University of Padova

Department of Department of Mathematics

Master Thesis in Data Science

Adaptive evolution strategies for blackbox optimization

Supervisor
Francesco Rinaldi
University of Padova

Master Candidate
Alessandro Manente

Academic Year

2020-2021

To all those who helped me along the way.

Abstract

Optimization of black-box functions has been of interest to researchers for many years and has become more relevant since the advent of reinforcement learning problems, which goes along with the development of machine learning.

One of the ways used to tackle the problem is the use of evolutionary strategy algorithms. These are able to optimize the given function without the need to compute the gradient of the function itself, which is the main problem while dealing with black-box functions, and they also have theoretical guarantees for their ability to converge to an optimum. After a brief discussion of state-of-the-art algorithms, in this thesis a novel algorithm is presented and compared to them. The algorithm, called ASHGF, implements adaptivity of parameters to escape local global minima and use of historical gradients in order to deal with the exploration-exploitation trade-off. Some theoretical results are given, and it is further numerically validated.

All the algorithms are first compared on a standard testbed and then on a set of reinforcement learning problems. It will be shown that the algorithm can outperform all the other state-of-the-art algorithms.

Contents

Listing of figures

Listing of tables

Listing of acronyms

BO	Blackbox Optimization
DFO	Derivative-Free Optimization
ZO	Zeroth-Order
RS	Random Search
ES	Evolution Strategy
DGS	Directional Gaussian Smoothing
GH	Gauss-Hermite quadrature
RL	Reinforcement Learning

1
Introduction

1.1 Historical Introduction

In recent years, starting from the advance of robotics [1], [2], and then the popularization and spread of machine learning, the need for affordable and powerful algorithms able to optimize function without the need to compute their derivative has increased. These kind of algorithms trace back to the 1970s, with the definition of the Nelder-Mead strategy [3], and since then this field has been largely expanded and developed.

The paths taken by the literature are various:

- Zeroth-order algorithms: a branch of optimization focused on the study of algorithms that mimics first order methods. These algorithms implement the same structure adopted in their first order counterpart and, instead of computing the gradient and use it as descent direction, an estimation of the gradient is used. Relevant examples of this commitment are the work from Nesterov [4], who laid the theoretical ground for further works i.e. the use of Gaussian Smoothing to estimate the gradient of the function to be optimized, the adaptation to the Zeroth-order framework of the stochastic gradient descent [5], stochastic variance reduction [6], the largely used algorithm in deep learning AdaMM [7];

- Evolution Strategies: this branch focuses on the idea of evolution strategy, of which a prime example is CMA-ES [8]. This kind of research converged, de facto, into algorithms, of which the one presented in this thesis is an example, which are somehow related to random searches and Nesterov's algorithm. The connection with Zeroth-order algorithm is now becoming increasingly clear thanks to the use of the Gaussian Smoothing technique to compute a descent direction.

- Derivative-Free Optimization: A large number of algorithms and techniques fall into this class of methods. Depending on the problem that needed to be solved, various approaches were and are developed, like, e.g., directional direct-search methods [9], model-based methods [10, 11].

The method presented in this work does not use the usual techniques studied in the Zeroth-order literature; thus it is to be categorised as an Evolution Strategy. It implements a peculiar way of choosing the descent directions, a different way to estimate the gradient and adaptivity of some hyperparameters. Relevant works associated with this one are the algorithms SGES [12] and ASGF [13]. In fact, in ASGF the authors proposed an algorithm that uses:

- Alternative gradient estimation method: this substitutes a $d-$dimensional Gaussian Smoothing with d $1-$dimensional Gaussian Smoothing;
- The use of a random orthonormal basis of the space in which the first direction is the direction of estimated gradient at the previous iteration;
- The adaptivity of smoothing parameter and learning rate.

The relevance of SGES is given by the way in which it uses previously estimated gradients:

- it keeps track of the last k estimated gradients and uses them to influence the choice of some of the directions used to estimate the gradient in the following iteration;
- at each iteration it adapts the number of directions to be influenced by previously estimated gradients.

1.2 Problem Definition

1.2.1 Black-Box optimization

The general problem that the algorithm aims to solve is the one of finding global minima for a high-dimensional nonconvex objective function $F : \mathbb{R}^d \to \mathbb{R}$. Here is considered the unconstrained black-box optimization problem, parameterized by a d -dimensional vector $x = (x_1, \dots, x_d) \in \mathbb{R}^d$, i.e.,

$$\min_{x \in \mathbb{R}^d} F(x) \tag{1.1}$$

$F(x)$ is assumed to be of black-box type, and the gradient $\nabla F(x)$ is inaccessible, thus (1.1) is typically solved with a derivative- or gradient-free optimization method. Such a function will be defined as a black-box function.

1.2.2 Reinforcement Learning problem

The reinforcement learning problem is meant to frame a problem in which to achieve a goal an agent learns from interaction with the environment. Here we call an agent the learner and decision-maker, while we denominate the environment everything outside of the agent. More specifically here the problem is considered as an episodic problem, as defined in [14].

The cycle of learning is characterized by a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$, where at each time t, the state $S_t \in S$ (S set of all possible states), a representation of the environment at time step t, is given to the agent, who, depending on the state, chooses an action $A_t \in A(S_t)$ ($A(S_t)$ set of all actions available in step S_t). At the next step a reward $R_{t+1} \in R \subset \mathbb{R}$ and the new state of the environment are given to the agent. Note that the reward is stochastic in the sense that the agent does not always face the same problem within the same environment because the latter presents always random differences (think for example at the fact that when playing chess no play is the same as the one before that).

The choice of the action is done through a mapping of the state to probabilities of selecting each possible action. Such mapping is called a policy and is written as π_t, with $\pi_t(a|s)$ is the probability of choosing action a if at state t.

The goal of the agent is to maximize the total amount of reward received in the long run.

Reinforcement Learning via black-box oracle

When solving Reinforcement Learning problem, given $\pi_x : \mathbb{R}^d \to \mathbb{R}^p$ a policy parametrized by x, the average total reward of a given policy $\mathbb{E}_\xi \left[r\left(\pi_x, \xi\right)\right]$, whose maximization is the main goal, can be seen as a black-box function $F(x)$, hence the problem can be written as:

$$\max_{x \in \mathbb{R}^d} \mathbb{E}_\xi \left[r\left(\pi_x, \xi\right)\right] = \max_{x \in \mathbb{R}^d} F(x) \tag{1.2}$$

Through this interpretation of the problem, the algorithm is concerned with only the result of a whole episode, treating the function as a black-box oracle. This approach to the solution of reinforcement learning problems is not a novel idea. As [1] points out, reinforcement learning algorithms evolved during the years showing increasing similarities with evolution strategies. This in turns reflects the abandonment of the classical framework of reinforcement learning problems and the adoption of the concept of black-box oracle.

1.3 Aim of this work

The aim of this work is to take the idea of SGES and insert it into the framework of ASGF, specifically changing the update of the orthonormal basis, which will have a number of directions dependent on the previous gradients that will be adapted during the execution of the algorithm.

Then the algorithm is tested against some state-of-the-art algorithms on a suit of challenging functions taken from [15] and [16], and on some Reinforcement Learning problems taken from [17].

2 State-of-the-art algorithms

2.1 Random Search

The first relevant algorithm proposed in the literature to approach the problem (1.1), is the algorithm from [4], *Random Search*, with the aim to simulate the working of the gradient descent but translated in the zeroth-order optimization.

In order to overcome the limitations posed by the impossibility of computing the gradient, the idea of Nesterov, and the literature that followed, was to replace the objective function with a new function which keeps most of the properties of the original but makes the computation of the gradient possible. This strategy is the so called Gaussian Smoothing:

Definition 2.1 (Gaussian Smoothing). *Let* $\sigma > 0$ *the smoothing parameter, then we denote by* $F_\sigma(x)$ *the Gaussian Smoothing of* F *with radius* σ*, i.e.*

$$F_\sigma(x) = \frac{1}{\pi^{d/2}} \int_{\mathbb{R}^d} F(x + \sigma\xi) e^{-\|\xi\|_2^2} \, \mathrm{d}\xi = \mathbb{E}_{\xi \sim \mathcal{N}(0, \mathbb{I}_d)}[F(x + \sigma\xi)] \tag{2.1}$$

F_σ keeps all the features of F, such as convexity and the Lipschitz constant, while being always differentiable, even when F is not. This is relevant because, given that $\|F - F_\sigma\|$ can be bounded by the Lipschitz constant, (1.1) can be replaced with

$$\min_{x \in \mathbb{R}^d} F_\sigma(x) \tag{2.2}$$

Now, the gradient of the smoothed F is:

$$\nabla F_\sigma(x) = \frac{2}{\sigma \pi^{d/2}} \int_{\mathbb{R}^d} \xi F(x+\sigma\xi) e^{-\|\xi\|_2^2} \,\mathrm{d}\xi = \frac{2}{\sigma} \mathbb{E}_{\xi\sim\mathcal{N}(0,\mathbb{I}_d)}[\xi F(x+\sigma\xi)] \tag{2.3}$$

This is still a difficult object to compute, but it can be estimated easily. The estimator usually exploited to estimate this gradient is obtained via Monte Carlo sampling, with a central finite difference approach, which is used to reduce the variance of the estimation; namely, given $M \in \mathbb{N}_+$ the number of directions and $\{\xi_j\}_{j=1}^M \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$ the directions, then we have the following:

Definition 2.2 (Gradient of Gaussian Smoothed function).

$$\nabla F_\sigma(x) \approx \frac{1}{2\sigma M} \sum_{j=1}^{M} \xi_j \left(F\left(x+\sigma\xi_j\right) - F\left(x-\sigma\xi_j\right)\right) \tag{2.4}$$

Actually, Nesterov in its work first proposed and studied the theoretical properties of its algorithm with $M = 1$; following literature increased it.

