SAPIENZA
UNIVERSITÀ DI ROMA

# Statistical Black-Box Global Optimization


Ingegneria dell'Informazione, Informatica e Statistica
Computer Science - Informatica

**Lorenzo Levantesi**
ID number 1785694

Advisor
Prof. Enrico Tronci


Academic Year 2022/2023

**Statistical Black-Box Global Optimization**
Sapienza University of Rome



This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: levantesi.1785694@studenti.uniroma1.it

# Abstract

Black-box optimization is a challenging optimization problem, where the gradient information about the objective function is not available. The current black-box global optimizers may have global optimality convergence results, but using them in real-world problems may be difficult and expensive in terms of computational resources and execution time, as no indications are available regarding when to stop the optimization procedure. The new black-box optimization method presented is the *Statistical Black-Box Global Optimization* algorithm, which is based on Statistical Hypothesis Testing. *Statistical Black-Box Global Optimization* takes in input a black-box solver *BBO*, two values $\delta, \epsilon \in (0, 1)$ and a black-box function $f$. The algorithm returns in output a solution x where, with confidence $1 - \delta$, the probability that the optimizer *BBO* finds a solution on the objective function $f$ that is better than x is $< \epsilon$. This also means that, if a solver *BBO* has a probability $\geq \epsilon$ of finding the optimal solution of an objective function $f$, by setting $\delta$ to a very low value, with a high probability the optimal solution of $f$ is returned by the *Statistical Black-Box Global Optimization* method. Statistical Hypothesis Testing has been performed on two state-of-the-art black-box local optimizers, namely *Nevergrad* and *NOMAD*. The *Statistical Black-Box Global Optimization* method applied to *Nevergrad* and *NOMAD* has been tested on 25 well-known test functions for optimization, and the results have been compared with the ones obtained by the state of the art black-box global optimizer *DIRECT-GL*. The results show that the *Statistical Black-Box Global Optimization* method transforms both the *Nevergrad* and *NOMAD* local optimizers into global optimizers on all the test functions, while *DIRECT-GL* fails to solve some of the test functions with high dimensions. Even if *DIRECT-GL* fails to solve all the problems, it uses significantly fewer function evaluations of the objective function with respect to the *Statistical Black-Box Global Optimization* method on *Nevergrad* and *NOMAD*. By estimating a lower bound on the probability of *Nevergrad* to find an optimal solution, the value of $\epsilon$ has been increased as *Nevergrad* performed very well on the test functions. This resulted in much fewer function evaluations used by the *Statistical Black-Box Global Optimization* method on *Nevergrad*, outperforming the state-of-the-art black-box global optimizer *DIRECT-GL* also under this aspect, and, most importantly, the global optimality of *Nevergrad* under the *Statistical Black-Box Global Optimization* method has been kept.

# Contents

# Chapter 1

# Introduction

During the last few years, there has been a lot of interest in the black-box optimization (BBO) discipline, and many new derivative-free (or black-box) optimization algorithms have been devised. Given the rising of new and complex challenges where the objective function is a black-box, these methods have been employed to tackle down multidisciplinary real-world problems [32] [11] [4] [3] [23] [12] [1] [13] [38].

Black-box optimization is a field which encompasses a large number of different optimization algorithms, all with the same aim: given a black-box objective function $f$ with domain $\Omega$, find:

$$\mathrm{x}^* \in \arg \min_{\mathrm{x} \in \Omega \subseteq \mathbb{R}^n} f(\mathrm{x})$$

Black-box optimization addresses problems where the gradient information about $f$ is not available. These types of problems are frequent in real-world scenario, for example when $f(\mathrm{x})$ is the result of an experiment or a complex computer simulation, where derivatives of the objective function can not be computed due to the unavailability of the source code. Then, black-box optimizers must perform optimization on $f$ without using the gradient information.

The types of **derivative-free algorithms** that are most frequent in the literature studies are the *direct-search methods*, *model-based methods* and *implicit filtering methods*. *Direct-search methods* iteratively search an optimum point of $f$ by evaluating a poll of candidate points around the incumbent solution. This simple framework inspired many derivative-free algorithms [25] [7] [27] [28], where for some of them convergence results to local minima have been proved. *Model-based* derivative-free methods are a class of algorithms that, during the optimization phase, use and build an internal surrogate model of the objective function (or of any function of interest during the optimization) using only zero-order information. *Implicit filtering* is a method that uses finite differences, more specifically, it is a hybrid of a grid-search algorithm and a Newton-like local optimization method, where the gradient estimation for local optimization is done with the use of the *central differences*, in which the difference parameter is decreased when no lowest point is found around the current best point. This technique is similar to the one of *direct-search methods*, but implicit filtering is not polling opportunistically, i.e., all the candidate points are evaluated in each iteration, without breaking the iteration when a new incumbent solution is found.

Obviously, one would like to have a method which always converges to the global optimum of any black-box function (i.e., a black-box global optimizer), in a reasonable number of steps and using a feasible quantity of resources. The methods cited above, and many other *derivative-free* methods, are local optimizers, where convergence to the global optimum of any black-box function $f$ can not be achieved. In fact, under the assumption that $P \neq NP$ [14], there are no general algorithms that solve every global optimization problem in polynomial time [24] [26], even though there exist many methods which have guarantees of convergence to a global optimum for problems that satisfy specific requirements.

## 1.1 Motivations

Even if a global optimizer of any objective function $f$ cannot be obtained, there exist black-box optimization algorithms that have been proved to converge to the **global optimum** of a black-box function $f$, under small assumptions on $f$. Even if these methods have been proved to converge to the global optimum of a function $f$, there is no guarantee of when the global optimum is found, then it is not known when to stop the optimization to obtain the optimum point. Usually, these methods are stopped when a maximum number of function evaluations is reached or when the execution time exceeds a given threshold value.

These global optimizers also have difficulty in optimizing high dimensional objective functions, needing a large number of function evaluations and many computational resources to return a solution "accurate enough".

Finally, to guarantee global convergence properties, all these black-box global optimizers need the domain of the objective function to be constrained, i.e., $\Omega \subset \mathbb{R}^n$, which is not always the case for black-box optimization problems that can rise in the real-world.

## 1.2 Contributions

The new ***Statistical Black-Box Global Optimization*** method is presented. It is an algorithm for black-box optimization based on *Statistical Hypothesis Testing*, with which *statistical black-box global optimization* can be obtained. More specifically, the algorithm takes in input a black-box optimizer *BBO*, a black-box function $f$ and two values $\delta, \epsilon \in (0, 1)$. The algorithm will run the *BBO* optimizer multiple times, keeping always the best solution found, and it will stop when $N$ consecutive failures are obtained (i.e., no value better than the incumbent solution is found). The value of $N$ is computed from the values of $\delta$ and $\epsilon$, i.e., $\lceil \ln(\delta) / \ln(1-\epsilon) \rceil$. At termination, the solution returned by the algorithm is, with statistical confidence $1-\delta$, the optimal solution that *BBO* can find on $f$, and the probability that *BBO* returns a solution that is better than the one returned by the *Statistical Black-Box Global Optimization* algorithm is $< \epsilon$.

Then, if small enough values of $\delta$ and $\epsilon$ are given in input to the *Statistical Black-Box Global Optimization* method, the result obtained is, with high probability, the best result that can be obtained with *BBO* on $f$. Another important point of view is that, if *BBO* has a probability $\geq \epsilon$ to find the **global optimum** of the objective

function $f$, with high probability the solution returned by the *Statistical Black-Box Global Optimization* is the global optimal one. The probability with which this result is obtained depends on the choice of the values of $\delta$ and $\epsilon$.

The *Statistical Black-Box Global Optimization* is an important algorithm in order to obtain with high confidence the best result that a black-box solver can obtain when optimizating a black-box function $f$. One drawback of the algorithm is that, if the *BBO* has a low probability of finding the optimal solution of $f$, and so the value of $\epsilon$ is very low, the number of function evaluations used may become larger than the one used by a black-box global optimizer. Using a good solver that have high probability of finding an optimal solution, and then an higher value of $\epsilon$ can be set, uses much less function evaluations, outperforming also the state of the start methods of black-box global optimization in terms of function evaluations. This is done in Chapter 5, where the lower bound of the probability of the solver to find an optimal solution is estimated, outperforming a state of the art method for black-box global optimization, both in number of solved problems and number of function evaluations used.

## 1.3 Related work

One of the first **black-box global optimization** algorithms is *DIRECT* [20]. The ***DIRECT*** algorithm addresses only black-box optimization problems where the domain $\Omega$ of the objective function is bounded (i.e., $\Omega \subset \mathbb{R}^n$). *DIRECT* treat the function domain $\Omega$ as a unit hypercube and repeatedly split the "potentially optimal" rectangles, where each rectangle is evaluated by computing the function value on its centre point.

Another important black-box global optimization algorithm is the ***Multilevel Coordinate-Search (MCS)*** [18], which is inspired from *DIRECT*. The main difference between the two methods is that in *MCS* the rectangles are organized in levels. A level $s \in \{0, 1, \ldots, s_{\max}\}$ is assigned to each rectangle. When a rectangle is divided, the level of its descendants becomes $s+1$ (or $\min(s+2, s_{\max})$), while its level becomes 0 and will be ignored for the rest of the execution. Thus, rectangles with low levels represent large rectangles that are not been split many times, while rectangles with level $s_{\max}$ are considered too small to be split.

In each iteration, the method considers the rectangle in each level with the lowest function value to be split. Then, a global search in the algorithm is performed when considering rectangles of low level, while a local search is executed with rectangles of higher levels.

Another difference concerning DIRECT is that *MCS* consider the function values at boundary points, rather than the center points.

*MCS* is guaranteed to converge to a global optimum if the objective function is at least continuous in the neighbourhood of a global minimum.

The global optimizer *MCS* and a *DIRECT*-based optimizer *glcCluster* [17] have been tested in the benchmark of Rios and Sahinidis [30], with other 20 derivative-free solvers.

*glcCluster* was one of the best solvers in the **benchmark** of Rios and Sahinidis [30], in which 502 problems are considered, with the dimension of the functions

| Convex | | Nonconvex | | **Total** |
|---|---|---|---|---|
| Smooth | Nonsmooth | Smooth | Nonsmooth | |
| 79.50% | 42.90% | 72.20% | 22.20% | 62.20% |

**Table 1.2.** Percentage of problems solved by *glcCluster* in benchmark [30] with 1.1 as convergence test.

| Convex | | Nonconvex | | **Total** |
|---|---|---|---|---|
| Smooth | Nonsmooth | Smooth | Nonsmooth | |
| 75.60% | 12.40% | 71% | 27.80% | 51.40% |

**Table 1.4.** Percentage of problems solved by *MCS* in benchmark [30] with 1.1 as convergence test.

ranging from 1 to 300, and, furthermore, 263 of them are nonconvex functions. In the benchmark, the solvers are executed 10 times on each problem, for a maximum of $2,500$ function evaluations, with the starting point of each run drawn from a uniform distribution on the function's box-bounded domain. A problem $f$ is considered solved if the median x of the returned solutions of the 10 different runs is within 1% of the solution $\mathrm{x}^*$ of the global optimum:

$$f(\mathrm{x}) \leq \max(1.01 f(\mathrm{x}^*),\ f(\mathrm{x}^*) + 0.01) \tag{1.1}$$

The results obtained by *glcCluster* are represented in Table 1.2, while the ones obtained by *MCS* are in Table 1.4.

In the benchmark also the following convergence test is considered:

$$f(\mathrm{x}_0) - f(\mathrm{x}) \geq (1 - \tau)(f(\mathrm{x}_0) - f_L) \tag{1.2}$$

where $\tau > 0$ is a tolerance, $\mathrm{x}_0$ is the starting point of the algorithm, and $f_L$ is the smallest value of $f$ obtained by any solver considered in the benchmark within the limit of 2500 function evaluations. This convergence result tells us if the optimization algorithm has reached a point x such that the reduction $f(\mathrm{x}_0) - f(\mathrm{x})$ is at least $1 - \tau$ times the best possible reduction $f(\mathrm{x}_0) - f_L$. The results obtained by *glcCluster* when considering the convergence test 1.2 and the value $\tau = 10^{-6}$ are represented in Table 1.5, while the results obtained by *MCS* are in Table 1.6.

In the recent tests presented in the work describing the MATLAB toolbox for derivative-free global optimization DIRECTGO [34], DIRECT-based solvers, including *DIRECT-GL* [35] and *glcCluster* [17] have been tested on the test problems from the library DIRECTGOLib v1.0 [33], for a total of 81 problems whose dimension ranges between 2 and 15. Given that the global minima $f^*$ of all the problems is known, the algorithms are stopped when a point x with percent error (Equation 5.1) less than a value $\theta \geq 0$ is found. Otherwise, the algorithms are stopped when the number of function evaluations exceeds $2 \times 10^6$ or when the execution time exceeds $43,000.00$ seconds. *DIRECT-GL* is the solver that best performs in the

| Convex | | Nonconvex | | **Total** |
|---|---|---|---|---|
| Smooth | Nonsmooth | Smooth | Nonsmooth | |
| 100% | 71.40% | 93.10% | 55.60% | 85.90% |

**Table 1.5.** Percentage of problems solved by *glcCluster* in benchmark [30] with 1.2 as convergence test.

| Convex | | Nonconvex | | **Total** |
|---|---|---|---|---|
| Smooth | Nonsmooth | Smooth | Nonsmooth | |
| 100% | 57.10% | 94.30% | 55.60% | 81.90% |

**Table 1.6.** Percentage of problems solved by *MCS* in benchmark [30] with 1.2 as convergence test.

tests. With $\theta = 10^{-2}$, *DIRECT-GL* failed only on 3 problems, also *glcCluster* performed well by failing only 4 problems. When considering more accurate solutions by setting $\theta = 10^{-8}$, *DIRECT-GL* failed on 6 problems, while *glcCluster* failed on 29 problems.

The results from these two benchmarks led to the choice of considering ***DIRECT-GL*** as the state-of-the-art for black-box global optimization.

*DIRECT-GL* addresses the two main reasons why *DIRECT* may be slow to converge to the global minimum, where the first is that *DIRECT* may spend an excessive number of function evaluations in the region of a local minimum, performing a search that is too local. The second reason is that *DIRECT* may not do enough global search.

*DIRECT-GL* performs a global search by selecting also rectangles that are not "potentially optimal", by considering their rectangle size and the centerpoint. The local search, instead, is performed by considering the rectangles based on their rectangle size and the distance from the center point of the rectangle to the current best point.

## 1.4 Outline

In Chapter 2 the black-box optimization problem is briefly described, where a major attention is given to the optimality conditions, for both the constrained and unconstrained problems, which will be used in the successive chapters to prove convergence results of the black-box optimization algorithms.

Chapter 3 describes the *Statistical Black-Box Global Optimization* algorithm and the *MADS* algorithm from an algorithmic perspective, furthermore, the convergence results of *MADS* are analyzed.

Chapter 4 describes how the algorithms of the *Statistical Black-Box Global Optimization* method, *MADS* and *DIRECT-GL* have been implemented to perform the experiments.

In Chapter 5 the technical and computational evaluation of the methods is performed. Initially, the solvers are compared with each other on the results obtained, and then the results in terms of technical performance and usage of the resources are discussed individually for each optimizer.

Finally, Chapter 6 briefly sums up what has been obtained with *Statistical Black-Box Global Optimization* and what are the possible future developments of the method.

# Chapter 2

# Background

One way of defining **black-box** optimization problems is the following:

$$\begin{aligned} \underset{\mathrm{x}}{\text{minimize}} \quad & f(\mathrm{x}) \\ \text{subject to} \quad & \mathrm{x} \in \Omega \subseteq \mathbb{R}^n \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function with domain $\Omega$.

If $f$ is stochastic, the stochastic optimization problem is defined as:

$$\begin{aligned} \underset{\mathrm{x}}{\text{minimize}} \quad & f(\mathrm{x}) = \mathbb{E}_{\boldsymbol{\xi}} \left[ \tilde{f}(\mathrm{x}; \boldsymbol{\xi}) \right] \\ \text{subject to} \quad & \mathrm{x} \in \Omega \end{aligned} \tag{2.2}$$

Usually, black-box algorithms are also referred to as *Derivative-Free Optimization Algorithms*, which highlight the characteristic of these optimizers, where derivative information is not available, as the objective function $f$ is treated like a black-box. This increases the difficulty in finding an optimum of the function $f$ compared to gradient-based methods, in fact, if gradient information about $f$ can be retrieved/used, derivative-based methods should be used.

Various techniques exist to perform black-box optimization, the **direct-search** method is one of the most diffused. *Direct-search* methods consist of an iterative process where, starting from an initial point, a poll of candidate points is selected and evaluated. If an evaluated point has a better value than the incumbent solution, the search in the next iteration will continue from this point. This simple search framework is the basis of various *direct-search* algorithms that have been devised, which differ in how they choose the points to be added to the poll.

One of the most famous types of direct-search method is the ***Directional Direct-Search*** method. These algorithms take in input a starting point and then iteratively execute two main steps. The first is the *search step*, an optional step in which a finite number of candidate points are evaluated on the objective function, and these points can be selected using any heuristic. The second and less flexible step is the *poll step*, where, from the incumbent solution, a set of candidate points are chosen following directions contained in a selected set, whose step length varies as the algorithm runs.

In order to study the **convergence** results of the black-box methods, some **optimality conditions** must be defined.

**Definition 2.0.1** (Descent direction)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $\mathrm{x} \in \mathbb{R}^n$*. A vector* $d \in \mathbb{R}$*, with* $d \neq 0$*, is a descent direction for* $f$ *in* x *if there exists a value* $t^d_{\max} > 0$ *such that*

$$f(\mathrm{x} + td) < f(\mathrm{x}), \text{ for all } t \in (0, t^d_{\max}).$$

The first type of optimality conditions discussed is the one for **constrained problems**, which are problems described in Equation 2.1, with $\Omega \subset \mathbb{R}^n$.

**Definition 2.0.2** (Feasible direction)**.** *Let* $s \subseteq \mathbb{R}^n$ *and* $\mathrm{x} \in S$*. A vector* $d \neq 0$ *is a feasible direction in* x *if there exists* $t^f_{\max} > 0$ *such that*

$$\mathrm{x} + td \in S, \text{ for all } t \in \left[0, t^t_{\max}\right]$$

**Theorem 2.0.1** (Necessary condition of local minimizer)**.** *Let* $\mathrm{x}^* \in S$ *be a local minimizer of the problem in Equation 2.1, then there exists no feasible direction in* $\mathrm{x}^*$ *which is also descent for* $f$*.*

Theorem 2.0.1 defines the necessary condition for $\mathrm{x}^*$ to be a local minimizer of $f$.

**Theorem 2.0.2** (First order descent condition)**.** *Let* $f$ *be continuously differentiable in a neighbourhood of* $\mathrm{x} \in \mathbb{R}^n$ *and let* $d \neq 0 \in \mathbb{R}^n$*. If*

$$\nabla f(\mathrm{x})^\intercal d < 0,$$

*then* $d$ *is a descent direction for* $f$ *in* x*.*

Theorem 2.0.2 becomes a necessary condition when the objective function $f$ is convex:

**Theorem 2.0.3.** *If* $f$ *is convex then* $d$ *is a descent direction for* $f$ *in* x *if and only if*

$$\nabla f(\mathrm{x})^\intercal < 0$$

It is important to note that among descent directions, the most relevant is the *antigradient* $d = -\nabla f(\mathrm{x})$. In fact, if $\nabla f(\mathrm{x}) \neq 0$, $d = -\nabla f(\mathrm{x})$ is always a descent direction (i.e., $\nabla f(\mathrm{x})^\intercal d = -\nabla f(\mathrm{x})^\intercal \nabla f(\mathrm{x}) = -\|\nabla f(\mathrm{x})\|^2 < 0$)

Instead, if $\nabla f(\mathrm{x})^\intercal d > 0$, then $d$ is an *uphill direction* for $f$ in x. When $\nabla f(\mathrm{x}) \neq 0$, the gradient direction $d = \nabla f(\mathrm{x})$ is always *uphill* in x.

**Definition 2.0.3.** *Let* $f$ *be continuously differentiable in a neighbourhood of* $\mathrm{x} \in \mathbb{R}^n$ *and let* $d \neq 0 \in \mathbb{R}^n$*.*

1. *if* $\nabla f(\mathrm{x})^\intercal d < 0$ *then* $d$ *is a descent direction for* $f$ *in* x
2. *if* $\nabla f(\mathrm{x})^\intercal d > 0$ *then* $d$ *is a uphill direction for* $f$ *in* x
3. *if* $\nabla f(\mathrm{x})^\intercal d = 0$ *then it is not possible to state if* $d$ *is descent, uphill or constant value direction for* $f$ *in* x

When $f$ is **twice continuously differentiable** we can use second derivatives for further characterization:

**Definition 2.0.4** (Negative curvature direction)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable in a neighbourhood of $\mathrm{x} \in \mathbb{R}^n$ and let $d \neq 0 \in \mathbb{R}^n$. If $d^\intercal \nabla^2 f(\mathrm{x}) d < 0$, the direction $d \in \mathbb{R}^n$ is called a negative curvature direction.*

**Theorem 2.0.4** (Second-order descent condition)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable in a neighbourhood of $\mathrm{x} \in \mathbb{R}^n$ and let $d \neq 0 \in \mathbb{R}^n$. Assume that*

$$\nabla f(\mathrm{x})^\intercal d = 0,$$

*and $d$ satisfies $d^\intercal \nabla^2 f(\mathrm{x}) d < 0$. Then $d$ is a descent direction for $f$ in $\mathrm{x}$.*

Let $\mathrm{x}^* \in S$ be a local minimizer of problem 2.1, then there exists no feasible direction $d \in \mathbb{R}^n$ in $\mathrm{x}^*$ such that $\nabla f(\mathrm{x}^*)^\intercal d < 0$.

**Theorem 2.0.5** (First order necessary condition)**.** *Let $\mathrm{x}^* \in S$ be a local minimizer of problem 2.1, then we have*

$$\nabla f(\mathrm{x}^*)^\intercal d \geq 0 \quad \textit{for all the feasible directions } d \in \mathbb{R}^n \textit{ in } \mathrm{x}^*.$$

We can also use the second-order characterization of descent direction to get second-order optimality condition.

**Theorem 2.0.6** (Second-order necessary condition)**.** *Let $\mathrm{x}^* \in S$ be a local minimizer of problem 2.1, then for all the feasible directions in $\mathrm{x}^*$ such that $\nabla f(\mathrm{x}^*)^\intercal d = 0$ we have*

$$d^\intercal \nabla^2 f(\mathrm{x}^*) d \geq 0 \textit{ for all the feasible directions } d \in \mathbb{R}^n \textit{ in } \mathrm{x}^*.$$

In case of **unconstrained problems**, the problem 2.1 is defined with $\Omega = \mathbb{R}^n$, where $f : \mathbb{R}^n \to \mathbb{R}$ is nonlinear and continuously differentiable.

