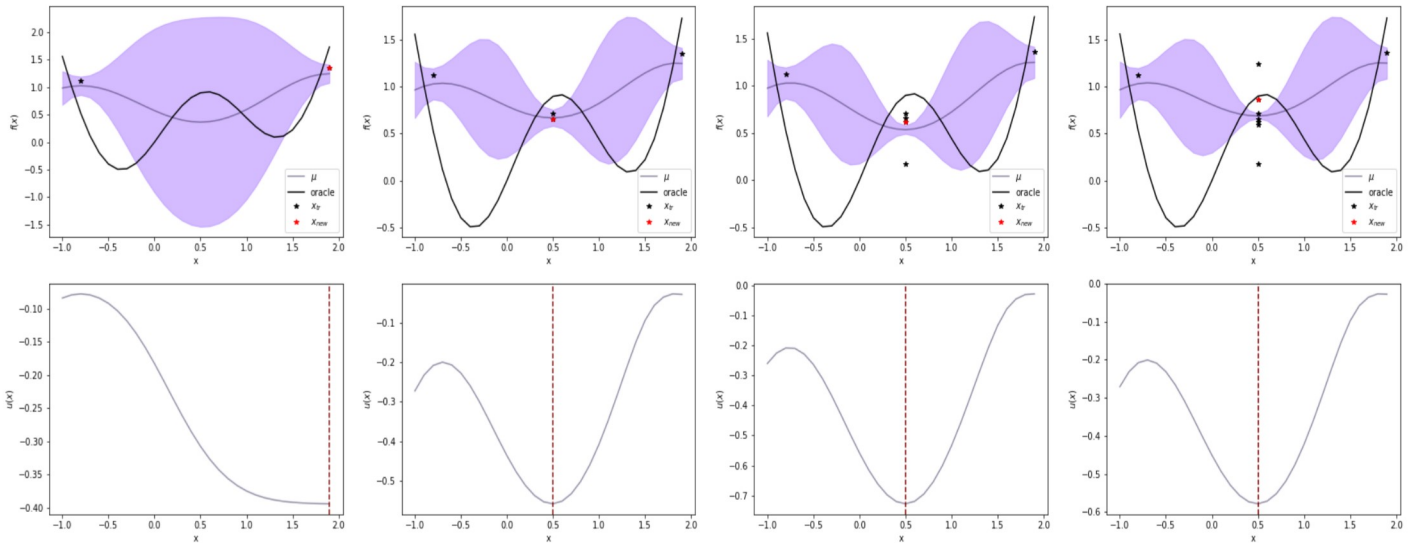
Finally, one of the last acquisition functions that we would like to cover in this work is the Lower Confidence Bound, which is also used in BADS (Acerbi and Ma 2017). The function has been designed to guide the optimization task to choose point with highly uncertain observations which have a wide range of plausible values, therefore the heuristic encourages the exploration of plausible optimal locations. The acquisition function has a very simple form and it corresponds to the quantile function:

$$x_{\text{new}} = \arg\min_x \alpha_{LCB}(x) = \arg\min_x \left(\mu(x) - \sqrt{\nu}\sigma(x)\right). \tag{2.15}$$

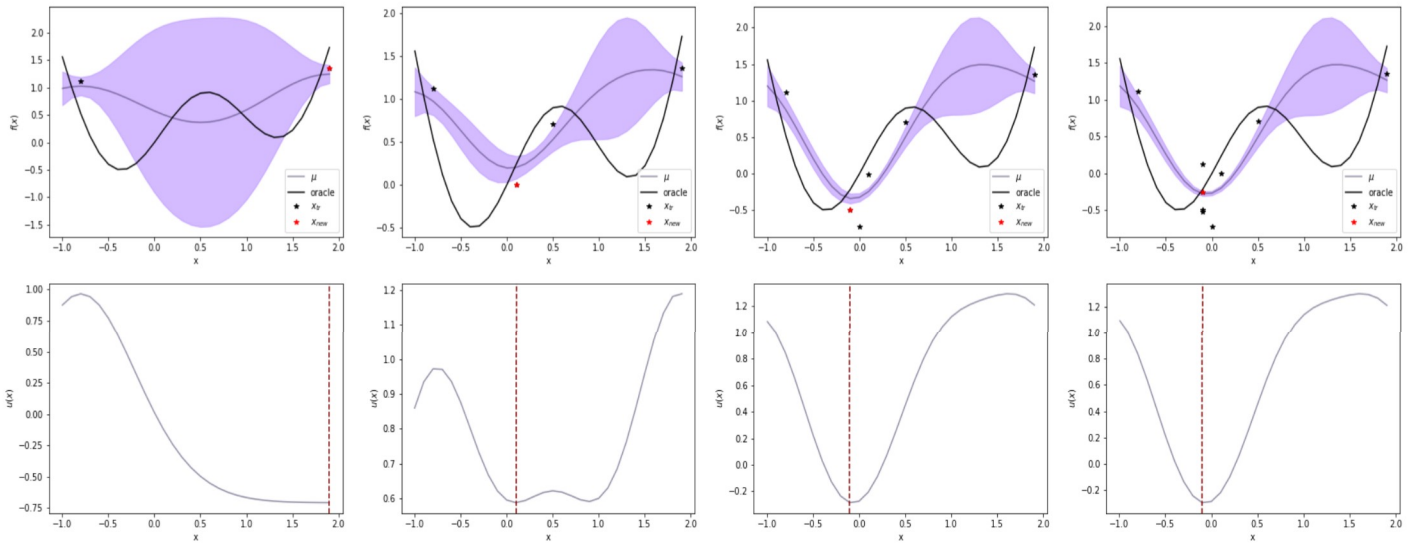
The ν hyper-parameter sets a confidence value and determines the confidence value: with relative low value exploitation is favoured by giving little credit to locations with high uncertainty, instead high values of ν heavily favour exploration.

For designing these acquisition functions there has been a wide range of functions, and a full review about of them can be seen in (Garnett 2022). The choice of acquisition functions is a non trivial task; its choice is still often unclear, and it is still an open question in the field. Several different functions have been proposed but none of them works well for all classes of functions (Hoffman, Brochu, and N. d. Freitas 2011). However, the most common and well-known acquisition functions that have been used in many applications of Bayesian Optimization are the Expected Improvement and the Lower/Upper Confidence Bound functions that we have been covered in the previous paragraph.

Instead of a single acquisition function, another possible strategy can be carried out by using a portfolio of acquisition functions. This method involves the usage of a pool of acquisition functions and defines a policy by an online multi-armed bandit strategy for picking a function from the pool. Such a method has been used in the work of Hoffman et al. where they have shown that it outperformed individual acquisition functions in some common benchmarks of Bayesian Optimization, making this strategy a promising method to use in some cases.



(a) Example of the BO algorithm using the Expected Improvement (EI) acquisition function. Unfortunately, with this configuration the algorithm get stuck in a local minima of the GP.



(b) Example of the BO algorithm using the Lower Confidence Bound (LCB) acquisition function. This configuration of the algorithm achieves a point close the Oracle solution with the same amount of iterations used for the EI function.

Figure 2.4: Summary of a shallow Bayesian optimization algorithm showing its performance after 8 iterations on a bi-modal stochastic function $f(x) = \sin(3x) + x^2 - 0.7x + \xi$ where $\xi \sim \mathcal{N}(0, 1)$. The examples report two configurations of acquisition functions (EI and LCB). In the first row of each figure, the black line describes the Oracle function, the blue line is the predicted mean of the GP and the purple area represents its estimated variance. The point elected by the acquisition function is reported with the red star. Instead, the second row reports the utility function $u(x)$ and a vertical line showing the elected minimum point of the utility function, which corresponds to the acquisition point.

Algorithm 3: General and basic BO algorithm

Input: initial dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$, prior $\boldsymbol{\gamma}$, parameters $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\boldsymbol{\gamma})$, unknown f
while *Stopping Criteria* **do**
    Approximate $p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta},\boldsymbol{\gamma})p(\boldsymbol{\theta}|\boldsymbol{\gamma})}{p(\mathcal{D})} \propto p(\mathcal{D}|\boldsymbol{\theta},\boldsymbol{\gamma})p(\boldsymbol{\theta}|\boldsymbol{\gamma})$
    $\boldsymbol{x} = \arg\max_{\boldsymbol{x'} \in \mathcal{X}} \alpha(\boldsymbol{x'}; \mathcal{D})$
    Observe $y_n = f(\boldsymbol{x})$
    $\mathcal{D} = \mathcal{D} \cup \{\boldsymbol{x}, y\}$
end
Return the candidate minimum point of the model.

To describe and fully capture the work-flow of the Bayesian Optimization framework we reported a general scheme in Algorithm 3, summarizing the main steps required by a BO algorithm. The framework requires: an initial dataset or point, and a statistical model that provides a parametrized posterior distribution retrieved using Bayesian inference, where the parameters come from a prior α, thus $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\boldsymbol{\gamma})$. The algorithm at each iteration first update its parameters and the posterior probability distribution of the surrogate model using the chosen inference method. Afterwards, it uses the acquisition function to find the new point to evaluate. Finally, it evaluates the function on the new point and adds it in the dataset. The method continues with the previous step until a stopping criterion is satisfied and returns the found minimum point with its uncertainty measure.
A common choice for the surrogate model in Black Box Optimization are Gaussian Processes. They have been the building blocks of Bayesian optimization (P. Frazier 2018) and they have found a lot of applications in BBO like in BADS (Acerbi and Ma 2017) and for tuning hyperparameters in Deep Neural Networks (Xiao, Xing, and Neiswanger 2021). They have been also used in statistical inference problems to boost inference methods like it has been done in Variational Bayesian Monte Carlo (VBMC, Acerbi 2020) and in the work of Gutmann, Cor, and er 2016. Moreover, they are commonly used in Bayesian Optimization because they provide closed form solutions when designing activation functions. Other types of surrogates like ensembles of Deep Neural Networks can also be used when considering non expensive functions as it has been done in the work of Swersky et al. 2020.

2.4 Bayesian Adaptive Direct Search (BADS)

Thanks to the general scheme of MADS it is possible to extend the framework and incorporate a search method that can significantly speed up the convergence toward a stationary point. Several approaches have been proposed in the literature by integrating surrogate models in the search method of MADS (Audet, Béchard, and Digabel 2008; Serafini et al. 1998; Digabel and Gramacy 2011). Contrary to these works, Bayesian Adaptive Direct Search (BADS, Acerbi and Ma 2017) method has stood out from the existing work because it does not only include a surrogate model in the search phase, but it encapsulates the Bayesian Optimization framework into the adaptive direct search method of MADS.
The algorithm combines both frameworks, it exploits the direct search method to guarantee theoretical convergence to a Clarke stationary point, and Bayesian active methods combined with a Gaussian Process to guide the sequence of decisions in the optimization problem and making the algorithm sample efficient. MADS has been designed for deterministic nonsmooth functions, integrating Bayesian Optimization and the Gaussian Process extends its usage also to noisy black-box functions. Indeed, BO has been considered as a state-of-the-art for black box optimization problems, including stochastic targets (Jones, Schonlau, and Welch 1998; Brochu, Cora, and N. d. Freitas 2010). Furthermore, BADS empirically has been proved to find better solutions or comparable ones against other optimizers on several model fitting problems with real data and models retrieved from cognitive, behavioural and computational neuroscience, which included noisy observations (Acerbi and Ma 2017).

We now describe BADS, which is summarized in Algorithm 4. The algorithm is divided in two stages like in MADS: the POLL phase and the SEARCH phase. Contrary to MADS, that selects the points in the mesh grid that most minimize the unknown target function only using the function evaluations, BADS in both stages exploits the Gaussian Process as a surrogate model to approximate the unknown function, and the acquisition function to select the next point of the Poll or Search set based on the variance estimated by the GP.
To summarize, BADS alternates between the two stages. During the POLL phase, it performs a failsafe exploration of the optimization problem, by gathering local information of the objective function using a model-free method – as it is carried out in MADS. Each observation made during the POLL is then added to the training set of the GP. The aim of this phase is not only to minimize the objective function by using the model-free method, but also to provide an adequate surrogate for the SEARCH stage when the POLL phase ends.
As soon as the algorithm switches to the SEARCH phase, the method exploits the built model

Algorithm 4: Bayesian Adaptive Direct Search

Input: initial dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$, objective function f, starting point $\boldsymbol{x}_0$, hard bounds LB, UB, optional plausible bounds PLB, PUB, optional barrier function c, additional options
Do
  for $1...n_{search}$; ▷ SEARCH stage
  do
    $\boldsymbol{x}_{\text{search}} \leftarrow$ SearchOracle ; ▷ Local BO step
    Observe $f(\boldsymbol{x}_{\text{search}})$
    if $\boldsymbol{x}_{search}$ *is a success* **then** $\boldsymbol{x}_{\text{new}} \leftarrow \boldsymbol{x}_s$; **break**;
  end
  if *SEARCH not sucessful* **then**
    compute poll set $\boldsymbol{P}_k$; ▷ POLL stage
    evaluate opportunistically f on $\boldsymbol{P}_k$ sorted by $\alpha_{LCB}(\boldsymbol{P}_k)$
    if $\boldsymbol{x}_p$ *is a success* **then** $\boldsymbol{x}_{\text{new}} \leftarrow \boldsymbol{x}_p$;
  end
  if *iteration* k *is sucessful* **then**
    $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_{\text{new}}$; ▷ Grid parameters update
    if *POLL is was a success* **then** $\Delta^m_{k+1} \leftarrow 2\Delta^m_k, \Delta^p_{k+1} \leftarrow 2\Delta^p_k$;
    else $\Delta^m_{k+1} \leftarrow \frac{1}{2}\Delta^m_k, \Delta^p_{k+1} \leftarrow \frac{1}{2}\Delta^p_k$;
  end
  $k \leftarrow k + 1$
until *fevals* > *MaxFunEvals and StoppingCriteria*;
Return the candidate minimum point of the model.

to effectively optimize the space by computing local fast optimization steps. This phase stops when the model is not able to find a better improvement in the minimization problem – which can be caused by a misspecified model or excess uncertainty – then the algorithm switches back to the POLL stage by updating first the parameters of the search space (poll size, mesh size).

2.4.1 Gaussian process setup

By default the observations $y^{(i)}$ are assumed i.i.d Gaussian, $y^{(i)} \sim \mathcal{N}(f(\boldsymbol{x}^{(i)}), \sigma^2)$ with $\sigma > 0$. Even when f is deterministic, we assume a small noise $\sigma > 0$, as it helps for the numerical stability of the Gaussian Process and its predictive accuracy (Gramacy and H. K. Lee 2012). The GP is configured with a constant mean function $m(\boldsymbol{x}) \in \mathbb{R}$ and a smooth automatic relevance

determination (ARD) rational quadratic (RQ) kernel,

$$k_{\mathrm{RQ}}\left(\boldsymbol{x}, \boldsymbol{x}'\right)=\sigma_f^2\left[1+\frac{1}{2\alpha} r^2\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right]^{-\alpha}, \quad \text{with} \quad r^2\left(\boldsymbol{x}, \boldsymbol{x}'\right)=\sum_{d=1}^{D} \frac{1}{\ell_d^2}\left(x_d-x_d'\right)^2 \tag{2.16}$$

where $l_1, ..., l_D$ are the kernel length scales for each coordinate direction, $\alpha > 0$ describes the shape, and σ_f^2 is the variance parameter. All these hyperparameters $\boldsymbol{\theta} = (m, l_1, ..., l_D, \alpha, \sigma^2)$ are modelled by independent prior distributions with hard bound constraint as described in Table 2.1. The priors are defined based on the data and they are updated using a quick heuristic approximation called Empirical Bayes (Supplementary Material A.3 Acerbi and Ma 2017). Moreover, $\boldsymbol{\theta}$ is estimated using maximum a posteriori (MAP) estimation to fit the GP via a gradient-based non-linear optimizer, and finally update the posterior distribution.
This task is computed either every $2D$ to $5D$ function evaluations or whenever the GP is inaccurate according to a normality test of the residuals assuming independent observations, $z^{(i)} = \left(y^{(i)} - \mu(\boldsymbol{x}^{(i)})\right) / \sqrt{s^2(\boldsymbol{x^{(i)}} + \sigma^2)}$.
In addition, to improve the Gaussian Process stability and to build a more precise local approximation around the incumbent x_k, the training set $\boldsymbol{X}$ of the GP is designed to select a subset of the points evaluated so far (Quinonero-Candela, Ramussen, and Williams 2007). At each time that either the GP parameters $\boldsymbol{\theta}$ are fitted or the incumbent is moved to a new point, the algorithm rebuilds the training set of the model by first sorting the points based on their $\boldsymbol{l}$-scaled distance r^2 (Equation 2.16) from the incumbent x_k, then adds the closest $n_{min} = 50$ to the training set. Secondly, additional points up to $10D$ that satisfies $r < 3\rho_{RQ}(\alpha)$ are added into $\boldsymbol{X}$, where $\rho_{RQ}(\alpha) = \sqrt{\alpha\ e^{1/\alpha} - 1}$ is a *radius function* that depends on the decay of the kernel (see Supplementary Material of BADS, Acerbi and Ma 2017). Finally each new point is added in the set using fast rank-one updated of the GP posterior.