Algorithm 1 Random Search

```
    Input: function $f$, state $x_0$, smoothing parameter $\sigma$, stepsize $\lambda$
1:  for $t = 0, 1, 2, \ldots$ do
2:      Sample $\xi \sim \mathcal{N}(0, I_n)$
3:      Compute returns $F_+ = F(x_t + \sigma\xi)$, $F_- = F(x_t - \sigma\xi)$
4:      Set $x_{t+1} = x_t - \frac{\lambda}{2\sigma}(F_{t_+} - F_{t_-})\xi$
```

The main theoretical result proved by Nesterov for smooth convex functions is the following.

At first, note that this method generates a random sequence $\{x_k\}_{k\geq 0}$. Denote by

$$\mathcal{U}_k = (\xi_0, \ldots, \xi_k) \tag{2.5}$$

a random vector composed by i.i.d. variables $\{u_k\}_{k\geq 0}$ related to each iteration of the scheme. Denote $\phi_0 = F(x_0)$, and $\phi_k \overset{\text{def}}{=} \mathbb{E}_{\mathcal{U}_{k-1}}[F(x_k)], \quad k \geq 1$. Then it holds:

Theorem 2.1.1. *Let F have Lipschitz Continuos Gradient, and sequence $\{x_k\}_{k\geq 0}$ be generated by 1 with*

$$\lambda = \frac{1}{4(d+4)L} \tag{2.6}$$

Then, for any $N \geq 0$, we have

$$\frac{1}{N+1}\sum_{k=0}^{N}(\phi_k - f^*) \leq \frac{4(d+4)L\,\|x_0 - x^*\|^2}{N+1} + \frac{9\sigma^2(d+4)^2L}{25} \tag{2.7}$$

Let function F be strongly convex. Denote $\delta_\sigma = \frac{18\sigma^2(d+4)^2}{25\tau}L$. Then

$$\phi_N - F^* \leq \frac{1}{2}L\left[\delta_\sigma + \left(1 - \frac{\tau}{8(d+4)L}\right)^N \left(\|x_0 - x^*\|^2 - \delta_\sigma\right)\right] \tag{2.8}$$

Considering the strongly convex case, the smoothing parameter μ has to be chosen such that it satisfies the inequality $\frac{1}{2}L\delta_\mu$. In this case, the number of iterations needed by the method to satisfy $\phi_N - F^* \leq \varepsilon$, is of the order of $\mathcal{O}(d\log\frac{1}{\varepsilon})$. It is interesting to note that the original gradient method requires $\mathcal{O}(\log\frac{1}{\varepsilon})$ iterations, which shows that this method, and all the other methods presented in this work, suffer from the curse of dimensionality. This problem can not be avoided: in fact using $M = d$ directions to estimate the gradient would lead to $\mathcal{O}(\log\frac{1}{\varepsilon})$ iterations, at the cost of increasing the computational time for each iteration by a factor of d.

2.2 ES

The following algorithm, from [18], is essentially an analogous of Nesterov's Random Search which has its main focus on the increment on the number M of directions used during the estimation of the gradient.

Algorithm 2 ES algorithm

```
   Input: function $f$, state $x_0$, smoothing parameter $\sigma_0$, stepsize $\lambda$
1: for $t = 0, 1, 2, \ldots$ do
2:     Sample $\xi_1, \ldots, \xi_n \sim \mathcal{N}(0, I_n)$
3:     Compute returns $F_{i_+} = F(x_t + \sigma\xi_i)$, $F_{i_-} = F(x_t - \sigma\xi_i)$, for $i = 1, \ldots, n$
4:     Set $x_{i+1} = x_i - \frac{\lambda}{2n\sigma}\sum_{i=1}^{n}(F_{i_+} - F_{i_-})\xi_i$
```

The authors focused on the use of the algorithm to solve RL problems, as in 1.1, and the possibility of parallelizing the execution of the algorithm which showed the strength of the algorithm to train AI's for this aim. This is proven to be able to led to an incredible advantage against other RL algorithms.

2.3 ASEBO

Inspired by the classical RL literature, in [19], the authors tried to embed the idea of the exploration-exploitation tradeoff into the framework given by Algorithm 1 and 2. To achieve this they introduced the idea of active subspaces, originally presented in [20], and they devised an adaptive method that tries to balance between exploration and exploitation. In their work they identified exploration as the choice of directions independent of previous gradients directions, while exploitation was associated to choice of random directions dependent of previous gradients.

Algorithm 3 ASEBO algorithm

```
    Input: function $F$, state $x_0$, smoothing parameter $\sigma_0$, stepsize $\lambda$, decay rate $\nu$, termination tolerance $\varepsilon$, number of iterations of full-sampling $l$, total iterations maxiter
    Output: point of minimum $\tilde{x}$
 1: Set $\tilde{x} = x_0$, $\sigma = \sigma_0$ and let $\Sigma$ be a random orthonormal basis
 2: Initialize an archive $\mathcal{G}$ of maximum capacity $k$
 3: for $t = 0$ to maxiter do
 4:     if $t < l$ then
 5:         Let $n_t = d$. Sample $\xi_1, \dots, \xi_{n_t}$ from $\mathcal{N}(0, I_d)$
 6:     else
 7:         Take top $r$ eigenvalues $\lambda_i$ of $\mathrm{Cov}_t$, where $r$ is smallest such that: $\sum_{i=1}^{r} \dot{\lambda}_i \geq \epsilon \sum_{i=1}^{d} \lambda_i$ using its SVD as described in text and take $n_t = r$.
 8:         Take the corresponding eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_r \in \mathbb{R}^d$ and let $\mathbf{U} \in \mathbb{R}^{d \times r}$ be obtained by stacking them together. Let $\mathrm{U}^{\mathrm{act}} \in \mathbb{R}^{d \times r}$ be obtained from stacking together some orthonormal basis of $\mathcal{L}_{\text{active}}^{\text{ES}} \overset{\text{def}}{=} \operatorname{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$. Let $\mathbf{U}^{\perp} \in \mathbb{R}^{d \times (d-r)}$ be obtained from stacking together some orthonormal basis of the orthogonal complement $\mathcal{L}_{\text{active}}^{\text{ES},\perp}$ of $\mathcal{L}_{\text{active}}^{\text{ES}}$
 9:         Sample $n_t$ vectors $\mathbf{g}_1, \dots, \mathbf{g}_{n_t}$ as follows: with probability $1 - p^t$ from $\mathcal{N}(0, \mathbf{U}^{\perp}(\mathbf{U}^{\perp})^{\top})$ and with probability $p^t$ from $\mathcal{N}(0, \mathbf{U}^{\mathrm{act}}(\mathbf{U}^{\mathrm{act}})^{\top})$.
10:         Renormalize $\mathbf{g}_1, \dots, \mathbf{g}_{n_t}$ such that marginal distributions $\|\mathbf{g}_i\|_2$ are $\chi(d)$.
11:     Compute $\widehat{\nabla}_{\mathrm{MC}}^{\mathrm{AT}} F(x_t)$ as: $\widehat{\nabla}_{\mathrm{MC}}^{\mathrm{AT}} F(x_t) = \frac{1}{2 n_t \sigma} \sum_{j=1}^{n_t} \left(F(x_t + \mathbf{g}_j) - F(x_t - \mathbf{g}_j)\right) \mathrm{g}_j$
12:     Set $\mathrm{Cov}_{t+1} = \lambda \,\mathrm{Cov}_t + (1-\lambda)\Gamma$, where $\Gamma = \widehat{\nabla}_{\mathrm{MC}}^{\mathrm{AT}} F_\sigma(x_t) \left(\widehat{\nabla}_{\mathrm{MC}}^{\mathrm{AT}} F_\sigma(x_t)\right)^{\top}$.
13:     Set $p^{t+1} = p_{\mathrm{opt}}$ for $p_{\mathrm{opt}}$ output by Algorithm 2 and: $x_{t+1} = x_t - \eta \widehat{\nabla}_{\mathrm{MC}}^{\mathrm{AT}} F(x_t)$.
```

Definition 2.3. *F has a τ-smooth third order derivative tensor with respect to $\sigma > 0$, so that $F(x + \sigma\mathbf{g}) = F(x) + \sigma\nabla F(x)^\top \mathbf{g} + \frac{\sigma^2}{2}\mathbf{g}^\top H(x)\mathbf{g} + \frac{1}{6}\sigma^3 F'''(x)[\mathbf{v},\mathbf{v},\mathbf{v}]$ for some $\mathbf{v} \in \mathbb{R}^d$ $(\|\mathbf{v}\|_2 \le \|\mathbf{g}\|_2)$ satisfying $| F'''(x)[\mathbf{v},\mathbf{v},\mathbf{v}] \le \tau\|\mathbf{v}\|_2^3 \le \tau\|\mathbf{g}\|_2^3$*

The authors proved the following bounds:

Theorem 2.3.1. *If F has Lipschitz Continuos Gradient and τ-smooth third order derivative tensor, the estimators $\widehat{\nabla}^{AT,base}_{MC,k=1}F_\sigma(x)$ and $\widehat{\nabla}^{AT,asebo}_{MC,k=1}F_\sigma(x)$ are close to the true gradient $\nabla F(x)$, i.e.:*

$$\left\|\mathbb{E}_{\mathbf{g}\sim\mathcal{N}(0,\mathbf{I}_d)}\left[\widehat{\nabla}^{\text{AT},\,base}_{\text{MC},k=1}F_\sigma(x)\right] - \nabla F(x)\right\| \le \epsilon \tag{2.9}$$

and

$$\left\|\mathbb{E}_{\mathbf{g}\sim\widehat{P}}\left[\widehat{\nabla}^{\text{AT},\,asebo}_{\mathbf{MC},k=1}F_\sigma(x)\right] - \nabla F(x)\right\| \le \epsilon \tag{2.10}$$