We observe that in the unconstrained case, feasibility does not play any role because it is possible to move along any vector $d \in \mathbb{R}^n$. Hence we get the following result: Let $\mathrm{x}^* \in S = \mathbb{R}^n$ be a local minimizer of problem 2.1, then there exists no descent direction for $f$ in $\mathrm{x}^*$ (i.e., there exists no direction $d \in \mathbb{R}^n$ such that $\nabla f(\mathrm{x}^*)^\intercal d < 0$).

**Theorem 2.0.7** (First order unconstrained necessary condition)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable on $\mathbb{R}^n$ and let $\mathrm{x}^* \in S = \mathbb{R}^n$. If $\mathrm{x}^* \in S$ is an unconstrained local minimizer of problem 2.1, then*

$$\nabla f(\mathrm{x}^*) = 0.$$

**Definition 2.0.5** (Stationay point)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable on $\mathbb{R}^n$. A point $\bar{\mathrm{x}}$ satisfying $\nabla f(\bar{\mathrm{x}}) = 0$ is called a **stationary point** of $f$.*

**Definition 2.0.6.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable on $\mathbb{R}^n$ and let $\mathrm{x}^* \in \mathbb{R}^n$ such that $\nabla f(\mathrm{x}^*) = 0$, then one of the following assertions is true*

1. *$\mathrm{x}^*$ is a local minimizer.*

2. $\mathrm{x}^*$ *is a local maximizer.*

3. $\mathrm{x}^*$ *is a saddle point (in* $\mathrm{x}^*$ *there are both descent and uphill directions).*

When the function is **convex** the condition becomes also sufficient, so we get:

**Theorem 2.0.8** (Necessary and sufficient condition convex function)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be continuously differentiable on* $\mathbb{R}^n$*. Assume that* $f$ *is convex. The point* $\mathrm{x}^*$ *is a global minimizer of* $f$ *on* $\mathbb{R}^n$ *if and only if* $\nabla f(\mathrm{x}^*) = 0$*. Furthermore, if* $f$ *is strictly convex on* $\mathbb{R}^n$ *and* $\nabla f(\mathrm{x}^*) = 0$*, then* $\mathrm{x}^*$ *is the unique global minimizer of* $f$*.*

We the objective function $f$ is twice-differentiable, can also use characterization of the **second-order**:

**Theorem 2.0.9** (Second-order unconstrained necessary condition)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be twice continuously differentiable on* $\mathbb{R}^n$*. Let* $\mathrm{x}^* \in S$ *be a local minimizer of problem 2.1, then for any* $d$ *such that* $\nabla f(\mathrm{x}^*)^\intercal d = 0$*, it holds*

$$d^\intercal \nabla^2 f(\mathrm{x}^*)^\intercal d \geq 0 \quad \textit{for all} \quad d \in \mathbb{R}^n$$

**Theorem 2.0.10** (Second-order unconstrained necessary condition)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be twice continuously differentiable in a neighbourhood of* $\mathrm{x}^*$*. Let* $\mathrm{x}^* \in S$ *be a local minimizer of problem 2.1, then we have:*

1. $\nabla f(\mathrm{x}^*) = 0$

2. $\nabla^2 f(\mathrm{x}^*)$ *is positive semidefinite (i.e.,* $y^\intercal \nabla^2 f(\mathrm{x}^*) y \geq 0$*, for all* $y \in \mathbb{R}^n$*)*

When the Hessian matrix $\nabla^2 f(x^*)$ is positive definite, we get a stronger result.

**Theorem 2.0.11** (Second-order sufficient condition)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be twice continuously differentiable in a neighbourhood of* $\mathrm{x}^*$*.*

*Assume that the following conditions hold:*

1. $\nabla f(\mathrm{x}^*) = 0$

2. *The Hessian matrix is positive definite in* $\mathrm{x}^*$ *(i.e.,* $y^\intercal \nabla^2 f(\mathrm{x}^*) y > 0$ *for all* $y \in \mathbb{R}^n$*, with* $y \neq 0$*)*

*Then* $\mathrm{x}^*$ *is a strict local minimizer.*

When $\nabla f(\mathrm{x}^*) = 0$ and the Hessian $\nabla^2 f(\mathrm{x}^*)$ is *indefinite* (i.e., there exists vectors $d$ such that $d^\intercal \nabla^2 f(\mathrm{x}^*) d > 0$ and others such that $d^\intercal \nabla^2 f(\mathrm{x}^*) d < 0$, then $\mathrm{x}^*$ is neither a local minimizer nor a local maximizer and $\mathrm{x}^*$ is usually called *saddle point*, which is a point where there are descent directions $d^\intercal \nabla^2 f(\mathrm{x}^*) d < 0$ and uphill directions $d^\intercal \nabla^2 f(\mathrm{x}^*) d > 0$).

We can use this information to perform the **second partial derivative test**:

**Definition 2.0.7** (Second partial derivative test)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be twice continuously differentiable in a neighbourhood of* $\mathrm{x}$*, the following test can be applied at any critical point* $\mathrm{x}$ *for which the Hessian matrix is invertible:*

1. *If the Hessian is positive definite (i.e., has all eigenvalues positive) at* x*, then* $f$ *attains a local minimum at* x*.*

2. *If the Hessian is negative definite (i.e., has all eigenvalues negative) at* x*, then* $f$ *attains a local maximum at* x*.*

3. *If the Hessian has both positive and negative eigenvalues, then* x *is a saddle point for* $f$*.*

*In those cases not listed above, the test is inconclusive.*

# Chapter 3

# Methods

The selected black-box optimizers are now described from an algorithmic perspective.

Given that no documentation is available about the internal structure of the *Nevergrad*'s algorithm selection wizards `NGOopt` [29] [10], then only the ***Statistical Black-Box Global Optimization*** algorithm and ***MADS*** algorithm [7] [2] [8] are described in this chapter, proving also the *correctness* and *convergence* results of the methods.

## 3.1 Statistical Black-Box Global Optimization Algorithm

The ***Statistical Black-Box Global Optimization*** **algorithm** (Algorithm 1) is simple and easily parallelizable.

**Algorithm 1** Statistical Black-Box Global Optimization

**Input:** A black-box optimizer $BBO$, a black-box function $f$, two real numbers $\epsilon, \delta \in (0, 1)$, a tolerance value $\sigma \geq 0$

$N \leftarrow \lceil \ln(\delta) / \ln(1 - \epsilon) \rceil$
$\mathrm{x} \leftarrow BBO.optimize(f)$
$S' \leftarrow f(\mathrm{x})$
**repeat**
    $S \leftarrow S'$
    **for** $i \leftarrow 1$ **to** $N$ **do**
        $\mathrm{x} \leftarrow BBO.optimize(f)$
        **if** $f(\mathrm{x}) < S - \sigma$ **then**
            $S' \leftarrow f(\mathrm{x})$
            **break**
**until** $S' = S$
**return** $S$

Once the value of $N$ is computed and $S$ is initialized with the solution of a run of the optimizer $BBO$ on $f$, the algorithm iterates $N$ times, searching for a better value of $S$. In each iteration, a run of $BBO$ on $f$ is executed and the returned solution x is retrieved. If $f(\mathrm{x})$ is strictly less than $S$ minus a tolerance value $\sigma$

given in input, $f(\mathrm{x})$ is the new $S$ value and the $N$ iterations are executed again.

As in the work of Tronci et al. [37], which has been inspired by the randomized Monte Carlo algorithm for Statistical Model Checking of Grosu and Smolka [15], the *Statistical Black-Box Global Optimization* method performs **Statistical Hypothesis Testing** to compute the value of $S$.

Let the *null hypothesis* $H_0(S)$ be the following stating: "*given a solution* x *obtained by a run of* $BBO.optimize(f)$*, the probability that* $f(\mathrm{x}) < S - \sigma$*, is greater than* $\epsilon$". The *null hypothesis* simply states that the probability that any solution returned by $BBO$ on $f$ is "better" than the one returned by the *Statistical Black-Box Global Optimization* algorithm, is $\geq \epsilon$. When the algorithm terminates, $H_0(S)$ is rejected with statistical confidence $1-\delta$, rejecting $H_0(S)$ means that the probability of obtaining a solution from $BBO$ that is better than $S$, is $< \epsilon$.

If $\epsilon$ and $\delta$ are set to very small values, the *Statistical Black-Box Global Optimization* algorithm will return a value $S$ such that, with high statistical confidence, the probability that a run of $BBO$ returns a value "better" than $S$ is very low.

Theorem 3.1.1 proves the ***correctness*** of the *Statistical Black-Box Global Optimization* algorithm.

**Theorem 3.1.1** (*correctness*)**.** *Given the values* $\delta, \epsilon \in (0,1)$ *and the value* $S$ *returned by the Statistical Black-Box Global Optimization algorithm (Algorithm 1), with confidence* $1-\delta$*, the following is satisfied:*

$$P(\mathrm{x} = BBO.optimize(f) \wedge f(\mathrm{x}) < S - \sigma) < \epsilon$$

*Proof.* Let $\xi$ be the event "$\mathrm{x} = BBO.optimize(f) \wedge f(\mathrm{x}) < S - \sigma$", we can associate $\xi$ to a Bernoulli random variable $Z$ with probability $p_Z$ of success and with probability $q_Z = 1 - p_Z$ of failure, i.e., $P(\xi) = p_Z$.

From the random variable $Z$, a *geometric* random variable $X$ can be defined, where, with parameter $p_Z$, its values represent the number of independent trials of $Z$ until success is obtained. The *probability mass function* of $X$ is $p(N) = P(X = N) = q_Z^{N-1} p_Z$ and the *cumulative distribution function* (CDF) of $X$ is

$$F(N) = P(X \leq N) = \sum_{n \leq N} p(n) = 1 - q_Z^N$$

By requiring that $F(N) \geq 1 - \delta$ and that $p_Z \geq \epsilon$, the following is obtained:

$$\begin{aligned} F(N) &\geq 1 - \delta \\ 1 - q_Z^N &\geq 1 - \delta \\ (1 - p_Z)^N &\leq \delta \\ N \ln(1 - p_Z) &\leq \ln(\delta) \\ N &\geq \frac{\ln(\delta)}{\ln(1-\epsilon)} \geq \frac{\ln(\delta)}{\ln(1-p_Z)} \end{aligned}$$

This provides a lower bound on the number $N$ of trials needed to achieve success with confidence ratio $\delta$ and with $\epsilon$ as the lower bound of $p_Z$. Then, if after $N$ trials a success is not obtained, with statistical confidence $1 - \delta$, $p_Z < \epsilon$.

Given that the *Statistical Black-Box Global Optimization* algorithm stops only after $N$ failing trials, $P(\xi) = p_Z < \epsilon$, with confidence $1 - \delta$.

□

## 3.2 Mesh Adaptive Direct Search (MADS) Algorithm

**Algorithm 2** Mesh Adaptive Direct Search (MADS) algorithm

**Input:** A black-box function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, a starting point $\mathrm{x}^0 \in \Omega$ and a barrier function $f_\Omega(\mathrm{x})$

1: $\Delta^0 \in (0, \infty)$ ▷ initial frame size parameter
2: $D = GZ$ ▷ postive spanning matrix
3: $\tau \in (0, 1)$ ▷ rational number of the mesh size adjustment parameter
4: $\varphi$ ▷ stopping criterion, e.g., minimum mesh size, maximum function evaluations, etc...
5: **while** $\varphi = False$ **do**
6:
**Mesh Size Parameter Update**
7: update the mesh size parameter to $\delta^k = \min\left\{\Delta^k, \left(\Delta^k\right)^2\right\}$
8:
**Search**
9: **if** $f_\Omega(t) < f_\Omega\left(\mathrm{x}^k\right)$ for some $t$ in a finite subset $S^k$ of the mesh $M^k$ **then**
10: $\mathrm{x}^{k+1} \leftarrow t$
11: $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$ ▷ increase frame size parameter
12: go to line 21
13:
**Poll**
14: select a positive spanning set $\mathbb{D}_\Delta^k$ such that the set of *poll points* $P^k = \left\{\mathrm{x}^k + \delta^k d : d \in \mathbb{D}_\Delta^k\right\}$ is a subset of the frame $F^k$ with frame size $\Delta^k$
15: **if** $f_\Omega(t) < f_\Omega\left(\mathrm{x}^k\right)$ for some $t \in P^k$ **then**
16: $\mathrm{x}^{k+1} \leftarrow t$
17: $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$ ▷ increase frame size parameter
18: **else**
19: $\mathrm{x}^{k+1} \leftarrow \mathrm{x}^k$
20: $\Delta^{k+1} \leftarrow \tau\Delta^k$ ▷ decrease frame size parameter
21: $k \leftarrow k + 1$
22: **return** $\mathrm{x}^k$

The ***Mesh Adaptive Direct Search (MADS)*** algorithm [7] (Algorithm 2) is a *Directional Direct Search* method which is able to address both *constrained* and *unconstrained* black-box optimization problems.

In *MADS* each evaluated point lies on a ***mesh*** $M^k$ of coarseness $\delta^k$.

**Definition 3.2.1** (*Mesh*)**.** *Let $G \in \mathbb{R}^{n\times n}$ be an invertible matrix and $Z \in \mathbb{Z}^{n\times p}$ be such that the columns of $Z$ form a positive spanning set for $\mathbb{R}^n$. With $D = GZ$, the* mesh *of coarseness $\delta^k > 0$ generated by $D$, centred at the incumbent solution $\mathrm{x}^k \in \mathbb{R}^n$, is defined as*

$$M^k := \left\{ \mathrm{x}^k + \delta^k D y \ : \ y \in \mathbb{N}^p \right\} \subset \mathbb{R}^n,$$

*where $\delta^k$ is called the* mesh size *parameter.*

In the Poll step of Algorithm 2, in addition to requiring the points to be in the *mesh*, the candidates in $P^k$ must lie inside a ***frame*** of size $\Delta^k$.

**Definition 3.2.2** (*Frame*)**.** *Let $G \in \mathbb{R}^{n\times n}$ be an invertible matrix and $Z \in \mathbb{Z}^{n\times p}$ be such that the columns of $Z$ form a positive spanning set for $\mathbb{R}^n$. By selecting a* mesh size *parameter $\delta^k > 0$ and a* frame size *parameter $\Delta^k$ such that $\delta^k \leq \Delta^k$, with $D = GZ$, the frame of extent $\Delta^k$ generated by $D$ and centred at the incumbent solution $\mathrm{x}^k \in \mathbb{R}^n$, is defined as*

$$F^k := \left\{ \mathrm{x} \in M^k \ : \ \left\| \mathrm{x} - \mathrm{x}^k \right\|_\infty \leq \Delta^k b \right\}$$

*with $b = \max\{\|d'\|_\infty : d' \in \mathbb{D}\}$.*

At every iteration of the *MADS* algorithm, $\delta^k$ and $\Delta^k$ always satisfy $0 < \delta^k \leq \Delta^k$, this is ensured by the operation in Line 6 of the algorithm, where the *mesh size* parameter decreases at a faster rate than the *frame size* parameter, where a thinner *mesh* increases the number of *MADS*' options of directions in the *poll step*.

Figure 3.1 shows 4 **examples** of *meshes*, *frames* and *poll sets* in $\mathbb{R}^2$. The points that lie in the *mesh* are the ones in the intersection of the thin grey lines, while the thick black box defines the *frame*. In each example, a *poll set* $P^k = \{p^1, p^2, p^3\}$ is selected. In all 4 examples, the parameters $\delta^k$ and $\Delta^k$ satisfy $0 < \delta^k \leq \Delta^k$. Given that the directions $d$ are taken from a positive spanning set $\mathbb{D}^k_\Delta$, there is high flexibility on which points can be selected in the *poll step*, with the only requirement that $\mathrm{x}^k + \delta^k d$ lies inside the *frame* $F^k$.

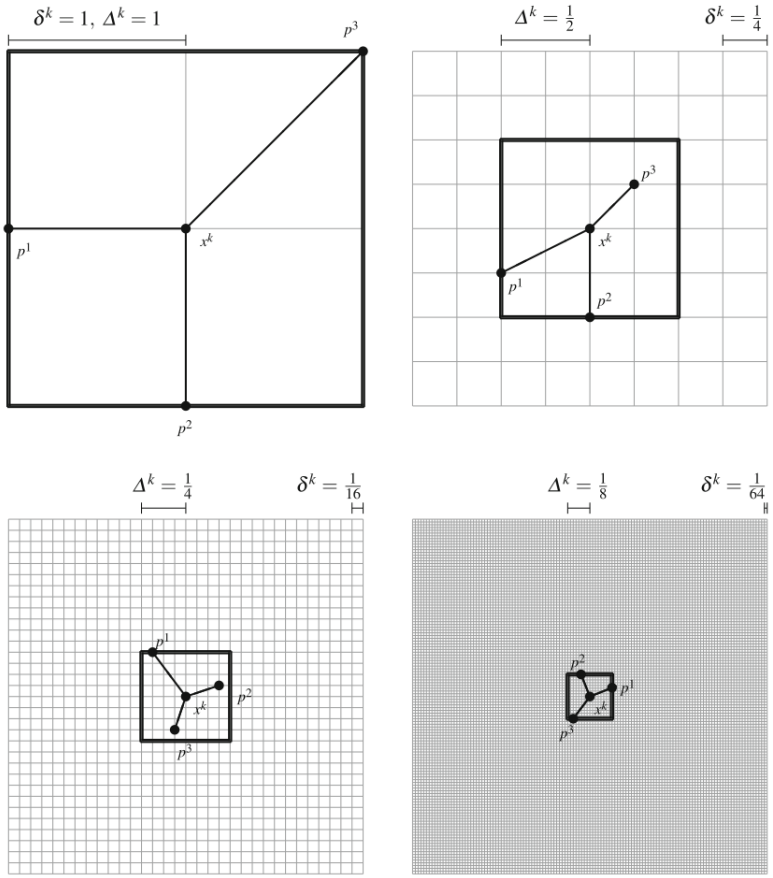



**Figure 3.1.** Examples of *meshes* and *frames* in $\mathbb{R}^2$ for different values of $\delta^k$ and $\Delta^k$. (Image taken from the book of Audet and Hare [9])

The algorithm is divided into two **main steps**. Initially, a *search* step is performed, where any strategy can be used to select a finite number (also none) of points of the *mesh* to be evaluated on the objective function $f$. If the *search* step finds a new incumbent solution, the *frame size* parameter is increased by a factor of $\tau^{-1}$ and the *search* step is executed again, otherwise, the *poll* step is performed. In the *poll* step, the set of candidate points are selected following the direction inside the positive spanning set $\mathbb{D}_{\Delta}^{k}$, with the only requirement that the candidate point $\mathrm{x}^k + \delta^k d$ lies inside the frame $F^k$.

In order to handle objective functions $f$ with a domain $\Omega \subset \mathbb{R}^n$ the algorithm uses the **extreme barrier function** $f_\Omega$, in fact, the optimization is performed on the function $f_\Omega$ rather than $f$.

**Definition 3.2.3.** *Given an objective function* $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *and a constraint set* $\Omega \subset \mathbb{R}^n$*, the* extreme barrier function $f_\Omega : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *is defined as*

$$f_\Omega(\mathrm{x}) = \begin{cases} f(\mathrm{x}) & \text{if } \mathrm{x} \in \Omega, \\ \infty & \text{if } \mathrm{x} \notin \Omega. \end{cases}$$

forcing the *MADS* algorithm to consider only points that, according to the constraints that define $\Omega$, are feasible.

The **convergence analysis** of *MADS* is divided into two steps. The first step shows that both $\liminf_{k\to\infty} \delta^k = 0$ and $\liminf_{k\to\infty} \Delta^k = 0$, the second step proves that the generalised directional derivative at *refined points* of the algorithm is non-negative for all the *refined directions*.

During the convergence analysis, it is assumed that the level sets of $f$ are bounded and that the initial point $\mathrm{x}^0$ is a feasible point (i.e., $\mathrm{x}^0 \in \Omega$).