2.4.2 Acquisition function in BADS

BADS embodies the acquisition function in MADS by applying it on the Poll set or in the Search set to elect the candidate point which will be evaluated using f and then later added in the training set of the GP. By default, the algorithm uses the Lower Confidence Bound (LCB) acquisition function, which elects the point by taking in consideration the uncertainty in the observation and encourage possible optimal regions with the aim of reducing the variance present in such location.

Hyperparameter	Prior	Bounds
GP Kernel		
Length scales l_d	$\ln \ell_d \sim \mathcal{N}_{\mathrm{T}}\left(\frac{1}{2}\left(\ln r_{\max} + \ln r_{\min}\right), \frac{1}{4}\left(\ln r_{\max} - \ln r_{\min}\right)^2\right)$	$\left[\Delta_{\min}^{\mathrm{poll}}, L_d\right]$
Signal variability σ_f	$\ln \sigma_f \sim \mathcal{N}_{\mathrm{T}}\left(\ln \mathbf{SD}(\boldsymbol{y}), 2^2\right)$	$[10^{-3}, 10^{9}]$
RQ kernel shape α	$\ln \alpha \sim \mathcal{N}_{\mathrm{T}}(1, 1)$	$[-5, 5]$
GP observation noise σ	$\ln \sigma \sim \mathcal{N}_T\left(\ln \sigma_{est}, 1\right)$	$[4 \cdot 10^{-4}, 150]$
deterministic f	$\sigma_{est} = \sqrt{10^{-3}\Delta_k^p}$	
noisy f	$\sigma_{est} = 1$	
GP mean μ	$\mu \sim \mathcal{N}\left(\mathbf{Q}_{0.9}(\boldsymbol{y}), \frac{1}{5^2}\left(\mathbf{Q}_{0.9}(\boldsymbol{y}) - \mathbf{Q}_{0.5}(\boldsymbol{y})\right)^2\right)$	$(-\infty, \infty)$

Table 2.1: GP hyperparameters priors (Acerbi and Ma 2017). Empirical Bayes priors and bounds for GP hyperparameters. $\mathcal{N}_T$ denotes the *truncated* normal, defined within the bounds specified in the last column. r_{max} and r_{min} are the maximum (resp., minimum) distance between any two points in the training set. The L_d is the parameter range $(UB_d - LB_d)$, for $1 \leq d \leq D$. $\mathbf{SD}(\cdot)$ is the standard deviation of a set of elements, instead $\mathbf{Q}$ is the $q-$th quantile of a set of elements.

The function is defined as:

$$\alpha_{LCB}(\boldsymbol{x}, \{\boldsymbol{X}, \boldsymbol{y}\}, \boldsymbol{\theta}) = \mu(\boldsymbol{x}) - \sqrt{\nu\beta_t s^2((x))}, \quad \beta_t = 2\ln\left(\frac{Dt^2\pi^2}{6\delta}\right), \tag{2.17}$$

where ν and δ are both positive hyper-parameters of the function $(\nu, \delta) \in R^+ \setminus \{0\}$, the former one is the confidence value and the latter one describes the probabilistic tolerance, t is the number of function evaluations so far and β_t represents the learning rate chosen to minimize the cumulative regret under certain assumptions.
For BADS, it is recommended to use the values $\nu = 0.2$ and $\delta = 0.1$ (Acerbi and Ma 2017). Other types of acquisition functions have been also tested for the experimental analysis carried out in BADS, but the LCB function has been found to perform better than others (Acerbi and Ma 2017).

2.4.3 Initial configuration

At first the algorithm is initialized by giving a starting point $\boldsymbol{x}_0$, the objective function, the hard bounds $(\mathrm{LB}_D, \mathrm{UB}_D)$ and optionally the plausible ones $(\mathrm{PLB}_D, \mathrm{PUB}_D)$, then it checks if the given function is noisy by evaluating multiple times from the starting point and assessing if the difference between the observations is more than a fixed tolerance noise. After this assessment, the algorithm generates $n_{\text{init}} = D$ additional points using a space-filling quasi-random Sobol sequence, and sets the incumbent $\boldsymbol{x}_1$ as the point with the minimum function value. All the input points require to be inside the hard bounds, otherwise they are projected in the plausible bounds, which identify a region in parameter space where most solutions are expected to lie. Moreover, all the variables are scaled into the standardized box $[-1, 1]^D$, such

that the box bounds correspond to $[\text{PLB}_D, \text{PUB}_D]$, in the original space. In the case that all the bounds are positive and the plausible range spans at least one order of magnitude of difference, i.e PUB/PLB ≥ 10, a log-scaling is applied on the input variables of the function. The algorithm allows the hard bounds to be infinite, but then it needs the plausible bounds to be finite. In such a way, it supports both constrained and unconstrained optimization problems, and a constraint can be passed as an input to BADS via a *barrier* function c.
After having standardized the bounds, the algorithm initializes the MADS parameters, i.e the frame size and the mesh size, by setting $\Delta_0^p = 1, \Delta_0^m = 2^{-10}$ and $\tau = 2$. In such a way, the algorithm can span the plausible regions and the mesh grid is relatively fine and the increment in step size (τ) is applied only after a successful POLL. Finally, before starting the optimization problem, the Gaussian Process is initialized by setting the prior distributions of the hyperparameters and fitting the model on the initial training set made by the observations returned by the space-filling method.

2.4.4 BADS Implementation: integrating MADS into BO

Poll stage in BADS

The first iteration of the algorithm is by default characterized with the POLL phase, the subsequent iterations of the POLL instead are performed when the SEARCH method fails. BADS during the POLL phase explores the mesh grid using the LTMADS (Audet and Dennis 2006) algorithm for constructing the dense poll set of directions $\boldsymbol{D}_k$, and additionally it rescales each direction vector proportionally to the GP length scale l_d. The new candidate point is retrieved from the acquisition function applied on the $\boldsymbol{x}_k + \boldsymbol{D}_k$ input set. The elected point is then evaluated with the target function and added in the training set of the Gaussian Process. During this task the GP is fitted when computing the first observation of the poll set or when it is not well calibrated. The POLL is opportunistic, thus when a successful improvement is found it stops. Otherwise if it does not find such improvement it continues to process the poll set until either all the new points $\boldsymbol{x}_k + \boldsymbol{D}_k$ have been discarded or the poll iterations exceeded twice the number of the input dimensionality.

The search stage and ES

The search strategy has the aim of exploring the local region of the mesh grid in an effective way by computing fast optimization steps thanks to an adequate Gaussian Process surrogate

model. BADS combines different search strategies inspired by the CMA-ES numerical optimization method (Hansen, Müller, and Koumoutsakos 2003) adopting a portfolio allocation strategy (also called *hedge* search) for allocating and choosing between different kinds of evolution strategies. The hedge search tracks the record of cumulative improvements among all the strategies according to the Hedge algorithm (Hoffman, Brochu, and N. d. Freitas 2011).
All the search methods share a general rule of sampling points in the neighbourhoods of the incumbent $\boldsymbol{x}_k$, which are drawn from a multivariate Normal distribution $\mathcal{N}(\boldsymbol{x}_s, \lambda^2(\Delta_k^p)\boldsymbol{\Sigma})$, where $\boldsymbol{x}_s$ is the current search focus, λ a scaling factor and $\boldsymbol{\Sigma}$ a *search covariance matrix*. Samples that violate non-bound constraint ($c(\boldsymbol{x}) > 0$) or the hard bounds are projected to the closest mesh point inside the bounds. The difference among the search methods lies on the covariance matrix passed to the Normal distribution. BADS uses two different strategies: the first one is called ES-WCM method and it constructs a matrix $\boldsymbol{\Sigma}_{WCM}$ proportional to the weighted covariance matrix of point in the training set of the Gaussian Process. Instead, the other method is ES-ELL that uses a diagonal matrix made of the re-scaled length scales of the Gaussian Process, $diag(l_1^2/|\boldsymbol{l}|^2, ..., l_D^2/|\boldsymbol{l}|^2)$.
Once drawn the samples from the Normal distribution of the search method, the acquisition selects the $\boldsymbol{x}_{\text{search}}$ candidate point, which is later evaluated using the target function, and finally the observation is added to the training set of the GP.
The method performs a maximum of $n_{\text{search}} = \max\{D, \lfloor 3 + D/2 \rfloor\}$ unsuccessful SEARCH steps, if they exceed the trials it switches to the POLL phase. On the other hand, when a successful improvement is found, the incumbent is moved and a new SEARCH from scratch is started from the new point.

Incumbent and update rule

Both stages assess if a new point presents an improvement in the minimization problem, depending on its results the mesh size and the incumbent are both updated.
BADS defines an improvement depending on the incumbent function value and the new observed value. If $f(x_{\text{new}}) - f(x_k) < 0$ then it is an improvement and the incumbent is moved towards the new point. In addition to this rule, BADS defines also a *sufficient improvement*, which occurs when $f(x_{\text{new}}) - f(x_k) < (\Delta_k^p)^{3/2}$. The latter heuristic does not just move the incumbent, but defines a successful point and updates the mesh grid parameters. Indeed when a successful point is found during the POLL, the mesh size and the poll size are incremented by a factor $\tau = 2$. Instead, if the point is found during the SEARCH strategy then the poll is skipped, the incumbent moved and the grid parameters remain the same. When no successful

improvements have been found in both of the strategies, then the mesh size and the poll size are both reduced by $\tau = 2^{-1}$.

$$\begin{cases} \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_{\text{new}};\ \ \Delta^m_{k+1} \leftarrow 2\Delta^m_k, \Delta^p_{k+1} \leftarrow 2\Delta^p_k & f(\boldsymbol{x}_{\text{new}}) - f(\boldsymbol{x}_k) < 0 \text{ and } f(\boldsymbol{x}_k) < (\Delta^p_k)^{3/2} \\ \quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_{\text{new}} & f(\boldsymbol{x}_{\text{new}}) - f(\boldsymbol{x}_k) < 0 \\ \Delta^m_{k+1} \leftarrow 2^{-1}\Delta^m_k, \Delta^p_{k+1} \leftarrow 2^{-1}\Delta^p_k & \text{otherwise} \end{cases}, \tag{2.18}$$

Uncertainty handling

Noisy tasks are more challenging problems and when they are encountered in BADS they are addressed by the Gaussian Process which quantifies the uncertainty in the predictions and by the Bayesian optimization framework. Although this approach can be sufficient in simple cases, due to the noise we can not simply rely on the y_i output values of the function as a ground truth value, especially when evaluating the decision rule for the incumbent update. For this reason, BADS replaces y_i with the GP latent quantile function following the work carried out by V. Picheny et. al. (Picheny et al. 2013):

$$q_\beta(\boldsymbol{x}; \{\mathbf{X}, \boldsymbol{y}\}, \boldsymbol{\theta}) \equiv q_\beta(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \Phi^{-1}(\beta)s(\boldsymbol{x}), \quad \beta \in [0.5, 1), \tag{2.19}$$

where $\Phi^{-1}(\cdot)$ is the quantile function of the standard normal distribution, $s(\boldsymbol{x})$ the estimated variance at $\boldsymbol{x}$ and β the statistical significance threshold rate used as a hyperparameter. By default $\beta = 0.5$ for promoting exploration, making the algorithm more stochastic and less conservative when moving the incumbent. In addition, to have a better accounting of the uncertainty BADS keeps a set of incumbents $\{\boldsymbol{x}_i\}^k_{i=1}$ and at the end of each POLL it re-evaluates q_β for all the incumbents and select the the point with the lowest q_β.
At the end of the algorithm, $\boldsymbol{x}_{end}$ is selected in a conservative and robust manner using $\beta = 0.999$. Moreover, when BADS minimizes a stochastic target it adds some initial configurations for dealing better with a noisy function. It doubles the minimum number of the GP training data to a maximum size of the 200, it increases the minimum number of Sobol's sequence samples to be at least $n_{init} = 20$, and it doubles the number of allowed stalled iterations. Finally, when the algorithm needs to retrieve the final function value, BADS returns either the GP prediction value $\mu(\boldsymbol{x}_{end})$, or an unbiased estimates of $\mathbb{E}[f(\boldsymbol{x}_{end})]$ using a simple Monte Carlo

sampling method.

Stopping criteria

BADS uses several stopping criteria, the simplest one being when the algorithm exceeds the function evaluations allowed by a defined budget (default $500D$) or the number of iterations allowed. But more sophisticated rules are also applied, like the detection of the minimal frame, i.e when the poll size shrinks towards zero which is checked when $\Delta_k^p > 10^{-6}$ making the algorithm to stop. Another stopping criteria is used when there is no significant improvement on the objective function for more than $4 + \lfloor D/2 \rfloor$ iterations.
When a stopping criterion occurs, before the algorithm returns the final $\boldsymbol{x}_{end}$ it transforms the incumbent to the original coordinate space.

3
Methods

BADS is available in an open and accessible GitHub repository[1] and it is developed in MATLAB. The algorithm has been shown to perform equally well or better than many other popular optimizers like *fminsearch*, *fmincon*, and *cmaes* (Acerbi and Ma 2017).
A Python version has been much requested by several labs and for this reason we developed PyBADS, the Python version of the existing BADS MATLAB library. We reported in this thesis a detailed description of the new version and its performance analysis against BADS.
The objective of this chapter is to describe the work carried out for porting BADS, which did not only consist of a mere porting but also included the analysis of a new method integrated in PyBADS for the optimization of stochastic targets. The aim was to improve its convergence rate towards stationary points and providing a theoretical result for its convergence. The study involved the integration of the existing BADS with the recent Stochastic Mesh Adaptive Direct Search (Sto-MADS) framework (Audet, Dzahini, et al. 2021), which guarantees theoretical convergence towards stationary points for non-smooth **stochastic** functions. This chapter also presents a generic black-box optimization benchmark library in Python used to evaluate and assess different optimizers for black-box optimization problems. In this thesis work, we restricted its usage to the comparison between BADS and PyBADS, by testing them on some optimization problems using synthetic functions as reported in our experimental results.

Chapter 3 is structured as follows: it first briefly describes the porting of the MATLAB

[1]BADS: https://github.com/acerbilab/bads

code to the Python version, followed by the new stochastic variant method included in PyBADS. Since the new approach has been adapted from the existing work of Sto-MADS (Audet, Dzahini, et al. 2021), we dedicated a section for summarizing the main algorithm steps of Sto-MADS and its theoretical results for convergence guarantees. Section 3.2 describes the studies and the changes made on PyBADS to integrate it with Sto-MADS.
The section 3.3 of the chapter presents a robust evaluation method with high statistical power, designed for deterministic and stochastic black box optimization problem. The description of the evaluation method provides the scheme used for assessing the performance of the optimizer on a target function. Finally, the chapter reports a brief description of the Black Box Optimization (BBO) benchmark, since it includes the implementation of the evaluation method.

3.1 PyBADS

PyBADS is a black box optimizer developed using Python, and is a port version of BADS. The new Python optimizer is based on NumPy (Harris et al. 2020) and SciPy [1] as scientific computing libraries. In addition, it also utilizes a lightweight Gaussian Process regression library developed by the Machine and Human Intelligence research group (PI: prof. Luigi Acerbi) [2], called GPyReg[3]. This library is currently a private repository and it is used for providing a wrapper object of a Gaussian Process. The main aim of this work is to provide a modular implementation of the GP in a single place to make its usage fairly simple for users. By just providing some configurations to GPyReg, a Gaussian Process object can be quickly ready to use for fitting a dataset with a Gaussian Process and making predictions of unobserved points.

The work carried out for developing the Python version of BADS did not consist of a simple line-by-line porting, but it was a non-trivial task. It involved a complex algorithm with a large code-base, and also required to develop changes in the GPyReg library, for example the implementation of the ARD-RQ kernel function with its related tests.
PyBADS has been designed in an Object-Oriented Programming method by providing a neat structure in order to make it simple to use for users. A user only requires to configure and run the optimizer on a target function, by instantiating a `BADS` object class and using the `optimize` function to run the optimizer. Although it seems very simple to use, the main class hides many complex classes and functions involved when using the optimizer. Therefore, the code has been

[1] SciPy: scipy.org
[2] Machine and Human Intelligence: machine-and-human-intelligence
[3] GPyReg: https://github.com/acerbilab/gpyreg

developed by dividing it in many modules such that it can be easy to understand, test, maintain and exploit modularized classes and functions. More detailed examples and description of code can be found in the repository of PyBADS [4], soon to be released with its related documentation. In addition, we also reported in the Appendix A.2 some examples of its usage by providing brief descriptions of the main functions.

PyBADS preserves almost all the features of the MATLAB version. However, the two libraries do slightly differ from each other from a computational point of view, and in the structure and implementation of the code. The difference in the computational aspect arises because of two main reasons. The first one comes from the different use of the Memory and CPU of the two programming languages. Secondly, they do differ because of the different numerical computing libraries (e.g Numpy) and optimization tools they used.