Algorithm 4 ASEBO subroutine

```
    **Hyperparameters:** smoothing parameter $\sigma$, horizon $C$, learning rate $\alpha$, probability regularizer $\beta$, initial probability parameter $q_0^t \in (0,1)$
    **Input:** subspaces: $\mathcal{L}^{\text{ES}}_{\text{active}}$, $\mathcal{L}^{\text{ES},\perp}_{\text{active}}$, function $F$, vector $x_t$
    **Output:** $p_C$
1:  **for** $t = 0$ to *maxiter* **do**
2:      Compute $p_{l-1}^t = (1-2\beta)q_{l-1}^t + \beta$ and sample $a_l^t \sim \text{Ber}\left(p_l^t\right)$
3:      If $a_l^t = 1$, sample $\mathbf{g}_l \sim \mathcal{N}\left(0, \sigma\mathbf{I}_{\mathcal{L}^{\text{ES}}_{\text{active}}}\right)$, otherwise sample $\mathbf{g}_l \sim \mathcal{N}\left(0, \sigma\mathbf{I}_{\mathcal{L}^{\text{ES},\perp}_{\text{active}}}\right)$
4:      Compute $v_l = \frac{1}{2\sigma}\left(F\left(x_t + \mathbf{g}_l\right) - F\left(x_t - \mathbf{g}_l\right)\right)$
5:      Set $\mathbf{e}_l = (1-2\beta)\begin{bmatrix}\left(-\frac{a_l^t\left(\dim\left(\mathcal{L}^{\text{ES}}_{\text{ative}}\right)+2\right)}{\left(p_l^t\right)^3}\right)\\ \left(-\frac{\left(1-a_l^t\right)\left(\dim(\mathcal{L}_{\text{active}})+2\right)}{\left(1-p_l^t\right)^3}\right)\end{bmatrix} v_l^2$
6:      Set $q_l^t = \frac{q_{l-1}^t\exp(-\alpha\mathbf{e}_l(1))}{q_{l-1}^t\exp(-\alpha_l(1))+\left(1-q_{l-1}^t\right)\exp(-\alpha\mathrm{e}_l(2))}$
```

Theorem 2.3.2. *The following holds for $s_{\mathbf{U}^{act}} = \left\|\left(\mathbf{U}^{\text{act}}\right)^\top \nabla F(x)\right\|_2^2$ and $s_{\mathbf{U}^\perp} = \left\|\left(\mathbf{U}^\perp\right)^\top \nabla F(x)\right\|_2^2$:*

- *The variance of $\widehat{\nabla}^{\text{AT},asebo}_{\text{MC},k=1}F_\sigma(x)$ is close to Γ, i.e. $\left|\text{Var}\left[\widehat{\nabla}^{\text{AT},\,asebo}_{\text{MC},k=1}F_\sigma(x)\right] - \Gamma\right| \le \epsilon$*

- *The choice of p^t that minimizes Γ satisfies* $p_*^t := \frac{\sqrt{(s_{Uact})\left(d_{active}+2\right)}}{\sqrt{(s_{Uact})(d_{active}+2)}+\sqrt{(s_{U}\perp)(d_{U\perp}+2)}}$ *and the optimal variance* Var_{opt} *corresponding to* p_*^t *satisfies:* $| \, Var_{opt} - \Delta \, | \leq \epsilon$ *for*

$$\Delta = \left[\sqrt{\left(s_{\mathbf{U}^{act}}\right)(d_{active}+2)} + \sqrt{\left(s_{\mathbf{U}^{\perp}}\right)(d_{\perp}+2)}\right]^2 - \|\nabla F(x)\|^2$$

- $\text{Var}_{\text{opt}} \leq \text{Var}\left[\widehat{\nabla}^{\text{AT},\, base}_{\text{MC},k=1} F_\sigma(x)\right] + \epsilon$
$- \underbrace{\left|\sqrt{(s_{\mathbf{U}^{\perp}})\,(d_{active}+2)} - \sqrt{(s_{\mathbf{U}^{\text{act}}})\,(d_{\perp}+2)}\right|^2 - 2\|\nabla F(x)\|^2.}_{\lambda}$

2.4 SGES

Following the work of [19], [12] tried to improve and simplify Algorithm 3.

Instead of keeping track of all the gradients via momentum-like update of a covariance matrix, a fixed-size buffer of gradients is kept updated and used to compute a covariance matrix; the routine responsible for the choice of directions is also simplified and more numerically stable. To do so, $k \ll d$ gradients are collected from the beginning, and then always the most recent k gradients in the buffer are kept. Then at each iteration $t \geq k$, given $\mathbf{G}_t \in \mathbb{R}^{nxk}$, which is a matrix made with the last k gradients, the algorithm generates the subspace $\mathcal{L}_{\mathbf{G}}$, using the vectors in the buffer. This subspace captures the structure of the optimization problem and, in the reinforcement learning optic, it captures the directions related to exploitation. Then the algorithm samples the search directions from:

$$\begin{cases} \xi \sim \mathcal{N}\left(0, \mathrm{G}^{\perp}\mathrm{G}\right) & \text{with probability } \alpha \\ \xi \sim \mathcal{N}(0, \mathrm{I}) & \text{with probability } 1-\alpha \end{cases} \tag{2.11}$$

$\alpha \in (0,1)$ represents the trade-off between exploitation, choosing a direction from $L_{\mathbf{G}}$, and exploration, from the whole space. The algorithm adapts α at each iteration such that the amount of exploitation and exploration varies depending on how the algorithm is performing. To do that the following are computed:

$$\widehat{r}_{\mathbf{G}} = \frac{1}{M}\sum_{i=1}^{M} \min_{m=1,\ldots,m_i} \left\{F\left(x + \sigma\xi_i\right)\right\} \tag{2.12}$$

$$\widehat{r}_{\mathbf{G}}^{\perp} = \frac{1}{d-M} \sum_{i=M+1}^{d} \min_{m=1,\ldots,m_i} \{F(x+\sigma\xi_i)\} \tag{2.13}$$

Then, at iteration t, α is chosen as:

$$\alpha_t = \begin{cases} \min\{\delta\alpha_{t-1}, \kappa_1\} \text{ if } \widehat{r}_{\mathbf{G}} \text{ does not exist or } \widehat{r}_{\mathbf{G}} < \widehat{r}_{\mathbf{G}}^{\perp} \\ \max\left\{\frac{1}{\delta}\alpha_{t-1}, \kappa_2\right\} \text{ if } \widehat{r}_{\mathbf{G}}^{\perp} \text{ does not exist or } \widehat{r}_{\mathbf{G}} \geq \widehat{r}_{\mathbf{G}}^{\perp} \end{cases} \tag{2.14}$$

with $\delta > 1$ a scaling factor, k_1 and k_2 are upper and lower bounds of α. Intuitively, if $\widehat{r}_{\mathrm{G}} < \widehat{r}_{\mathrm{G}}^{\perp}$ or $\widehat{r}_{\mathrm{G}}$ does not exist, than the algorithm increases α, otherwise it is decreased.

Algorithm 5 SGES algorithm

```
    Input: function $F$, state $x_0$, smoothing parameter $\sigma_0$, learning rate $\lambda$, hyper-parameters $k$, total iterations maxiter, warmup iteration $T_w \geq k$
    Output: point of minimum $\tilde{x}$
 1: Initialize an archive $\mathcal{G}$ of maximum capacity $k$
 2: for $t = 0$ to maxiter do
 3:     if $t < T_w$ then
 4:         Sample search directions $\xi_1, \ldots, \xi_n$ from $\mathcal{N}(0, I_n)$
 5:     else
 6:         Obtain gradient matrix $\mathbf{G} \in \mathbb{R}^{nxk}$ from $\mathcal{G}$
 7:         Generate subspaces $L_{\mathbf{G}}$ and $L_{\mathbf{G}}^{\perp}$
 8:         Sample search directions $\varepsilon_1, \ldots, \varepsilon_n$ from (2.11)
 9:         Normalize these search directions
10:     Compute gradient estimate $\nabla F_\sigma(x_i)$ by (2.2)
11:     Update the parameters $x_{i+1} = x_i - \lambda \nabla F_\sigma(x_i)$
12:     Add the gradient estimate $\nabla F_\sigma(x_i)$ to $\mathcal{G}$
13:     if $i < T_w$ then
14:         Adaptively adjust $\alpha$ as in (2.14)
```

Their main theoretical result is:

Theorem 2.4.1. *Assuming the objective function has Lipschitz Continous Gradient, 2.3 and* $\sigma < \frac{1}{35}\sqrt{\frac{\varepsilon \min\{\alpha, 1-\alpha\}}{\tau n^3 \max\{L, 1\}}}$ *for some precision parameter* $\varepsilon > 0$, *it holds*

$$\left\|\mathbb{E}_{\epsilon\sim\mathcal{P}}\left[\hat{g}_{sges}\right] - \nabla f(x)\right\| \leq \varepsilon \left|\mathrm{Var}\left[\hat{g}_{sges}\right] - \Omega\right| \leq \varepsilon \tag{2.15}$$

where

$$\Omega = \frac{k+2}{\alpha} \cdot \left\|\mathbf{U}^\top \nabla F(x)\right\|^2 - \|\nabla F(x)\|^2 + \frac{n-k+2}{1-\alpha} \cdot \left\|\left(\mathbf{U}^\perp\right)^\top \nabla F(x)\right\|^2 \tag{2.16}$$

2.5 ASGF

This algorithm from [13] embeds a number of changes with respect to the previous literature.