**Lemma 3.2.1** (*MADS* iterates lie on *mesh*)**.** *Let $G \in \mathbb{R}^{n\times n}$ be an invertible matrix, $Z \in \mathbb{Z}^{n\times p}$ be such that the columns of $Z$ form a positive spanning set for $\mathbb{R}^n$, and $D = GZ$. Let $M^k$ be the mesh centred at the incumbent solution $\mathrm{x}^k$ with mesh size parameter $\delta^k$,*

$$M^k = \left\{ \mathrm{x}^k + \delta^k D y \ : \ y \in \mathbb{N}^p \right\}.$$

*Then $\mathrm{x}^{k+1} \in M^k$.*

*Proof.* If $\mathrm{x}^{k+1}$ is found in the *search* step then the result holds by design. If instead $\mathrm{x}^{k+1}$ is found in the *poll* step, then $\mathrm{x}^{k+1} = \mathrm{x}^k + \delta^k d$ for some $d \in \mathbb{D}^k_\Delta$, that is $\mathrm{x}^{k+1} = \mathrm{x}^k + \delta^k D e_i$ for some coordinate direction $e_i \in \mathbb{N}^p$. The result trivially holds when the iteration fails to find a new incumbent solution. □

**Lemma 3.2.2** (Minimal distance between *mesh* points)**.** *Let $G \in \mathbb{R}^{n\times n}$ be an invertible matrix, $Z \in \mathbb{Z}^{n\times p}$ be such that the columns of $Z$ form a positive spanning set for $\mathbb{R}^n$, and $D = GZ$. Let $M^k$ be the* mesh *centred at the incumbent solution $\mathrm{x}^k$ with mesh size parameter $\delta^k > 0$. Then,*

$$\min_{u\neq v\in M^k} \|u - v\| \geq \frac{\delta^k}{\|G^{-1}\|}.$$

*Proof.* Let $u$ and $v$ be two different points in the mesh $M^k$. By definition, there exist $y_u \in \mathbb{N}^p$ and $y_v \in \mathbb{N}^p$ with $y_u \neq y_v$, such that $u = \mathrm{x}^k + \delta^k D y_u$ and $v = \mathrm{x}^k + \delta^k D y_v$. Then,

$$\|u - v\| = \delta^k \|D\left(y_u - y_v\right)\| = \delta^k \|GZ\left(y_u - y_v\right)\|.$$

Given that, for any invertible matrix $G$ and vector $d$, the inequality $\|G^{-1}\| \cdot \|d\| \geq \|G^{-1}d\|$ holds, we can write

$$\delta^k \|GZ\left(y_u - y_v\right)\| \geq \delta^k \frac{\|G^{-1}GZ\left(y_u - y_v\right)\|}{\|G^{-1}\|} = \delta^k \frac{\|Z\left(y_u - y_v\right)\|}{\|G^{-1}\|}.$$

Since $Z\left(y_u - y_v\right)$ is a nonzero integer vector, its norm is of at least 1, then $\|u - v\|$ can be lower bounded as follows:

$$\|u - v\| \geq \delta^k \frac{\|Z\left(y_u - y_v\right)\|}{\|G^{-1}\|} \geq \frac{\delta^k}{\|G^{-1}\|}$$

□

**Lemma 3.2.3** (Upper bound on the *mesh* coarseness)**.** *Let $\left\{\delta^k\right\}$ be the sequence of* mesh size *parameters produced from an execution of the* MADS *algorithm to a function $f : \mathbb{R}^n \to \mathbb{R}$. Then, there exists a positive integer $\hat{z}$ such that $\delta^k \leq \delta^0 (\tau)^{\hat{z}}$ for every integer $k \geq 0$.*

*Proof.* All the incumbent solution found by *MADS* belongs to the level set $\mathscr{L}\left(f\left(\mathrm{x}^0\right)\right) = \left\{\mathrm{x} \in \mathbb{R}^n : f\left(\mathrm{x}\right) \leq f\left(\mathrm{x}^0\right)\right\}$. Since $\mathscr{L}$ is bounded, there exists $\gamma > 0$ such that $\mathscr{L}\left(f\left(\mathrm{x}^0\right)\right) \subseteq B_\gamma\left(\mathrm{x}^0\right)$, where $B_\gamma\left(\mathrm{x}^0\right)$ is the open ball of radius $\gamma$ centred at $\mathrm{x}^0$. If the *mesh size* parameter is large enough such that $\delta^k > 2\gamma\|G^{-1}\|$, then Lemma 3.2.2 with $v = \mathbf{x}^k$ ensures that any point $u \in M^k$ different from $\mathrm{x}^k$ would lie outside of $\mathscr{L}\left(f\left(\mathrm{x}^0\right)\right)$. This means that any iteration where the *mesh size* parameter exceeds $2\gamma\|G^{-1}\|$ is unsuccessful. If iteration $k-1$ is successful and if $\delta^{k-1} \leq 2\gamma\|G^{-1}\| < \delta^k$, then iteration $k$ must be unsuccessful and

$$\delta^{k+1} = \delta^{k-1} \leq 2\gamma\|G^{-1}\| < \delta^k = \delta^{k-1}\tau^{-1} \leq 2\gamma\|G^{-1}\|\tau^{-1}.$$

Then, every *mesh size* parameter $\delta^k$ is upper bounded by $2\gamma\|G^{-1}\|\tau^{-1}$, and the integer $\hat{z}$ can be simply selected so that $\delta^0\left(\tau\right)^{\hat{z}} \geq 2\gamma\|G^{-1}\|\tau^{-1}$. □

**Theorem 3.2.4.** *Let* $\left\{\delta^k\right\}$ *and* $\left\{\Delta^k\right\}$ *be the sequence of* mesh size *and* frame size *parameters produced by executing the* MADS *to a function* $f : \mathrm{R}^n \to \mathrm{R}$ *with bounded level sets. Then, the sequences satisfy*

$$\liminf_{k\to\infty} \Delta^k = \liminf_{k\to\infty} \delta^k = 0.$$

*Moreover,* $\left\{\delta^k\right\}_{k\in K}$ *is a subsequence (for some subset of indices* $K$*) with* $\lim_{k\in K} \delta^k = 0$ *if and only if* $\left\{\delta^k\right\}_{k\in K}$ *is a subsequence with* $\lim_{k\in K} \Delta^k = 0$.

*Proof.* Let's assume by contradiction that there exists an integer $\check{z}$ such that $0 < \delta^0 \leq \delta^k$ for all $k \geq 0$. By Lemma 3.2.3 and the *MADS* update rule of $\delta^k$, we have that for any $k \geq 0$, $\delta^k = \delta^0\left(\tau\right)^{r^k}$ where $r^k$ takes its value from the integers of the finite set $\{\check{z}, \check{z}+1, \ldots, \hat{z}\}$, for some $\hat{z} \in \mathbb{N}$. Since, by Lemma 3.2.1, $\mathrm{x}^{k+1}$, we know that $\mathrm{x}^{k+1} = \mathrm{x}^k + \delta^k D z^k$ for some $z^k \in \mathbb{N}^p$. From Lemma 3.2.3, we can substitute $\delta^k$ with $\delta^0\left(\tau\right)^{r^k}$, then for any integer $N \geq 1$:

$$\begin{aligned}\mathrm{x} = \mathrm{x}^0 + \sum_{k=0}^{N-1} \delta^k D z^k &= \mathrm{x}^0 + \delta^0 D \sum_{k=0}^{N-1} \left(\tau\right)^{r^k} z^k \\ &= \mathrm{x}^0 + \frac{(p)^{\check{z}}}{(q)^{\check{z}}}\delta^0 D \sum_{k=0}^{N-1} (p)^{r^k-\check{z}}(q)^{\hat{z}-r^k} z^k\end{aligned}$$

where $p$ and $q$ are relatively prime integers satisfying $\tau = \frac{p}{q}$. Since for any $k$ the term $(p)^{r^k-\check{z}}(q)^{\hat{z}-r^k} z^k$ appearing the last sum is an integer, it follows that all iterates lie on the translated integer lattice generated by $\mathrm{x}^0$ and the columns of the matrix $\frac{(p)^{\check{z}}}{(q)^{\check{z}}}\delta^0 D$.

Since all iterates belong to the level set $\mathscr{L}\left(f\left(\mathrm{x}^0\right)\right)$, and $f$ has bounded level sets, we know all iterates belong to a compact set. Given that the intersection of a translated integer lattice and a compact set is finite, it follows that there are only finitely many different iterates, and one of them must be visited infinitely many times. But this cannot happen as the iteration can never return to an earlier iterate once it has left that point because of the requirement that the objective function value decrease in a successful iteration. Then, the increase rule $\delta^{k+1} = \tau\delta^k$ is only

applied finitely many times, so the decrease rule $\delta^{k+1} = \tau\delta^k$ is applied infinitely many times. This is in contradiction with the hypothesis that $\delta^0(\tau)^{\check{z}}$ is a lower bound for the *mesh size* parameter.

The same procedure applies for $\Delta^k$, as $\delta^k = \min\left\{\Delta^k, \left(\Delta^k\right)^2\right\}$, from which follows also the last part of the theorem. □

This **second part** of the convergence analysis uses the definition of *refining subsequences, points* and *directions.*

**Definition 3.2.4** (Refining subsequences, points and directions)**.** *A convergent subsequence of mesh local optimizers* $\left\{\mathrm{x}^k\right\}_{k\in K}$ *(for some subset of indices K) is said to be a* refining subsequence *if and only if* $\lim_{k\in K}\delta^k = 0$*. The limit* $\hat{\mathrm{x}}$ *of* $\left\{\mathrm{x}^k\right\}_{k\in K}$ *is called a* refined point*. Given a refining subsequence* $\left\{\mathrm{x}^k\right\}_{k\in K}$ *and its corresponding refined point* $\hat{\mathrm{x}}$*, a direction d is said to be a* refining direction *if and only if there exists an infinite subset* $L \subseteq K$ *with poll directions* $d^k \in \mathbb{D}_\Delta^k$ *such that* $\mathrm{x}^k + \delta^k d^k \in \Omega$ *and* $\lim_{k\in L}\frac{d^k}{\|d^k\|} = \frac{d}{\|d\|}$*.*

In the work of Audet and Dennis [6] it is shown that there exists at least one convergent *refining subsequence.*

In order to study convergence results with ***nonsmooth*** **functions** $f$, the notion of the *generalised directional derivative* is needed, which is also known as the *Clarke directional derivative.*

**Definition 3.2.5.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be Lipschitz near* $\mathrm{x} \in \mathbb{R}^n$*. The generalised directional derivative of f at* x *in the direction* $d \in \mathbb{R}^n$ *is*

$$f^\circ(\mathrm{x}; d) = \limsup_{y\to \mathrm{x}, t\searrow 0}\frac{f(y+td) - f(y)}{t}.$$

The first-order optimality, with generalised directional derivatives, is defined by Theorem 3.2.5

**Theorem 3.2.5.** *If f is Lipschitz continuous near a local minimiser* $\mathrm{x}^*$*, then* $f^\circ(\mathrm{x}^*; d) \geq 0$ *for every* $d \in \mathrm{R}^n$*.*

The optimality condition should also consider problems where $\Omega$ is defined by constraints. Then, the notion of *cone* and *hypertangent cone* must be defined.

**Definition 3.2.6** (Cone)**.** *A set* $T \subseteq \mathrm{R}^n$ *is said to be a cone if and only if* $\lambda d \in T$ *for every scalar* $\lambda > 0$ *and for every* $d \in T$*.*

**Definition 3.2.7** (Hypertangent Cone)**.** *A vector* $d \in \mathrm{R}^n$ *is said to be a hypertangent vector to the set* $\Omega \subset \mathrm{R}^n$ *at the point* $x \in \Omega$ *if and only if there exists a scalar* $\epsilon > 0$ *such that*

$$y + tw \in \Omega$$

*for all* $y \in \Omega \cap B_\epsilon(\mathrm{x})$*,* $w \in B_\epsilon(d)$ *and* $0 < t < \epsilon$*.*

*The set of all hypertangent vectors to* $\Omega$ *at* x *is called the hypertangent cone to* $\Omega$ *at* x*, and is denoted by* $T_\Omega^H(\mathrm{x})$*.*

**Theorem 3.2.6** (Constrained first order optimality via $f^\circ$)**.** *If $f$ is Lipschitz continuous near $\mathrm{x}^*$, a local minimiser of $f$ over the constraint set $\Omega$, then $f^\circ(\mathrm{x}^*; d) \geq 0$ for every direction $d \in T_\Omega^H(\mathrm{x}^*)$.*

The Lemma defines a lower bound on the generalised directional derivative with directions in $T_\Omega^H(\hat{\mathrm{x}})$.

**Lemma 3.2.7** (Lower Bound on $f^\circ$)**.** *Suppose $f : \Omega \to \mathrm{R}^n$ is Lipschitz near $\hat{\mathrm{x}} \in \Omega \subseteq \mathrm{R}^n$, and $d \in T_\Omega^H(\hat{\mathrm{x}})$ is an hypertangent direction, then*

$$f^\circ(\hat{\mathrm{x}}; d) \geq \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{f(\mathrm{x} + tv) - f(\mathrm{x})}{t}. \tag{3.1}$$

*Proof.* By denoting the Lipschitz constant of $f$ by $M$:

$$\begin{aligned}
&\limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{f(\mathrm{x} + tv) - f(\mathrm{x})}{t} \\
&\quad = \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{f(\mathrm{x} + tv) - f(\mathrm{x} + td) + f(\mathrm{x} + td) - f(\mathrm{x})}{t} \\
&\quad \leq \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{f(\mathrm{x} + td) - f(\mathrm{x})}{t} + \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{|f(\mathrm{x} + tv) - f(\mathrm{x} + td)|}{t} \\
&\quad \leq f^\circ(\mathrm{x}; d) + \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} \frac{M\|tv - td\|}{t} \\
&\quad = f^\circ(\mathrm{x}; d) + \limsup_{\mathrm{x} \to \hat{\mathrm{x}}, v \to d, t \searrow 0} M\|v - d\| = f^\circ(\hat{\mathrm{x}}; d)
\end{aligned}$$

□

Theorem 3.2.8 defines the convergence of *MADS* for constrained problems. The theorem for convergence with unconstrained problems is similar and follows the same idea of Theorem 3.2.8. Even if the *MADS* algorithm performs optimization on the extreme barrier function $f_\Omega$, Theorem 3.2.8 requires only local smoothness on $f$, instead of $f_\Omega$.

**Theorem 3.2.8** (Convergence of *MADS*)**.** *Let $f$ be Lipschitz near a refined point $\hat{\mathrm{x}} \in \Omega$, and $d \in T_\Omega^H(\hat{\mathrm{x}})$ be a refining direction for $\hat{\mathrm{x}}$. Then the generalised directional derivative of $f$ at $\hat{\mathrm{x}}$ in the hypertangent direction $d$ is nonnegative, i.e., $f^\circ(\hat{\mathrm{x}}; d) \geq 0$.*

*Proof.* Let $\left\{x^k\right\}_{k \in K}$ be a refining subsequence converging to the refined point $\hat{\mathrm{x}}$ and let $d \in T_\Omega^H(\hat{\mathrm{x}})$ be a refining direction. Therefore, there exists an infinite subsequence $L$ of the set of indices $K$ of unsuccessful iterations, with poll directions $d^k \in \mathbb{D}_\Delta^k$ such that $\lim_{k \in L} \frac{d^k}{\|d^k\|} = \frac{d}{\|d\|}$. Furthermore, Definition 3.2.7 of an hypertangent vector guarantees that $\mathrm{x}^k + \delta^k d^k \in \Omega$ for all sufficiently large $k \in L$.

By applying Lemma 3.2.7 with sequences $\mathrm{x}^k \to \hat{\mathrm{x}}$, $d^k/\|d^k\| \to d/\|d\|$, and $\delta^k\|d^k\| \searrow 0$, given that $\delta^k\|\delta^k\| \leq \Delta^k b \searrow 0$, we have:

$$f^\circ\left(\hat{\mathrm{x}}; \frac{d}{\|d\|}\right) \geq \limsup_{\mathrm{x}\to\hat{\mathrm{x}}, v\to d/\|d\|, t\searrow 0} \frac{f(\mathrm{x}+tv)-f(\mathrm{x})}{t}$$
$$\geq \limsup_{k\in L} \frac{f\left(\mathrm{x}+\delta^k\|\delta^k\|\frac{d^k}{\|d^k\|}\right)-f\left(\mathrm{x}^k\right)}{\delta^k\|\delta^k\|} \geq 0$$

because $x^k$ is a refining subsequence, so $x^k+\delta^k d^k \in \Omega$ resulted in an unsuccessful *poll* step, i.e., $f\left(x^k+\delta^k d^k\right) = f_\Omega\left(\mathrm{x}^k+\delta^k d^k\right) \geq f_\Omega\left(x^k\right) = f\left(x^k\right)$ for all $k \in L$. □

From Theorem 3.2.8, the following hierarchy of convergence behaviour for *MADS* can be defined as:

1. With no additional assumptions, there must be at least one refining subsequence and refined point $\hat{\mathrm{x}}$.

2. If $f$ is continuous, then every refined point has the same objective function value.

3. If $f$ is Lipschitz near a refined point $\hat{\mathrm{x}}$, then the generalised directional derivatives satisfy $f^\circ(\hat{\mathrm{x}}; d) \geq 0$ for any refining direction $d \in T_\Omega^H(\hat{\mathrm{x}})$.

4. If $f$ is Lipschitz near a refined point $\hat{\mathrm{x}}$ and regular (i.e., the generalised directional derivative equals the directional derivative, for every direction $d$) at $\hat{\mathrm{x}}$, then the directional derivatives satisfy $f'(\hat{\mathrm{x}}; d) \geq 0$ for any refining direction $d \in T_\Omega^H(\hat{\mathrm{x}})$.

5. If $f$ is differentiable and $\Omega$ is convex near a refined point $\hat{\mathrm{x}}$, then $v^\intercal \nabla f(\hat{\mathrm{x}}) \geq 0$ for any direction $v \in conv(V)$ (i.e., convex hull of set $V$), where $V = \left\{d \in T_\Omega^H(\hat{\mathrm{x}}) : d \text{ is a refining direction}\right\}$.

# Chapter 4

# Implementation

## 4.1 Statistical Black-Box Global Optimization Software

The global optimization method based on *hypothesis testing* has been implemented in Python. It consists in a simple program that executing $N = \lceil \frac{\ln \delta}{\ln(1-\epsilon)} \rceil$ independent runs of a black-box optimizer of choice, with the values of $\delta$ and $\epsilon$ given in input.

In order to leverage the independence of the independent runs of the optimizer, the optimizations can be executed in parallel. This is implemented with the `multiprocessing` Python package, where multiple processes are in charge of execute in parallel the optimization runs with the chosen optimizer. The main process has total control of the multiple processes, more precisely, it is in charge of creating and killing them and it also has to signal the multiple processes when to execute an optimization run, and finally, it retrieves the information about the run's result. The communication from the main process to the parallel processes is performed with a one-producer multi-consumer FIFO queue, shared among all the processes. When there is the need for an optimization run, the main process "put" into the queue the integer 1, which symbolizes the request for a run; each parallel process will wait in the blocking call `queue.get()` until an element can be removed from the queue and then a run of the optimization solver is started. The concurrent access to the queue by the processes is handled by the locks/semaphores present in the `queue` object implementation. When an optimization run is terminated, the parallel processes will put in a second multi-producer one-consumer FIFO queue the information about the result of the run; the main process will then continuously retrieve the results with the blocking call `queue.get()` until all the requested runs are terminated.

It is important to note that the software for the method based on *hypothesis testing* is a separate module from the software of the solver, in fact, each run of the solver can be also executed by any external process, the only requirement is that the solution of the optimization can be retrieved by the parallel process and transmitted to the main process. This modularity ensures easy reusability of the software when using different solvers.

## 4.2 Nevergrad

**Nevergrad** [29] [10] is an open-source platform for black-box optimization implemented in Python, which provides the implementation of a series of state-of-the-art black-box optimizers (e.g., Covariance Matrix Adaptation [16], Particle swarm optimization [21], Bayesian Optimization [31] etc...) and sets of benchmark problems with which compare different solvers.

The `NGOpt10` algorithm of Nevergrad has been considered in the tests of the Statistical Black Box Global Optimization method. `NGOpt10` is an algorithm selection wizard, this means that, based on a priori information about the problem and the available resources, it selects a set of optimizers and chooses how to execute and combine them in order to perform optimization.

Once initialized the `NGOpt10` optimizer by giving it in input the function's domain on which perform the search and the maximum number of function evaluations allowed, the optimizer is interfaced with the `ask()` method to obtain input points that will be evaluated on the objective function, and with the `tell()` method to update the optimizer with the evaluated values.

For each Nevergrad run, the **starting point** is selected at random from the domain's range.

To stop the optimizer when no better point than the incumbent solution can be found, the execution is terminated when the standard deviation of the last returned function evaluations is small. A maximum number of function evaluations is also chosen as the second **stopping criterion**.

The source code of the implementation of the *Statistical Black-Box Global Optimization* method with *Nevergrad* is available on GitHub (`https://github.com/loreleva/nevergrad_bbo_hypothesis_testing`).

## 4.3 NOMAD

**NOMAD 4** [5] is a C++ software for constrained black-box optimization that implements the MADS algorithm [7] [2] [8].

NOMAD 4 also provides a series of interfaces implemented in MATLAB, Python, C and Julia. In these tests, the Python interface called PyNomad is used, which makes it possible to perform optimization with NOMAD using only Python function calls.

In **PyNomad** the optimization is executed by giving in input to `PyNomad.optimize()` the following arguments:

- The name of the Python function that implements the objective function.
- The starting point of the optimizer.
- The lower and upper bounds of the objective function's domain
- The *NOMAD*'s parameters

The call to `PyNomad.optimize()` will return the number of function evaluations used by MADS and the best x and $f$ found during the optimization.

For each *NOMAD* run, the **starting** point is selected at random from the domain's range.

The *NOMAD* parameters are used to define the information about the objective function and the behaviour of *NOMAD*. As in *Nevergrad*, *NOMAD* has 2 **stopping criteria**: MADS will stop if the size of the *frame* is small or if the number of function evaluations exceeds a predefined maximum.

The source code of the implementation of the *Statistical Black-Box Global Optimization* method with *NOMAD* is available on GitHub (`https://github.com/loreleva/nomad_bbo_hypothesis_testing`).

## 4.4 DIRECT-GL

The DIRECT-based algorithm ***DIRECT-GL*** [35] has been tested using its MATLAB implementation, which is present in the MATLAB toolbox DIRECTGO [34].

The software that runs the test for *DIRECT-GL* has been implemented in Python. This program retrieves the information about the objective function of interest using the Python module for the test function and creates a MATLAB subprocess which is in charge of running *DIRECT-GL*. The Python main process captures the standard output of the MATLAB subprocess in order to retrieve the results of the *DIRECT-GL* run.

The **MATLAB** subprocess executes an `m` script, which contains the definition of the variables to be given in input to the implementation of *DIRECT-GL* called `dDirect_GL()`. `dDirect_GL()` takes in input information about the objective function (MATLAB function name, dimension and lower and upper bound of the domain, mandatory to run DIRECT-based algorithms). The MATLAB implementations of the test functions have been taken from the website of Surjanovic and Bingham [36].

A maximum number of function evaluations has been chosen as **stopping criterion** for *DIRECT-GL*, and to compare the results with the ones obtained by the *Statistical Black-Box Global Optimization* method with *Nevergrad* and *NOMAD*, the second stopping criterion will halt *DIRECT-GL* when the *percent error* (Equation 5.1) is below some given threshold.

The source code of the implementation of *DIRECT-GL* is available on GitHub (`https://github.com/loreleva/DIRECT_hypothesis_testing`).

# Chapter 5

# Experimental Results

The *Statistical Black-Box Global Optimization* method based on *Hypothesis Testing* has been tested on well-known test problems for optimization. These tests simulate a real-world scenario where a black-box function has to be optimized, furthermore, a function evaluation has to be easily obtainable in terms of required resources and time. A particular interest is given to test functions that have a high number of variables, since they are the most challenging problems to solve in optimization, especially when the objective function is a black-box.

In order to evaluate the accuracy of the solution returned by an optimization algorithm, the ***percent error*** is used:

$$pe = 100\% \times \begin{cases} \frac{f(\mathrm{x})-f^*}{|f^*|} & f^* \neq 0 \\ f(\mathrm{x}) & f^* = 0 \end{cases} \tag{5.1}$$

where $f$ is the objective function, $f^*$ is the function's global optimum and x is the solution returned by the solver.

These tests aim to show that the *Statistical Black-Box Global Optimization* method can be employed to obtain a **$\theta$-optimal solution** (i.e., a solution that satisfies $pe \leq \theta$) of any black-box function $f$. The only requirement, apart from the high efficiency in obtaining a function evaluation on $f$, is that the black-box optimizer to which *hypothesis testing* is performed has a probability of at least $\epsilon$ of finding a $\theta$-optimal solution for $f$.

The *hypothesis testing* method has been tested on two state of the art black-box local optimizers, namely ***Nevergrad*** [29] and ***NOMAD*** [5]. The results of the tests are validated and compared to the ones obtained, on the same test functions, by the black-box global optimizer ***DIRECT-GL*** [35], which is an algorithm based on *DIRECT* [20] [19].