3.1.1 PyBADS and BADS implementation differences

Despite these unavoidable differences, PyBADS presents also some little variations and choices in the code compared to BADS. For example, the initialization of the GP hyperparameters is made differently than BADS. In PyBADS we use a heuristic approach that calculates a High Posterior Density (HPD) of the initial training points, which consists of a fraction (default 80%) of the bottom training set (ascending ordered training set), and returns the estimators of the GP hyperparameters using the selected data. The estimator of the prior mean of the mean function is described by a Normal distribution centered with the median of the HPD observations. Instead, the prior distributions of the covariance function parameters are centered at the standard deviation of the data. One of the major difference in PyBADS compared to BADS comes with the optimization of the Gaussian Process hyperparameters. Indeed, contrary to BADS which uses the optimization toolbox of Matlab, PyBADS uses the GPyReg library which applies the SciPy optimizer for optimizing the hyperparameters. The optimization configuration is similar to BADS, but PyBADS exploits also multiple initial points for the hyperparameters optimization, designed by a quasi-random space filling of the GP objective function. The sample size of the initial points are defined by a decreasing polynomial function dependent on the size of training set of the Gaussian Process, with a default of maximum initial points set to 128, to a minimum of 8. This choice does not only help the hyperparameters optimization, but also speeds up the algorithm.

[4] PyBADS: github.com/acerbilab/pybads

Another minor change in PyBADS consists in the initial points sample size of the global optimization problem. PyBADS uses the Sobol's sequence rule with a different sample size than BADS for generating the evaluated points. It uses a sample size corresponding to $n = 2^{\lceil \log_2(n_0) \rceil}$, where n_0 for deterministic targets corresponds to the dimension problem $n_0 = D$, and for stochastic objective functions $n_0 = 20$. The power base choice is dictated by the recommendation of the SciPy Sobol method to preserve the balance properties during the quadrature rule used by the library for generating the initial points.
Finally, PyBADS includes a new configuration of the algorithm which follows the method proposed in Sto-MADS (Audet, Dzahini, et al. 2021) as discussed in the next section.

3.2 Sto-MADS and BADS

While BADS can already handle stochastic targets, its implementation follows a set of very effective heuristics. On the other side, the Mesh Adaptive Direct Search framework only supported deterministic function evaluations. However, in 2021 the MADS framework has been extended by Sto-MADS (Audet, Dzahini, et al. 2021) to support stochastic targets. The new approach provides theoretical guarantees and convergence of the algorithm towards a Clarke's stationary point also for a stochastic function. The work carried out by C. Audet et al. in the algorithm proves the extensibility of the MADS framework and provides a distinctive theoretical result in the field of noisy black-block functions, which has been a goal of intensive research efforts.
In this thesis, we first provide a short summary of the algorithm and the tools used by Sto-MADS. Afterwards, we report the changes made by us to integrate Sto-MADS into BADS's framework.

3.2.1 From MADS to Sto-MADS

Contrary to MADS, since the objective function is affected by some unknown random noise, the optimization problem turns to be stochastic, i.e we want to find $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \mathbb{E}_\Theta \left[f_\Theta(\mathbf{x}) \right]$.
Therefore, Sto-MADS comes with the main idea of evaluating the mesh points using an estimator function $\hat{f}$ to measure the uncertainty present in the points generated by the poll directions and control the variance of the estimator by reducing it as more samples evaluations are collected.
The algorithm scheme of Sto-MADS is very similar to MADS, but it presents two crucial differences. The first one is made by the usage of a function estimates $\hat{f}$, which is built using the

observations made by black-box evaluations, and it is used to give points estimations and assess the improvement of a new point in the optimization problem. By all means, all the predicted points by $\hat{f}$ require to be accurate up to a tolerance and satisfies some variance condition (discussed later) to prove the convergence of the algorithm.
The other main difference comes within the update rule of the mesh and poll size. Since we are dealing with a noisy target, Sto-MADS introduces the concept of uncertain interval $\mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k)$ for assessing a candidate point. Assuming that the estimator $\hat{f}$ is ε-accurate (see the definition in Equation 3.2) the algorithms checks the improvement which is categorized in three different states using the following rule:

$$\begin{cases} \text{successful} & \text{if } \hat{f}_s^k - \hat{f}_x^k \leq -\gamma\varepsilon_f \left(\Delta_p^k\right)^2 \\ \text{certain unsuccessful} & \text{if } \hat{f}_s^k - \hat{f}_x^k \geq \gamma\varepsilon_f \left(\Delta_p^k\right)^2 \\ \text{uncertain unsuccessful} & \text{if } \hat{f}_s^k - \hat{f}_x^k \in \mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k) := \left(-\gamma\varepsilon_f \left(\Delta_p^k\right)^2, \gamma\varepsilon_f \left(\Delta_p^k\right)^2\right) \end{cases} \tag{3.1}$$

where x describes the current incumbent, and s is the candidate point retrieved from the SEARCH or POLL method. $\mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k)$ is the so-called uncertainty interval which is reduced when an uncertain unsuccessful iteration happens, and γ is a positive constant between $(2, +\infty)$.
Depending on the case, Sto-MADS accordingly updates the mesh parameters by increasing or decreasing the mesh. When a *sufficient decrease condition* is satisfied on candidate point s, i.e $\hat{f}_s^k - \hat{f}_x^k \leq -\gamma\varepsilon_f \left(\Delta_p^k\right)^2$, the iteration is called *successful*. Therefore the incumbent is replaced with the new candidate point and the mesh parameters are incremented. More specifically, the mesh parameters are updated as $\Delta_p^{k+1} = \min\left\{\tau^{-2}\Delta_p^k, \tau^{-\hat{z}}\right\}$ and $\Delta_m^{k+1} = \min\left\{\Delta_p^{k+1}, (\Delta_p^{k+1})^2\right\}$, where $\tau \in (0,1) \cap \mathbb{Q}$ is a fixed constant and $\hat{z} \in \mathbb{Z}$ a large fixed integer such that the frame size is upper bounded by a positive constant.
For the unsuccessful cases, the incumbent is never replaced with the new candidate point. Although, we do not update the incumbent when an *uncertain unsuccessful* iteration happens the frame size is reduced just by τ, i.e $\Delta_p^{k+1} = \tau\Delta_p^k$. Instead, an aggressive decrease of τ^2 is applied on the poll size when a *certain unsuccessful* iteration is detected.

3.2.2 A general estimator function for Sto-MADS

The main aim of the function estimate $\hat{f}$ in the algorithm is to provide an accurate estimator at the evaluated points and ensure the theoretical convergence of the algorithm. Indeed, Sto-MADS needs the estimator to assess the improvement at a new candidate point, which has to

be sufficiently and probabilistically accurate. In other words, we need to assess if the estimator is accurate up to a tolerance for a fixed probability that does not require to be equal to one, but larger than a constant. By satisfying these conditions, Sto-MADS proves the convergence of the algorithm towards a stationary point by building a sequence of accurate estimators.

To theoretically prove the convergence of Sto-MADS, we need first to introduce some definitions and assumptions about the function f, its noise, and the functions estimate $\hat{f}$.
All the stochastic quantities considered from now on lie in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the sample space of the algorithm, $\mathcal{F}$ is the σ-algebra and $\mathbb{P}$ the probability measure.

Definition 1

$\hat{f}$ is said to be a ε-accurate estimator at a point x if:

$$|\hat{f}(x) - f(x)| \leq \varepsilon_f \left(\Delta_p^k\right)^2, \tag{3.2}$$

where ε_f is a positive fixed constant that adjusts the initial amplitude of the uncertainty Interval $\mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k)$. The accuracy of the estimator in Sto-MADS is made dependent on the current poll size parameter Δ_p^k, in this manner the poll size does not only update the resolution of the mesh but adaptively controls also the variance of the estimator (Audet, Dzahini, et al. 2021).

Definition 2

The ε-accurate estimator is subsequently extended by the notion of β-probabilistically ε_f-accurate estimator:

$$\mathbb{P}\left[|\hat{f}(x) - f(x)| \leq \epsilon_f \left(\Delta_p^k\right)^2\right] \leq \beta, \tag{3.3}$$

where $\beta \in (0, 1)$.

Assumption 1

f is locally Lipschitz continuous everywhere and all iterates x^k generated by Sto-MADS lie in a compact set $\mathcal{X}$.

Remarks: assumption 1 makes the poll size Δ_k^p and the function $f(X^k)$ integrable in the probability space, i.e $\Delta_k^p, f(X^k) \in \mathbb{L}^1(\Omega, \mathcal{G}, \mathbb{P})$ for all k (see **Proposition 2** of Audet, Dzahini, et al. 2021), which is necessary for the convergence analysis of Sto-MADS.

Assumption 2

The variance present in the target function is upper bounded by some constant $V > 0$,

$$\mathrm{Var}_\Theta\left[f_\Theta(x)\right] \leq V < +\infty \quad \text{for all } x. \tag{3.4}$$

All the above assumptions are very common in many optimization problems and they are used to introduce and define the problem. But to prove the convergence of Sto-MADS, the following core assumptions about the estimator function need to be satisfied.

Assumption 3

(i) A sequence of random estimates $\left\{\hat{f}_x^k, \hat{f}_s^k\right\}$ generated by the algorithm is β-probabilistically ε-accurate for some $\beta \in (0,1)$:

$$\begin{aligned}\mathbb{P}\Big(\Big\{\Big|\hat{f}_x^k(x^k) - f(x^k))\Big| \leq \varepsilon_f \left(\Delta_p^k\right)^2\Big\} \cap \\ \Big\{\Big|\hat{f}_s^k(s^k) - f(s^k))\Big| \leq \varepsilon_f \left(\Delta_p^k\right)^2\Big\} \mid \mathcal{F}_{k-1}^{\hat{f}}\Big) \geq \beta \quad \forall\, k > 0,\end{aligned} \tag{3.5}$$

where $\mathcal{F}_{k-1}^{\hat{f}}$ is the $\sigma-$algebra of events up to the choice of x_k.

(ii) There exists $\kappa_f > 0$ s.t the sequence of estimates $\left\{\hat{f}_x^k, \hat{f}_s^k\right\}$ generated by the algorithm satisfies the following κ_f-variance conditions for all $k > 0$ iterations:

$$\begin{aligned}\mathbb{E}\left[\left|\hat{f}_x^k(x^k) - f(x^k)\right|^2 |\mathcal{F}_{k-1}^{\hat{f}}\right] \leq (\kappa_f)^2 \left(\Delta_p^k\right)^4 \quad \text{and} \\ \mathbb{E}\left[\left|\hat{f}_s^k(s^k) - f(s^k)\right|^2 |\mathcal{F}_{k-1}^{\hat{f}}\right] \leq (\kappa_f)^2 \left(\Delta_p^k\right)^4.\end{aligned} \tag{3.6}$$

Assumption 3 states a key assumption about the accuracy of the estimator and defines a lower bound on β by any k iterations of the algorithm. Such a lower bound can be achieved by a function estimate that is defined in terms of ε_f, κ_f, and by the sample size used by the estimator.

Satisfying **Assumption 3** allows to derive the following **1** that plays a crucial part in the convergence analysis.

Lemma 1

Let **Assumption 3** hold, and suppose that all the estimators at each point are generated by Sto-MADS described by J_k, a sequence of random estimate $\left\{\hat{f}_x^k, \hat{f}_s^k\right\}$ of β-probabilistically ε_f-accurate such that $\mathbb{P}\left[J_k|\mathcal{F}_{k-1}^{\hat{f}}\right] = \mathbb{E}\left[\mathbb{1}_{J_k}|\mathcal{F}_{k-1}^{\hat{f}}\right]$. Denote its negated form by $\bar{J}_k$, i.e a set

made by bad estimates. Then

$$\begin{aligned} \mathbb{E}\left[\mathbb{1}_{\bar{J}_k}\left|\hat{f}_x^k(x^k)-f(x^k)\right|^2|\mathcal{F}_{k-1}^{\hat{f}}\right] &\leq (1-\beta)^{1/2}\kappa_F\left(\Delta_p^k\right)^2 \\ \text{and } \mathbb{E}\left[\mathbb{1}_{\bar{J}_k}\left|\hat{f}_s^k(s^k)-f(s^k)\right|^2|\mathcal{F}_{k-1}^{\hat{f}}\right] &\leq (1-\beta)^{1/2}\kappa_F\left(\Delta_p^k\right)^2 \end{aligned} \tag{3.7}$$

In a few words, the above Lemma demonstrates the relationship between the κ-variance conditions and the probability of obtaining a bad estimates, which can be controlled by the number of samples size used by the estimator. A full detailed proof of the Lemma can be seen in **Lemma 1** of Sto-MADS (Audet, Dzahini, et al. 2021). Therefore, if the function estimates satisfies **Assumptions 3**, we can exploit **Lemma 1** and use it for proving **Theorem 1** of Sto-MADS (Audet, Dzahini, et al. 2021), which guarantees the convergence of the algorithm to a stationary point using the designed function estimates and the scheme of the algorithm proposed by Sto-MADS.

3.2.3 The Sto-MADS probabilistic estimate

As we have seen until now the Sto-MADS framework does not state many assumptions about the noise of the target or the dependencies of the observed values, by leaving the method general and adaptable to different choice of function estimates.
Indeed, more tightening assumptions can be added accordingly to the function estimates with the aim of proving Assumption 3 which ensures the convergence of the algorithm.
For example, Sto-MADS assumes unbiased noise $\mathbb{E}_\Theta\left[f_\Theta(x)\right] = f(x)$, identically independent distribution observations, and it uses the Monte Carlo method for estimating the function values of the target function. More formally, let x be the incumbent point and s the candidate point, we have the corresponding independent estimators $\hat{f}_x^k = \frac{1}{p^k}\sum_{i=1}^{p^k} f_{\Theta_{x,i}}(x^k)$ and $\hat{f}_s^k = \frac{1}{p^k}\sum_{i=1}^{p^k} f_{\Theta_{s,i}}(s^k)$, where p^k denotes the sample size and $\Theta_{x,1}, \Theta_{x,2}...\Theta_{x,p^k}$ and $\Theta_{s,1}, \Theta_{s,2}...\Theta_{s,p^k}$ are indeed independent random samples of Θ_x and Θ_s. Using this function estimates makes Assumption 3 easy to prove.
Indeed, by exploiting Chebyshev's inequality and noticing that the estimators are unbiased and that the expected value and variance of the estimators are $\mathbb{E}\left(\hat{f}_x^k\right) = f(x^k)$ and $\text{Var}\left(\hat{f}_x^k\right) =$

$\frac{V}{p^k}$, it follows that:

$$\mathbb{P}\left(\left|\hat{f}_x^k - f\left(x^k\right)\right| > \varepsilon_f \left(\Delta_p^k\right)^2\right) \leq \frac{\mathrm{Var}\left(\hat{f}_x^k\right)}{\left(\varepsilon_f\right)^2 \left(\Delta_p^k\right)^4} \leq \frac{V}{p^k \left(\varepsilon_f\right)^2 \left(\Delta_p^k\right)^4}, \tag{3.8}$$

and by choosing

$$p^k \geq \frac{V}{\left(\varepsilon_f\right)^2 \left(1 - \sqrt{\beta}\right) \left(\Delta_p^k\right)^4} \tag{3.9}$$

we obtain that

$$\mathbb{P}\left(\left|\hat{f}_x^k - f\left(x^k\right)\right| \leq \varepsilon_f \left(\Delta_p^k\right)^2\right) \geq \sqrt{\beta}. \tag{3.10}$$

Since $\hat{f}_x^k$ and $\hat{f}_s^k$ are independent we similarly get that $\mathbb{P}\left(\left|\hat{f}_s^k - f\left(x^k + s^k\right)\right| \leq \varepsilon_f \left(\Delta_p^k\right)^2\right) \geq \sqrt{\beta}$, and it follows that

$$\mathbb{P}\left(\left\{\left|\hat{f}_x^k - f\left(x^k\right)\right| \leq \varepsilon_f \left(\Delta_p^k\right)^2\right\} \cap \left\{\left|\hat{f}_s^k - f\left(x^k + s^k\right)\right| \leq \varepsilon_f \left(\Delta_p^k\right)^2\right\}\right) \geq \beta, \tag{3.11}$$

proves the point (i) of Assumption 3. Finally, since the estimators are unbiased we can also easily prove point (ii) of Assumption 3 as following

$$\mathbb{E}\left(\left|\hat{f}_x^k{}^k - f\left(x^k\right)\right|^2\right) = \mathrm{Var}\left(\hat{f}_x^k - f\left(x^k\right)\right) = \mathrm{Var}\left(\hat{f}_x^k\right) \leq \frac{V}{p^k} \leq \left(\varepsilon_f\right)^2 \left(1 - \sqrt{\beta}\right) \left(\Delta_p^k\right)^4, \tag{3.12}$$

and by choosing $\left(\kappa_F\right)^2 \geq \left(\varepsilon_f\right)^2 \left(1 - \sqrt{\beta}\right)$, we finally prove the second point of the assumption, taking in consideration that the same result holds also for $\hat{f}_s^k$.

From the previous results we can see the control effect on κ_f-variance condition and ensure an accurate estimator by a finite number of observed values described by the p^k sample size of the estimator. Indeed, this result is particularly exploited in the case of a bad estimate in the proof of Theorem 1 of Sto-MADS's convergence analysis, which ensures that an accurate estimator can be obtained by increasing the sample size to satisfy the desired variance condition. Although, we can prove theoretically the convergence of the algorithm using this estimator, from Equation 3.9 we can see that the sample size required to satisfy the variance condition is very high in the case of small magnitude values of the poll size and ε_f-accurate tolerance. For this reason, in the actual implementation of Sto-MADS they proposed a biased estimator based on Monte Carlo method to reduce the number of function evaluations. The proposed estimator can be seen in Equation 7 of Sto-MADS (Audet, Dzahini, et al. 2021), and this scheme has

been adopted by the authors because of its efficiency in the practical case of expensive black-boxes functions. Despite this fact, the Monte Carlo estimator allows to prove easily Assumption 3 required by Sto-MADS to satisfy the accuracy condition on the function estimates.