At first, as it was done in [21], it replaces Gaussian Smoothing with Directional Gaussian Smoothing. Where, instead of doing a single $d-$dimensional Gaussian Smoothing, d $1-$dimensional Gaussian Smoothing are performed, which can be approximated via Gauss-Hermite quadrature.

To achieve this, a function $G : \mathbb{R} \to \mathbb{R}$ is defined as a cross section of F along ξ as

$$G(y \mid x, \xi) = F(x + y\xi), \quad y \in \mathbb{R} \tag{2.17}$$

It is further defined the Gaussian smoothing of $G(y)$, denoted as $G_\sigma(y)$, by

$$G_\sigma(y \mid x, \xi) := \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} G(y + \sigma v \mid x, \xi) \mathrm{e}^{-\frac{v^2}{2}} dv = \mathbb{E}_{v \sim \mathcal{N}(0,1)}[G(y + \sigma v \mid x, \xi)] \tag{2.18}$$

which is the Gaussian smoothing of $F(x)$ along ξ near x. Using a one-dimensional expectation, the derivative of $G(y \mid x, \xi)$ at $y = 0$, is

$$\mathcal{D}\left[G_\sigma(0 \mid x, \xi)\right] = \frac{1}{\sigma} \mathbb{E}_{v \sim \mathcal{N}(0,1)}[G(\sigma v \mid x, \xi) v] \tag{2.19}$$

Now let $\Xi = (\xi_1, \ldots, \xi_d)$ an orthonormal basis, than the following vector Directional Gaussian Smoothing gradient (henceforth DGS gradient), can be defined:

Definition 2.4 (DGS Gradient).

$$\nabla_{\sigma,\xi}[F](x) := \left[\mathcal{D}\left[G_\sigma\left(0 \mid x, \xi_1\right)\right], \cdots, \mathcal{D}\left[G_\sigma\left(0 \mid x, \xi_d\right)\right]\right] \Xi \tag{2.20}$$

At this point, the key is that each of the components of the DGS gradient requires a one-dimensional integral, which is approximated with high accuracy with the Gauss-Hermite rule. Hence we obtain the following estimator for $\mathcal{D}\left[G_\sigma(0 \mid x, \xi)\right]$:

$$\widetilde{\mathcal{D}}^M\left[G_\sigma(0 \mid x, \xi)\right] = \frac{1}{\sqrt{\pi}\sigma}\sum_{m=1}^{M} w_m F\left(x + \sqrt{2}\sigma v_m \xi\right)\sqrt{2}v_m \tag{2.21}$$

Thus we define the DGS Gradient Estimator as

Definition 2.5 (DGS Gradient Estimator).

$$\widetilde{\nabla}^M_{\sigma,\Xi}[F](x) = \left[\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_1\right)\right], \cdots, \widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_d\right)\right]\right]\Xi \tag{2.22}$$

The authors then fix the balance between exploration and exploitation by assigning, at each iteration, the previous estimated gradient as first vector of the new set of direction, keeping the remaining directions randomly chosen. It then estimates for each direction a Local Lipschitz Constant to update, with a momentum-like mechanism, the learning rate like in 3.2.

Algorithm 6 ASGF algorithm

Input: function f, state x_0, smoothing parameter σ_0, termination tolerance ε, hyper-parameters α, k, total iterations *maxiter*

Output: point of minimum $\tilde{x}$

```
1:  Set $\tilde{x} = x_0$, $\sigma = \sigma_0$ and let $\Xi$ be a random orthonormal basis
2:  for $i = 0$ to maxiter do
3:      for $j = 1$ to $d$ do
4:          compute $\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_j\right)\right]$ by (2.21) and estimate local Lipschitz constants $L_j$
5:      Average Lipschitz constant $L_\nabla$ and compute learning rate $\lambda$
6:      Assemble the gradient $\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$ by (2.22) and update $x_{i+1} = x_i - \lambda\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$
7:      Add the gradient estimate to $\mathbf{G}$
8:      if $f\left(x_{i+1}\right) < f(\tilde{x})$ then
9:          Update $\tilde{x} = x_{i+1}$
10:     if $\left\|x_{i+1} - x_i\right\|_2 < \varepsilon$ then
11:         break
12:     else
13:         Update smoothness parameter $\sigma$ and the search directions $\Sigma$ by Algorithm (7)
```

Algorithm 7 Parameter update

Input: smoothing parameter σ, gradient $\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$, local Lipschitz constants $L_1, \ldots, L_d$

Data: number of resets r and reset factor ρ, decay rate γ_σ, threshold parameters A, B and their change rates A_+, A_-, B_+, B_-

Output: smoothing parameter σ and directions Σ

```
1:  if $r > 0$ and $\sigma < \rho\sigma_0$ then
2:      Assign $\Xi$ to be a random orthonormal basis and set $\sigma = \sigma_0$
3:      Set $A, B$ to their initial values and set $r = r - 1$
4:  else
5:      Update directions $\mathcal{D}$
6:      if $\max_{1\leq j\leq d} \left|\widetilde{\mathcal{D}}^M \left[G_\sigma\left(0 \mid \boldsymbol{x_i}, \xi_j\right)\right]/L_j\right| < A$ then
7:          Decrease smoothing $\sigma = \sigma * \gamma_\sigma$ and lower threshold $A = A * A_-$
8:      else if $\max_{1\leq j\leq d} \left|\widetilde{\mathcal{D}}^M \left[G_\sigma\left(0 \mid \boldsymbol{x_i}, \xi_j\right)\right]/L_j\right| > B$ then
9:          Increase smoothing $\sigma = \sigma/\gamma_\sigma$ and upper threshold $B = B * B_+$
10:     else
11:         Increase lower threshold $A = A * A_+$ and decrease upper threshold $B = B * B_-$
```

3 The Algorithm

In this chapter the proposed algorithm is described in Section 3.4 while the previous sections explain the various pieces that compose it.

3.1 Historical Gradients

As in Algorithm 5, the proposed method uses a buffer of previously calculated gradients in order to sample directions which strives to achieve a better trade-off between exploration and exploitation. The main difference with respect to Algorithm 5 lies in the fact that the presented algorithm orthonormalizes the set of directions produced at each iteration, this being needed for the estimation of the gradient.

3.2 The Gradient Estimator

Like in Algorithm 6, the algorithm does not exploit Gaussian Smoothing, but rather makes use Directional Gaussian Smoothing in order to estimate the gradient of the function which is to be optimized. The main difference between Algorithm 6 and 8 is that the last one uses a equal fixed amount of quadrature points both for directions chosen completely at random and those dependant on the previous estimated gradients as in (2.11), while the former uses a smaller amount of points for completely random directions and for the single direction dependent on the gradient, it increases the number of points until a threshold of given accuracy is reached. Both from a practical and theoretical point of view the latter approach should be preferable because, at the same time, reduces the time needed to execute the algorithm, reduces the number of function evaluations required by the algorithm and then makes it feasible to talk about convergence results.

3.3 Adaptation of the Parameters

Lastly, instead of keeping for the whole execution of the algorithm some fixed values of the smoothing parameter σ and the learning rate λ, it modifies them at each iteration, in order to estimate their best values.

3.3.1 Learning rate

To update the learning rate, at each iteration the directional local Lipschitz constants L_j are estimated as:

$$L_j = \max_{\{i,k\}\in I} \left| \frac{F\left(x+\sigma p_i \xi_j\right) - F\left(x+\sigma p_k \xi_j\right)}{\sigma\left(p_i - p_k\right)} \right| \tag{3.1}$$

where $I = \left\{\{i,k\} \in \{1,\dots,m\}^2 \;\middle|\; \mid i - \lfloor \frac{m}{2} \rfloor - 1 \mid \neq \mid k - \lfloor \frac{m}{2} \rfloor - 1 \mid \right\}$. Then the learning rate is computed as:

$$L_\nabla \leftarrow \left(1-\gamma_L\right) L_{\mathbf{G}} + \gamma_L L_\nabla \quad \text{and} \quad \lambda = \sigma / L_\nabla \tag{3.2}$$

where $L_{\mathbf{G}}$ is the max of the estimated Lipschitz constants associated to the directions sampled from $\mathcal{L}_{\mathbf{G}}$. Note that until iteration T_w, $L_{\mathbf{G}}$ is the max of all the local Lipschitz constants.

3.3.2 Smoothing parameter

At each iteration, to adapt the smoothing parameter to the local geometry, $\forall j = 1,\dots,d$, $\widetilde{\mathcal{D}}^M \left[G_\sigma\left(0 \mid x, \xi_j\right)\right]$ is compared to the corresponding local Lipschitz constant L_j. If the max of all the ratios of the absolute value of the two quantities is sufficiently large, σ is increase, else it is increased.

Furthermore, the algorithm has a given number of reset of the smoothing parameter σ and directions to initial conditions. This is implemented in order to help the algorithm escaping local minima that would otherwise make it stop. This happens whenever the smoothing parameter becomes sufficiently small, $\sigma < \rho\sigma_0$.