*DIRECT-GL* has been chosen as the **state of the art of black-box global optimization** because it performed slightly better than the *glcCluster* algorithm, a *DIRECT*-based optimizer present in the MATLAB environment TOMLAB [17], when optimizing test functions with up to 15 variables [34].

## 5.1 Test problems

The solvers have been tested on **25 well-known functions** in the optimization scenario [30] [39] [22] [34]. Functions for which the global minimum is known have been selected from the Virtual Library of Simulation Experiments of Surjanovic and Bingham [36], where, for each optimization problem, the following information is provided:

- **Description**: consists of the mathematical formula of the objective function, along with the dimension of the function's input and the description of the function's parameters, if any.
- **Input Domain**: a suggested range for the function's domain, to be used as box constraint during the optimization phase.
- **Global Minimum**: input for the function's global minimum (i.e., $\mathrm{x}^*$) and the minimum value of the function (i.e., $f^* = f(\mathrm{x}^*)$).
- **Code**: MATLAB and R implementations of the function.
- **References**: Citation of resources where the problem has been considered.

In order to easily evaluate the test problems during the execution of the solvers, a **Python module** for the evaluation of the functions has been implemented and made available on GitHub (`https://github.com/loreleva/bbo-functions/tree/main/sfu`). This module implements a class `objective_function()` that takes as argument the name of the objective function and it can be easily evaluated at a point `input` with the `objective_function.evaluate(input)` method. The data about the test functions have been saved in a JSON file, while their implementations have been written from scratch from the formula description on the website of Surjanovic and Bingham, with the use of the Python library NumPy.

The functions used in the tests are listed in Appendix A, where they are grouped according to their physical properties and shapes.

Some of the most **challenging test functions**, which are the ones that have **many local minima**, are now described along with their plot and mathematical formula.

The ***Ackley Function*** (Figure 5.1) is a $d$-dimensional function evaluated on the hypercube $x_i \in [-32.768, 32.768]$, for all $i = 1, \ldots, d$. With $d = 2$, the function presents a large hole at the centre, with the rest of the surface being a nearly flat region. Its global minimum is $f(\mathrm{x}^*) = 0$ at $\mathrm{x}^* = [0, \ldots, 0]$.

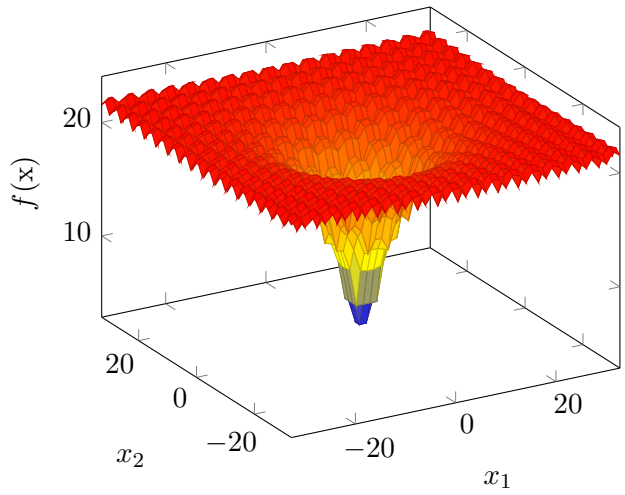



**Figure 5.1.** Ackley Function
$\boldsymbol{f(\mathbf{x})} = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e.$

The ***Bukin Function N.*** 6 (Figure 5.2) is a 2-dimensional function evaluated on the rectangle $x_1 \in [-15, -5], x_2 \in [-3, 3]$. The function has many local minima located on a ridge. Its global minimum is $f(\mathrm{x}^*) = 0$ at $\mathrm{x}^* = [-10, 1]$.

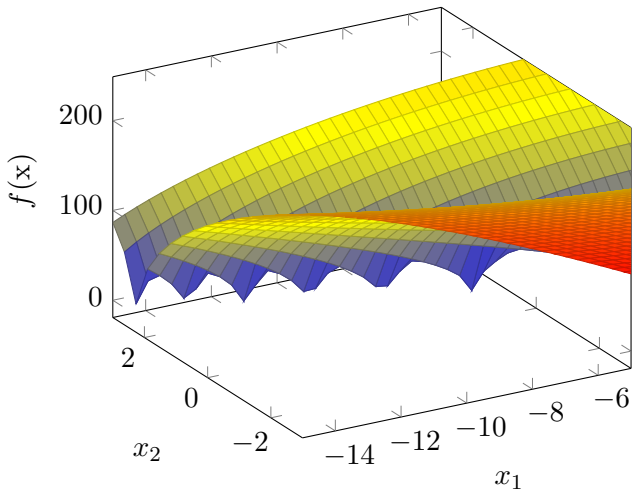



**Figure 5.2.** Bukin Function N. 6
$\boldsymbol{f(\mathbf{x})} = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01\,|x_1 + 10|.$

The ***Cross-in-Tray Function*** (Figure 5.3) is a 2-dimensional function evaluated on the square $x_i \in [-10, 10]$, for all $i = 1, 2$. The function has multiple global minima $\mathrm{x}^* = [1.3491, -1.3491], [1.3491, 1.3491], [-1.3491, 1.3491]$ and $[-1.3491, -1.3491]$, all with the optimum value $f(\mathrm{x}^*) = -2.06261$.

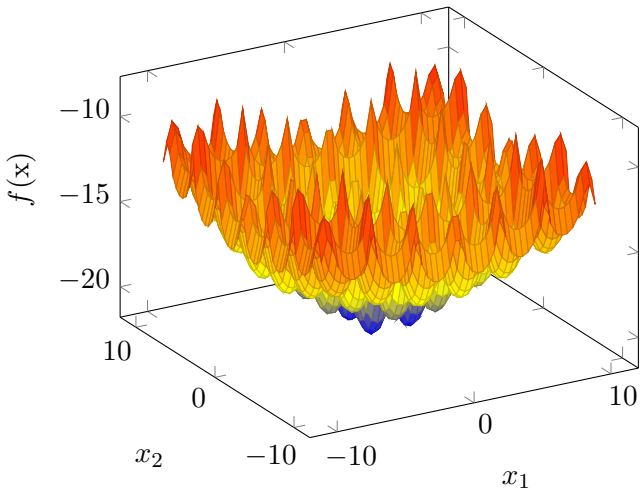



**Figure 5.3.** Cross-in-Tray Function
$$\boldsymbol{f}\,(\mathbf{x}) = -0.0001\left(\left|\sin\left(x_1\right)\sin\left(x_2\right)\exp\left(\left|100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|\right)\right| + 1\right)^{0.1}.$$

The ***Drop-Wave Function*** (Figure 5.4) is a 2-dimensional function evaluated on the square $x_i \in [-5.12, 5.12]$, for all $i = 1, 2$. The function is multimodal and highly complex, where its global minimum is $f\left(\mathrm{x}^*\right) = -1$ at $\mathrm{x}^* = [0, 0]$.

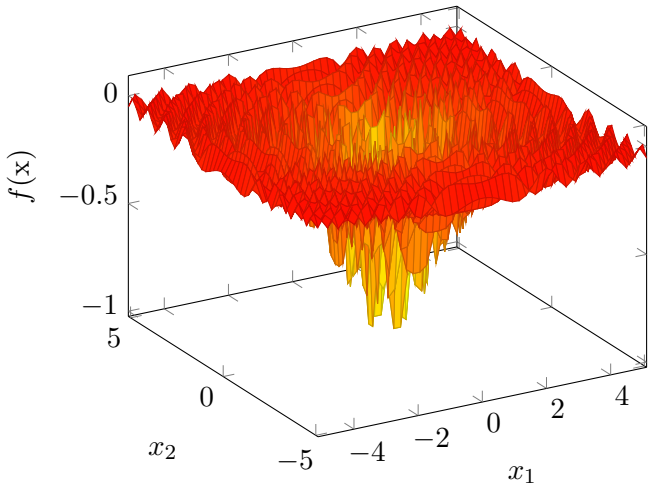



**Figure 5.4.** Drop-Wave Function
$$\boldsymbol{f}\,(\mathbf{x}) = -\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5\left(x_1^2+x_2^2\right)+2}.$$

The ***Eggholder Function*** (Figure 5.5) is a 2-dimensional function evaluated on the square $x_i \in [-512, 512]$, for all $i = 1, 2$. The function presents many local minima, where its global minimum is $f\left(\mathrm{x}^*\right) = -959.6407$ at $\mathrm{x}^* = [512, 404.2319]$.

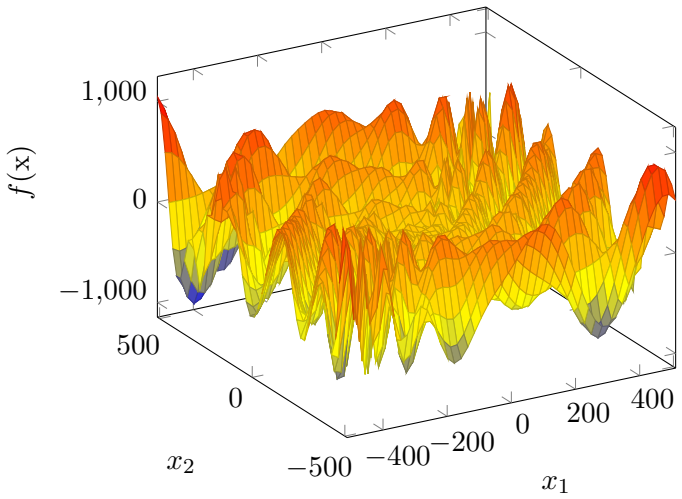



**Figure 5.5.** Eggholder Function
$\boldsymbol{f}(\mathbf{x}) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)$.

The ***Griewank Function*** (Figure 5.6) is a $d$-dimensional function evaluated on the hypercube $x_i \in [-600, 600]$, for all $i = 1, \ldots, d$. With $d = 2$, The function presents many local minima, all regularly distributed. The global minimum is $f(\mathrm{x}^*) = 0$ at $\mathrm{x}^* = [0, \ldots, 0]$.

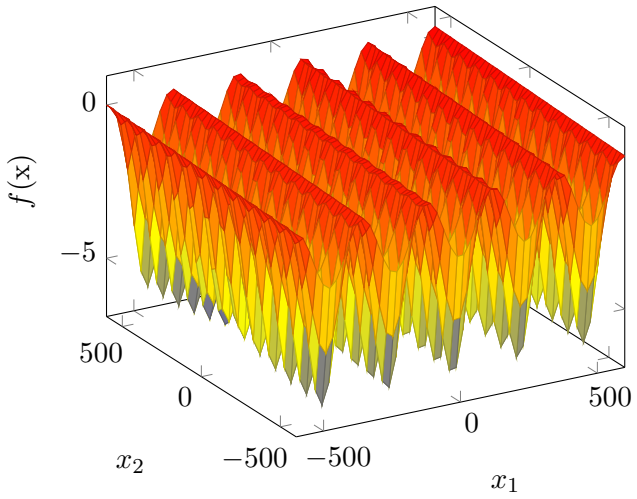



**Figure 5.6.** Griewank Function
$\boldsymbol{f}(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$.

The ***Levy Function N.*** 13 (Figure 5.7) is a 2-dimensional function evaluated on the square $x_i \in [-10, 10]$, for all $i = 1, 2$. The function presents many local minima, all following a similar pattern. The global minimum is $f(\mathrm{x}^*) = 0$ at $\mathrm{x}^* = [1, 1]$.

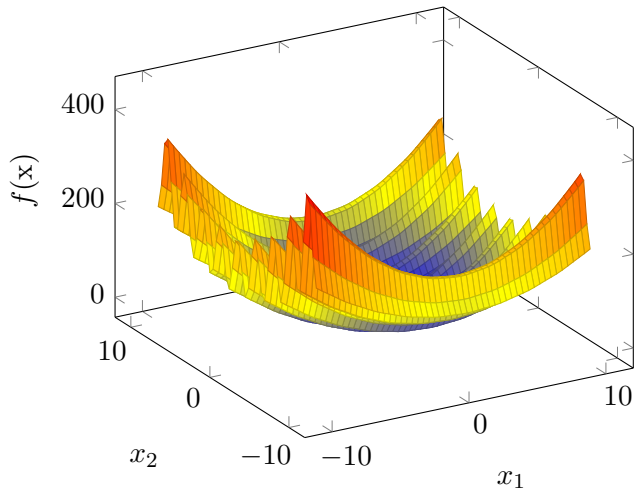



**Figure 5.7.** Levy Function N. 13
$\boldsymbol{f}(\mathbf{x}) = \sin^2(3\pi x_1) + (x_1 - 1)^2 \left[1 + \sin^2(3\pi x_2)\right] + (x_2 - 1)^2 \left[1 + \sin^2(2\pi x_2)\right]$.

The ***Rastrigin Function*** (Figure 5.8) is a $d$-dimensional function evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, \ldots, d$. The function presents many local minima, all regularly distributed. The global minimum is $f(\mathrm{x}^*) = 0$ at $\mathrm{x}^* = [0, \ldots, 0]$.

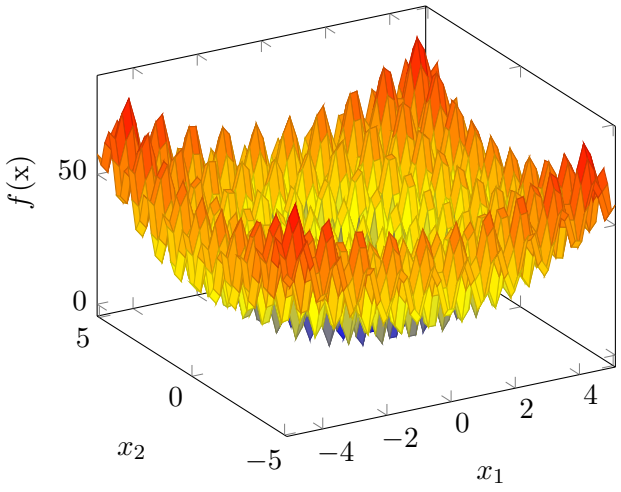



**Figure 5.8.** Rastrigin Function
$\boldsymbol{f}(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i)\right]$.

The ***Shubert Function*** (Figure 5.9) is a 2-dimensional function evaluated on the square $x_i \in [-10, 10]$, for all $i = 1, 2$. The function presents both many local minima and 18 global minima. The global minimum is $f(\mathrm{x}^*) = -186.7309$.

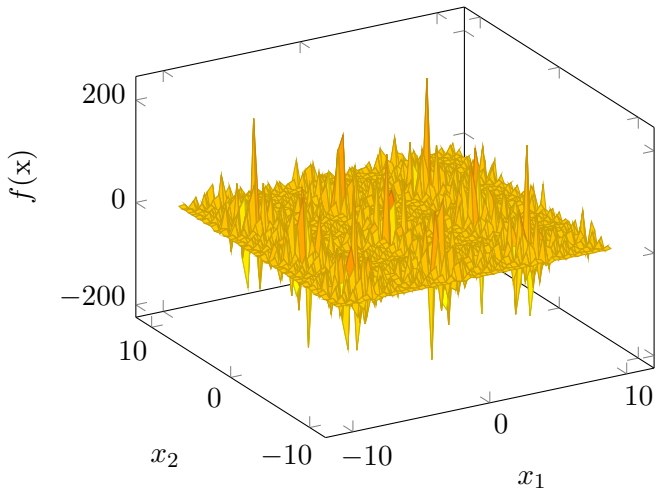



**Figure 5.9.** Shubert Function
$\boldsymbol{f}(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos\left((i+1)\,x_1+1\right)\right)\left(\sum_{i=1}^{5} i\cos\left((i+1)\,x_2+1\right)\right)$.

## 5.2 Algorithmic settings

Each method has been tested only once on each **test function** and corresponding dimension. For each function that is $d$-dimensional, 11 tests have been executed, with $d$ equal to each of the values in $\{2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. In the case of functions that accept parameters, the default values suggested in Surjanovic and Bingham's website have been used (e.g., for the *Ackley Function* the tests have been performed with the following parameters values: $a = 20, b = 0.2, c = 2\pi$). For each test function, also the domain boundaries suggested in Surjanovic and Bingham's website are used. An exception is done for *DIRECT-GL*, given that its initial point is always the centre point of the domain, if the global optimum of a test function lies in the centre, the upper bound of the domain is increased by 10, e.g., given that the $d$-dimensional *Ackley Function* has the optimum at the origin, which is the centre of the domain $[-32.768, 32.768]^d$, the new domain of the function when testing *DIRECT-GL* on the *Ackley Function* is $[-32.768, 42.768]^d$.

The values of the parameters for the ***Statistical Black-Box Global Optimization*** method have been set to $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$, obtaining the value $N = \lceil \frac{\ln(\delta)}{\ln(1-\epsilon)} \rceil = 6905$. To avoid restarting the main loop of $N$ iterations when a really small improvement is found, a solution $\mathrm{x}'$, where the incumbent solution is $\mathrm{x}^*$, is considered the new incumbent solution only if $f(\mathrm{x}') + 10^{-6} < f(\mathrm{x}^*)$.

This choice of parameters for the *Statistical Black-Box Global Optimization* method guarantees that, after the termination of the method, the null hypothesis (i.e., there exists a run of the solver that can return a better result than the incumbent solution) is **rejected with a high statistical confidence** of $(1 - \delta) = 0.999$. This also means that the probability for the null hypothesis to be true is less than $\epsilon = 0.001$, i.e., with high probability, when the method terminates, the best solution found by the solver is the best that the solver can do for such an objective function. From another point of view, if the solver has a probability of at least $\epsilon = 0.001$ of

finding a $\theta$-optimum solution for a given objective function, then, with a statistical confidence of $(1-\delta) = 0.999$, the *Statistical Black-Box Global Optimization* method will return a $\theta$-optimum solution with a probability of at least 0.999.

Given that the tests require a large number of resources and take a long time to complete, all the experiments have been executed on the University's cluster.

In order to leverage the **independence of the iterations** in the *Statistical Black-Box Global Optimization* algorithm, a maximum of 600 GB of RAM and 64 CPU cores are used to execute in parallel 64 independent optimization runs of the local solvers *Nevergrad* and *NOMAD*.

During the optimization phase of ***Nevergrad***, 64 function points are evaluated in each iteration of the algorithm. The execution of *Nevergrad* is terminated when the last 20 solutions returned by the solver have a *standard deviation* strictly smaller than $10^{-6}$, this is done to capture the moment in which the algorithm can no longer improve its solution. The other stopping criterion chosen is a maximum of $10^5$ evaluations of the objective function. For all the other parameters of *Nevergrad*, the default values have been used.

In each ***NOMAD*** run, the maximum number of function evaluations executed in a single iteration of the *MADS* algorithm is 64. *NOMAD* halts its execution when the value of the *frame size* parameter in each of its dimensions is smaller than $10^{-6}$. Another stop criterion considered is the maximum number of function evaluations set to $10^6$. For all the other parameters of *NOMAD*, the default values have been used.

***DIRECT-GL*** was run on a machine with 6 CPU cores and 300 GB of RAM. The maximum number of function evaluations allowed for ***DIRECT-GL*** are dependent from the results of *Nevergrad* and *NOMAD*. More precisely, given a test function, the maximum number of function evaluations is chosen from the highest number of function evaluations used by the *Statistical Black-Box Global Optimization* method with *Nevergrad* or *NOMAD* on such function. This ensures no disadvantages for *DIRECT-GL* during the tests with respect to the method based on *hypothesis testing*, allowing the global optimizer to use the same resources as the other two statistical global optimizers. *DIRECT-GL* will also stop its execution if the *percent error* of the best solution found is less than $10^{-4}$. For all the other parameters of *DIRECT-GL*, the default values have been used.

## 5.3 Results evaluation

The results obtained by *DIRECT-GL* and by the *Statistical Black-Box Global Optimization* method with *Nevergrad* and *NOMAD* are now described. During the description of the results, when referring to *Nevergrad* or *NOMAD*, it is implied that the method described is the *Statistical Black-Box Global Optimization* method applied to the two solvers. Whilst, with "single run" or simply "run", it is meant a single run of the *Nevergrad* or *NOMAD* optimizer, until a stopping criterion is satisfied.

The number of **problems solved** will be considered as the most important metric, with particular attention on the number of function evaluations used by a solver. A test problem is considered solved by a solver if the *percent error pe*

(Equation 5.1) satisfies $pe \leq \theta$, where $\theta \geq 0$ is a threshold value used to define when a result is "accurate enough".

Table 5.1 shows the number of test functions considered in the experiments, grouped by their dimension size.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No. of functions** | 20 | 1 | 3 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | **65** |

**Table 5.1.** Number of test functions grouped by their dimension size.

Due to time constraints, the tests of *NOMAD* on $d$-dimensional functions have been executed on 2 functions, until dimension 80. Table 5.2 shows the number of functions on which *NOMAD* has been tested.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No. of functions** | 18 | 1 | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 0 | **39** |

**Table 5.2.** Number of test functions on which *NOMAD* has been tested.

### 5.3.1 Results comparison

In Table 5.3 the number of **solved problems** by each optimizer is reported, where only very accurate solutions are considered by setting $\theta = 10^{-4}$ (e.g., if the global optimum of a function is 0, only solutions with a value $\leq 10^{-6}$ are considered correct). *NOMAD* is the most accurate solver, solving 94.87% of the problems. The second most accurate is *DIRECT-GL*, with 73.85% of the problems solved. *Nevergrad* also performs well with 70.77% of correct tests.

When considering **low-dimensional functions** (i.e., with $d \leq 6$), the 3 solvers are all wrong on only one test problem, solving all the others. Instead, when testing on high-dimensional functions ($d \geq 10$), *NOMAD* solved the highest percentual of problems (93.75%), while *DIRECT-GL* solved 60% of the tests and *Nevergrad* is the worst performing solver with 55% correct solutions.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 20/20 | 1/1 | 2/3 | 1/1 | 4/4 | 3/4 | 3/4 | 3/4 | 3/4 | 1/4 | 3/4 | 1/4 | 1/4 | 0/4 | **46/65** (**70.77%**) |
| ***NOMAD*** | 18/18 | 1/1 | 2/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 1/2 | 0/0 | 0/0 | **37/39** (**94.87%**) |
| ***DIRECT-GL*** | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65** (**73.85%**) |

**Table 5.3.** Number of problems solved by each solver, with $\theta = 10^{-4}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Table 5.4 shows the results when considering slightly less accurate solutions with $\theta = 10^{-3}$. *Nevergrad* is the best performer, failing only on one low-dimensional

problem and solving all the high-dimensional ones. The performance of *NOMAD* and *DIRECT-GL* remains very similar to the tests with $\theta = 10^{-4}$, where *DIRECT-GL* have difficulties finding accurate solutions with high-dimensional functions.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 20/20 | 1/1 | 2/3 | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | **64/65 (98.46%)** |
| ***NOMAD*** | 18/18 | 1/1 | 2/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 0/0 | 0/0 | **38/39 (97.44%)** |
| ***DIRECT-GL*** | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |

**Table 5.4.** Number of problems solved by each solver, with $\theta = 10^{-3}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Table 5.5 shows the results of the tests when allowing less accurate solutions with $\theta = 10^{-2}$. *Nevergrad* and *NOMAD* solve all the test problems, confirming the great performances of these local solvers, which with the *Statistical Black-Box Global Optimization* became $10^{-2}$-global optimizers of these test functions. The results of *DIRECT-GL* remain the same, solving 73.85% of the problems.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 20/20 | 1/1 | 3/3 | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | **65/65 (100.00%)** |
| ***NOMAD*** | 18/18 | 1/1 | 3/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 0/0 | 0/0 | **39/39 (100.00%)** |
| ***DIRECT-GL*** | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |

**Table 5.5.** Number of problems solved by each solver, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

It is now interesting to see how many **function evaluations** each solver needed to use in order to solve the problems. Figure 5.10 shows the **box plot of the function evaluations used by each solver**, for different function dimensions.