3.2.4 Integrating Sto-MADS in BADS

In this part of the section we introduce the changes made in the PyBADS to adapt the existing BADS scheme to the strategy proposed by C. Audet et al. in Sto-MADS (Audet, Dzahini, et al. 2021). As we have analyzed in the previous paragraphs Sto-MADS proposes a Monte Carlo method as function estimates, which can be a very expensive choice, as the estimator might require many samples to satisfy the variance condition set by the algorithm. In addition, Sto-MADS does not state many assumptions about the noise present in the target function, which might be affected by high and heteroskedastic noise. Therefore, choosing such an estimator would lead to underestimating the target and would require many samples to build an accurate estimator at the candidate point.
Because of the previous points, we chose to rely on the Gaussian Process of BADS as the function estimates in our implementation. More precisely, the estimator at the candidate point s^k is obtained from the posterior mean given from the Gaussian Process conditioned on the function evaluation at the candidate point, i.e we add the observed value to the training set of the GP and retrieve the posterior of the Gaussian Process. Once obtained the estimation point, we apply the criteria used in Sto-MADS, i.e we check the improvement presents with the new point, by following the rule based on the uncertainty Interval $\mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k)$ of Sto-MADS, and apply the update rule to the mesh parameters as described in Sto-MADS. A noteworthy decision we made on the uncertainty interval concerns about the ε_f parameter of $\mathcal{I}_{\gamma,\varepsilon_f}(\Delta_p^k)$. We defined it as the sum of the latent GP function uncertainty present in the incumbent and at the candidate point. Therefore, $\mathcal{I}_\gamma(\Delta_p^k) := \left(-\gamma\sqrt{\hat{\sigma}_{f_x}^2 + \hat{\sigma}_{f_s}^2}\left(\Delta_p^k\right)^2, \gamma\sqrt{\hat{\sigma}_{f_x}^2 + \hat{\sigma}_{f_s}^2}\left(\Delta_p^k\right)^2\right)$, where $\hat{\sigma}_{f_x}^2$ and $\hat{\sigma}_{f_s}^2$ are the posterior variance of the GP at the incumbent $\boldsymbol{x}$ and at the candidate point $\boldsymbol{s}$. γ is chosen by the user as a hyperparameters of the algorithm, as it also done in Sto-MADS.

Despite Sto-MADS uses a new method for generating the polling directions based on the Householder matrix to meet the nonsmooth optimality condition (Section 3.2 of Sto-BADS, Audet, Dzahini, et al. 2021), the LT-MADS (Audet and Dennis 2006) implementation of the polling directions still matches the nonsmooth optimality condition required for the convergence of the algorithm (Audet, Dzahini, et al. 2021). Therefore, in our implementation we kept the LT-MADS method, which is already present in BADS and used it for generating the polling

directions during the poll phase.

With all the above changes introduced in BADS and assuming that the Gaussian Process can ensure the variance condition of the function estimate, we satisfy all the requirements of Sto-MADS and fully adopt its framework. To prove the choice of the Gaussian Process as a the function estimates, we also worked on the theoretical aspect by providing a sketch of the proof reported in the Appendix A.1.

The implementation of this stochastic variant can be configured in PyBADS by passing the `stobads` flag to the `BADS` object, and a more detailed example of its usage is described in Appendix A.2).

3.3 Evaluation method

Evaluating and comparing result for global optimization problems is not a trivial task. This problem is more emphasized either in the case of stochastic objective functions, where the optimization problem is not characterized just by a single solution, or in high dimensional target problems. Indeed, sometimes the evaluation method is underestimated, and some algorithms can present biased results, showing great performance but performing very poor in the mean case. For example, rather than reporting the central tendency of the solution, some works have been reporting just the best result of the optimizer, thus making the evaluation unfair and biased. For these reasons, the assessing criteria needs to be fair and transparent by containing a detailed evaluation study of the solutions reported.

3.3.1 The bootstrapping evaluation method

Taking in consideration fairness and transparency, we report in this section the evaluation method used in our experiments, which has been designed for benchmarking BADS, by Acerbi and Ma 2017, and implemented in the *Inference Benchmark*[5] repository. The criteria for assessing the solution of the optimization problem is based on a bootstrapping method, and assesses separately deterministic with the stochastic cases Differentiating the two cases is an important matter, indeed when dealing with a stochastic target it's non-trivial to determine what an algorithm would return at an arbitrary point during the trajectory, because it's not necessarily the last point nor the best observed point - which instead we can do easily for a deterministic target, just get the best point until that point of the trajectory.

[5] infbench: https://github.com/lacerbi/infbench

Now, we briefly introduce the bootstrapping method and the benchmark procedure used in the experiments followed by some notations . All the optimizers are evaluated on a set of problems (also called tasks) $\mathcal{P} = \{P_1, ..., P_p\}$, every Problem P_i is defined by a target function, the dimension of the problem, its bounds, and the budget of maximum function evaluations. For each Problem P, we run n (default $n = 50$) independent optimization runs $R = \{r_1, r_2, ..., r_n\}$, with randomized starting points within the specific problem bounds. Each run r_i describes the solution of the optimization problem and contains the information of the trajectories of the optimization run made by the algorithm. For a fair comparison when comparing different optimizers the optimization runs are started from the same randomized starting points.

Once having all the runs results, we apply a bootstrapping method, i.e we randomly sample from R and concatenate the runs until we reach the maximum budget of function evaluations allowed by the problem. In the case that the last sampled run exceeds the problem budget, the last solution from the trajectory of the iterative algorithm satisfying the budget is taken. This process is repeated t times (e.g $t = 10000$), until we have constructed a set of bootstrap samples $\mathcal{S} = \{S_1, S_2, ..., S_t\}$ where $S_i = \{r_1, ..., r_b\}$ is the single bootstrap sample, with $r_i \in R$ describing the sampled run and r_b the last sample run satisfying the budget problem. Finally, we compute the mean across the chosen statistics of the bootstrap samples, $\mathrm{FSR} = \sum_{i=1}^{t} \frac{\mathrm{stat}(S_i)}{t}$, where the chosen statistics follows two different methods depending if the target is noisy or not that we are going to see separately. The result of the mean in both cases corresponds to a measurement expressed as a percentage corresponding to the *fraction solved rate* (FSR) of the selected problem. This metric is used in our experiments to report the performances of the optimizers in function of number of function evaluations (deterministic case) or in function of the error tolerance present in the solution (stochastic case).

To assess the error present in the solution proposed by the optimizer $\hat{f}_{\boldsymbol{x}_{end}}$, we assume to have access to the Oracle function value at the global minimum f_{min}, and we check if the error made by the algorithm is less than a given error tolerance ε, i.e $|\hat{f}_{\boldsymbol{x}_{end}} - f_{min}| \leq \varepsilon$ where $\varepsilon \in [0.01, 10]$ (log-spaced).

This bootstrapping method presents the excellent property of having a high statistical power because of the large runs sample size used and the randomic sampling method used for constructing the approximation of the final s, making the method detect the true solutions.

Deterministic target case

The default budget of function evaluations for solving a deterministic target is set to $500 \times D$, where D represents the dimension of the problem. The statistics used by the bootstrapping

method for the deterministic target corresponds to the fraction of *successful runs*, where a *successful run* is defined as the **current** best function value (taken from the trajectories of the optimizer) within the given error tolerance ϵ from the true solution of the oracle f_{min}. The solution in the deterministic case is represented by the **current best** function value incumbent that first satisfies the error tolerance. The fraction of successful runs is computed as a function of number of objective evaluations, and is averaged over different chosen ε. An example of a range of ε can be a 100 dimension row vector with linear spaced values between $[\exp(\log(0.01))... \exp(\log(10))]$. Moreover, to take in consideration the dimensionality of the problem the number function evaluations are divided by the the dimension of the problem.

Stochastic target case

When considering a noisy objective function, the default budget of function evaluations is set to $200 \times D$. The successful run is described by the last solution of the optimizer, $\boldsymbol{x}_{end}$. The choice of only considering the $\boldsymbol{x}_{end}$ solution is due to the noise present in the objective function and because we can not know which solution would be returned by the algorithm at each iteration. Indeed the solution of the algorithm may not be the lowest observed value.

For the stochastic case, we also compute the fraction of successful runs, but contrary to the deterministic case, we do not average across the error tolerance but we track the fraction rate as function of ε. In this case, the ε is defined in the range of $[\exp(\log(0.1))... \exp(\log(10))]$.

3.3.2 BBO-benchmark

The previous bootstrapping evaluation method for optimization problems has been implemented in an stand-alone Python project called Black Box Optimization (BBO) benchmark [6]. It has been developed with the aim of accommodating any black box optimizer, potentially independent from the programming language used.

BBO benchmark does not only include the bootstrapping evaluation method, but it is a whole benchmark tool that allows to test and compare different optimizers on several problems with multiple evaluation methods. Moreover, the code architecture of the benchmark has been developed to be general and easy to follow when an optimizer is needed to integrate or test in the framework. In addition, BBO benchmark is developed based on the Hydra[7] framework, an elegant tool that allows to compose and run configuration of an application by avoiding boil-

[6] BBO-benchmark: https://github.com/acerbilab/bbo-benchmark

[7] Hydra framework: https://hydra.cc

erplate codes and giving a highly adaptable architecture for different applications.
The benchmark comes with the following main key features:

- Hierarchical and composable implementation of an optimization benchmark.
- Run simulation of a configured optimizer on a designed benchmark.
- Integrate and implement a set of optimization problems.
- Run multiple simulations jobs with different arguments with a single command on a local machine or remotely on a High Performance Computing cluster.
- Provide performance evaluation of the optimizers on the defined problems.
- Evaluate optimizers made in different programming languages than Python, by loading and mapping their results into the benchmark

The hierarchical and composable structure comes from the implementation of general classes. Indeed, our framework describes a benchmark by defining a set of Problems called Tasks, represented by the object class `Task`, a configured optimizer algorithm described by the `Algorithm` class, and finally the `AlgEvaluator` class that describes which method is used for evaluating the optimization runs. All these components together defines an implementation of a benchmark, which are contained in the `OptimizationBenchmark` object class.
A concrete example of a benchmark instance can be described by the `BBOOptimizationBenchmark` object class designed for benchmarking black box optimizers. It takes as input an optimizer like PyBADS described by the wrapper class `PyBADS` of BBO benchmark, a target function that extends the `Task` class, and an implementation of the evaluator like the bootstrapping evaluation method described in the 3.3.1 section, which is implemented in the `PyBADSBootstrappingEvaluator` class. The `BBOOptimizationBenchmark` has the aim of running n independent runs of the optimizer with random starting points on the selected task and finally use the evaluator to assess the performance of the algorithm.
A full detailed description of the benchmark can be seen in the GitHub repository[8], and several examples can also be seen in the Appendix of this thesis, which provides some use cases for configuring and running different tasks on the benchmark A.2.

[8] bbo-benchmark https://github.com/acerbilab/bbo-benchmark

4

Experiments and Results

We provide in this chapter the experimental analysis carried out in our work. The main objective of our investigation is to analyze the performance of PyBADS against BADS . The comparison between them not only provides the information about the performance among the two algorithms, but it has been a crucial part of the work for debugging and testing PyBADS in order to detect possible issues present in its code. Moreover, we also compared the stochastic variant of PyBADS that integrates the existing features of BADS into the Sto-MADS framework.
This chapter first describes the design of the BBO benchmark by reporting the problems set and the procedure we followed for evaluating the algorithms. Section 4.2 compares BADS and PyBADS on several problems and discusses the main results we obtained by testing them on the benchmark. Afterwards, the last section describes the comparison of results we obtained with the stochastic variant of PyBADS (introduced in Section 3.2.4).

4.1 Benchmark

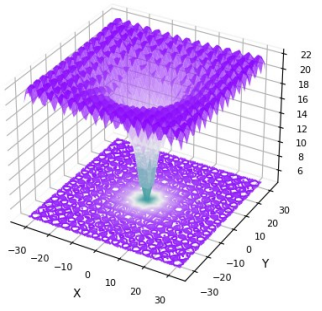
Both BADS and PyBADS have been evaluated by designing the BBO benchmark, which has been configured with 18 objective functions including noisy functions benchmark. Afterwards, the last section describes the comparison of results we obtained with the stochastic variant of PyBADS (introduced in Section 3.2.4). Moreover, since BADS is also able to handle heteroskedas-

tic noises, we also designed problems within this noise configuration, for a total of 20 objective functions in the presented benchmark.

All the deterministic functions and their stochastic variants are part of the standard benchmark for black box optimizations, and they have been taken and adapted from the IEEE-CEC 2014 expensive optimization test bed (Erlich et al. 2014). We evaluated all the problems on $D = \{2, 3, 6, 10\}$ dimensions. We configured the BBO benchmark for each optimizer with 50 independent runs per optimization problem. We used the bootstrapping evaluation method described in Section 3.3.1 for assessing and comparing the algorithms. This method suffices for evaluating the optimization problems because of its high statistical power. We set a budget of $500 \times D$ function evaluations for the deterministic targets and $200 \times D$ for the noisy cases. In all plots we omitted the error bars since the standard errors would be about the size of the line markers or less.

Before diving into the results, we first list all the problems and their configuration used in the benchmark to evaluate the algorithms.

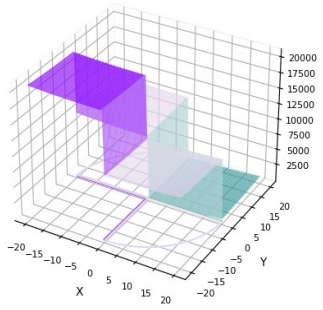
Ackley function



$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d} \cos(cx_i)\right) + a + \exp(1) \qquad (4.1)$$

Where $a = 20$, $b = 0.2$ and $c = 2\pi$ such that its global minimum is at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$. The function is characterized by a nearly flat outer region, and a large hole at the centre, with many local minima points. The bounds of the optimization problems are defined within the $x_i = [-32, 32]\, \forall i = 1, ..., D$ hypercube.

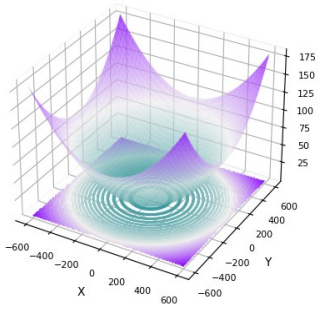
Cliff Function



$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 + 10^4 \mathbb{1}_{\left(\sum_{i=1}^{d} x_i < 0\right)} \tag{4.2}$$

The function is defined as a non-continuous step function characterized by some high step values. The domain of the problem is restricted in the hypercube made of $x_i = [-20, 20] \, \forall i = 1, ..., D$. The global minimum of the function is at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$.

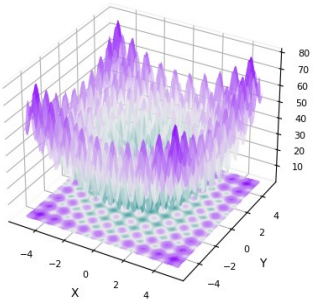
Griewank function



$$f(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{4.3}$$

This target is made by widespread local minima, which are regularly distributed due to the presence of the cosinusoidal functions. The bounds of the optimization problem in our case are defined in the range of $x_i = [-600, 600] \, \forall i = 1, ..., D$. The global minimum of the function is at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$.

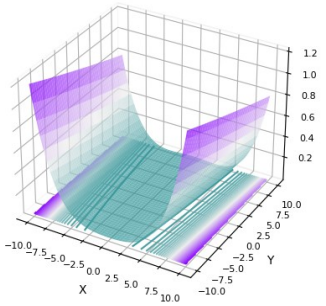
Rastrigin function



$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos\left(2\pi x_i\right) \right] \tag{4.4}$$

The function is made by several local minima and it is a highly multimodal function as we see from the picture, but locations of the minima are regularly distributed. The global minimum is located at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$. The domain of the function in our case are defined in the hypercube of $x_i = [-20, 20]\, \forall i = 1, ..., D$.

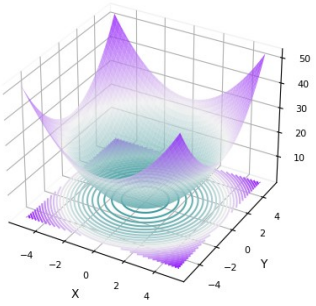
Rosenbrock's banana function



$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1} \left[b\left(x_{i+1} - x_i^2\right)^2 + a\left(x_i - 1\right)^2 \right] \tag{4.5}$$

The global minimum of this function is inside a long, narrow parabolic shaped flat valley, where finding the valley is trivial but the convergence to the global minimum is not a piece of cake. We set $a = 1$ and $b = 100$ such that its global minimum is at $\boldsymbol{x}^* = (1, ..., 1)$ with $f(\boldsymbol{x}^*) = 0$. The domain of optimization problem is defined within the $x_i = [-5, 5]\, \forall i = 1, ..., D$ hypercube.