3.4 The Algorithm

Algorithm 8 ASHGF algorithm

```
    Input: function $f$, state $x_0$, smoothing parameter $\sigma_0$, termination tolerance $\varepsilon$, hyper-
parameters $\alpha, k$, total iterations maxiter, warmup iteration $T_w = k$
    Output: point of minimum $\tilde{x}$
 1: Set $\tilde{x} = x_0, \sigma = \sigma_0$ and let $\Xi$ be a random orthonormal basis
 2: Initialize an archive $\mathcal{G}$ of maximum capacity $k$
 3: for $i = 0$ to maxiter do
 4:     for $j = 1$ to $d$ do
 5:         compute $\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_j\right)\right]$ by (2.21) and estimate local Lipschitz constants $L_j$
            by (3.1)
 6:     Average Lipschitz constant $L_\nabla$ and compute learning rate $\lambda$ by (3.2)
 7:     Assemble the gradient $\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$ by (2.22) and update $x_{i+1} = x_i - \lambda\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$
 8:     Add the gradient estimate to $\mathcal{G}$
 9:     if $f\left(x_{i+1}\right) < f(\tilde{x})$ then
10:         Update $\tilde{x} = x_{i+1}$
11:     if $\|x_{i+1} - x_i\|_2 < \varepsilon$ then
12:         break
13:     else
14:         if $i < T_w$ then
15:             Set $historical = False$
16:         else
17:             if $i >= T_w + 1$ then
18:                 Adaptively adjust $\alpha$ by (2.14)
19:             Set $historical = True$
20:         Update smoothness parameter $\sigma$ and the search directions $\Xi$ by Algorithm (9)
```

The algorithm, shown in Algorithm 8, is a modification of the random search defined by [4] and the Evolutionary Strategy by [18]. As said in the previous sections, the three main differences, with respect to the algorithms from the literature, are:

- the use of previously generated gradients to pick the direction used to get the gradient estimate;

- the adaptivity of the learning rate and of the smoothing parameter;
- the use of Gauss-Hermite quadrature instead of finite differences.

Furthermore, it is to be noted that the algorithm does not use any kind of technique which stems from the zeroth-order optimization literature. In fact it does not exploit variance reduction, momentum, the sign of gradient, estimated hessian or other approaches. It is, in this regard, a simple random search.

Algorithm 9 Parameter update

```
Input: smoothing parameter $\sigma$, gradient $\widetilde{\nabla}^M_{\sigma,\Xi}[F](x_i)$, local Lipschitz constants $L_1, \dots, L_d$, $historical$, archive of gradients $\mathcal{G}$
Data: number of resets $r$ and reset factor $\rho$, decay rate $\gamma_\sigma$, threshold parameters $A, B$ and their change rates $A_+, A_-, B_+, B_-$
Output: smoothing parameter $\sigma$ and directions $\Xi$
1: if $r > 0$ and $\sigma < \rho\sigma_0$ then
2:     Assign $\Xi$ to be a random orthonormal basis and set $\sigma = \sigma_0$
3:     Set $A, B$ to their initial values and set $r = r - 1$
4: else
5:     if $historical$ then
6:         Obtain gradient matrix $\mathbf{G} \in \mathbb{R}^{dxk}$ from $\mathcal{G}$
7:         Generate $d$ directions as in (2.11)
8:         Ortho-normalize the directions $\mathcal{D}$
9:         while $\mathcal{D}$ does not generate $\mathbb{R}^d$ do
10:            Complement $\mathcal{D}$ with random vectors from $\mathcal{N}(0, \mathbf{I})$ and orthonormalize
11:    else
12:        Set as $\mathcal{D}$ a random orthonormal basis
13:    Update $\Xi = \mathcal{D}$
14:    if $\max_{1 \le j \le d} \left| \widetilde{\mathcal{D}}^M [G_\sigma(0 \mid x_i, \xi_j)] / L_j \right| < A$ then
15:        Decrease smoothing $\sigma = \sigma\gamma_\sigma$ and lower threshold $A = A * A_-$
16:    else if $\max_{1 \le j \le d} \left| \widetilde{\mathcal{D}}^M [G_\sigma(0 \mid, \xi_j)] / L_j \right| > B$ then
17:        Increase smoothing $\sigma = \sigma/\gamma_\sigma$ and upper threshold $B = B * B_+$
18:    else
19:        Increase lower threshold $A = A * A_+$ and decrease upper threshold $B = B * B_-$
```

3.4.1 Implementation Details

Below the table with the parameters used can be found:

parameter	m	A	B	A_-	A_+	B_-	B_+	γ_L	γ_σ	r	ρ	k	$maxiter$	ε
value	5	0.1	0.9	0.95	1.02	0.98	1.01	0.9	0.9	10	0.01	50	10000	10^{-8}

Table 3.1: Table of the parameters used in 8 and 9.

The strength of this algorithm lies also in the fact that varying these parameters, keep almost the same performance on average, with some fluctuations depending on the function that is to be optimised.

3.5 Theoretical Results

3.5.1 Background notions and results

In this section the theoretical background needed in order to prove convergence results for the algorithm is presented.

Definition 3.1 (Lipschitz Continuos Gradient). *Let $F \in \mathcal{C}^1(\mathbb{R})$. F has Lipschitz Continuos Gradient if there exists $L > 0$ such that $\|\nabla F(x+\xi) - \nabla F(x)\| \leq L\,\|\xi\|$, $\forall x, \xi \in \mathbb{R}^d$.*

Definition 3.2 (Strong Convexity). *Let $F \in \mathcal{C}^1(\mathbb{R}^d)$. F is a strongly convex function if there exists a positive number τ such that $\forall x, \xi \in \mathbb{R}^d, F(x+\xi) \geq F(x) + \langle \nabla F(x), \xi \rangle + \frac{\tau}{2}\|\xi\|^2$. τ is called the convexity parameter of F.*

Theorem 3.5.1 (Gauss-Hermite quadrature error). *Let $\widetilde{\mathcal{D}}^M$ and $\mathcal{D}$ as in 2.21 and 2.19, respectively. Then the error of the one-dimensional GH quadrature is*

$$\left|\left(\widetilde{\mathcal{D}}^M - \mathcal{D}\right)[G_\sigma]\right| \leq C\frac{M!\sqrt{\pi}}{2^M(2M)!}\sigma^{2M-1}, \tag{3.3}$$

with $C > 0$ a constant independent of M and σ.

Proposition 3.5.2 (Lemma 3 - [4]). *If F ha Lipschitz Continuos Gradient with constant L, then*

$$\|\nabla F_\sigma(x) - \nabla F(x)\| \leq \frac{\sigma}{2}L(d+3)^{3/2} \tag{3.4}$$

For $F \in C^2(\mathbb{R}^d)$ such that $\|\nabla^2 F(x+\xi) - \nabla^2 F(x)\| \leq L\,\|\xi\|$, $\forall x, \xi \in \mathbb{R}^d$ with constant L, we can guarantee that

$$\|\nabla F_\sigma(x) - \nabla F(x)\| \leq \frac{\sigma^2}{6}L(d+4)^2 \tag{3.5}$$

Proposition 3.5.3 (Theorem 2.1.5 - [22]). *Let $x, y \in R^n$ and $\alpha \in [0,1]$. The following expressions are equivalent to say that F has Lipschitz Continous Gradient:*

$$0 \leq F(y) - F(x) - \langle F'(x), y - x \rangle \leq \frac{L}{2} \|x - y\|^2, \tag{3.6}$$

$$F(x) + \langle F'(x), y - x \rangle + \frac{1}{2L} \left\|F'(x) - F'(y)^2\right\| \leq F(y), \tag{3.7}$$

$$\frac{1}{L} \|F'(x) - F'(y)\|^2 \leq \langle F'(x) - F'(y), x - y \rangle, \tag{3.8}$$

$$\langle F'(x) - F'(y), x - y \rangle \leq L \|x - y\|^2, \tag{3.9}$$

$$\alpha F(x) + (1-\alpha)F(y) \geq F(\alpha x + (1-\alpha)y) + \frac{\alpha(1-\alpha)}{2L} \|F'(x) - F'(y)\|^2, \tag{3.10}$$

$$\alpha F(x) + (1-\alpha)F(y) \leq F(\alpha x + (1-\alpha)y) + \alpha(1-\alpha)\frac{L}{2} \|x - y\|^2. \tag{3.11}$$

3.5.2 Convergence of the algorithm

A first result is a bound on the difference between ∇F and the DGS estimator.

Proposition 3.5.4. *Let $\Xi = (\xi_1, \ldots, \xi_d)$ be a orthonormal basis of $\mathbb{R}^d$ and let F have a Lipschitz Continous Gradient with constant L. Then*

$$\left\|\widetilde{\nabla}^M_{\sigma,\Xi}[F](x) - \nabla F(x)\right\|^2 \leq \frac{2C^2 \pi d (M!)^2}{4^M ((2M)!)^2} \sigma^{4M-2} + 32 d L^2 \sigma^2 \tag{3.12}$$

Proof. First, adapting 3.5.2 to $1-$dimensional Gaussian smoothing, for any ξ being a unit vector in $\mathbb{R}^d$, there holds

$$|\mathcal{D}\left[G_\sigma(0 \mid x, \xi)\right] - \nabla_\xi F(x)| \leq 4\sigma L, \tag{3.13}$$

where $d = 1$, being a $1-$dimensional Gaussian smoothing.

From (3.3) and (3.13), we have

$$
\begin{aligned}
\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-\nabla_{\xi_i} F(x)\right|^2 &\leq 2\left|\left(\widetilde{\mathcal{D}}^M-\mathcal{D}\right)\left[G_\sigma\right]\right|^2+2\left|\mathcal{D}\left[G_\sigma\right]-\nabla_{\xi_i} F(x)\right|^2 \\
&\leq \frac{2C^2(M!)^2\pi}{4^M((2M)!)^2}\sigma^{4M-2}+32\sigma^2L^2, \forall i \in\{1, \ldots, d\}
\end{aligned} \tag{3.14}
$$

Summing 3.14 from $i=1$ to d, given that the bound does not depend on i, gives 3.12.

□

The next result aims to give a bound on the difference in the value of the function at an iteration t with respect to its global minima. For the following results the stepsize λ and the smoothing parameter σ are assumed to be constant.