A box plot shows the dispersion and skewness of 1-dimensional data in a dataset, using the following values:

- *Minimum* (or $0th$ percentile): is the minimum value of the dataset that is not an outlier and it is depicted as a whisker below the box.
- *Maximum* (or $100th$ percentile): is the maximum value of the dataset that is not an outlier and it is depicted as a whisker above the box.
- *Median* (or $50th$ percentile): is the median value of the dataset and it is depicted as a whisker in the middle of the box.
- *First quartile* (or $25th$ percentile): is the median value of the lower half of the dataset and it is depicted as the bottom of the box.
- *Third quartile* (or $75th$ percentile): is the median value of the upper half of the dataset and it is depicted as the top of the box.
- *Outliers*: outlier values are depicted as single dots.

From Figure 5.10 it can be seen that *DIRECT-GL* is the solver that uses the **smallest number of function evaluations**, in fact, the values have an order of magnitude of 2 for low-dimensional functions, until an order of magnitude of 8 for higher dimensional functions. The outliers, or upper whiskers, in the box plot with a large value, represent the functions on which *DIRECT-GL* has not found a $10^{-4}$-optimal solution, and then uses the maximum number of function evaluations that it can use (i.e., the maximum number of function evaluations used by *Nevergrad* or *NOMAD*). *NOMAD* is the most expensive solver in terms of function evaluations, obtaining values with an order of magnitude of 8 for functions of dimension $\geq 30$. *Nevergrad* also used a number of function evaluations that has an order of magnitude of 8, but the values are smaller than the ones obtained by *NOMAD*.

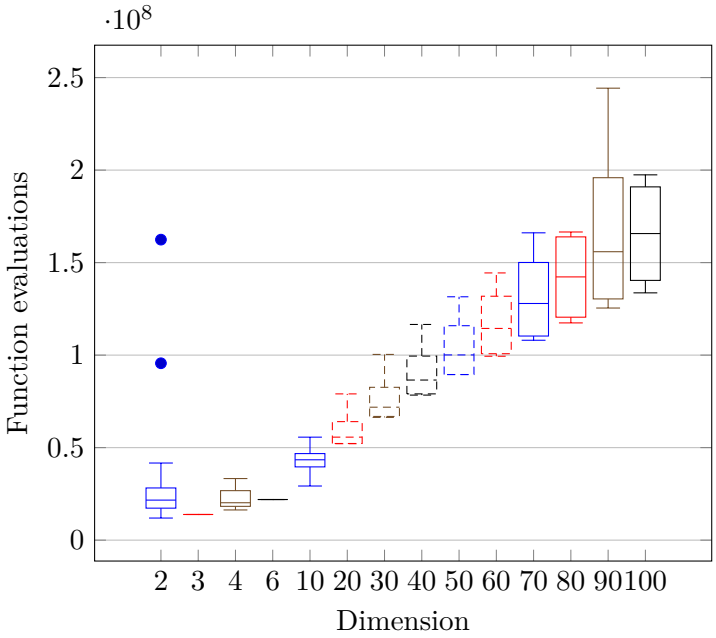
Even if *DIRECT-GL* did not solve as many problems as *Nevergrad* and *NOMAD*, it certainly uses fewer function evaluations than the *Statistical Black-Box Global Optimization* method with $\epsilon = \delta = 10^{-3}$, recalling that the value of $N$ is 6905, and so requiring a large number of function evaluations. It is important to analyze this type of performance of the *Statistical Black-Box Global Optimization* method in order to show how the method outperforms the state-of-the-art black-box optimizer *DIRECT-GL* under all the aspects.

Table 5.6 shows (with $\theta = 10^{-4}$) the number of test problems solved by each optimizer and in which the **minimum number of function evaluations**, with respect to all the other solvers is used (e.g., in Table 5.6, for the test functions of dimension 20, *Nevergrad* found for 3 times the correct solution, but only for 1 solution it has used the minimum number of function evaluations with respect to *NOMAD* and *DIRECT-GL*). The result of the solver that obtained the highest ratio for a given function dimension is underlined. As expected, *DIRECT-GL* obtained the best result, with 100% of the returned correct solutions using the minimum number of function evaluations with respect to *Nevergrad* and *NOMAD*.

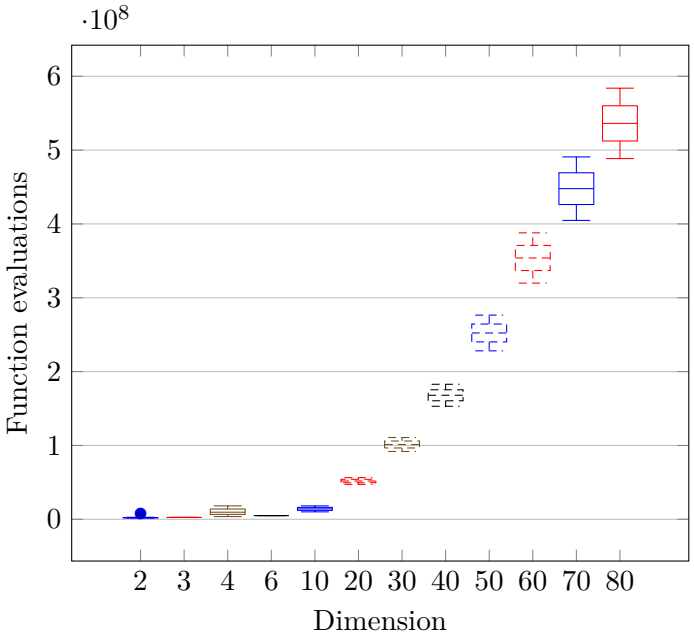
| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 0/20 | 0/1 | 0/2 | 0/1 | 1/4 | 1/3 | 1/3 | 1/3 | 1/3 | 1/1 | 1/3 | 1/1 | 0/1 | 0/0 | **8/46 (17.39%)** |
| ***NOMAD*** | 0/18 | 0/1 | 0/2 | **1/1** | 1/2 | 0/2 | 0/2 | 0/2 | 0/2 | 1/2 | 1/2 | 0/1 | 0/0 | 0/0 | **4/37 (10.81%)** |
| ***DIRECT-GL*** | **20/20** | **1/1** | **3/3** | 0/0 | **2/2** | **3/3** | **3/3** | **3/3** | **3/3** | **2/2** | **2/2** | **2/2** | **2/2** | **2/2** | **48/48 (100.00%)** |

**Table 5.6.** The ratio between the number of problems in which a solver has found a $\theta$-optimal solution using the minimum number of function evaluations and the number of $\theta$-optimal solutions returned, with $\theta = 10^{-4}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

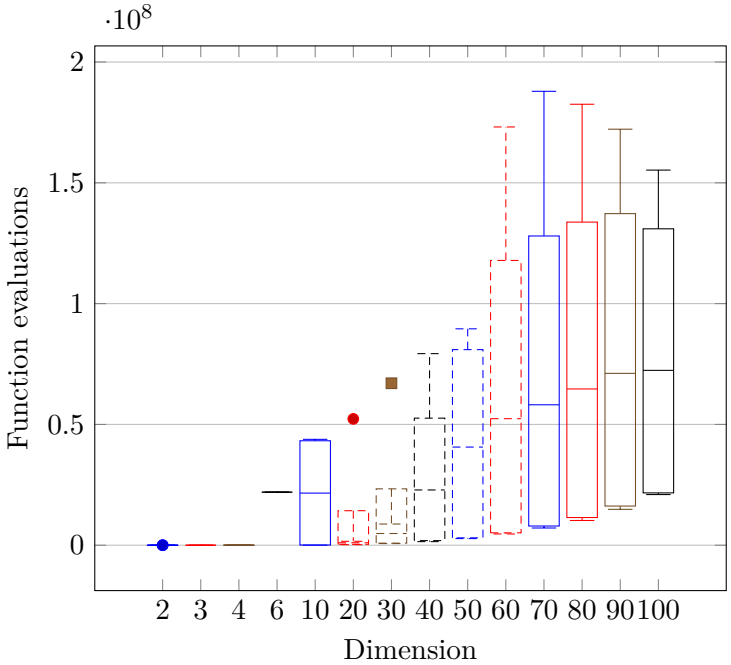
Table 5.7 and Table 5.8 shows respectively the results with $\theta = 10^{-3}$ and $\theta = 10^{-2}$. The results remain unchanged, with *DIRECT-GL* being the solver that uses the lowest number of function evaluations when returning a $\theta$-optimal solution.




(a) ***Nevergrad***




(b) ***NOMAD***




(c) ***DIRECT-GL***

**Figure 5.10.** Number of function evaluations used by each solver, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 0/20 | 0/1 | 0/2 | 0/1 | 1/4 | 1/4 | 1/4 | 1/4 | 1/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **15/64 (23.44%)** |
| ***NOMAD*** | 0/18 | 0/1 | 0/2 | **1/1** | 1/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/0 | 0/0 | **2/38 (5.26%)** |
| ***DIRECT-GL*** | **20/20** | **1/1** | **3/3** | 0/0 | **2/2** | **3/3** | **3/3** | **3/3** | **3/3** | **2/2** | **2/2** | **2/2** | **2/2** | **2/2** | **48/48 (100.00%)** |

**Table 5.7.** The ratio between the number of problems in which a solver has found a $\theta$-optimal solution using the minimum number of function evaluations and the number of $\theta$-optimal solutions returned, with $\theta = 10^{-3}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 0/20 | 0/1 | 0/3 | 0/1 | 1/4 | 1/4 | 1/4 | 1/4 | 1/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **15/65 (23.08%)** |
| ***NOMAD*** | 0/18 | 0/1 | 0/3 | **1/1** | 1/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/0 | 0/0 | **2/39 (5.13%)** |
| ***DIRECT-GL*** | **20/20** | **1/1** | **3/3** | 0/0 | **2/2** | **3/3** | **3/3** | **3/3** | **3/3** | **2/2** | **2/2** | **2/2** | **2/2** | **2/2** | **48/48 (100.00%)** |

**Table 5.8.** The ratio between the number of problems in which a solver has found a $\theta$-optimal solution using the minimum number of function evaluations and the number of $\theta$-optimal solutions returned, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

In order to use **fewer function evaluations** with the *Statistical Black-Box Global Optimization* method, the value of $\epsilon$ (i.e., the probability that in a single run the solver returns the $\theta$-optimal solution) can be slightly increased with good performing solvers, reducing the number of consecutive failing runs $N$ required by the method to stop.

This part of the results will **focus on *Nevergrad***, given that it has showed during that tests that it uses much fewer function evaluations with respect to ***NOMAD***.

*Nevergrad*, for most of the test functions, has frequently returned an $\theta$-optimal solution during the runs, making it a **reliable solver**. The value $\epsilon$ of the *Statistical Black-Box Global Optimization* method, then, can be safely increased in order to use less function evaluations with *Nevergrad*, keeping its statistical global convergence property on the test functions. How much to increase $\epsilon$ depends on the probability that *Nevergrad* has to return the $\theta$-optimal result, in a single run, for a general function of dimension $\leq 100$. If the value of $\epsilon$ is increased to this probability, the number of problems solved by the optimizer with $N = 6905$ has still to be the same with this new value of $\epsilon$, and so a smaller value of $N$.

In order to choose a value of $\epsilon > 10^{-3}$ for *Nevergrad* that is a more accurate estimation of the probability that *Nevergrad* returns a $\theta$-optimal solution for general functions of dimension $\leq 100$, we can compute this estimation from the $N = 6905$ runs of *Nevergrad* with $\epsilon = 10^{-3}$, i.e., when the probability that *Nevergrad* returns a $\theta$-optimal solution is underestimated by $\epsilon$.

Given that *Nevergrad* is a $10^{-2}$-optimizer of the 66 test functions (Table 5.5), from now on a result x will be considered correct if and only if $pe(\mathrm{x}) \leq 10^{-2}$.

By considering a test of *Nevergrad* on any function $f$, the following variables are defined:

- *no_runs*: the total number of runs executed by *Nevergrad* on $f$.

- *no_corrects*: the number of runs in which *Nevergrad* has returned a correct solution.

Then, an estimation of the probability that *Nevergrad* returns a correct result in a single run on $f$ is:

$$\frac{no\_corrects}{no\_runs} \tag{5.2}$$

This estimation will be called ***correctness ratio***. When computing the *correctness ratio* of *Nevergrad* on all the test functions, the *Eggholder Function* is the function with the lowest value, with a correctness ratio of $\approx 0.13$. We can then consider the *Eggholder Function* as the worst case for *Nevergrad* and use the obtained *correctness ratio* as the lower bound of the probability that *Nevergrad* returns a correct result in a single run, for any function $f$ with dimension $\leq 100$.

If the value of $\epsilon$ is modified to 0.1, resulting in a reduction of the value of $N$ to 66, *Nevergrad* is still expected to solve all the test problems (i.e., to obtain the same results as Table 5.5), with significantly fewer function evaluations.

Table 5.9 confirms what is stated above. *Nevergrad* still solves all the 65 problems, even with the value $\epsilon = 0.1$ and $N = 66$, obtaining the same results in Table 5.5, where $\epsilon = 10^{-3}$ and $N = 6905$.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 20/20 | 1/1 | 3/3 | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | **65/65 (100.00%)** |
| ***NOMAD*** | 17/18 | 1/1 | 3/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 0/0 | 0/0 | **38/39 (97.44%)** |
| ***DIRECT-GL*** | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |

**Table 5.9.** Number of problems solved by each solver, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 0.1$ ($N = 66$).

We can now **compare** the new *Statistical Black-Box Global Optimization* method that uses *Nevergrad* (with $\delta = 10^{-3}$ and $\epsilon = 0.1$) with *DIRECT-GL* on the number of problems solved by the optimizers that use the minimum number of function evaluations.

Table 5.10 shows, for each optimizer, the number of problems solved using the minimum number of function evaluations. Unlike Table 5.8, in which the results for $\epsilon = 10^{-3}$ are shown, with $\epsilon = 0.1$ *Nevergrad* is the most efficient solver in new 16 problems. The most important fact is that *Nevergrad* is the most efficient solver for problems with dimension >= 40, outperforming the state-of-the-art black-box optimizer *DIRECT-GL* on the function evaluations required to solve high-dimensional problems. Even if for low-dimensional problems *DIRECT-GL* is more efficient, with high-dimensional functions the solver requires more function evaluations than *Nevergrad*, and in some cases it fails to find a $10^{-2}$-optimal solution, making *Nevergrad* a **more reliable solver** in terms of global optimality and number of function evaluations required.

| Dimension | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Nevergrad*** | 0/20 | 0/1 | 0/3 | 0/1 | 1/4 | 1/4 | 1/4 | **4/4** | **4/4** | **4/4** | **4/4** | **4/4** | **4/4** | **4/4** | **31/65 (47.69%)** |
| ***NOMAD*** | 0/17 | 0/1 | 0/3 | **1/1** | 1/2 | 1/2 | 1/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/0 | 0/0 | **4/38 (10.53%)** |
| ***DIRECT-GL*** | **20/20** | **1/1** | **3/3** | 0/0 | **2/2** | **2/3** | **2/3** | 0/3 | 0/3 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | **30/48 (62.50%)** |

**Table 5.10.** The ratio between the number of problems in which a solver has found a $\theta$-optimal solution using the minimum number of function evaluations and the number of $\theta$-optimal solutions returned, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 0.1$ ($N = 66$).

### 5.3.2 Nevergrad

Table 5.11 summarizes the number of problems solved by *Nevergrad*, with different values of $\theta$. *Nevergrad* has difficulty finding $10^{-4}$-optimal solutions for higher-dimensional function, despite this, it returns $10^{-3}$-optimal solutions to almost all the problems, except one. When considering the $10^{-2}$-optimal solutions, *Nevergrad* finds them for all 65 problems.

| Dimension | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{\theta = 10^{-4}}$ | 20/20 | 1/1 | 2/3 | 1/1 | 4/4 | 3/4 | 3/4 | 3/4 | 3/4 | 1/4 | 3/4 | 1/4 | 1/4 | 0/4 | **46/65 (70.77%)** |
| $\boldsymbol{\theta = 10^{-3}}$ | 20/20 | 1/1 | 2/3 | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | **64/65 (98.46%)** |
| $\boldsymbol{\theta = 10^{-2}}$ | 20/20 | 1/1 | 3/3 | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | **65/65 (100.00%)** |

**Table 5.11.** Number of problems solved by *Nevergrad*, for different values of $\theta$ and with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Another metric that helps to find out how good is a solver to find a $\theta$-optimal solution is the ***correctness ratio*** (Equation 5.2). Table 5.12 shows the 5 functions on which *Nevergrad* has obtained the lowest *correctness ratio* values and it can be seen that most of them have a high dimension. The unique 2 dimensional function in the ranking is the *Eggholder Function*, a challenging function to optimize with many local minima.

| | Function | Dimension | Correctness Ratio |
|---|---|---|---|
| 1. | Griewank Function | 80 | 0.028236316246741965 |
| 2. | Levy Function | 100 | 0.059079061685490875 |
| 3. | Eggholder Function | 2 | 0.13002022536839064 |
| 4. | Rastrigin Function | 100 | 0.1440776136692731 |
| 5. | Griewank Function | 70 | 0.16594265855777585 |

**Table 5.12.** Lowest 5 correctness ratios obtained by *Nevergrad*, with $\theta = 10^{-4}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Table 5.13 and Table 5.14 show the 5 functions with the lowest ***correctness ratio*** obtained by *Nevergrad*, respectively with $\theta = 10^{-3}$ and $\theta = 10^{-2}$. There is no difference between the two tables, which reflects the similarity of the results in Table 5.11, for the two values of $\theta$. The newly obtained values of the correctness ratio are much higher than the ones in Table 5.12, reflecting the good performances of *Nevergrad* with functions of dimension $\leq 100$. In fact, the function with the second lower correctness ratio has a value of $\approx 0.47$, meaning that *Nevergrad*, for this function and all the others (with an exception for the function in position 1), has an estimated probability of $>\approx 47\%$ of returning a $10^{-2}$-optimal solution in a single run. Nevertheless, *Nevergrad* has difficulty with the *Eggholder Function*, in fact, the correctness ratio obtained for this function was used in Section 5.3.1 as a lower bound on the probability of correctness of *Nevergrad* with functions of dimension $\leq 100$.

| | **Function** | **Dimension** | **Correctness Ratio** |
|---|---|---|---|
| 1. | Eggholder Function | 2 | 0.13002022536839064 |
| 2. | Griewank Function | 2 | 0.4758670520231214 |
| 3. | Levy Function | 100 | 0.5466261222125688 |
| 4. | Rastrigin Function | 10 | 0.6420503909643788 |
| 5. | Levy Function | 90 | 0.6905174906042209 |

**Table 5.13.** Lowest 5 correctness ratios obtained by *Nevergrad*, with $\theta = 10^{-3}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

| | **Function** | **Dimension** | **Correctness Ratio** |
|---|---|---|---|
| 1. | Eggholder Function | 2 | 0.13002022536839064 |
| 2. | Griewank Function | 2 | 0.4758670520231214 |
| 3. | Levy Function | 100 | 0.5466261222125688 |
| 4. | Rastrigin Function | 10 | 0.6420503909643788 |
| 5. | Levy Function | 90 | 0.6905174906042209 |

**Table 5.14.** Lowest 5 correctness ratios obtained by *Nevergrad*, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Figure 5.11 is a box plot of the values of ***percent error*** obtained by *Nevergrad* for each dimension. It can be seen that the ***percent error*** values obtained by

*Nevergrad* are all below 0.0016, confirming the good performances of the solver and its $10^{-2}$-global optimality on the test functions.

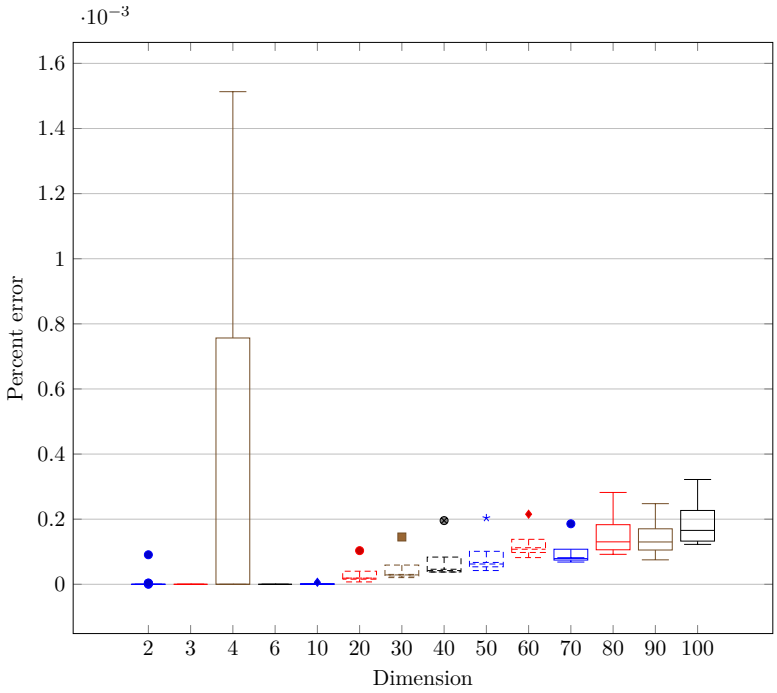



**Figure 5.11.** Box plots of the *percent error* values obtained by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Figure 5.12 shows the box plot of the **function evaluations** required by *Nevergad*, for each dimension of the test functions. In the figure, it can be seen the increasing trend of the number of function evaluations as the dimensionality of the functions increases. Most of the values have an order of magnitude of 8, showing that *Nevergrad* is an expensive method in terms of function evaluations, with the values $\delta = \epsilon = 10^{-3}$ ($N = 6905$) of the *Statistical Black-Box Global Optimization* method.