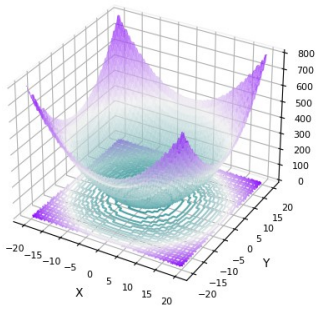
Sphere Function



$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{4.6}$$

The function has D local minima except for the global one which is at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$. This objective function is a convex and unimodal but it is non trivial to optimize due to different local minima in higher dimension and when is affected by some noise. The domain of he optimization problem is defined in $x_i = [-20, 20] \, \forall i = 1, ..., D$.

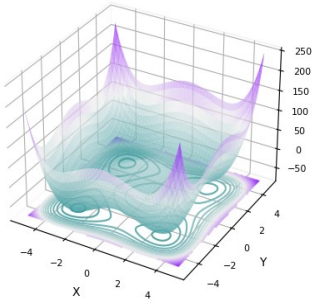
Step function



$$f(\mathbf{x}) = \sum_{i=1}^{d} \lfloor x_i + 0.5 \rfloor^2 \tag{4.7}$$

The function is defined as a non-continuous step function with a parabolic shape. The domain of the problem is restricted in the $x_i = [-20, 20] \, \forall i = 1, ..., D$ hypercube. The global minimum of the function is at $\boldsymbol{x}^* = (0, ..., 0)$ with $f(\boldsymbol{x}^*) = 0$.

Styblinskitang Function



$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{d}\left(x_i^4 - 16x_i^2 + 5x_i\right) \tag{4.8}$$

The function is very commonly used in many optimization benchmark. It presents a global minimum at $\boldsymbol{x}^* = (2.903534, , 2.903534)$ with $f(\boldsymbol{x}^*) = 39.16599D$. The domain of optimization problem is defined within the $x_i = [-5, 5]\,\forall i = 1, ..., D$ hypercube.

Note that the previous examples we have just mentioned are all deterministic functions, but in the benchmark we also considered stochastic targets. We implemented them by adding an i.i.d Gaussian noise with variance 1 to the function, i.e the noisy target is built as $g(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon_i \quad \varepsilon_i \sim \mathcal{N}(0, 1)$. A simple 3-D example of a stochastic function can be seen in Figure 4.1. Since BADS supports heteroskedastic noise, we also tested the optimizer on this configuration.

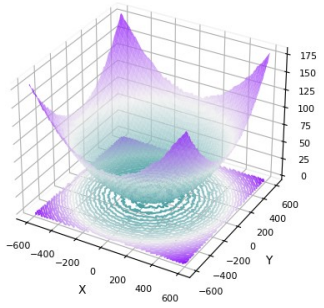


Figure 4.1: Noisy Griewank function
$f(\mathbf{x}) = \left(\sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right) + \varepsilon \quad \varepsilon \sim \mathcal{N}(0, 1)$

The two examples of functions with heteroskedastic noise chosen in the benchmark are based

on the Ackley and Rosenbrock functions such that the variance of the observations increases for solutions away from the optimum. More precisely, they correspond respectively to:

$$g(\mathbf{x}) = f_{Ackley}(\mathbf{x}) + \log \mathcal{N}(0,1) + f_{Ackley}(\mathbf{x}), \tag{4.9}$$

$$q(\mathbf{x}) = f_{Rosenbrock}(\mathbf{x}) + \log \mathcal{N}(0,1) + (f_{Rosenbrock}(\mathbf{x}) - 1). \tag{4.10}$$

4.2 BADS vs PyBADS

In this part of the chapter we analyze the results obtained from PyBADS and compare its performance with BADS. For a fair comparison, both algorithms have been set with their default configuration, as presented in Section 2.4 and suggested in BADS's paper (Acerbi and Ma 2017). To review the complete description of the configuration set for the optimizers, it is possible to access at the public GitHub repository of BADS [1], and follow the default options presented in the code.

Deterministic targets

To start with the comparison of the two algorithms, we first compared all the deterministic targets from a low dimension problem up to $D = 10$. In Figure 4.2, we reported some of the main results we obtained by running the algorithms on all the deterministic functions with $D = \{2, 3\}$ dimensions using the bootstrapping method of BBO benchmark. We reported in the x-axis, the number of function evaluations divided by the dimensionality of the problem and in the y-axis the fraction solved rate of the problem. We omitted to report some of the target functions of the benchmark for this dimensional configuration since their results were completely overlapping with the performance of BADS. We can see some little differences between the performance of the two optimizers in Figure 4.2. Both achieve very similar results and they show the same overall convergence rate to the optimum points.

However, we have found that the major differences in terms of performance between the algorithms are raised up by increasing the dimensionality of the problems. Indeed, in Figure 4.3 we reported the results obtained on the deterministic targets for $D = \{6, 10\}$ dimensional problems showing different performances among the two optimizers. We can see that for the Rastrigin target function BADS shows to perform better in the case of $D = 6$ as it is able to achieve a lower minimum point. On the other hand, PyBADS on the same target function achieves a lower minimum point when $D = 10$ with less number of function evaluations. De-

[1] BADS configuration: `https://github.com/acerbilab/bads/blob/master/bads.m#L184`

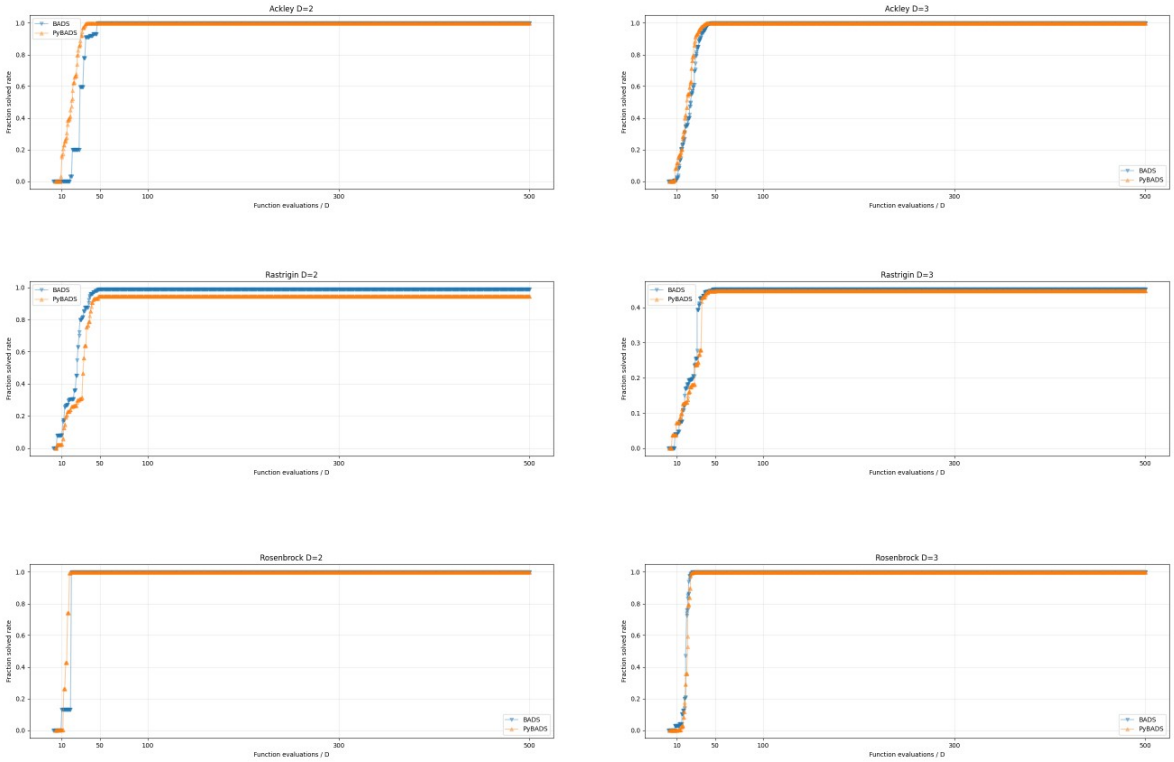


Figure 4.2: BBO benchmark deterministic functions for $D = \{2, 3\}$. Main optimization problems with the most relevant differences when comparing PyBADS with BADS. In the x-axis are reported the number of function evaluations divided by the dimension problems, and in the y-axis the Fraction of successful runs (for $\varepsilon \in [0.01, 10]$).

spite this fact, we have also to highlight that by increasing the dimensionality of the problem the optimization results in a more complex optimization task, and with the Rastrigin function both optimizers results in a solution that is far away from the global minimum - the maximum fraction solved rate achieved is less than 0.03.

Although we see these differences, sometimes PyBADS performs better (e.g when applied to Cliff) and other times BADS achieves better outcomes. If we average the results across the target functions and their dimension problems for both algorithms as shown in Figure 4.4, we see that PyBADS performs on average equally as BADS when considering all the deterministic functions of the benchmark.

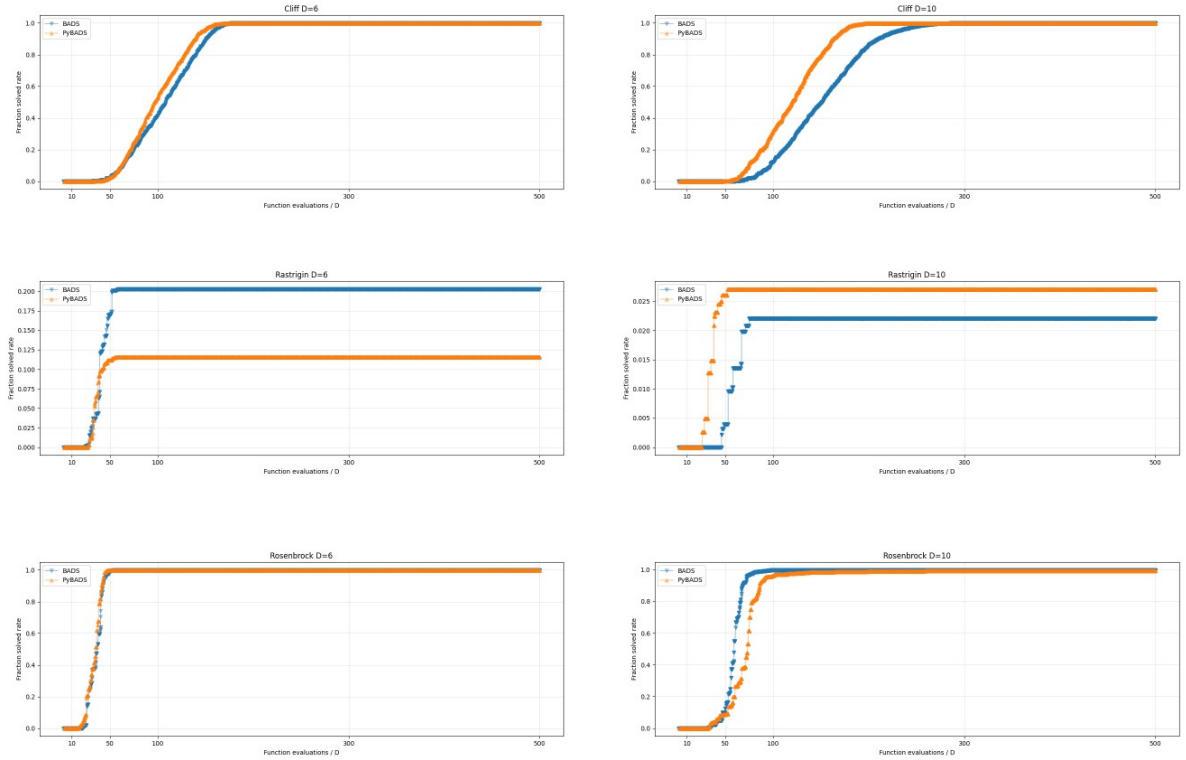


Figure 4.3: BBO benchmark deterministic functions for $D = \{6, 10\}$. Main optimization problems with relevant differences when comparing PyBADS and BADS. The x-axis reports the number of function evaluations divided by the dimension problems, and the y-axis presents the Fraction of successful runs (for $\varepsilon \in [0.01, 10]$).

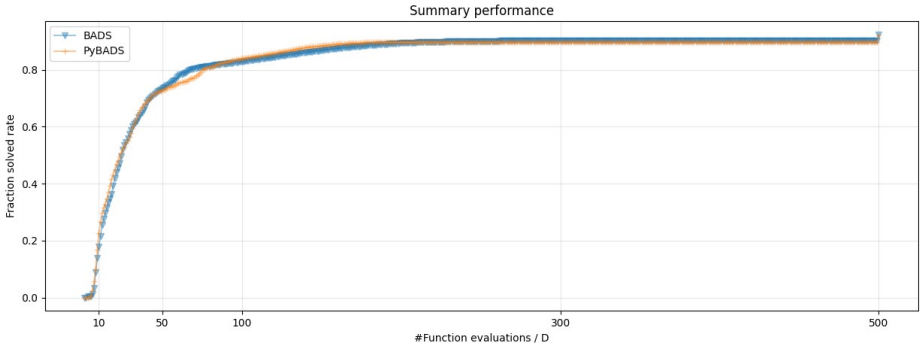


Figure 4.4: BBO benchmark deterministic targets for $D = \{2, 3, 6, 10\}$. Overall performance of PyBADS and BADS averaged across the 8 deterministic functions and among D dimensions. The x-axis reports the number of function evaluations divided by the dimension problems, and the y-axis presents the Fraction of successful runs (for $\varepsilon \in [0.01, 10]$).

Homoskedastic noisy targets

To fully investigate and asses PyBADS, we also need to analyze the stochastic cases and evaluate the behaviour of PyBADS in these cases. Therefore, in Figure 4.5 and Figure 4.6 we reported lower and higher dimensional problems of the BBO benchmark function with homoskedastic noise. Although, we run the algorithms on all the designed problems, we reported the tasks that presented relevant differences in terms of performance with BADS. We omitted the tasks which showed same results as BADS. Note that this time, we reported in the x-axis the error tolerance present in the algorithm since we are dealing with stochastic targets. When considering the low dimension functions, we clearly see that PyBADS outperformed or equally performed BADS. However, when taking in consideration higher dimension as shown in Figure 4.6, BADS performed slightly better.

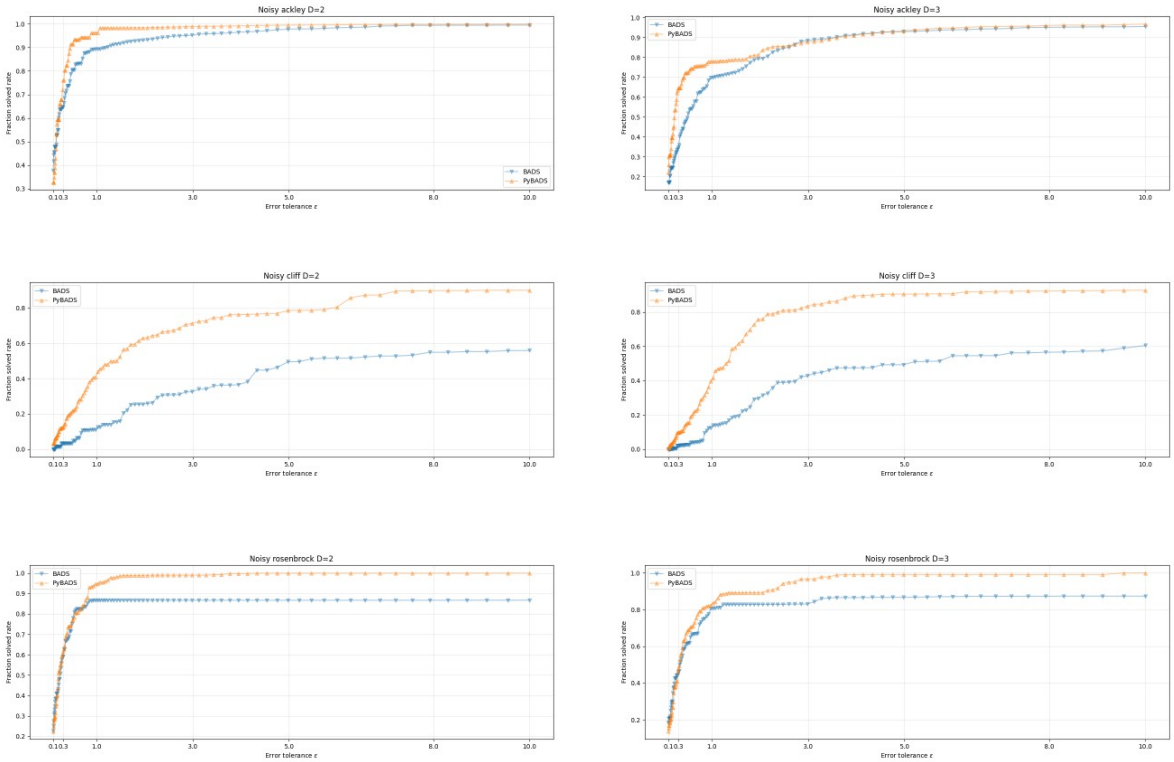


Figure 4.5: BBO benchmark noisy functions for $D = \{2, 3\}$. Performance comparison between PyBADS and BADS on relevant optimization problems with homoskedastic noise. The x-axis reports error tolerance, and the y-axis presents the Fraction of successful runs (for $\varepsilon \in [0.1, 10]$).