Theorem 3.5.5 (Bound on the descent). *Let F be a strongly convex function with Lipschitz Continuos Gradient, L being its Lipschitz constant, x^* the global minimizer of F and the sequence $\{x_t\}_{t\geq 0}$ be generated by Algorithm 8 with $\lambda = \frac{1}{8L}$. Then, for any $t \geq 0$,*

$$
F\left(x_t\right)-F\left(x^*\right) \leq \frac{1}{2} L\left[\delta_\sigma+\left(1-\frac{\tau}{16L}\right)^t\left(\left\|x_0-x^*\right\|^2-\delta_\sigma\right)\right]. \tag{3.15}
$$

Here,

$$
\delta_\sigma=\left(\frac{128}{\tau^2}+\frac{16}{\tau L}\right)L^2d\sigma^2+\left(\frac{8}{\tau^2}+\frac{1}{2\tau L}\right)\frac{C^2(M!)^2\pi d}{4^M((2M)!)^2}\sigma^2. \tag{3.16}
$$

Proof. Firstly, let's find an upper bound for $\left\|\widetilde{\nabla}_{\sigma,\Xi}^M[F](x)\right\|$. Recall that

$$
\left\|\widetilde{\nabla}_{\sigma,\Xi}^M[F](x)\right\|^2=\sum_{i=1}^d\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]\right|^2. \tag{3.17}
$$

Each term of the sum defined above can be bounded as

$$
\begin{aligned}
\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]\right|^2 &= \left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-0\right|^2 \\
&\leq 2\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-\mathcal{D}\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]\right|^2+ \\
&\quad 2\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-0\right|^2 \\
&= 2\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-\mathcal{D}\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]\right|^2+ \\
&\quad 2\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]\right|^2 \\
&\leq \frac{2C^2(M!)^2\pi}{4^M((2M)!)^2}\sigma^{4M-2}+4\left|\widetilde{\mathcal{D}}^M\left[G_\sigma\left(0 \mid x, \xi_i\right)\right]-\nabla_{\xi_i}F(x)\right|^2+ \\
&\quad 4\left|\nabla_{\xi_i}F(x)\right|^2 \\
&\leq \frac{2C^2(M!)^2\pi}{4^M((2M)!)^2}\sigma^{4M-2}+64\sigma^2L^2+4\left|\nabla_{\xi_i}F(x)\right|^2,
\end{aligned}
\tag{3.18}
$$

where (3.13) and (3.3) were used. This bound can then used in (3.17), in order to obtain

$$
\left\|\widetilde{\nabla}^M_{\sigma,\Xi}[F](x)\right\|^2 \leq \frac{2C^2(M!)^2d\pi}{4^M((2M)!)^2}\sigma^{4M-2}+4\sum_{i=1}^d\left|\nabla_{\xi_i}F(x)\right|^2+64d\sigma^2L^2. \tag{3.19}
$$

Now, let's define that $r_t=\|x_t-x^*\|$. Then

$$
\begin{aligned}
r_{t+1}^2 &= \left\|x_t-\lambda\widetilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right)-x^*\right\|^2 \\
&= r_t^2-2\lambda\left\langle\widetilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right), x_t-x^*\right\rangle+\lambda^2\left\|\widetilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right)\right\|^2 \\
&\overset{(3.19)}{\leq} r_t^2-2\lambda\left\langle\widetilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right)-\nabla F\left(x_t\right), x_t-x^*\right\rangle-2\lambda\left\langle\nabla F\left(x_t\right), x_t-x^*\right\rangle \\
&\quad +4\lambda^2\sum_{i=1}^d\left|\nabla_{\xi_i}F\left(x_t\right)\right|^2+64\lambda^2L^2d\sigma^2+\frac{2C^2\lambda^2(M!)^2\pi d}{4^M((2M)!)^2}\sigma^{4M-2}.
\end{aligned}
\tag{3.20}
$$

Now, the aim is to bound the right side of (3.20). Given the strong convexity of F,

$$-2\lambda\left\langle \nabla F\left(x_t\right), x_t - x^*\right\rangle \leq 2\lambda F\left(x^*\right) - 2\lambda F\left(x_t\right) - \lambda\tau\left\|x^* - x_t\right\|^2. \tag{3.21}$$

Now, given that

$$(a+b)\cdot(a+b) = \|a\|^2 + 2a\cdot b + \|b\|^2 \geq 0 \Longrightarrow \|a\|^2 + \|b\|^2 \geq -2a\cdot b,$$

then it holds:

$$\begin{aligned}
&-2\lambda\left\langle \tilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right) - \nabla F\left(x_t\right), x_t - x^*\right\rangle \\
&\leq \frac{2\lambda}{\tau}\left\|\tilde{\nabla}^M_{\sigma,\Xi}[F]\left(x_t\right) - \nabla F\left(x_t\right)\right\|^2 + \frac{\lambda\tau}{2}\left\|x_t - x^*\right\|^2 \\
&\overset{(3.12)}{\leq} \frac{4\lambda}{\tau}\cdot\frac{C^2\pi d(M!)^2}{4^M((2M)!)^2}\sigma^{4M-2} + \frac{64\lambda}{\tau}L^2 d\sigma^2 + \frac{\lambda\tau}{2}\left\|x_t - x^*\right\|^2.
\end{aligned} \tag{3.22}$$

Using a bound for functions with Lipschitz Continuos Gradients from 3.5.3, it gives

$$4\lambda^2\sum_{i=1}^{d}\left|\nabla_{\xi_i}F\left(x_t\right)\right|^2 = 4\lambda^2\left\|\nabla F\left(x_t\right)\right\|^2 \leq 8\lambda^2 L\left(F\left(x_t\right) - F\left(x^*\right)\right). \tag{3.23}$$

Now, joining (3.20) and (3.23), it holds

$$\begin{aligned}
r_{t+1}^2 \leq\, & r_t^2 - \left(2\lambda - 8\lambda^2 L\right)\left(F\left(x_t\right) - F\left(x^*\right)\right) \\
&+ \left(\frac{64\lambda}{\tau} + 64\lambda^2\right)L^2 d\sigma^2 + \left(\frac{4\lambda}{\tau} + 2\lambda^2\right)\frac{C^2(M!)^2\pi d}{4^M((2M)!)^2}\sigma^{4M-2}.
\end{aligned} \tag{3.24}$$

Given the strong convexity of F and that $\lambda^* = \frac{1}{8L}$ is the maximum for $8\lambda^2 L - 2\lambda$, we have that

$$-\left(2\lambda - 8\lambda^2 L\right)\left(F\left(x_t\right) - F\left(\boldsymbol{x}^*\right)\right) \leq \frac{\tau}{16L}\left\|x_t - x^*\right\|^2 \tag{3.25}$$

Let's now assume that $\sigma < 1$. Then, from (3.24), (3.25) and (3.16), holds that

$$r_{t+1}^2 - \delta_\sigma \leq \left(1 - \frac{\tau}{16L}\right)\left(r_t^2 - \delta_\sigma\right), \tag{3.26}$$

which gives

$$r_t^2 - \delta_\sigma \leq \left(1 - \frac{\tau}{16L}\right)^t \left(r_0^2 - \delta_\sigma\right). \tag{3.27}$$

Now, since for F holds $F\left(x_t\right) - F\left(x^*\right) \leq \frac{1}{2}L\left\|x_t - x^*\right\|^2$, because of (3.6), the conclusion is achieved.

$\square$

Thanks to the theorem above, it can be proved the global linear rate of convergence with ASHGF, which is to be expected in the case of strongly convex functions. In order to prove it, let $\varepsilon > 0$. Then to guarantee $F\left(x_t\right) - F\left(x^*\right) \leq \varepsilon$, the smoothing parameter has to be such that $\sigma \leq O\left(\sqrt{\frac{\varepsilon}{d}}\right)$ and the number of iterations have to be $t = O(\log\frac{1}{\varepsilon})$, which yields the result.

It is relevant to note that the number of iterations required by Algorithm 8 is independent of the dimensionality of the problem considered.

4
Numerical Results

4.1 Minimization of functions

The first set of tests used to verify the goodness of the proposed algorithm focused on a collection of functions taken from [15] and [16], some characterized by deep valleys, others by extended plateaus or the presence of many local minima. The optimization problems were set at three different dimensions: 10, 100, 1000. Moreover ES, SGES, ASEBO and ASGF were tested alongside ASHGF, to show how well it would score against SOTA available algorithms. It appears that all code for the algorithms, excluding ASEBO which was available through the GitHub page of the author [*], were written following exactly the related papers. Also, the implementation of ASGF is different from the one presented in [13]: at one step the algorithm should have tried to estimate the value of the gradient along the direction of the previous estimated gradient, increasing the number of quadrature points until the difference between the last estimation and the previous one was less then a threshold, meanwhile in this comparison it was kept a fixed number of quadrature point. This was done for two main reasons: decrease the computational time required by the algorithm to be executed; ease the comparison of the number of function evaluations required by the algorithms.

For ASEBO, ES and SGES the step-size was fixed to 10^{-4}, the smoothing parameter to 10^{-4}.

Now, to test the performance and efficiency of the algorithms, performance and data profiles

[*]https://github.com/jparkerholder/ASEBO

are used in this work, as defined in [23]. Let's call f_L the smallest value of f obtained by any solver for each function within μ_L function evaluations, the number of function evaluations allowed during the execution of the solvers; thus we can define:

Definition 4.1 (Convergence test). *Let $p \in \mathcal{P}$ be a problem, $s \in \mathcal{S}$ a solver and μ_L a fixed number of function evaluations. A solver is said to satisfy the convergence test if holds*

$$F_{x_k} \leq F_L + \tau(F(x_0) - F_L), \tag{4.1}$$

with $0 < \tau < 1$ a parameter indicating the tolerance of distance from F_L, which is the best objective function value achieved among all the solvers for problem p.