It is now interesting to check in how many runs *Nevergrad* has found the final returned solution, i.e., in terms of the *Statistical Black-Box Global Optimization* method, for how many runs *Nevergrad* has been executed in order to find the $\boldsymbol{S}$ **value** that is returned as a final solution, and then resulted in $N$ consecutive failing iterations. Figure 5.13 shows that *Nevergrad* has almost always found the returned solution in the very early iteration of the algorithm. The outliers that can be seen in the figure are mostly small improvements that are irrelevant when considering $10^{-3}$ and $10^{-2}$ optimal results (e.g., for the *Ackley Function* of dimension 90, *Nevergrad* found an $S$ value of $3.51 \times 10^{-6}$ in the first run, which is then updated at the $2562^{nd}$ run with the new $S$ value $2.477 \times 10^{-6}$.

Following the discussion in Section 5.3.1, the results of *Nevergrad* with the value $\boldsymbol{\epsilon}$**= 0.1** (i.e., $\epsilon = 0.1$ is an estimated lower bound of the probability of *Nevergrad* to find a $10^{-2}$-optimal result in a single run, the estimation has been obtained from the lowest *correctness ratio* value in Table 5.14), and so $\boldsymbol{N}$**= 66** , are now reported.

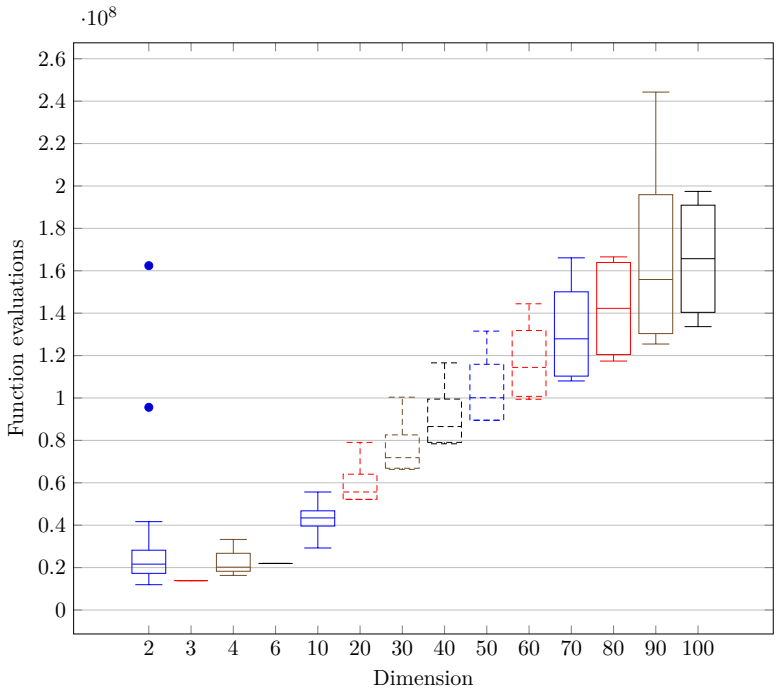



**Figure 5.12.** Box plots of the function evaluations used by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Figure 5.14 shows the ***percent error*** values obtained by *Nevergrad*. This result is similar to Figure 5.11 (*percent error* values with $\epsilon = 10^{-3}$ and $N = 6905$), in fact, the *percent error* values are all below 0.0016, confirming the $10^{-2}$-optimality of *Nevergrad* also with a lower value of $N$.

The most important advantage when using a well-performing solver with the *Statistical Black-Box Global Optimization* method is that, with a lower value of $N$, the number of function evaluations used are lower with respect to other less reliable solvers. Figure 5.15 shows the box plot of the **function evaluations** used by *Nevergrad* with $\boldsymbol{N}$**= 66**, for each dimension. The number drops drastically from the results with $\epsilon = 10^{-3}$ (Figure 5.12), with values of the order of magnitude of 6.

Also the number of runs used by *Nevergrad* to find the ***S* value** returned as a final solution is lower (Figure 5.16) than the ones obtained with $\epsilon = 10^{-3}$ (Figure 5.13), still obtaining the $10^{-2}$-optimality on all the test functions.

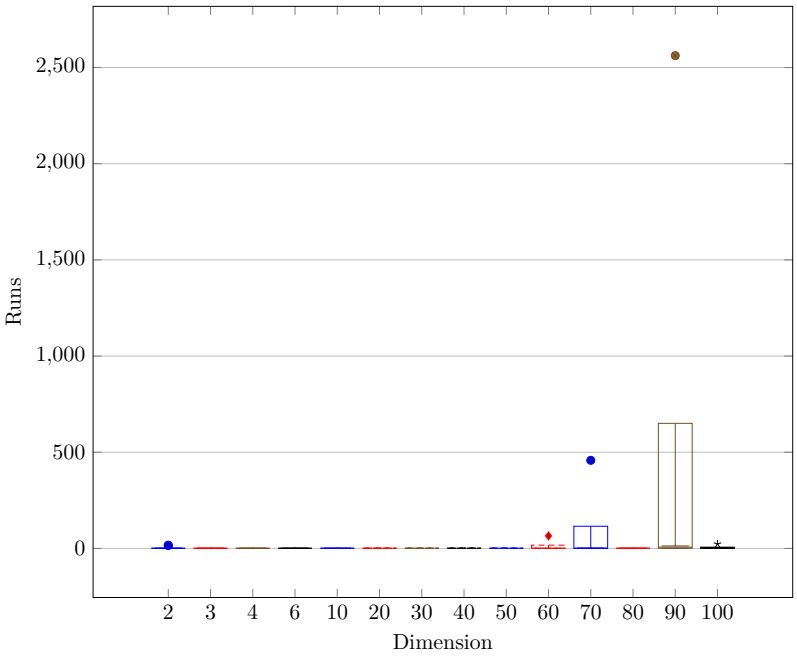



**Figure 5.13.** Box plots of the number of runs used to find the final solution returned by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

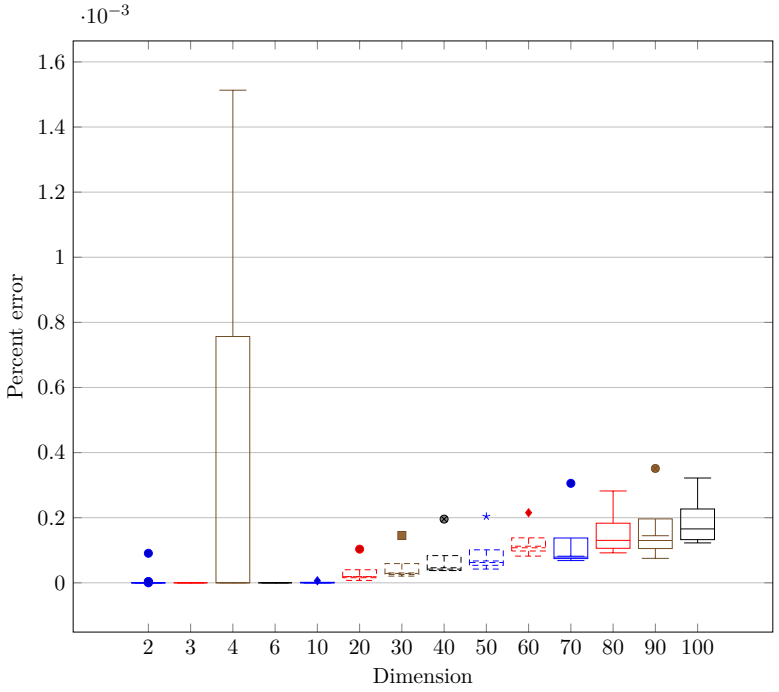



**Figure 5.14.** Box plots of the *percent error* values obtained by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 0.1$ ($N = 66$).

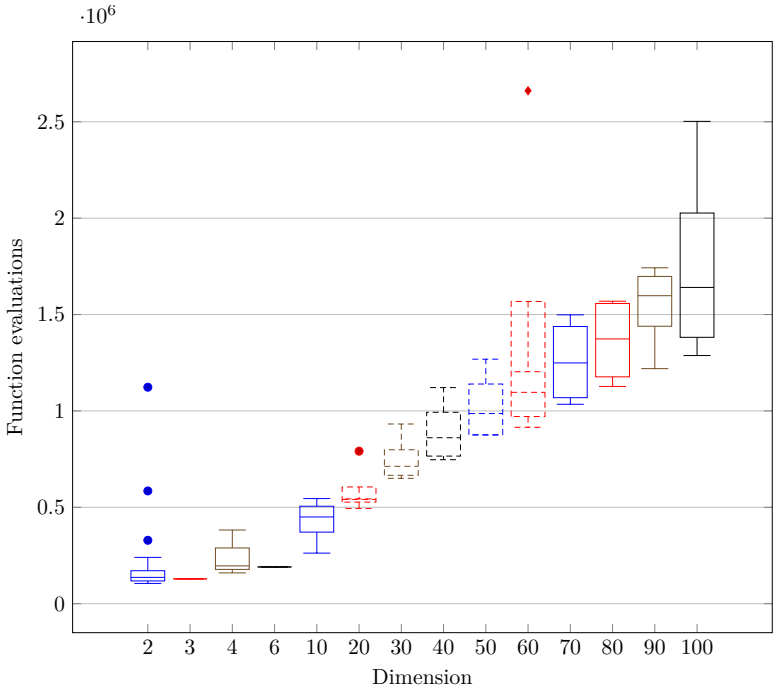



**Figure 5.15.** Box plots of the function evaluations used by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 0.1$ ($N = 66$).

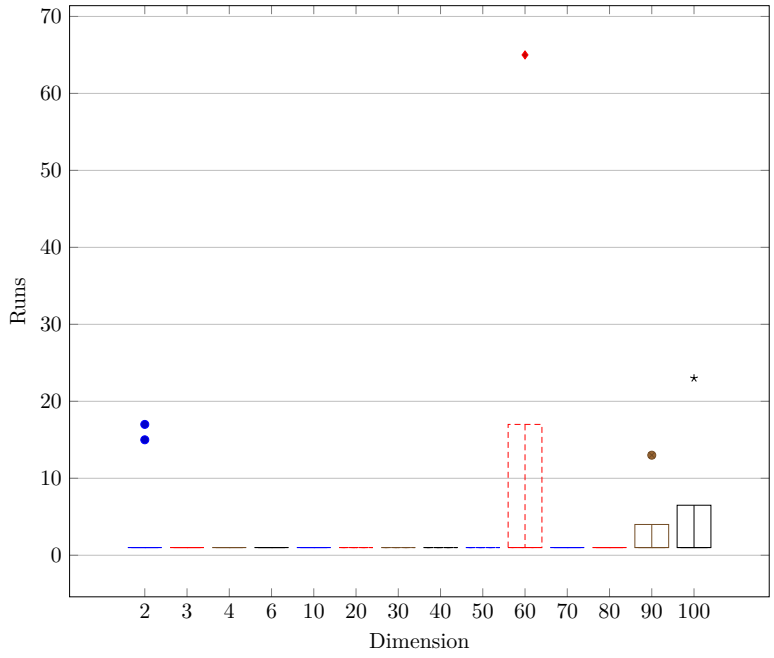



**Figure 5.16.** Box plots of the number of runs used to find the final solution returned by *Nevergrad*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 0.1$ ($N = 66$).

### 5.3.3 NOMAD

Table 5.15 shows the number of **problems solved** by *NOMAD*. The solver generally returns very accurate solutions, as it fails on only 2 problems when considering $10^{-4}$-optimal solutions. It instead reaches global optimality on all the test functions when $\theta = 10^{-2}$. *NOMAD* performs well on high-dimensional functions, finding a $10^{-3}$-optimal solution for all of them.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{\theta = 10^{-4}}$ | 18/18 | 1/1 | 2/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 1/2 | 0/0 | 0/0 | **37/39 (94.87%)** |
| $\boldsymbol{\theta = 10^{-3}}$ | 18/18 | 1/1 | 2/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 0/0 | 0/0 | **38/39 (97.44%)** |
| $\boldsymbol{\theta = 10^{-2}}$ | 18/18 | 1/1 | 3/3 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 0/0 | 0/0 | **39/39 (100.00%)** |

**Table 5.15.** Number of problems solved by *NOMAD*, with different values of $\theta$ and with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

The lowest 5 ***correctness ratio*** (Equation 5.2) values obtained by *NOMAD* are shown, for different values of $\theta$, in Table 5.16, Table 5.17 and Table 5.18. The values are the same for all the 3 different values of $\theta$, meaning that, for these functions, if *NOMAD* has returned a wrong solution, this solution has a *percent error* $> 10^{-2}$. The values are very low, which reflects the fact that, for a function of dimension $\leq 100$, the estimated lower bound on the probability that *NOMAD* returns a solution with *percent error* $\leq 10^{-2}$ is 0.041. Given this fact, the value $\epsilon = 10^{-3}$ used in the *Statistical Black-Box Global Optimization* method let *NOMAD* to still return solutions of these test functions with a *percent error* $\leq 10^{-2}$.

| | **Function** | **Dimension** | **Correctness Ratio** |
|---|---|---|---|
| 1. | Griewank Function | 2 | 0.041859122401847575 |
| 2. | Drop Wave Function | 2 | 0.08688097306689835 |
| 3. | Easom Function | 2 | 0.09366980325064157 |
| 4. | Eggholder Function | 2 | 0.09459068556551924 |
| 5. | Schaffer Function N 2 | 2 | 0.113958876339415 |

**Table 5.16.** Lowest 5 correctness ratios obtained by *NOMAD*, with $\theta = 10^{-4}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

Figure 5.17 shows the box plot of the ***percent error*** values obtained by *NOMAD* for each function dimension. The results returned by *NOMAD* are generally very accurate, with values lower than $10^{-3}$, this high accuracy is also reported in Table 5.15, where, with $\theta = 10^{-4}$, *NOMAD* solves 37 problems on a total of 39. An exception is made for functions of dimension 4, where the highest *percent error* value obtained is $\approx 1.51 \times 10^{-3}$ on the *Shekel Function*, a 4-dimensional function with 10 local minima.

| | Function | Dimension | Correctness Ratio |
|---|---|---|---|
| 1. | Griewank Function | 2 | 0.041859122401847575 |
| 2. | Drop Wave Function | 2 | 0.08688097306689835 |
| 3. | Easom Function | 2 | 0.09366980325064157 |
| 4. | Eggholder Function | 2 | 0.09459068556551924 |
| 5. | Schaffer Function N 2 | 2 | 0.113958876339415 |

**Table 5.17.** Lowest 5 correctness ratios obtained by *NOMAD*, with $\theta = 10^{-3}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

| | Function | Dimension | Correctness Ratio |
|---|---|---|---|
| 1. | Griewank Function | 2 | 0.041859122401847575 |
| 2. | Drop Wave Function | 2 | 0.08688097306689835 |
| 3. | Easom Function | 2 | 0.09366980325064157 |
| 4. | Eggholder Function | 2 | 0.09459068556551924 |
| 5. | Schaffer Function N 2 | 2 | 0.113958876339415 |

**Table 5.18.** Lowest 5 correctness ratios obtained by *NOMAD*, with $\theta = 10^{-2}$, $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

The high accuracy obtained by *NOMAD* comes at a cost. Figure 5.18 shows the box plot of the number of **function evaluations**, for each dimension, used by *NOMAD*. For test functions of dimension $\geq 30$, the number of function evaluations used has an order of magnitude of 8, where the maximum number of calls to the objective function has been reached for the *Griewank Function* of dimension 80, with $\approx 583$ million calls.

Figure 5.19 shows the box plot of the **number of runs** that *NOMAD* has used to find the final returned solution. It can be seen that the solver has always found the final solution in the early runs, with only 3 outliers for dimension 2, which are small improvements to the incumbent solution.

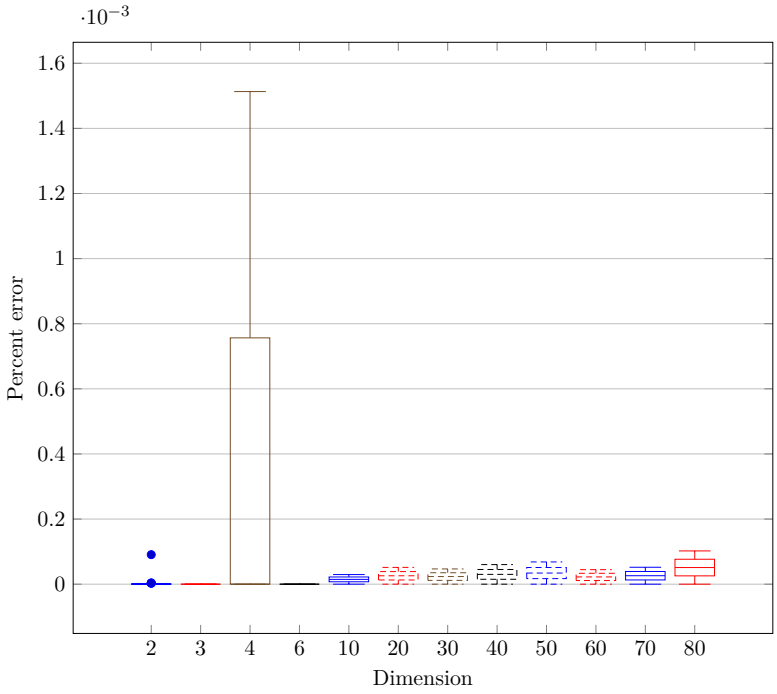



**Figure 5.17.** Box plots of the *percent error* values obtained by *NOMAD*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

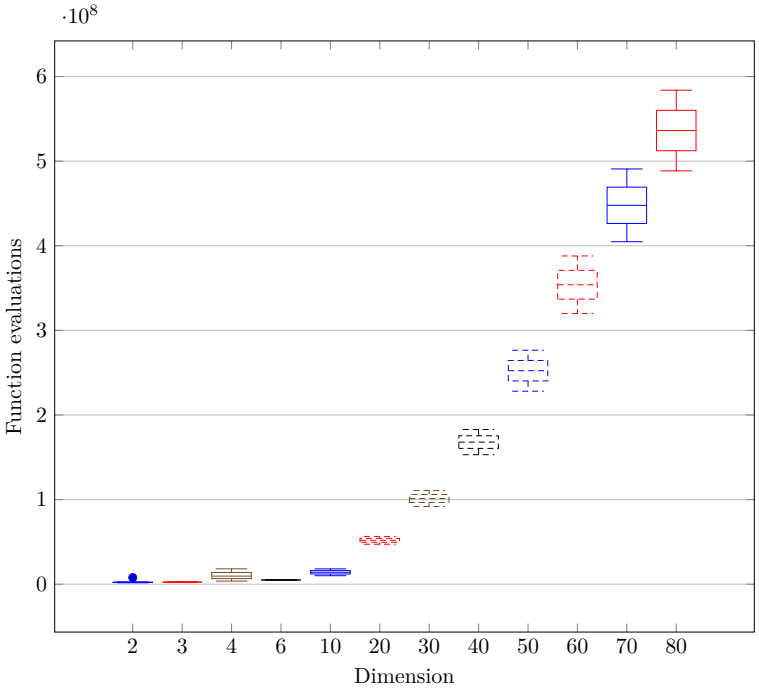



**Figure 5.18.** Box plots of the function evaluations used by *NOMAD*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

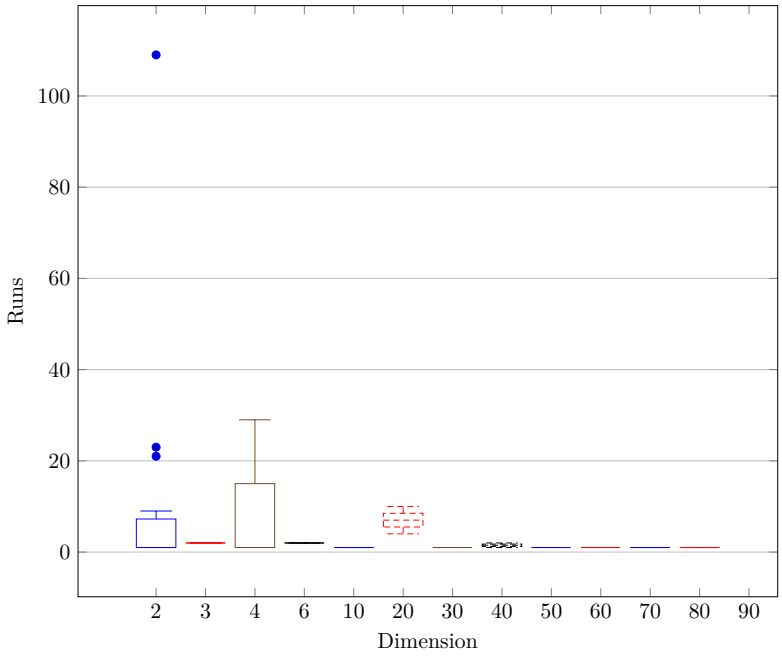



**Figure 5.19.** Box plots of the number of runs used to find the final solution returned by *NOMAD*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

### 5.3.4 DIRECT-GL

The number of **problems solved** by the black-box global optimizer *DIRECT-GL* are shown in Table 5.19. The results are the same for all the values of $\theta$, this means that the wrong solutions returned by *DIRECT-GL* have a *percent error* $> 10^{-2}$. *DIRECT-GL* performs well on low-dimensional functions, solving all the test problems with dimension $\leq 4$. Instead, for functions of higher dimensions, *DIRECT-GL* has difficulty on solving all the problems, but it still finds a $10^{-4}$-optimal solution to at least half of them.

Figure 5.20 shows the box plot of the ***percent error*** values obtained by *DIRECT-GL*. It can be seen that the solver performs well on low-dimensional functions, returning solutions with low *percent error*. By increasing the dimensionality of the problem, the solutions returned on some of the test functions have a large *percent error*, suggesting that the solver still needs to use more function evaluations in order to find the global optimum, but this number becomes prohibitive as the maximum number of function evaluations used in the *Statistical Black-Box Global Optimization* method has been used as an upper bound, which has already an order of magnitude of 8.