Among all these tasks, BADS demonstrated to achieve lower minimum points especially for the noisy Ackley function when $D = \{6, 10\}$, but also when considering the noisy Styblinskitang function with dimension $D = 6$. For these cases, we investigated the behaviour of PyBADS by analysing and comparing the runs of the optimization problems with BADS.

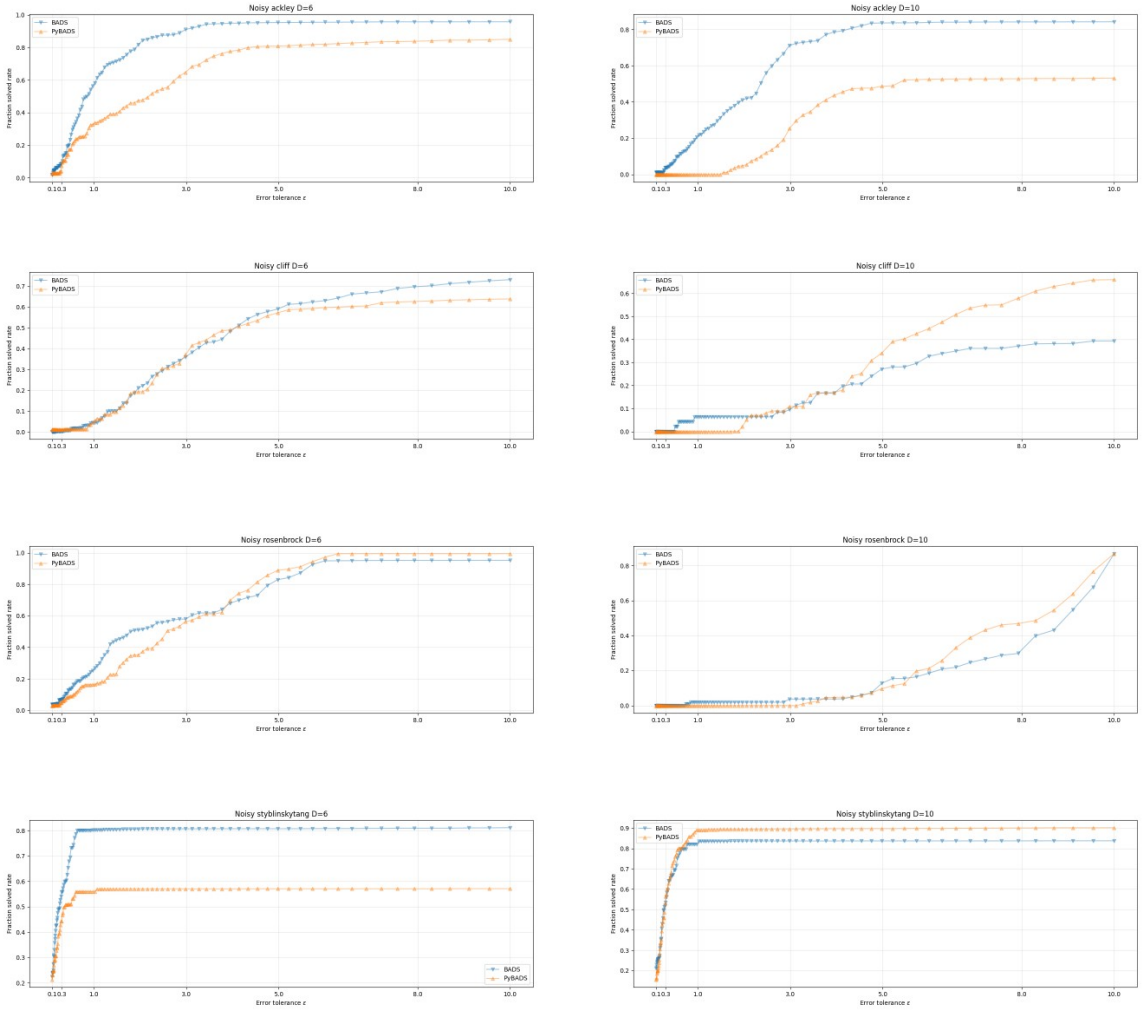


Figure 4.6: BBO benchmark noisy functions for $D = \{6, 10\}$. Main optimization problems with homoskedastic noise which have shown relevant differences between PyBADS and BADS. The x-axis reports error tolerance, and the y-axis presents the Fraction of successful runs (for $\varepsilon \in [0.1, 10]$).

By looking at the trajectories of the GP and the Oracle of each run as reported for the Ackley example in Figure 4.7, we can see that the results obtained by PyBADS is affected by some optimization runs where the algorithm is trapped in a minimum point in the beginning of the optimization. The same behaviour is also present in BADS as we can see on the right side of the Figure. However, this issue in BADS is less pronounced than PyBADS.

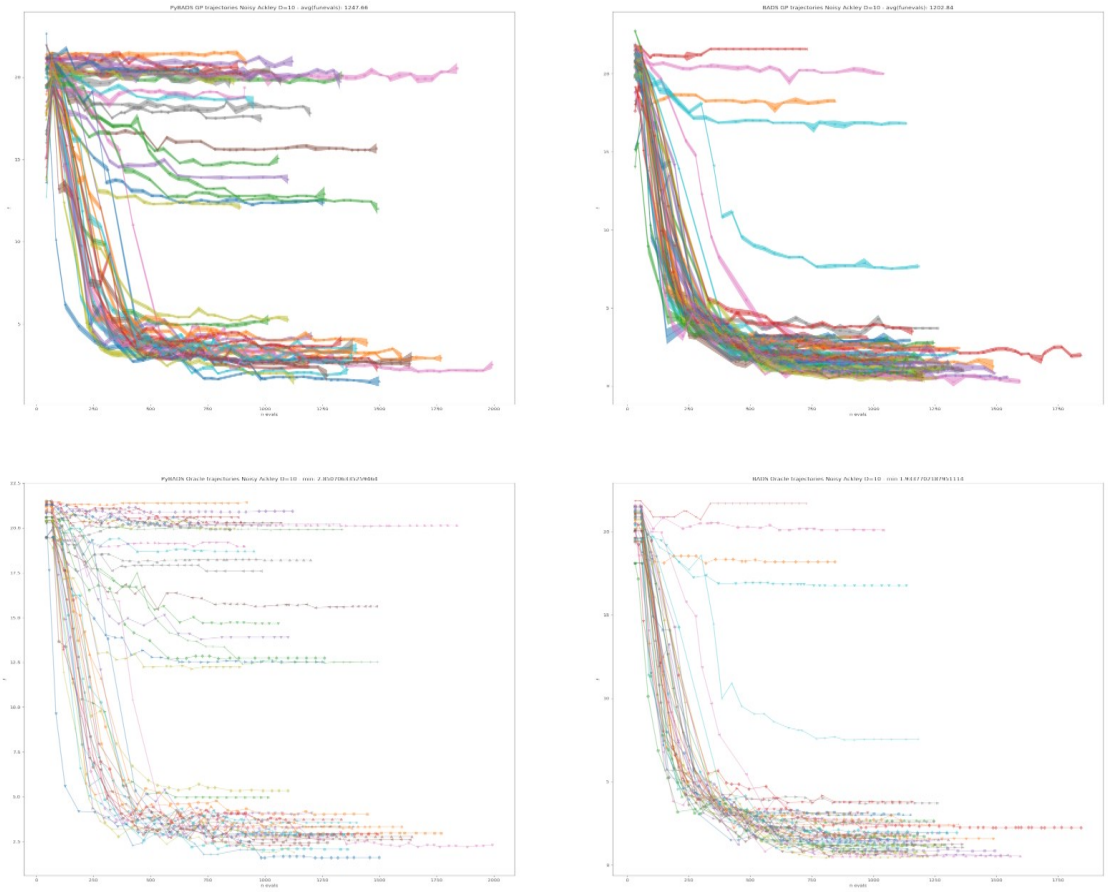


Figure 4.7: Ackley jittering. Example of trajectories line charts used to diagnose the noisy Ackley problem with $D = 10$. We report on x-axis the number of function evaluations and on the y-axis the oracle function evaluations or the GP prediction. The example shows the convergence of the algorithms and highlights the presence of some jittering behaviours in the beginning of the optimization.

The problem comes in both algorithms just with the stochastic targets, and by debugging the code we have concluded that this issue might raised up due to the combination of several heuristic methods used when dealing with a noisy target (recall the paragraph on Uncertainty Handling of Section 2.4). These policies might cause a zig-zagging behaviour when updating and assessing the incumbent with a candidate point. This issues happens rarely and it only happened in high dimensional problems for two particular configuration against the whole benchmark optimization problems we considered until now. Moreover, the heuristic methods applied for the noisy targets have shown in many optimization problems to be a good rule for making BADS exploring the mesh space and avoid local minima for the noisy targets.
Despite this particular issue, we can see from Figure 4.8 that also for the noisy targets PyBADS results on average across dimensions and tasks to perform the same or even better than BADS.

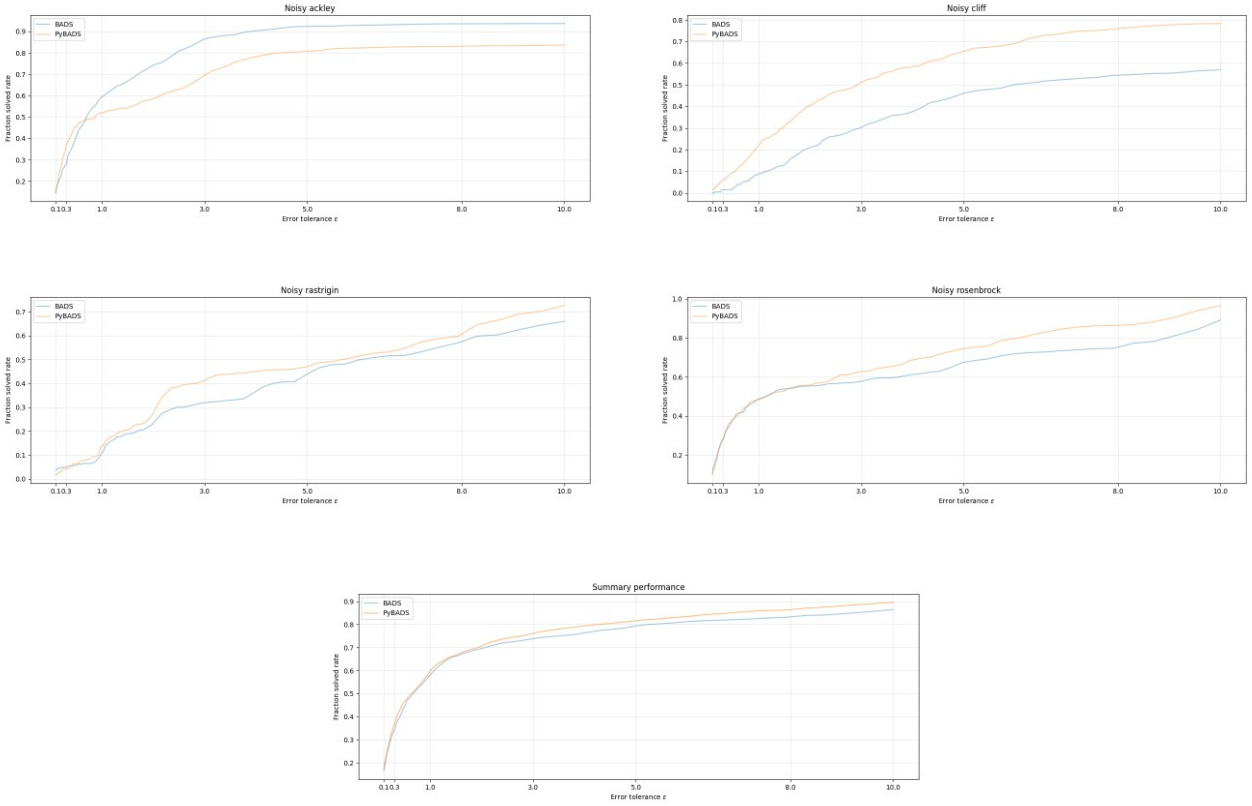


Figure 4.8: Summary of BBO benchmark noisy targets for $D = \{2, 3, 6, 10\}$. Overall performance of PyBADS and BADS on all 8 homoskedastic noisy functions. We first report the overall performance of PyBADS and BADS grouped by the most relevant target functions, by averaging the fraction solved rate across the D dimensions. Instead, in the last Figure we report the overall performance across dimensions and functions.

Briefly,we would also like to report that PyBADS as well handles heteroskedastic noise and we tested it by adding the designed noise to the Rosenbrock and Ackley functions. From Figure 4.9 we can see the comparison of the algorithms on the two targets function by evaluating them on $D = \{2, 3, 6, 10\}$ dimensions. The overall result highlights PyBADS to perform better than BADS for these cases.

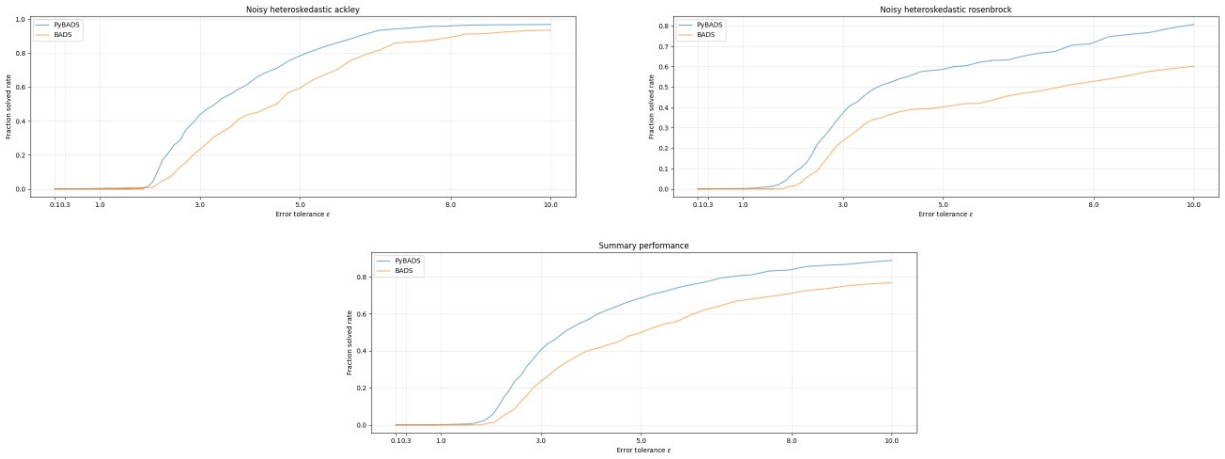


Figure 4.9: Heteroskedastic examples for $D = \{2, 3, 6, 10\}$. Overall performance on the Ackley and Rosenbrock heteroskedastic noisy functions across dimensions $D = \{2, 3, 6, 10\}$, and in the last figure we port the average performances among the two functions. The x-axis reports the error tolerance, and the y-axis presents the average of successful fraction runs (for $\varepsilon \in [0.1, 10]$).

4.3 Stochastic PyBADS (Sto-PyBADS)

In this section we analyze the Sto-PyBADS introduced in Section 3.2.4, and compare it with PyBADS and BADS using the BBO benchmark with the same homoskedastic noise configuration problems used when we compared BADS and PyBADS. We investigated Sto-PyBADS by running different instances of the algorithm for each problem of the benchmark. Every instance of the algorithm corresponded to a different configuration, that was obtained by setting the main γ hyperparameter of Sto-MADS, which defines the width of the uncertainty interval $\mathcal{I}_\gamma(\Delta_p^k)$ used for assessing the improvement of a candidate point.
We set $\gamma = \{1.96, 2, 5, 10, 15, 20, 25, 30\}$, other range of values have been also tried but we omitted them in this thesis because they led worse results. In Figure 4.10 we summarize outcomes of the algorithms by giving a summary of the performance across dimensions and tasks.

Unfortunately the algorithm has not been able to perform better, but just equally or sometimes worse than BADS and PyBADS. Despite for the Rosenbrock and Rastrigin noisy tasks, Sto-PyBADS showed to achieve lower minimum point. In addition, it is not clear which γ parameter should be chosen as a general rule. Sometimes small magnitude values of γ performed better other times we got the opposite results, and other times the parameter did not show much effect in the results. The overall performance of Sto-PyBADS across all the problems is just slightly worse than the optimizers of reference. However, a point in favour of Sto-PyBADS is that for most of the tasks it has been shown to resolve the given problems. Moreover, by proving the schematic proof proposed in Appendix A.1 it would demonstrate theoretical convergence of the algorithm towards stationary points for black box functions, which is currently lacking in the BADS/PyBADS for stochastic targets.