Then it can be defined

Definition 4.2 (Performance measure). *Given a problem $p \in \mathcal{P}$ and a solver $s \in \mathcal{S}$, fixed a number of function evaluations L. We define the performance measure of s for problem p the number of function evaluations to satisfy the convergence test for a given tolerance τ and computational budget μ_L. If the test is not satisfied, $t_{p,s}$ is set to ∞.*

Thus the following quantities can be defined:

Definition 4.3 (Performance and data profiles). *Let $s \in \mathcal{S}$ be a solver and μ_L fixed a number of function evaluations. Its performance and data profiles are, respectively,*

$$\rho_s(\alpha) = \frac{1}{|P|} \left| \left\{ p \in P : \frac{t_{p,s}}{\min\{t_{p,s'} : s' \in S\}} \leq \alpha \right\} \right| \tag{4.2}$$

$$d_s(\alpha) = \frac{1}{|P|} \left| \{ p \in P : t_{p,s} \leq \alpha\,(n_p + 1) \} \right|. \tag{4.3}$$

Note that n_p is the number of variables in $p \in \mathcal{P}$.

Ackley	$F(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + \exp(1)$
Almost Perturbed Quadratic	$F(x) = \sum_{i=1}^{n} i x_i^2 + \frac{1}{100}(x_1 + x_n)^2$
Cube	$F(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} 100\left(x_i - x_{i-1}^3\right)^2$
Extended Feudenstein and Roth	$F(x) = \sum_{i=1}^{n/2}\left[(-13 + x_{2i-1} + ((5 - x_{2i})x_{2i} - 2)x_{2i})^2 + (-29 + x_{2i-1} + ((x_{2i} + 1)x_{2i} - 14)x_{2i})^2\right]$
Extended Hiebert	$F(x) = \sum_{i=1}^{n/2}\left[(x_{2i-1} - 10)^2 + (x_{2i-1}x_{2i} - 50000)^2\right]$
Extended Himmelblau	$F(x) = \sum_{i=1}^{n/2}\left[\left(x_{2i-1}^2 + x_{2i} - 11\right)^2 + \left(x_{2i-1} + x_{2i}^2 - 7\right)^2\right]$
Extended PSC1	$F(x) = \sum_{i=1}^{n/2}\left[\left(x_{2i-1}^2 + x_{2i}^2 + x_{2i-1}x_{2i}\right)^2 + \sin^2(x_{2i-1}) + \cos^2(x_{2i})\right]$
Extended Rosenbrock	$F(x) = \sum_{i=1}^{n/2} 100\left(x_{2i} - x_{2i-1}^2\right)^2 + (1 - x_{2i-1})^2$
Extended Tridiagonal1	$F(x) = \sum_{i=1}^{n-1}\left[(x_i + x_{i+1} - 3)^2 + (x_i - x_{i+1} + 1)^4\right]$
Extended Tridiagonal2	$F(x) = \sum_{i=1}^{n-1}\left[(x_i x_{i+1} - 1)^2 + 0.1(x_i + 1)(x_{i+1} + 1)\right]$
Extended Trigonometric	$F(x) = \sum_{i=1}^{n}\left[\left(n - \sum_{j=1}^{n}\cos x_j\right) + i(1 - \cos x_i) - \sin x_i\right]^2$
Extended White and Holst	$F(x) = \sum_{i=1}^{n/2} 100\left(x_{2i} - x_{2i-1}^3\right)^2 + (1 - x_{2i-1})^2$
Generalized Quartic	$F(x) = \sum_{i=1}^{n-1}\left[x_i^2 + (x_{i+1} + x_i^2)^2\right]$
Generalized Rosenbrock	$F(x) = \sum_{i=1}^{n-1} c\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, c = 100$
Generalized White and Holst	$F(x) = \sum_{i=1}^{n-1}\left[c\left(x_{i+1} - x_i^3\right)^2 + (1 - x_i)^2\right], c = 100$
Griewank	$F(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Levy	$F(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}\left[(w_i - 1)^2\left[1 + 10\sin^2(\pi w_i + 1)\right]\right] + (w_d - 1)^2\left[1 + \sin^2(2\pi w_d)\right], \quad w_i = 1 + \frac{x_i - 1}{4}, \quad \forall i = 1, \ldots, d$
Perturbed Quadratic	$F(x) = \sum_{i=1}^{n} i x_i^2 + \frac{1}{100}\left(\sum_{i=1}^{n} x_i\right)^2$
Perturbed Quadratic Diagonal	$F(x) = \left(\sum_{i=1}^{n} x_i\right)^2 + \sum_{i=1}^{n}\frac{i}{100}x_i^2$
Power	$F(x) = \sum_{i=1}^{n}(i x_i)^2$
Rastrigin	$F(x) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right]$
Schwefel	$F(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$
Sphere	$F(x) = \sum_{i=1}^{d} x_i^2$
Sum of Different Powers	$F(x) = \sum_{i=1}^{d}\lvert x_i \rvert^{i+1}$
Trid	$F(x) = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d} x_i x_{i-1}$
Zakharov	$F(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^4$

Table 4.1: Table of the functions used to test the algorithms.

The functions used for the experiment are the ones listed in the Table 4.1; they are taken both from [15] and [16]. They range from strongly convex functions to non-convex functions, plateau functions, deep valley functions and others.

4.1.1 Performance profiles

This quantity aims to describe how much the solver performs with respect to the other solvers on the given set of problems $\mathcal{P}$. It is to be noted that for α sufficiently large, $\rho_s(\alpha)$ represents the fraction of problem solved by the solver s; also, for a given α, it represents the fraction of problems with $\frac{t_{p,s}}{\min t_{p,s'}}$ bounded by α, thus a solver with a high value of $\rho_s(\alpha)$ is preferable.

As it can be seen in the following figures, with the increase of problem dimensionality, ASHGF requires increased computational budget that, when it is given, let the algorithm to perform better than the other algorithms.