The metric on which *DIRECT-GL* outstand the *Statistical Black-Box Global Optimization* with *Nevergrad* and *NOMAD* is the number of ***function evaluations*** used during the optimization phase. Figure 5.21 shows the number of function evaluations used by *DIRECT-GL*. The solver uses a number of function evaluations that have an order of magnitude of 2 for low-dimensional functions, until an order of magnitude of 7 for higher-dimensional functions. The outliers, or upper whiskers, in the box plot with a large value represent the functions on which *DIRECT-GL* has not found a $10^{-4}$-optimal solution, and then uses the maximum number of function evaluations that it can use.

| **Dimension** | 2 | 3 | 4 | 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{\theta = 10^{-4}}$ | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |
| $\boldsymbol{\theta = 10^{-3}}$ | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |
| $\boldsymbol{\theta = 10^{-2}}$ | 20/20 | 1/1 | 3/3 | 0/1 | 2/4 | 3/4 | 3/4 | 3/4 | 3/4 | 2/4 | 2/4 | 2/4 | 2/4 | 2/4 | **48/65 (73.85%)** |

**Table 5.19.** Number of problems solved by *DIRECT-GL*, with different values of $\theta$ and with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

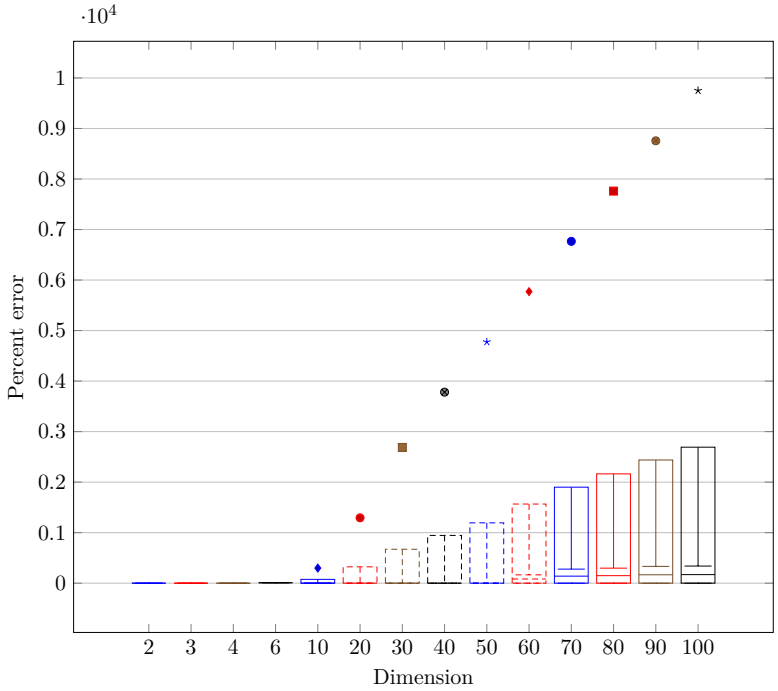



**Figure 5.20.** Box plots of the *percent error* values obtained by *DIRECT-GL*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

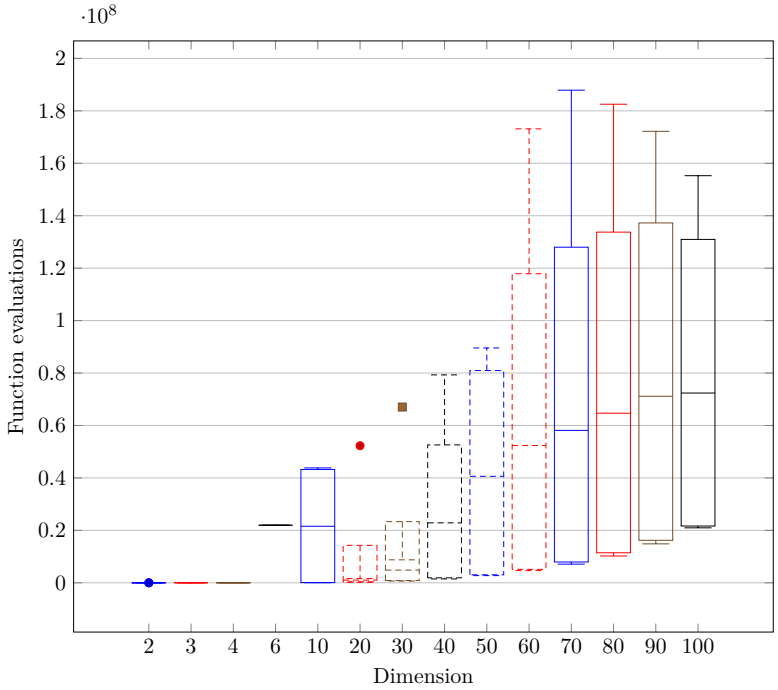



**Figure 5.21.** Box plots of the function evaluations used by *DIRECT-GL*, for each dimension, with $\delta = 10^{-3}$ and $\epsilon = 10^{-3}$ ($N = 6905$).

## 5.4 Computational evaluation

During the execution of the tests, the following ***metrics*** have been tracked for each optimizer:

- ***Maximum RAM usage***: Maximum *RAM* usage in Gibibyte (Gib) used during the optimization phase, for each function. More specifically, the maximum *Resident Set Size (RSS)* used by each process has been retrieved. The *RSS* is a portion of memory used by a process that is physically in RAM. Then, it does not include the memory from shared libraries that are not in RAM, memory that is allocated but not used and memory that is swapped out.

- ***Single-core execution time***: It is the sum of the CPU time of all the parallel processes used during the optimization of a test function. The CPU time is in seconds units and represents the amount of time for which a process has used the CPU. Then, the time (for example) elapsed when the process is in an idle state or when performing I/O operations is not included in the metric.

- ***Multicore execution time***: In order to track the execution time of the solvers, including the ones that use multiprocessing, the wall-clock time of the execution of the process is retrieved for each test, in seconds units.

- ***Speedup***: Represents the rate at which the single core computation is speeded up by using more than 1 CPU core. It is computed as $\frac{\text{Single core execution time}}{\text{multicore execution time}}$.

- ***Efficiency***: It is a measure of how effectively all the CPU cores have been used. It is computed as $\frac{\text{speedup}}{\text{\# of CPU cores}}$.

### 5.4.1 Nevergrad

The *Statistical Black-Box Global Optimization* method with *Nevergrad* has been executed on 64 CPU cores, where each independent run of *Nevergrad* was run in a single core.

Figure 5.22 shows the box plot of the **RAM usage** of *Nevergrad*, for each test function dimension. The maximum usage was reached for the *Ackley Function* of dimension 2, with a peak of RAM usage of $\approx 9.6594$ GiB during its optimization. Instead, the minimum RAM usage has been obtained for the *Hartmann 3-D Function* of dimension 3, with a maximum RAM usage during its optimization of $\approx 7.8342$ GiB.

Figure 5.23 shows the box plot of the sum of the CPU time (in seconds units) of all the 64 parallel processes, aimed to approximate the **execution time** of the tests on a **single CPU core**. The maximum sequential execution time is reached for the *Griewank Function* of dimension 100, with 154 day, 8 hours, 22 minutes and 55.8056 seconds. Instead, the minimum sequential execution time has been obtained with the *Levy Function* of dimension 2, with 16 hours, 13 minutes and 15.9837 seconds.

Figure 5.24 shows the box plot of the **multicore execution time** of the tests with *Nevergrad*, in seconds units. The maximum duration for an optimization test has been reached on the *Griewank Function* of dimension 100, where the test took

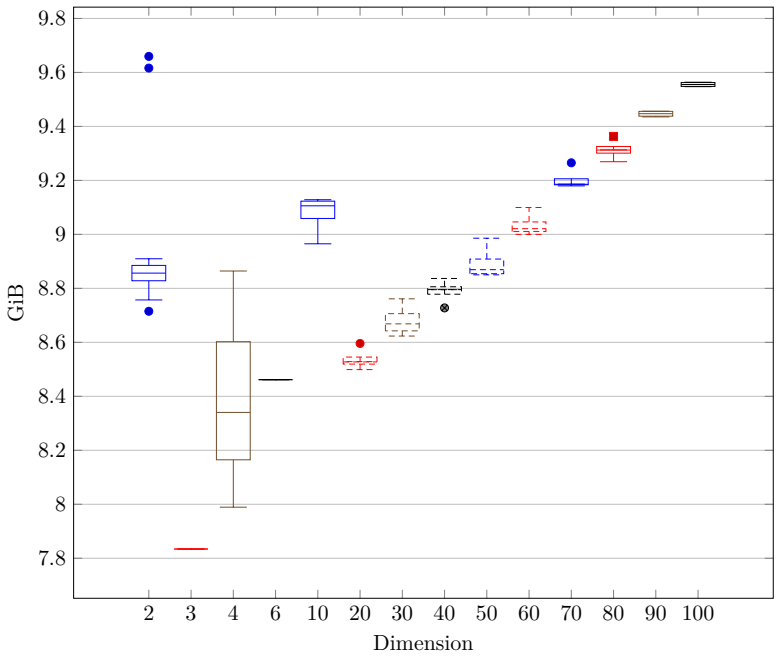



**Figure 5.22.** Box plot of the RAM usage in Gibibyte of *Nevergrad*, for each dimension.

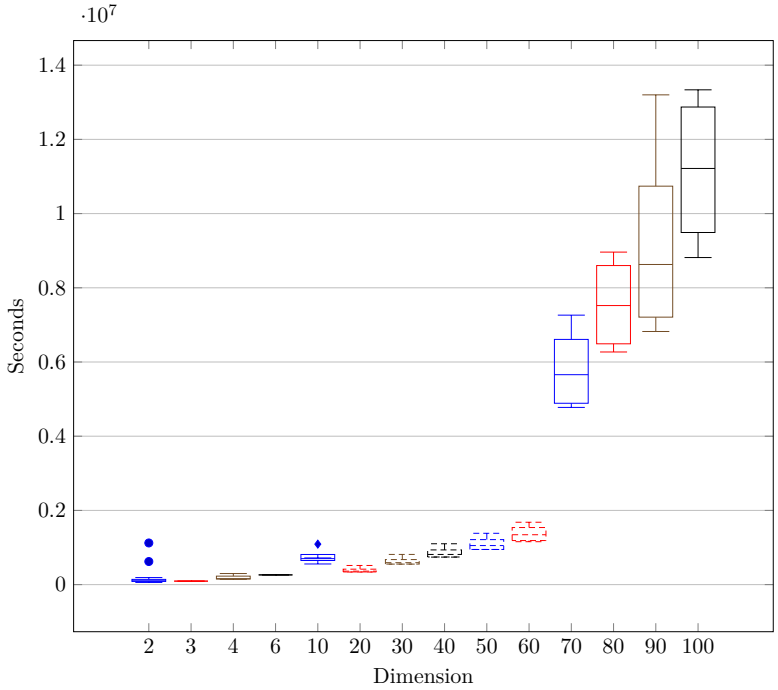



**Figure 5.23.** Box plot of the sequential exeuction time of *Nevergrad* in seconds units, for each dimension.

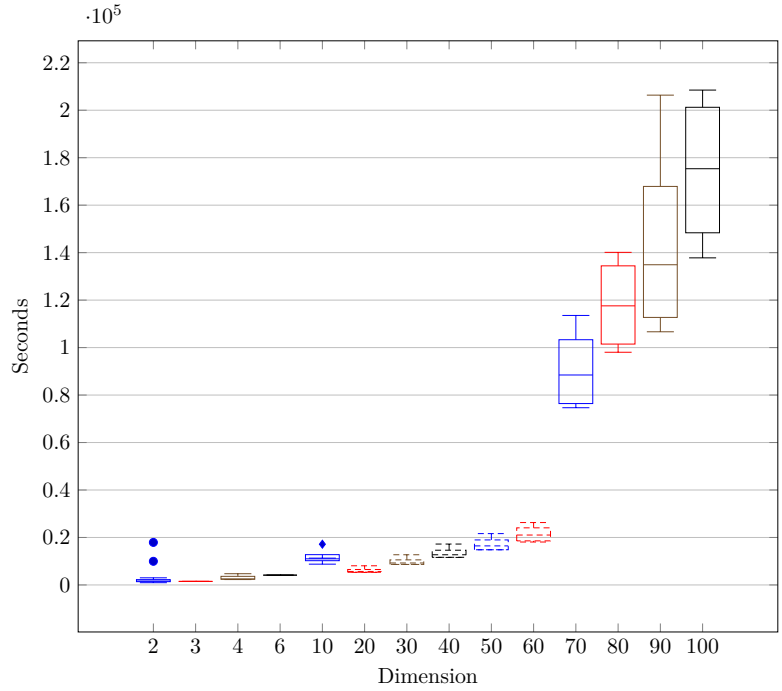



**Figure 5.24.** Box plot of the multicore time, in seconds units, otbained by *Nevergrad*, for each dimension.

2 days, 9 hours, 54 minutes and 44.98 seconds. Instead, on the *Levy Function* of dimension 2, *Nevergrad* performed the fastest optimization, taking just 15 minutes and 44.27 seconds.

Figure 5.25 and 5.26 show respectively the box plot of the ***speedup*** and ***efficiency*** values. From the plots, it can be seen the importance of parallel computation with the *Statistical Black-Box Global Optimization* method, where, with 64 CPU cores, a speedup of between 61% and 63% is obtained for low-dimensional test functions, while with higher dimensions the speedup is of the $\approx 63.8\%$. The *Statistical Black-Box Global Optimization* method also uses all the available CPU cores with a high efficiency of $> 96\%$.

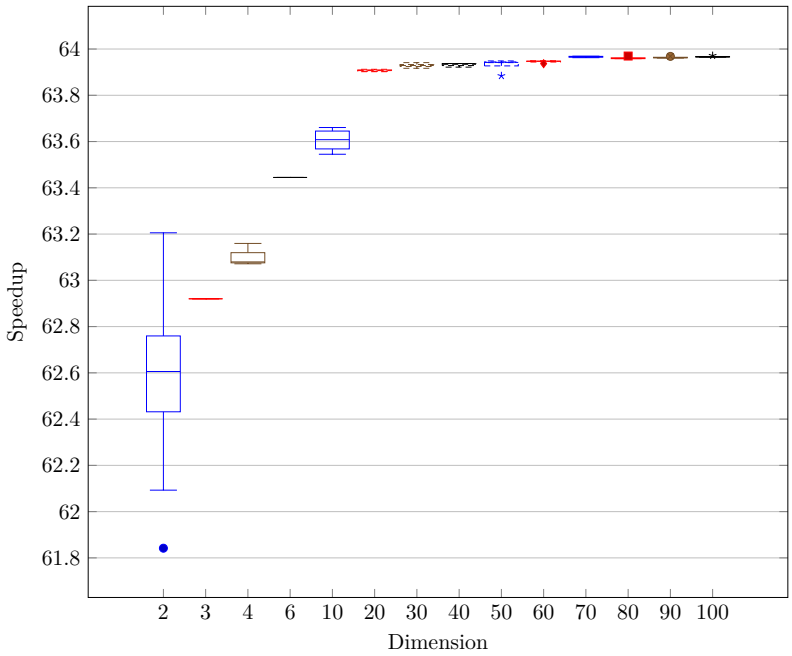



**Figure 5.25.** Box plot of the *speedup* values of the multicore execution of *Nevergrad*, for each dimension.

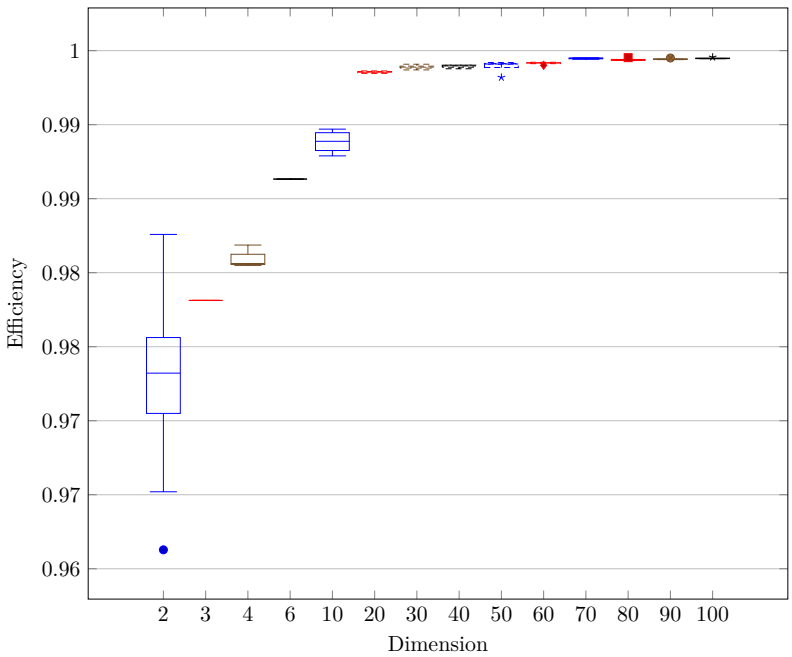



**Figure 5.26.** Box plot of the *efficiency* values of the multicore execution of *Nevergrad*, for each dimension.

### 5.4.2 NOMAD

Also for *NOMAD*, the *Statistical Black-Box Global Optimization* method has been executed on 64 CPU cores, where each independent run of *NOMAD* was run in a single core.

*NOMAD* is the method with the highest cost in terms of memory, where the maximum **RAM usage** value was $\approx$ 562.227 GiB with the *Griewank Function* of dimension 70. Instead, the minimum RAM usage was obtained with the *Matyas Function* of dimension 2, with $\approx$ 2.0944 GiB of RAM used.

*NOMAD* is also the solver with the highest execution time among the solvers. The maximum estimation of the **execution time** on a **single CPU core** has been obtained with the *Griewank Function* of dimension 80, where the solver needed 181 days, 16 hours, 35 minutes and 22 seconds. Instead, with the *Easom Function* of dimension 2, *NOMAD* performed optimization with the shortest single core execution time, requiring 15 hours, 39 minutes and 13.41 seconds.

Figure 5.29 shows the **execution time** of *NOMAD* when using **multiprocessing**. The longest optimization was done on the *Griewank Function* of dimension 80, taking 29 days, 8 hours, 8 minutes and 50.44 seconds, which is also the longest run among all the tests executed with all the solvers. Instead, the shortest optimization was performed on the *Easom Function* of dimension 2, which took 14 minutes and 59.42 seconds.

Figure 5.30 and Figure 5.31 show respectively the boxplot of the ***speedup*** and ***efficiency*** values obtain by the multiprocessing execution of *NOMAD*, for all the different dimensions of the test functions. Also in this case the *speedup* is between 62% and 63%, with an *efficiency* of almost 100%, confirming the high advantage that multiprocessing execution brings to the *Statistical Black-Box Global Optimization* method.