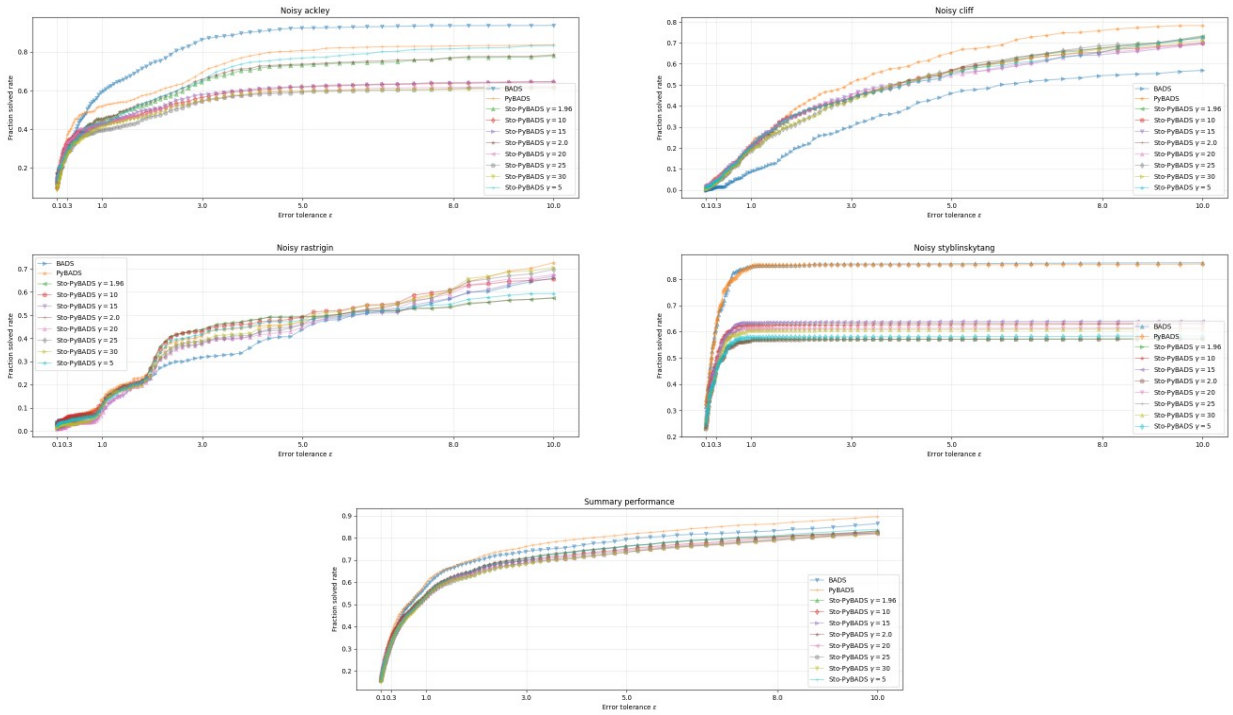


Figure 4.10: Summary of Sto-PyBADS on BBO benchmark noisy targets for $D = \{2, 3, 6, 10\}$ with different γ parameters. Overall performance of Sto-PyBADS on all 8 homoskedastic noisy functions. We reported the overall performance fot some of the noisy target functions.

5 Conclusion

From the experiments we carried out in this work, the results we obtained from the benchmark, as described in Chapter 4, have shown similar performance between PyBADS and BADS. The overall outcomes are very optimistic and PyBADS revealed to stand up its MATLAB version, especially for the stochastic targets. Moreover, seeing that the algorithms behaved as BADS for most of the tasks tested the correctness of the porting from Matlab to Python carried out in thesis. The study made in this work also involved the porting of BADS, but included the integration of Sto-MADS (Audet, Dzahini, et al. 2021) into BADS's framework, with the aim of improving the existing algorithm and providing a theoretical convergence guarantees of BADS for stochastic targets. This integration is contained in PyBADS as a configuration of the algorithm, and the major work involved by developing a new criteria for assessing and moving the incumbent by combining the uncertainty interval defined by Sto-MADS in its update rule and the usage of GP as function estimate. Although the method comes with neat structure, the experimental results have shown to not improve the existing performance of BADS. Despite the fact that Sto-PyBADS has been able to accomplish all the tasks, the overall outcomes are slightly worse than PyBADS or BADS. This result comes in favour of BADS and shows the effectiveness of the heuristics used by the method to deal with stochastic targets. However, the advantage of the approach proposed with the integration of Sto-MADS into BADS comes with the theoretical guarantees of the algorithm, which could be proved by following the schema of the proof described in the appendix.

The work accomplished by developing the Python implementation of BADS makes its usage

to a broader audience, which in last years has steadily risen each year, and we hope this trend to increase. Indeed, BADS has shown to compete different state-of-the-art optimizers on several benchmarks as described in its published paper (Acerbi and Ma 2017). More recently, it has also been tested on a new benchmark for black box optimization problems developed by D. Stenger and D. Abel for engineering applications (Stenger and Abel 2022), and achieved the top rank in most settings of the benchmark, resulting in the most sample efficient algorithm for $D = [3, 5]$ dimension problems only using $25 \times D$ function evaluations.

We end this work by leaving some discussion about future works. This thesis could still be extended in several ways and could involve different future plans. For example, a first priority would be to test PyBADS on more complex optimization tasks and model-fitting problems as carried out in BADS, using the CCN17 benchmark for cognitive and computation neuroscience studies (Acerbi and Ma 2017). This task would involve also to extend the developed BBO benchmark used for testing PyBADS and BADS.
Important future works would consist to provide a complete proof of BADS for theoretical zeroth-order convergence towards Clarke stationary point of non smooth functions, based on the new method we proposed in this thesis. Other possible horizons instead could consists in designing for BADS a smart multi-start optimization problem based on the existing work present in Bayesian Optimization, and investigate some of its heuristic approaches involved with stochastic targets by proposing other methods which could exploit approximate inference.

A
Appendix

A.1 Supplementary material

In this part of the Appendix, we give a sketch of the Sto-PyBADS proof to prove the convergence guarantees of the algorithm to the Clarke's stationary point, using the schema of the method described in Section 3.2.4 based on the framework proposed in Sto-MADS (Audet, Dzahini, et al. 2021). In particular, the aim of the proof is to demonstrate that the Gaussian Process surrogate model used in Sto-PyBADS matches the assumptions required by Sto-MADS for the convergence guarantees of the algorithm.
We first state the assumptions involved in the proof and the give the sketch of the proof.
Assumptions

- Let f the target function and defined as $f : \mathbb{R}^D \to \mathbb{R}$, affected by some unbiased noise, thus $\mathbb{E}_\Theta \left[f_\Theta(x) \right] = f(x)$
- Let y_i be the observation at x_i made on f, and defined as $y_i = f(x_i) + \tau(x_i)$, where $\tau \sim N\left(0, \sigma_\tau^2(x_i)\right)$ is a heteroskedastic noise. Moreover $\sigma_\tau^2\left(x_i\right) \leq V < +\infty$.
- Let the likelihood function distributed as $\mathbf{y} \mid X \sim \mathcal{N}\left(\mu_y, \Sigma_\tau\right)$
- Let GP $\left(\hat{f}_{\boldsymbol{\theta}}; \mu, \Sigma\right)$ be the prior GP with fixed hyperparameter $\boldsymbol{\theta}$ collecting the the mean function parameters, the length-scale and the signal variance of the covariance function, and the noise variance.

- Assume that the prior GP $\left(\hat{f}_{\boldsymbol{\theta}}; \mu, \Sigma\right)$ is smooth enough such that $f \in \left\{\text{GP}\left(\hat{f}_{\boldsymbol{\theta}}; \mu, \Sigma\right)\right\}$

From now on we note x^k as the incumbent and s^k as the candidate point at k-th iteration, retrieved from the poll or the search method.
The **objective of the proof** consists more formally into prove that:

(i) The sequence of estimates $\left\{\hat{f}^k\left(x^k\right), \hat{f}^k\left(s^k\right)\right\}$ is β-probabilistic ε_f–accurate for $\beta \in (0,1)$ and $\varepsilon_f > 0$, i.e :

$$\begin{aligned}\mathbb{P}\Big(\Big\{\Big|\hat{f}_x^k(x^k) - f(x^k)\Big| \leq \varepsilon_f \left(\Delta_p^k\right)^2\Big\} \cap \\ \Big\{\Big|\hat{f}_s^k(s^k) - f(s^k)\Big| \leq \varepsilon_f \left(\Delta_p^k\right)^2\Big\}\Big) \geq \beta \quad \forall\, k > 0,\end{aligned} \tag{A.1}$$

(ii) $\exists \kappa_f > 0$ such that the sequence of estimates $\left\{\hat{f}^k\left(x^k\right), \hat{f}^k\left(s^k\right)\right\}$ satisfies the following κ_f–variance conditions for all $k > 0$

$$\begin{aligned}\mathbb{E}\left[|\hat{f}_x^k - f_x^k|^2 | \hat{f}_s^k\right] \leq \kappa_f^2 \left(\Delta_p^k\right)^4 \text{ and} \\ \mathbb{E}\left[|\hat{f}_s^k - f_s^k|^2 | \hat{f}_x^k\right] \leq \kappa_f^2 \left(\Delta_p^k\right)^4 \forall k \geq 0.\end{aligned} \tag{A.2}$$

By Proving (i) +(ii), it follows from **Lemma** 1 of Sto-MADS (Audet, Dzahini, et al. 2021) that:

$$\begin{aligned}\mathbb{E}\left[\mathbb{1}_{\bar{J}_k} |\hat{f}_x^k - f_x^k|^2 | \hat{f}_s^k\right] \leq (1-\beta)^{1/2} \kappa_f \left(\Delta_p^k\right)^2 \text{ and} \\ \mathbb{E}\left[\mathbb{1}_{\bar{J}_k} |\hat{f}_s^k - f_s^k|^2 | \hat{f}_x^k\right] \leq (1-\beta)^{1/2} \kappa_f \left(\Delta_p^k\right)^2 \quad \forall k \geq 0,\end{aligned} \tag{A.3}$$

where $J_k = \left\{\hat{f}_x^k, \hat{f}_s^k \text{are } \beta\text{–probabilistic } \varepsilon_f\text{–accurate estimators}\right\}$ and
$\bar{J}_k = \left\{\hat{f}_x^k, \hat{f}_s^k \text{ are not } \beta\text{–probabilistic } \varepsilon_f\text{–accurate estimators}\right\}$.
Thanks to this Lemma, we satisfy the assumptions required in **Theorem** 1 of Sto-MADS (Audet, Dzahini, et al. 2021), and we fully integrate our function estimates into the framework of Sto-MADS by guaranteeing the convergence of the algorithm.

Proof steps

We now state the sketch of the proof to demonstrate points (i) + (ii), using the assumptions mentioned above.
Let $\left(y_x^k, y_s^k\right)$ be the observations at point $\left(x^k, s^k\right)$ at iteration k. Since the likelihood is nor-

mally distributed and the prior also follows a Normal distribution, the posterior and the prior distributions are conjugate distributions. Therefore, the probability distribution of the posterior at $\left(x^k, s^k\right)$ corresponds to a Bivariate Normal distribution and can be computed using the following closed form solution:

$$p(\hat{\boldsymbol{f}}^k|y_x^k, y_s^k) \sim \mathcal{N}\Big(\operatorname{diag}\left(\boldsymbol{\sigma}_\tau^{-2}\right)\left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_\tau^{-2}\right)\right)^{-1}\boldsymbol{y}^k + \Sigma^{-1}\left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_\tau^{-2}\right)\right)^{-1}\boldsymbol{\mu}, \left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_\tau^{-2}\right)\right)^{-1}\Big), \tag{A.4}$$

where $\boldsymbol{\sigma}_\tau^{-2}$ is the vector noise at $\left(x^k, s^k\right)$, $\Sigma^{-1} \in \mathbb{R}^{2\times 2}$ is the covariance matrix of the GP at the two points, and $\boldsymbol{\mu}$ the prior mean of the GP.
In addition, by sampling n_x and n_s new observations respectively at points x^k and s^k we obtain a more accurate posterior and reduce its variance, leading to the following distribution:

$$p(\hat{\boldsymbol{f}}^k|\bar{y}_x^k, \bar{y}_s^k) \sim \mathcal{N}\Big(\operatorname{diag}\left(\boldsymbol{\sigma}_{\tau_n}^{-2}\right)\left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_{\tau_n}^{-2}\right)\right)^{-1}\bar{\boldsymbol{y}}^k + \Sigma^{-1}\left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_{\tau_n}^{-2}\right)\right)^{-1}\boldsymbol{\mu}, \left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_{\tau_n}^{-2}\right)\right)^{-1}\Big), \tag{A.5}$$

where $\bar{\boldsymbol{y}}$ and $\boldsymbol{\sigma}_{\tau_n}^2$ are column vectors defined as:

$$\bar{y}_{x_n} = \frac{\sum_j^{n_x} y_x^{(j)}}{n_x}, \quad \operatorname{Var}\left[\bar{y}_{x_n}\right] = \sigma_{\tau_{x_n}}^2 = \frac{\sigma_{\tau_x}^2}{n_x}$$

$$\bar{y}_{s_n} = \frac{\sum_j^{s_x} y_s^{(j)}}{s_x}, \quad \operatorname{Var}\left[\bar{y}_{s_n}\right] = \sigma_{\tau_{s_n}}^2 = \frac{\sigma_{\tau_s}^2}{n_s}$$

$$\bar{\boldsymbol{y}} = \left[\bar{y}_{x_n}, \bar{y}_{s_n}\right]^\top \quad \boldsymbol{\sigma}_{\tau_n}^2 = \left[\sigma_{\tau_{x_n}}^2, \sigma_{\tau_{s_n}}^2\right]^\top.$$

We observe that by increasing the number of sampled observations at x^k and s^k the resulted variance $\boldsymbol{\sigma}_{\tau_n}^2$ will be driven to zero for the weak law of large numbers. On the other hand, by increasing the sample size at the two points $\bar{\boldsymbol{y}}$ will tend to the true expected value $\mathbb{E}_\Theta\left[f_\Theta(x)\right] = f(x)$ of the target function as the we assumed unbiased noise on f.
From the posterior distribution $p(\hat{\boldsymbol{f}}^k|y_x^k, y_s^k)$, we note that the last term of the mean function includes also the bias term $\Sigma^{-1}\left(\Sigma^{-1} + \operatorname{diag}\left(\boldsymbol{\sigma}_{\tau_n}^{-2}\right)\right)^{-1}\boldsymbol{\mu}$ made by the prior $\boldsymbol{\mu}$, which might be faraway from the true expected value $\mathbb{E}_\Theta\left[f_\Theta(\cdot)\right]$.

However, by increasing the number of the sample size at $\left(y_x^k, y_s^k\right)$, the resulting covariance matrix of the posterior distribution will drive the standard deviation at the two points to zero, and their expected value function will also converge to the true function by cancelling the bias term as more observations are collected. By having such control, there exists some n_x^* and n_s^* sample size such that the posterior of the GP satisfies the β–probabilistic ε_f–accurate condition for some $\beta \in (0, 1)$ (point (i) of the proof), and the κ_f–variance condition (point (ii) of the proof).
Assuring both conditions allows us to prove the results of **Lemma** 1 and **Theorem** 1 of Sto-MADS (Audet, Dzahini, et al. 2021), and to integrate our function estimates to the framework of Sto-MADS and ensure the convergence of the algorithm.

A.2 Code

A.2.1 Examples of usage of PyBADS

PyBADS defines an optimization problem by instantiating a `BADS` object, which receives as input the target function, the hard/plausible bounds, and optionally the initial point. Once defined the problem, to run the optimization we simply call the `run()` function, which returns a dictionary result containing the solution found by the optimizer and other useful information about the executed run.

A.2.2 Example 1 - Rosenbrock's banana function

In the first example of PyBADS, we show a basic example on the Rosenbrock's function (deterministic case). We define for this problem wide hard bounds and tighter plausible bounds that contain (hopefully) the solution, we set the Rosenbrock function as target function and passes as the initial point $x_0 = (0, 0)$. In the case that an initial point is not passed, PyBADS randomly samples it from the plausible/hard bounds. Note that the global minimum of the target function is found at $x^* = (1, 1)$

```
import numpy as np
from pybads import BADS

def rosenbrocks_fcn(x):
"""Rosenbrock's 'banana' function in any dimension."""
  x_2d = np.atleast_2d(x)
  return np.sum(100 * (x_2d[:, 0:-1]**2 - x_2d[:, 1:])**2 + (x_2d[:,
    0:-1]-1)**2, axis=1)

lb = np.array([-20, -20])      # Lower bounds
ub = np.array([20, 20])        # Upper bounds
plb = np.array([-5, -5])       # Plausible lower bounds
pub = np.array([5, 5])         # Plausible upper bounds
x0 = np.array([0, 0]);         # Starting point

bads = BADS(rosenbrocks_fcn, x0, lb, ub, plb, pub)
optimize_result = bads.optimize()
x_min = optimize_result['x']
fval = optimize_result['fval']
```

```
print(f"BADS minimum at: x_min = {x_min.flatten()}, fval = {fval:.4g}")
print(f"total f-count: {optimize_result['func_count']}, time: {round(optimize_result['total_time'], 2)} s")
print(optimize_result)
```

In the last part of the code we print the result of the optimization problem. The first two outputs show the minimum point and its estimated function found by the algorithm. In the last output we see the dictionary result of the optimization problem that contains all the useful information about the executed run, like the trajectories, the mesh size values per iteration and other useful attributes of the of the computed optimization.