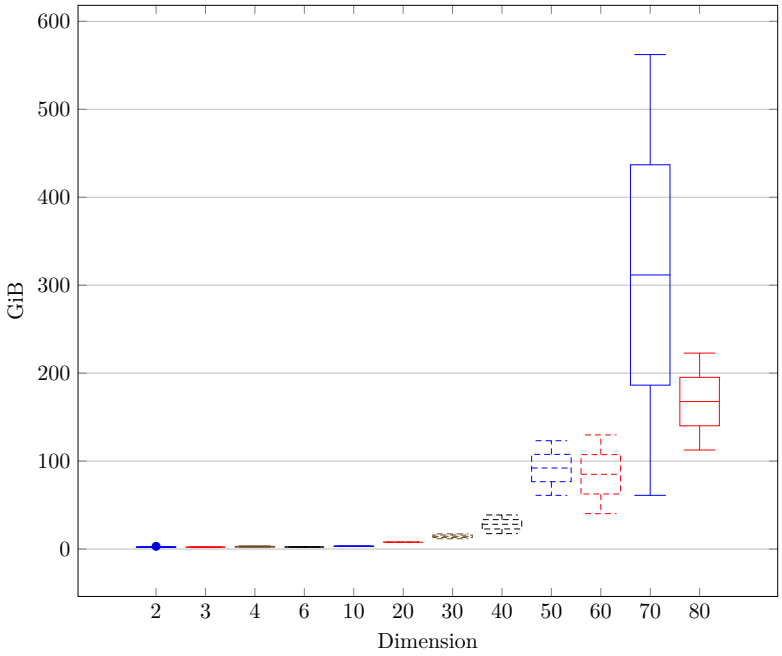



**Figure 5.27.** Box plot of the RAM usage in Gibibyte of *NOMAD*, for each dimension.

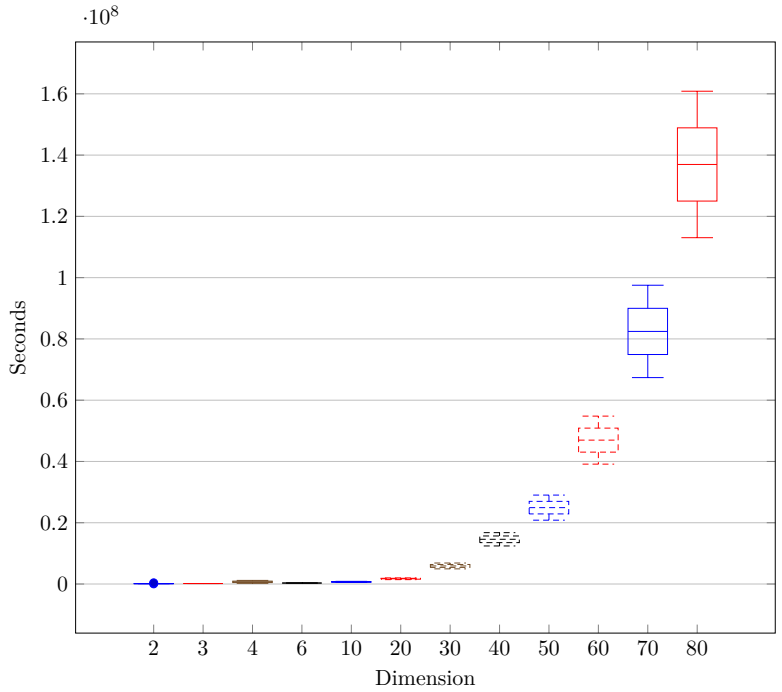



**Figure 5.28.** Box plot of the sequential exeuction time of *NOMAD* in seconds unit, for each dimension.

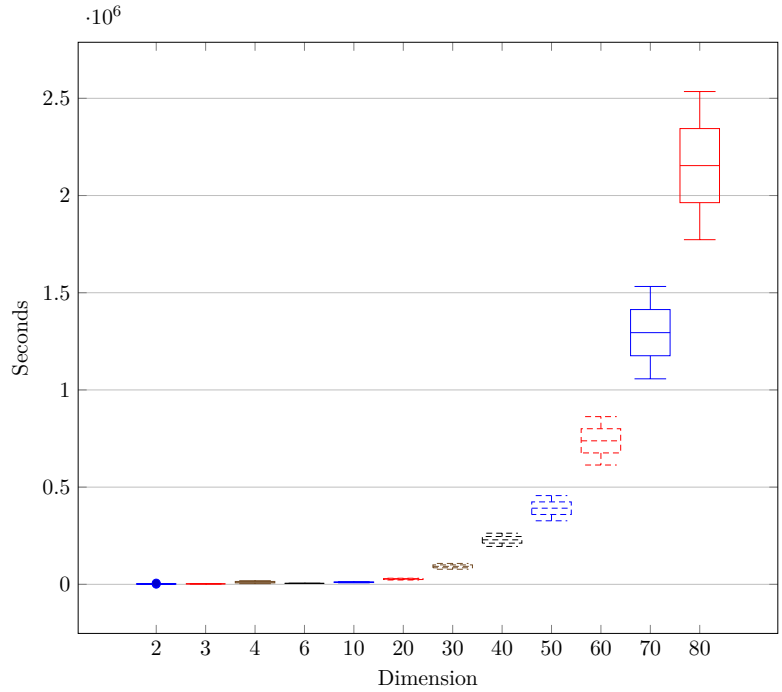



**Figure 5.29.** Box plot of the multicore time, in seconds unit, otbained by *NOMAD*, for each dimension.

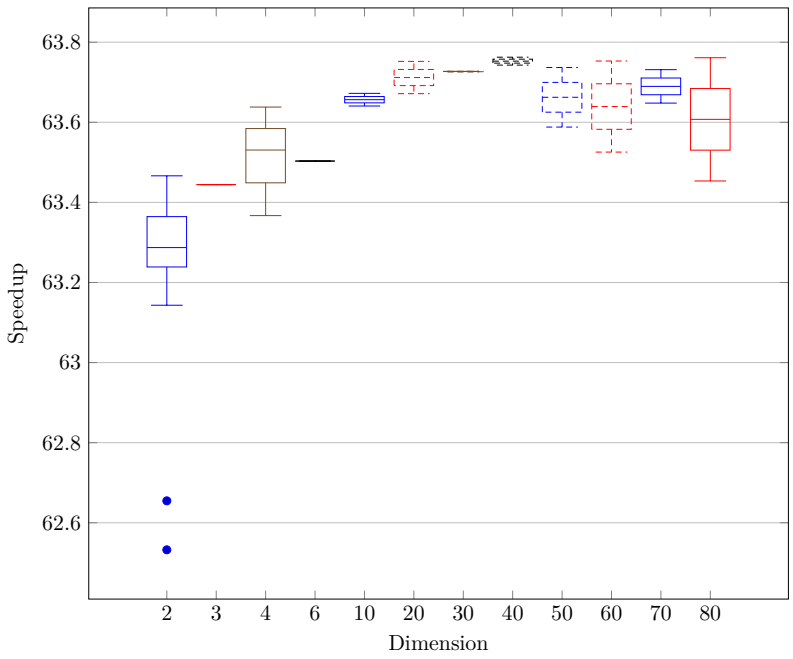



**Figure 5.30.** Box plot of the *speedup* values of the multicore execution of *NOMAD*, for each dimension.

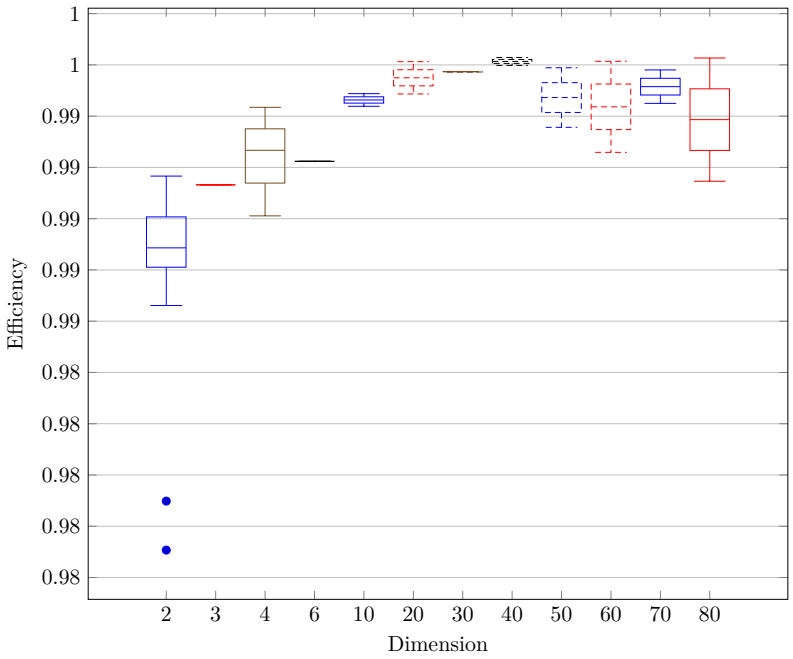



**Figure 5.31.** Box plot of the *efficiency* values of the multicore execution of *NOMAD*, for each dimension.

### 5.4.3 DIRECT-GL

Figure 5.32 shows the box plot of the **RAM usage** of *DIRECT-GL*, for each test function dimension. *DIRECT-GL* used a maximum of ≈ 247.547 GiB when optimizing the *Ackley Function* of dimension 100. Instead, the solver used the minimum amount of memory when optimizing the *Levy Function* of dimension 2, using ≈ 0.8298 GiB.

The *MATLAB* implementation of *DIRECT-GL* is not parallelized on multiple cores, nevertheless, *MATLAB* performs internally some parallelization of the code. For this, the estimations of the **sequential execution time** required by *DIRECT-GL*, which is the total CPU time of the process, are represented in the box plot in Figure 5.33. The maximum time required by *DIRECT-GL* to perform optimization on a single CPU core has been obtained with the *Ackley Function* of dimension 60, where the optimizer would run in 19 days, 12 hours, 36 minutes and 52.88 seconds to complete the execution with a single CPU core. The minimum sequential time required has been obtained with the *Levy Function* of dimension 2, taking only 0.57 seconds.

Figure 5.34 shows the box plot of the **"multicore" time** that *DIRECT-GL* used to complete the optimizations, for each function dimension. The optimization that took the most time was the one on *Ackley Function* with dimension 70, which took 7 days, 14 hours, 24 minutes and 1.08 seconds. The fastest optimization was the one on the *Easom Function* of dimension 2, where *DIRECT-GL* ran for only 0.55 seconds.

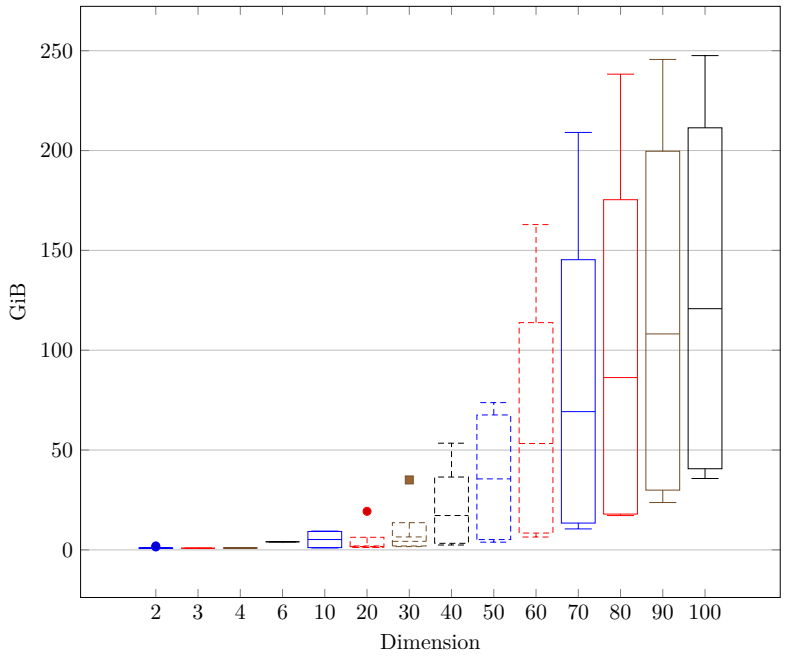



**Figure 5.32.** Box plot of the RAM usage in Gibibyte of *DIRECT-GL*, for each dimension.

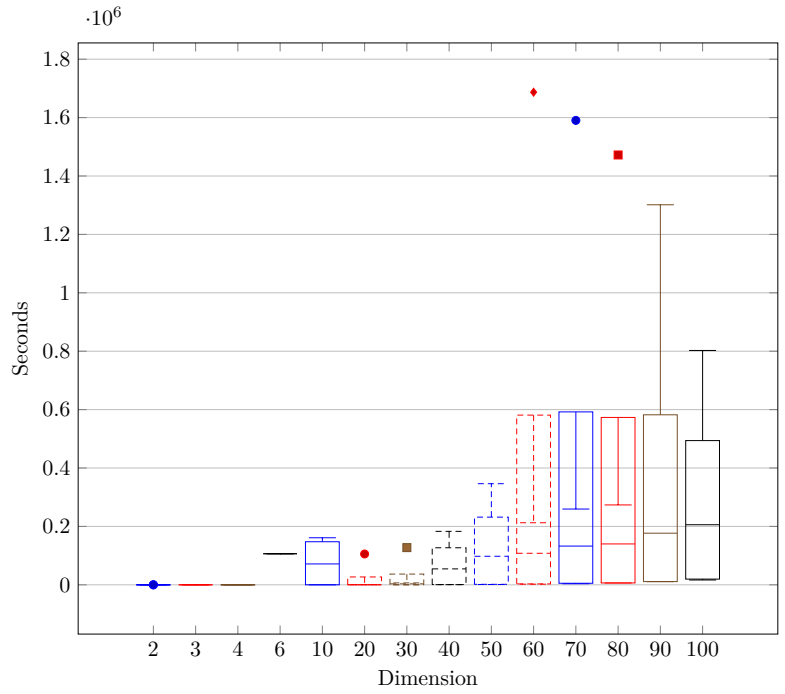



**Figure 5.33.** Box plot of the sequential exeuction time of *DIRECT-GL* in seconds unit, for each dimension.

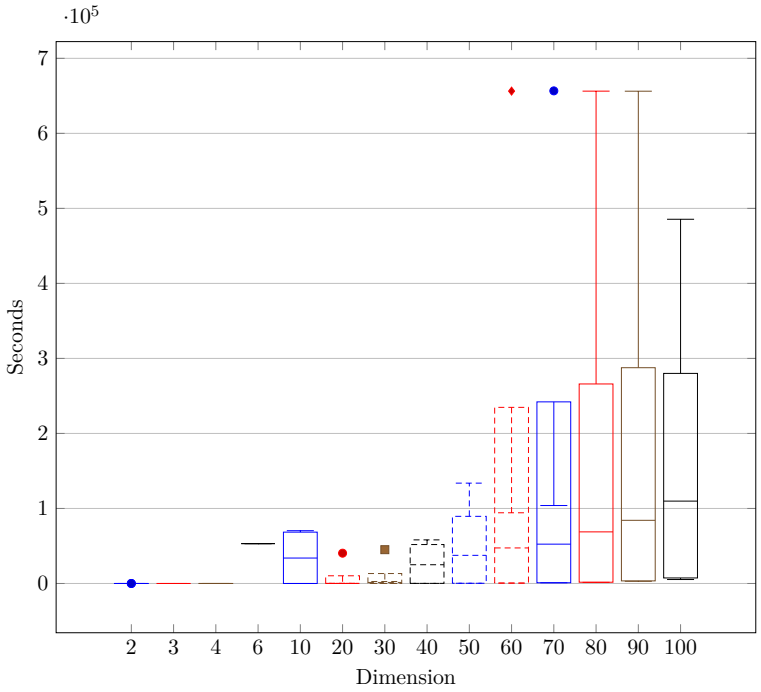



**Figure 5.34.** Box plot of the multicore time, in seconds unit, otbained by *DIRECT-GL*, for each dimension.

# Chapter 6

# Conclusions

The results of the experiments in Chapter 5 confirm the ***correctness*** of the *Statistical Black-Box Global Optimization* algorithm, i.e., given a black-box function $f$ and a black-box optimizer with probability $\geq \epsilon$ of finding a $\theta$-optimal solution x, the *Statistical Black-Box Global Optimization* algorithm returns x with probability $1 - \delta$.

In fact, by applying the *Statistical Black-Box Global Optimization* algorithm (with $\delta = \epsilon = 10^{-3}$ and so $N = 6905$) to the *Nevergrad* and *NOMAD* optimizers, which are the state-of-the-art of black-box local optimization and so they are expected to have a probability of finding a $\theta$-optimal solution with probability $\geq 10^{-3}$, it has been possible to obtain **$10^{-2}$-optimality** on all the test functions with dimension $\leq 100$, extracting the maximum potential of both methods in terms of optimization accuracy, which was not possible with a single optimization run on $f$.

The *Statistical Black-Box Global Optimization* method **outperformed** the state of the art black-box global optimizer *DIRECT-GL*, both in terms of the number of problems solved and function evaluations required to perform optimization. In fact, *DIRECT-GL* solved 73.48% of the problems when considering $10^{-2}$-optimal solutions, having difficulty with high-dimensional functions, while both *Nevergrad* and *NOMAD* solved 100% of them. In order to reduce the number of function evaluations used by the *Statistical Black-Box Global Optimization* method, a lower bound on the probability of *Nevergrad* to find a $10^{-2}$-optimal solution for functions with dimension $\leq 100$ is estimated from the previous tests with $\delta = \epsilon = 10^{-3}$ and $N = 6905$. By setting the value of $\epsilon$ to the estimation 0.1, and so $N = 66$, *Nevergrad* still obtained the $10^{-2}$-optimality on all the test functions, using a number of function evaluations that have an order of magnitude of 6, with respect to the order of magnitude 7 obtained by *DIRECT-GL*.

The method requires that the evaluation of the objective function $f$ is not expensive in terms of time and computational resources, given that a large number of function evaluations is required when the *optimizer* on which *hypothesis testing* is performed has a low probability of finding a $\theta$-optimal solution.

Furthermore, *Statistical Black-Box Global Optimization* is best suitable when addressing high-dimensional black-box objective functions. In fact, *DIRECT-GL* solved all the problems with dimension $< 6$ when considering $10^{-4}$-optimal solutions, using significantly less function evaluations with respect to the *Statistical*

*Black-Box Global Optimization* method when applied to *Nevergrad*.

Possible **future developments** of the *Statistical Black-Box Global Optimization* method could include an implementation of the algorithm in a distributed fashion. The statistical independence of the $N$ runs opens to a possible horizontal scalability of the resources employed to execute the method, i.e., rather than scale on CPU cores, the algorithm still executes $N$ independent runs on multiple cores, but also on multiple computers. This could result in a large speedup in terms of execution time, whith which a more expensive (in terms of resources required to obtain a function evaluation) objective function , also with a dimension $> 100$, could be addressed.

*Statistical Black-Box Global Optimization* can also be used in order to perform benchmarks on multiple solvers. With low values of the parameters $\delta$ and $\epsilon$, the algorithm returns with high confidence the minimum value that a solver can find in any objective function $f$. These experiments could yield a local optimizer to outperform, in terms of solution accuracy and function evaluations used, the best-performing solver in this work *Nevergrad*.

Finally, it is also interesting to test the *Statistical Black-Box Global Optimization* on real-world black-box optimization problems, ranging in multiple fields [4] (e.g., material science, astrophysics, energy distribution and transmission management).

# Appendix A

# Test functions table

| Name | $f^*$ | Dimension | Domain |
|---|---|---|---|
| | ***Many Local Minima*** | | |
| Ackley Function | 0 | $d$ | $[-32.768, 32.768]$ |
| Cross-in-Tray Function | -2.06261 | 2 | $[-10, 10]$ |
| Drop-Wave Function | -1 | 2 | $[-5.12, 5.12]$ |
| Eggholder Function | -959.6407 | 2 | $[-512, 512]$ |
| Griewank Function | 0 | $d$ | $[-600, 600]$ |
| Levy Function | 0 | $d$ | $[-10, 10]$ |
| Levy Function N. 13 | 0 | 2 | $[-10, 10]$ |
| Rastrigin Function | 0 | $d$ | $[-5.12, 5.12]$ |
| Schaffer Function N. 2 | 0 | 2 | $[-100, 100]$ |
| Shubert Function | -186.7309 | 2 | $[-10, 10]$ |
| | ***Bowl-Shaped*** | | |
| Bohachevsky Functions | 0 | 2 | $[-100, 100]$ |
| | ***Plate-Shaped*** | | |
| Booth Function | 0 | 2 | $[-10, 10]$ |
| Matyas Function | 0 | 2 | $[-10, 10]$ |
| McCormick Function | -1.9133 | 2 | $x_1 \in [-1.5, 4], x_2 \in [-3, 4]$ |
| | ***Valley-Shaped*** | | |

| | | | |
|---|---|---|---|
| Three-Hump Camel Function | 0 | 2 | $[-5, 5]$ |
| Six-Hump Camel Function | 0 | 2 | $x_1 \in [-3, 3], x_2 \in [-2, 2]$ |
| ***Steep Ridges/Drops*** | | | |
| Easom Function | -1 | 2 | $[-100, 100]$ |
| ***Others*** | | | |
| Beale Function | 0 | 2 | $[-4.5, 4.5]$ |
| Branin Function | 0.397887 | 2 | $x_1 \in [-5, 10], x_2 \in [0, 15]$ |
| Colville Function | 0 | 4 | $[-10, 10]$ |
| Goldstein-Price Function | 3 | 2 | $[-2, 2]$ |
| Hartmann 3-D Function | -3.86278 | 3 | $[0, 1]$ |
| Hartmann 6-D Function | -3.32237 | 6 | $[0, 1]$ |
| Powell Function | 0 | $d$ | $[-4, 5]$ |
| Shekel Function | -10.5364 | 4 | $[0, 10]$ |

**Table A.1.** Optimization test functions selected from the online library of Surjanovic and Bingham [36]. The table lists the function used in the evaluation of both the *Statistical Black-Box Global Optimization* method and the *DIRECT-GL* [35] algorithm.

# Bibliography

[1] Junaid Aasi, J Abadie, BP Abbott, Richard Abbott, TD Abbott, Matthew Abernathy, Timothee Accadia, Fausto Acernese, Carl Adams, Thomas Adams, et al. Einstein@ home all-sky search for periodic gravitational waves in ligo s5 data. *Physical Review D*, 87(4):042001, 2013.

[2] Mark A Abramson, Charles Audet, John E Dennis Jr, and Sébastien Le Digabel. Orthomads: A deterministic mads instance with orthogonal directions. *SIAM Journal on Optimization*, 20(2):948–966, 2009.

[3] Stéphane Alarie, Charles Audet, Vincent Garnier, Sébastien Le Digabel, and Louis-Alexandre Leclaire. Snow water equivalent estimation using blackbox optimization. *Pac J Optim*, 9(1):1–21, 2013.

[4] Stéphane Alarie, Charles Audet, Aïmen E Gheribi, Michael Kokkolaras, and Sébastien Le Digabel. Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, 9:100011, 2021.

[5] C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes. Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 48(3):35:1–35:22, 2022.

[6] Charles Audet and John E Dennis Jr. Analysis of generalized pattern searches. *SIAM Journal on optimization*, 13(3):889–903, 2002.

[7] Charles Audet and John E Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

[8] Charles Audet and John E Dennis Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on optimization*, 20(1):445–472, 2009.

[9] Charles Audet and Warren Hare. Derivative-free and blackbox optimization. 2017.

[10] Pauline Bennet, Carola Doerr, Antoine Moreau, Jeremy Rapin, Fabien Teytaud, and Olivier Teytaud. Nevergrad: black-box optimization platform. *ACM SIGEVOlution*, 14(1):8–15, 2021.

[11] Kasra Bigdeli, Warren Hare, and Solomon Tesfamariam. Configuration optimization of dampers for adjacent buildings under seismic excitations. *Engineering Optimization*, 44(12):1491–1509, 2012.

[12] Ibrahim M Chamseddine and Michael Kokkolaras. Nanoparticle optimization for enhanced targeted anticancer drug delivery. *Journal of Biomechanical Engineering*, 140(4), 2018.

[13] Diego Desani, Veronica Gil-Costa, Cesar AC Marcondes, and Hermes Senger. Black-box optimization of hadoop parameters using derivative-free optimization. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 43–50. IEEE, 2016.

[14] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[15] Radu Grosu and Scott A Smolka. Monte carlo model checking. In *Tools and Algorithms for the Construction and Analysis of Systems: 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005. Proceedings 11*, pages 271–286. Springer, 2005.

[16] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[17] Kenneth Holmström, Anders O Göran, and Marcus M Edvall. User's guide for tomlab 7. Technical report, Technical Report. Tomlab Optimization Inc, 2010.

[18] Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14:331–355, 1999.

[19] Donald R Jones. Direct global optimization algorithm. *Encyclopedia of optimization*, 1(1):431–440, 2009.

[20] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79:157–181, 1993.

[21] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[22] Jie Liu, Huachao Dong, and Peng Wang. Multi-fidelity global optimization using a data-mining strategy for computationally intensive black-box problems. *Knowledge-Based Systems*, 227:107212, 2021.

[23] Marie Minville, Dominique Cartier, Catherine Guay, Louis-Alexandre Leclaire, Charles Audet, Sébastien Le Digabel, and James Merleau. Improving process representation in conceptual hydrological model calibration using climate simulations. *Water Resources Research*, 50(6):5044–5073, 2014.

[24] Katta G Murty and Santosh N Kabadi. Some np-complete problems in quadratic and nonlinear programming. Technical report, 1985.

[25] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[26] Panos M Pardalos and Georg Schnitger. Checking local optimality in constrained quadratic programming is np-hard. *Operations Research Letters*, 7(1):33–35, 1988.

[27] Michael JD Powell. Uobyqa: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[28] Michael JD Powell. The newuoa software for unconstrained optimization without derivatives. *Large-scale nonlinear optimization*, pages 255–297, 2006.

[29] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

[30] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56:1247–1293, 2013.

[31] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[32] MC Spillane, E Gica, and VV Titov. Tsunameter network design for the us dart® array. In *AGU Fall Meeting Abstracts*, volume 2009, pages OS43A–1368, 2009.

[33] Linas Stripinis and Remigijus Paulavicius. DIRECTGOLib - a library of global optimization problems for DIRECT-type methods.

[34] Linas Stripinis and Remigijus Paulavičius. Directgo: A new direct-type matlab toolbox for derivative-free global optimization. *ACM Transactions on Mathematical Software*, 48(4):1–46, 2022.

[35] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. Improved scheme for selection of potentially optimal hyper-rectangles in direct. *Optimization Letters*, 12:1699–1712, 2018.

[36] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. `http://www.sfu.ca/~ssurjano/optimization.html`.

[37] Enrico Tronci, Toni Mancini, Ivano Salvo, Stefano Sinisi, Federico Mari, Igor Melatti, Annalisa Massini, Francesco Davi, Thomas Dierkes, Rainald Ehrig, et al. Patient-specific models from inter-patient biological models and clinical records. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 207–214. IEEE, 2014.

[38] Shinji Watanabe and Jonathan Le Roux. Black box optimization for automatic speech recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3256–3260. IEEE, 2014.

[39] Aref Yelghi and Cemal Köse. A modified firefly algorithm for global minimum optimization. *Applied Soft Computing*, 62:29–44, 2018.