A.2.3 Example 2 - Rosenbrock's banana function with constraint violations

PyBADS supports not just box constraints problems , but also handles function constraints violation. In the next example, we see PyBADS running with the previous target function but with unit circle constraint violations implemented by the `circle_constr` function.

```
import numpy as np
from pybads import BADS

def rosenbrocks_fcn(x):
    """Rosenbrock's 'banana' function in any dimension."""
    x_2d = np.atleast_2d(x)
    return np.sum(100 * (x_2d[:, 0:-1]**2 - x_2d[:, 1:])**2 + (x_2d[:, 0:-1]-1)**2, axis=1)

x0 = np.array([0, 0]);          # Starting point
lb = np.array([-1, -1])         # Lower bounds
ub = np.array([1, 1])           # Upper bounds

def circle_constr(x):
    """Return constraints violation outside the unit circle."""
    x_2d = np.atleast_2d(x)
    return np.sum(x_2d**2, axis=1) > 1

bads = BADS(rosenbrocks_fcn, x0, lb, ub, non_box_cons=circle_constr)
optimize_result = bads.optimize()

```

```
x_min = optimize_result['x']
fval = optimize_result['fval']

print(f"BADS minimum at: x_min = {x_min.flatten()}, fval = {fval:.4g}")
print(f"total f-count: {optimize_result['func_count']}, time: {round(optimize_result['total_time'], 2)} s")
print(f"Problem type: {optimize_result['problem_type']}")
```

A.2.4 Example 2 - quadratic homoskedastic noisy function

The second examples, involves a quadratic noisy function with i.i.d Gaussian noise. In this case, PyBADS detects automatically if the target function is noisy, by performing two consecutive function evaluations at the initial points and checking if they differ more than $1.5 \cdot 10^{-11}$. However, it is a good practice to set the `uncertainty_handling` option to `True` as shown in the example section A.2.5.

```
import numpy as np
from pybads import BADS
def noisy_sphere(x,sigma=1.0):
"""Simple quadratic function with added noise."""
  x_2d = np.atleast_2d(x)
  f = np.sum(x_2d**2, axis=1)
  noise = sigma*np.random.normal(size=x_2d.shape[0])
  return f + noise

x0 = np.array([-3, -3]);        # Starting point
lb = np.array([-5, -5])         # Lower bounds
ub = np.array([5, 5])           # Upper bounds
plb = np.array([-2, -2])        # Plausible lower bounds
pub = np.array([2, 2])          # Plausible upper bounds
bads = BADS(rosenbrocks_fcn, x0, lb, ub, plb, pub)
optimize_result = bads.optimize()
print(optimize_result)
```

A.2.5 Example 3 - quadratic heteroskedastic noisy function

Based on the previous target function, we now provide an example of a heteroskedastic function with user-specified noise, i.e the function not only returns the noisy estimate but also the

standard deviation of the noisy evaluation.
For this configuration we also specify to PyBADS some options about the problem:

- the `"uncertainty_handling": True` option set to True. It tells that the target is noisy, although it can be automatically detected, it is a good practice to specify it.

- the `"specify_target_noise": True` option set to True. It tells that the noisy target function provides the standard deviation of the noisy evaluations.

- the `"noise_final_samples": 100` option se to 100. It tells that the solution point is re-evaluated 100 times to build an accurate estimate of the function. In this case, we set it to 100 since the target function is not expensive. By default PyBADS uses 10 samples.

```
def noisy_sphere_estimated_noise(x, scale=1.0):
    """Quadratic function with heteroskedastic noise; also return
    noise estimate."""
    x_2d = np.atleast_2d(x)
    f = np.sum(x_2d**2, axis=1)
    sigma = scale*(1.0 + np.sqrt(f))
    y = f + sigma*np.random.normal(size=x_2d.shape[0])
    return y, sigma

x0 = np.array([-3, -3]);        # Starting point
lb = np.array([-5, -5])         # Lower bounds
ub = np.array([5, 5])           # Upper bounds
plb = np.array([-2, -2])        # Plausible lower bounds
pub = np.array([2, 2])          # Plausible upper bounds

options = {
  "uncertainty_handling": True,
  "specify_target_noise": True,
  "noise_final_samples": 100
}
bads = BADS(noisy_sphere_estimated_noise, x0, lb, ub, plb, pub,
    options=options)
optimize_result = bads.optimize()
```

A.3 Examples of usage of the BBO benchmark

In this part of the Appendix we provide some use case examples for running the BBO benchmark on the optimization problems we reported in the experiments chapter of this work (see Section 4).
To run an instance of the benchmark on the designed optimization problems, we need to configure four main components of the benchmark: the algorithm used for optimizing the function, the target to minimize, the evaluation method used, and the benchmark name instance. The BBO benchmark is based on the Hydra framework, which makes its configuration easy to configure for running multiple instances of the benchmark.
The results of the evaluation method are saved by default in the directory from which you executed the experiments under either the `outputs` or `multiruns` directories, where they are further organized by date and time. All the results of the evaluation method are stored as dictionaries in `.json` files. The output folders will also contain some default line charts, plotting the performance of the optimizer on the optimization problem ran in the benchmark.

We now provide some examples for running PyBADS on the BBO benchmark on several optimization problems, by starting from the most shallow configuration to more complex ones. All the examples we reported below use the bootstrapping method for evaluating the results of the algorithm, and the default configuration options of the benchmark described in Section 3.3.1, like the number of runs per optimization problem (50), the budget of the optimization problem ($200 \times D$ or $500 \times D$ number of function evaluations depending if the target is noisy or not).

The first example consists in running a single instance of BBO benchmark on the Rosenbrock's function with $D = 2$ dimensions.

```
[~/bbobench]$ python bbobench benchmark=bbo_benchmark algorithm=pybads
    evaluator=pybads_evaluator task=rosenbrock
```

Listing A.1: Simple Rosenbrock's function benchmark run

To run an optimization problem with a bigger dimension problem we just need to override the default configuration of the task by specifying the `task.options.D` attribute, e.g for $D = 6$ we have:

```
[~/bbobench]$ python bbobench benchmark=bbo_benchmark algorithm=
  pybads evaluator=pybads_evaluator task=rosenbrock ++task.options.D
  =6
```

From now on, we omit the `evaluator` argument since BBO benchmark uses by default the bootstrapping method for evaluating the results of the algorithms.
The BBO benchmark supports also multi-run operations, which allows the benchmark to be ran over a range of configurations with one single command lines using `-m` or `--multirun` flag.
For example, to run all the deterministic targets designed in our experiments we just need to run the following line on the bash:

```
[~/bbobench]$ python3 bbobench benchmark=bbo_benchmark algorithm=
  pybads task=ackley,rosenbrock,cliff,griewank,rastrigin,sphere,
  stepfunction,styblinskytang ++task.options.D=2,3,6,10 +seed=3 -m
```

Listing A.2: Command for benchmarking deterministic targets

With the previous commands we lunch the BBO benchmark on all the deterministic targets with $D = \{2, 3, 6, 10\}$ dimensions. In particular, we create an instance of the benchmark for each optimization problem which is identifies by the dimension D and the target function. The sequence of the instances can be executed in parallel when exploiting HPC services otherwise it will try to parallelize each configuration on the local machine. For enabling the HPC feature we just need to uncomment in the appropriate lines in `bbobench/config/default.yaml`:

```
- submitit-slurm-options
- override hydra/launcher: submitit_slurm
```

We also report a single line command example for running all the stochastic targets used in the experiments. For good practice we also specified the `uncertainty_handling` attribute of PyBADS since in this example we involve all noisy functions. But it can be omitted as PyBADS can automatically detect if the target function is noisy.

```
[~/bbobench]$ python3 bbobench benchmark=bbo_benchmark algorithm=
  pybads ++algorithm.kwargs.options.uncertainty_handling=True task=
  noisy_ackley,noisy_rosenbrock,noisy_cliff,noisy_griewank,
  noisy_rastrigin,noisy_sphere,noisy_stepfunction,
  noisy_styblinskytang ++task.options.D=2,3,6,10 +seed=3 -m
```

Listing A.3: Commnads for benchmarking homoskedastic noisy targets

Finally, we report the last two examples of this tutorial. The first one configures the BBO benchmark for evaluating the performance of PyBADS on the heteroskedastic target functions designed in our experiments.

```
[~/bbobench]$ python3 bbobench benchmark=bbo_benchmark algorithm=
  pybads ++algorithm.kwargs.options.uncertainty_handling=True ++
  algorithm.kwargs.options.specify_target_noise=True task=
  noisy_he_ackley, noisy_het_rosenbrock +seed=3 -m
```

Listing A.4: Command line for benchmarking heteroskedastic noisy targets

The next example shows the command lines needed for running Sto-PyBADS on some noisy targets with different values of the γ parameter of the uncertainty interval of the method (see Section 3.2.4).

```
[~/bbobench]$  python3 bbobench benchmark=bbo_benchmark algorithm=
  sto_pybads ++algorithm.kwargs.gamma_uncertain_interval=1.96,2.0,5.
  task=noisy_ackley,noisy_cliff,noisy_griewank,noisy_rastrigin,
  noisy_rosenbrock,noisy_sphere,noisy_stepfunction,
  noisy_styblinskytang task.kwargs.D=3,6 +seed=3 -m
```

Listing A.5: Sto-PyBADS benchmark command line example

Bibliography

Acerbi, Luigi (2020). "Variational Bayesian Monte Carlo with Noisy Likelihoods." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 8211–8222.

Acerbi, Luigi and Wei Ji Ma (2017). "Practical Bayesian Optimization for Model Fitting with Bayesian Adaptive Direct Search." In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc.

Agnihotri, Apoorv and Nipun Batra (2020). "Exploring Bayesian Optimization." In: *Distill*. DOI: 10.23915/distill.00026. URL: https://distill.pub/2020/bayesian-optimization.

Audet, Charles, Vincent Béchard, and Sébastien Le Digabel (June 2008). "Nonsmooth optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search." In: *Journal of Global Optimization* 41.2, pp. 299–318. ISSN: 1573-2916. DOI: 10.1007/s10898-007-9234-1.

Audet, Charles and J. Dennis (Jan. 2006). "Mesh Adaptive Direct Search Algorithms for Constrained Optimization." In: *SIAM Journal on Optimization* 17, pp. 188–217. DOI: 10.1137/060671267.

Audet, Charles, Kwassi Joseph Dzahini, et al. (May 2021). "Stochastic Mesh Adaptive Direct Search for Blackbox Optimization Using Probabilistic Estimates." In: *Computational Optimization and Applications* 79.1, pp. 1–34. ISSN: 0926-6003. DOI: 10.1007/s10589-020-00249-0.

Belyaev, Mikhail, Evgeny Burnaev, and Yermek Kapushev (2014). *Exact Inference for Gaussian Process Regression in case of Big Data with the Cartesian Product Structure*. DOI: 10.48550/ARXIV.1403.6573.

Bergstra, James et al. (July 2015). "Hyperopt: A Python library for model selection and hyperparameter optimization." In: *Computational Science and Discovery* 8, p. 014008. DOI: 10.1088/1749-4699/8/1/014008.

Booker, A. J. et al. (Feb. 1999). "A rigorous framework for optimization of expensive functions by surrogates." In: *Structural optimization* 17.1, pp. 1–13. ISSN: 1615-1488. DOI: 10.1007/BF01197708.

Brochu, Eric, Vlad M. Cora, and Nando de Freitas (Dec. 2010). "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning." In: *CoRR* abs/1012.2599. DOI: 10.48550/ARXIV.1012.2599.

Brochu, Eric, Nando de Freitas, and Abhijeet Ghosh (2007). "Active Preference Learning with Discrete Choice Data." In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Vancouver, British Columbia, Canada: Curran Associates Inc., pp. 409–416. ISBN: 9781605603520.

Cisbani, E. et al. (May 2020). "AI-optimized detector design for the future Electron-Ion Collider: the dual-radiator RICH case." In: *Journal of Instrumentation* 15.05, P05009–P05009. DOI: 10.1088/1748-0221/15/05/p05009.

Clarke, F. H. (1990). *Optimization and Nonsmooth Analysis*. Reprint edition. Vol. 5. SIAM Pubblications.

Digabel, Sébastien Le and Robert B. Gramacy (2011). "The mesh adaptive direct search algorithm with treed Gaussian process surrogates." In.

Duvenaud, David (Apr. 2015). "Automatic model construction with Gaussian processes." In.

Erlich, I. et al. (Sept. 2014). "Solving the IEEE-CEC 2014 expensive optimization test problems by using single-particle MVMO." In: *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pp. 1084–1091. DOI: 10.1109/CEC.2014.6900517.

Frazier, P. (2018). "A Tutorial on Bayesian Optimization." In: *ArXiv* abs/1807.02811.

Frazier, Peter I. and Jialei Wang (Dec. 2015). "Bayesian Optimization for Materials Design." In: *Information Science for Materials Discovery and Design*. Springer International Publishing, pp. 45–75. DOI: 10.1007/978-3-319-23871-5_3.

Gardner, Jacob et al. (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration." In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc.

Garnett, Roman (2022). *Bayesian Optimization*. Cambridge University Press.

Görtler, Jochen, Rebecca Kehlbeck, and Oliver Deussen (2019). "A Visual Exploration of Gaussian Processes." In: *Distill*. DOI: 10.23915/distill.00017. URL: https://distill.pub/2019/visual-exploration-gaussian-processes.

Gramacy, Robert B. and Herbert K. Lee (May 2012). "Cases for the Nugget in Modeling Computer Experiments." In: *Statistics and Computing* 22.3, pp. 713–722. ISSN: 0960-3174. DOI: 10.1007/s11222-010-9224-x.

Gutmann, Michael U., Jukka Cor, and er (2016). "Bayesian Optimization for Likelihood-Free Inference of Simulator-Based Statistical Models." In: *Journal of Machine Learning Research* 17.125, pp. 1–47.

Hansen, Nikolaus, Sibylle Müller, and Petros Koumoutsakos (Feb. 2003). "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)." In: *Evolutionary computation* 11, pp. 1–18. DOI: 10.1162/106365603321828970.

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy." In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

Hoffman, Matthew, Eric Brochu, and Nando de Freitas (2011). "Portfolio Allocation for Bayesian Optimization." In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. UAI'11. Barcelona, Spain: AUAI Press, pp. 327–336. ISBN: 9780974903972.

Jones, Donald, Matthias Schonlau, and William Welch (Dec. 1998). "Efficient Global Optimization of Expensive Black-Box Functions." In: *Journal of Global Optimization* 13, pp. 455–492. DOI: 10.1023/A:1008306431147.

Kapoor, Ashish et al. (June 2010). "Gaussian Processes for Object Categorization." In: *International Journal of Computer Vision* 88.2, pp. 169–188. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0268-3.

Kim, Hyun-Chul and Jaewook Lee (2007). "Clustering Based on Gaussian Processes." In: *Neural Computation* 19.11, pp. 3088–3107. DOI: 10.1162/neco.2007.19.11.3088.

Lizotte, Daniel et al. (2007). "Automatic Gait Optimization with Gaussian Process Regression." In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., pp. 944–949.

Mockus, Jonas (2012). *Bayesian approach to global optimization: theory and applications*. Springer Science and Business Media.

Øksendal, Bernt (2003). "Some Mathematical Preliminaries." In: *Stochastic Differential Equations: An Introduction with Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 7–20. ISBN: 978-3-642-14394-6. DOI: 10.1007/978-3-642-14394-6_2.

Picheny, Victor et al. (2013). "Quantile-Based Optimization of Noisy Computer Experiments With Tunable Precision." In: *Technometrics* 55.1, pp. 2–13. DOI: 10.1080/00401706.2012.707580.

Quinonero-Candela, Joaquin, Carl Edward Ramussen, and Christopher K I Williams (2007). "Approximation Methods for Gaussian Process Regression." In: p. 24.

Rossi, Simone et al. (2020). *Sparse Gaussian Processes Revisited: Bayesian Approaches to Inducing-Variable Approximations*. DOI: 10.48550/ARXIV.2003.03080.

Serafini, David et al. (Feb. 1998). "Optimization Using Surrogate Objectives On a Helicopter Test Example." In: DOI: 10.1007/978-1-4612-1780-0_3.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc.

Stenger, David and Dirk Abel (2022). "Benchmark of Bayesian Optimization and Metaheuristics for Control Engineering Tuning Problems with Crash Constraints." In: *CoRR* abs/2211.02571. DOI: 10.48550/arXiv.2211.02571. arXiv: 2211.02571.

Swersky, Kevin et al. (Mar. 2020). "Amortized Bayesian Optimization over Discrete Spaces." In: *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*. Ed. by Jonas Peters and David Sontag. Vol. 124. Proceedings of Machine Learning Research. PMLR, pp. 769–778.

Tran, Dustin, Rajesh Ranganath, and David M. Blei (2015). *The Variational Gaussian Process*. DOI: 10.48550/ARXIV.1511.06499.

Wilson, Andrew Gordon, Christoph Dann, and Hannes Nickisch (2015). "Thoughts on Massively Scalable Gaussian Processes." In: *ArXiv* abs/1511.01870.

Xiao, Yuxin, Eric P. Xing, and Willie Neiswanger (2021). *Amortized Auto-Tuning: Cost-Efficient Bayesian Transfer Optimization for Hyperparameter Recommendation*. DOI: 10.48550/ARXIV.2106.09179.