Dimension 10

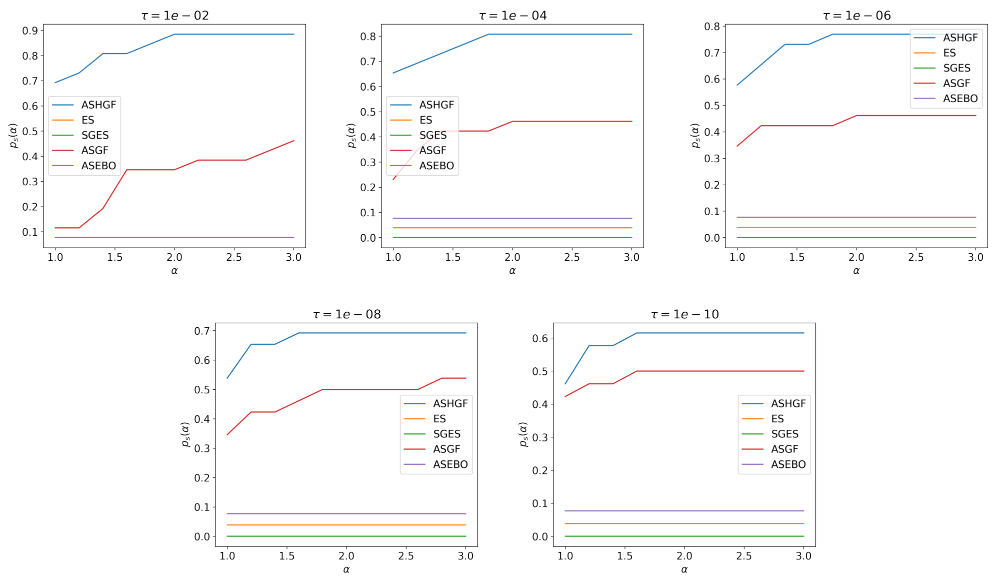

Figure 4.1: Performance profiles with $\mu_L = 10^4$

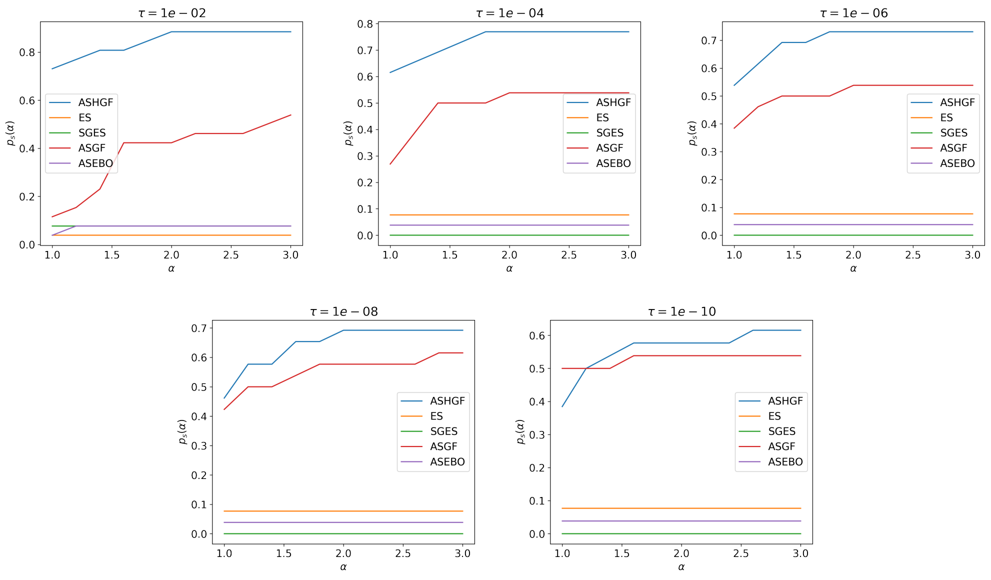

Figure 4.2: Performance profiles with $\mu_L = 10^5$

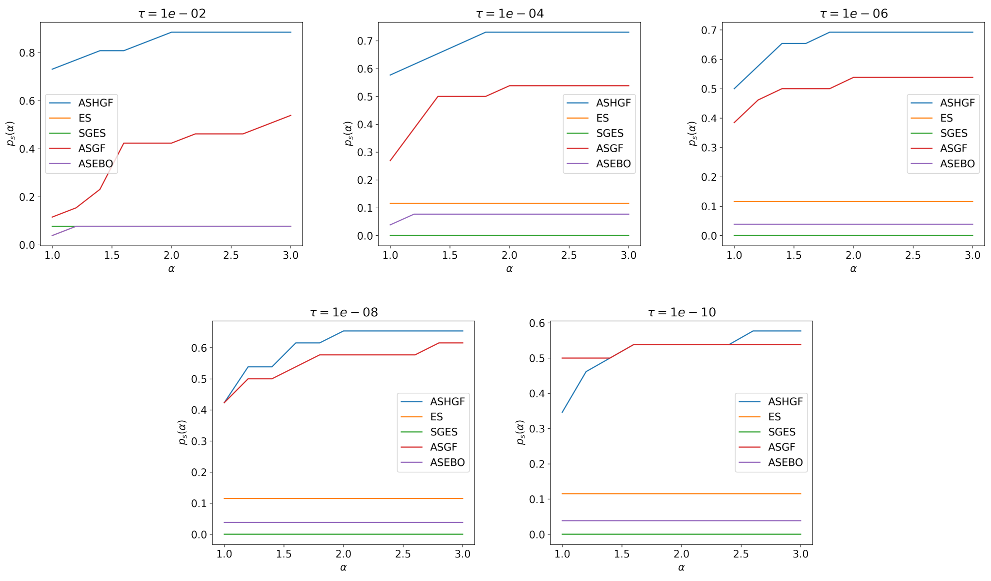

Figure 4.3: Performance profiles with $\mu_L = 10^6$

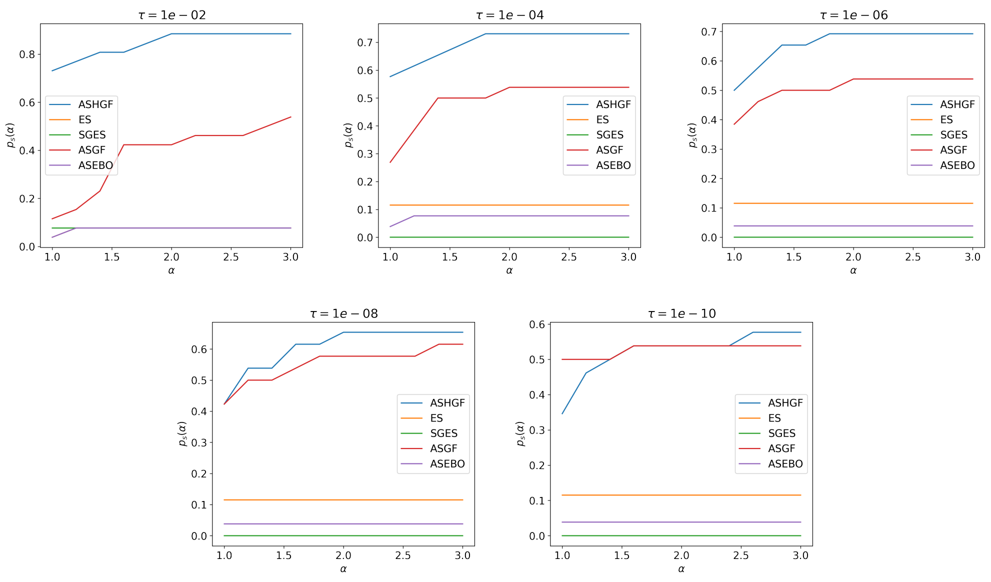

Figure 4.4: Performance profiles with $\mu_L = 10^7$

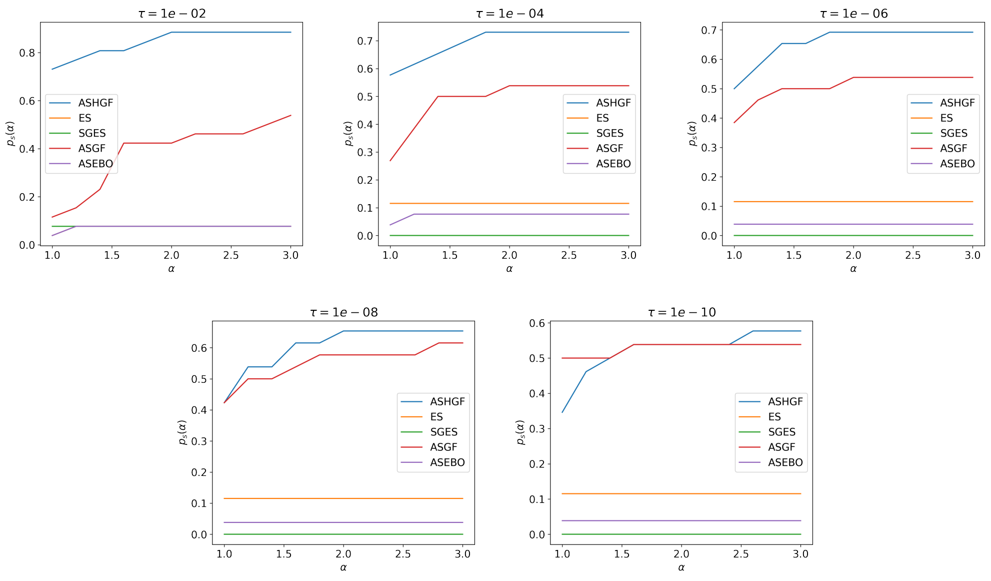

Figure 4.5: Performance profiles with $\mu_L = 10^8$

Dimension 100

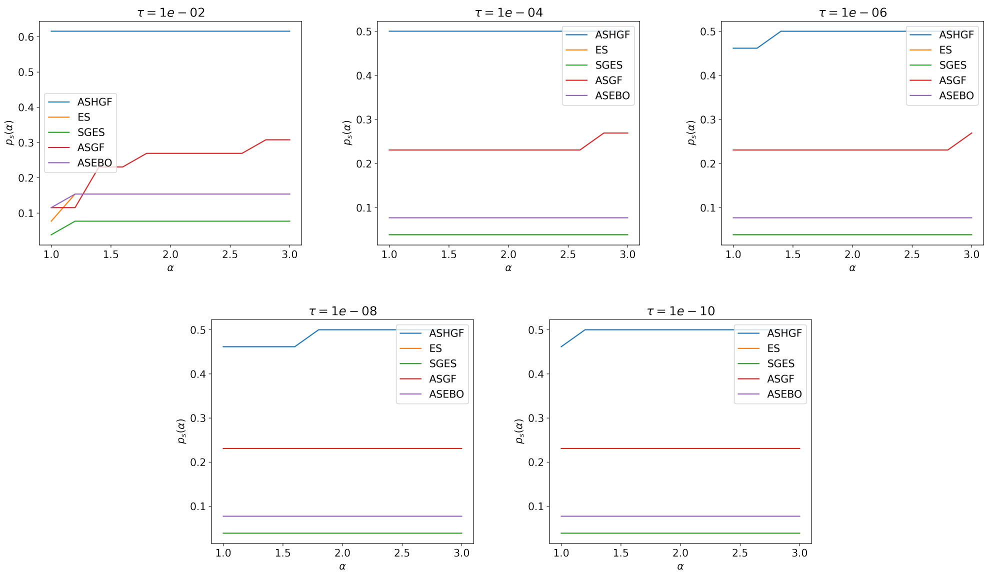

Figure 4.6: Performance profiles with $\mu_L = 10^4$

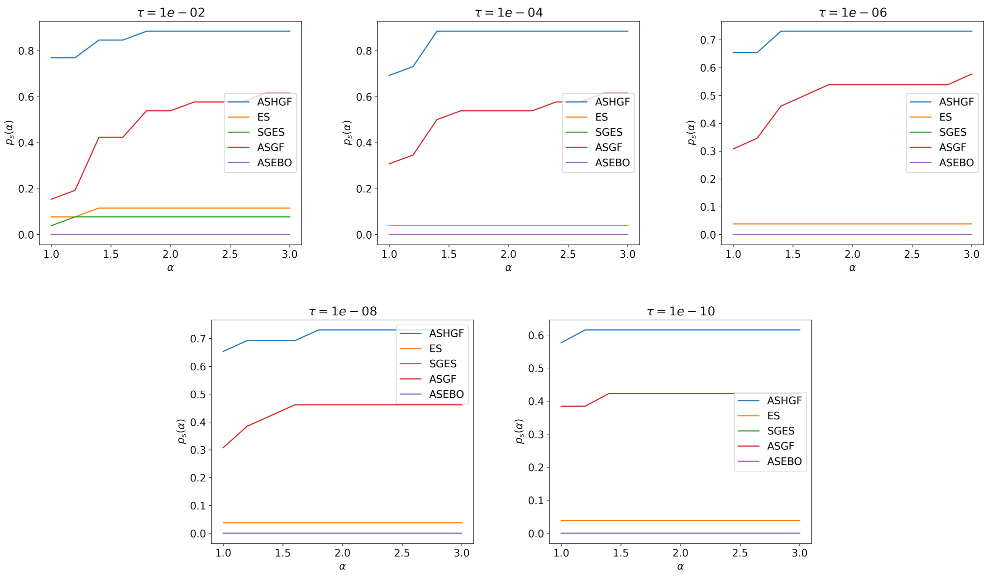

Figure 4.7: Performance profiles with $\mu_L = 10^5$

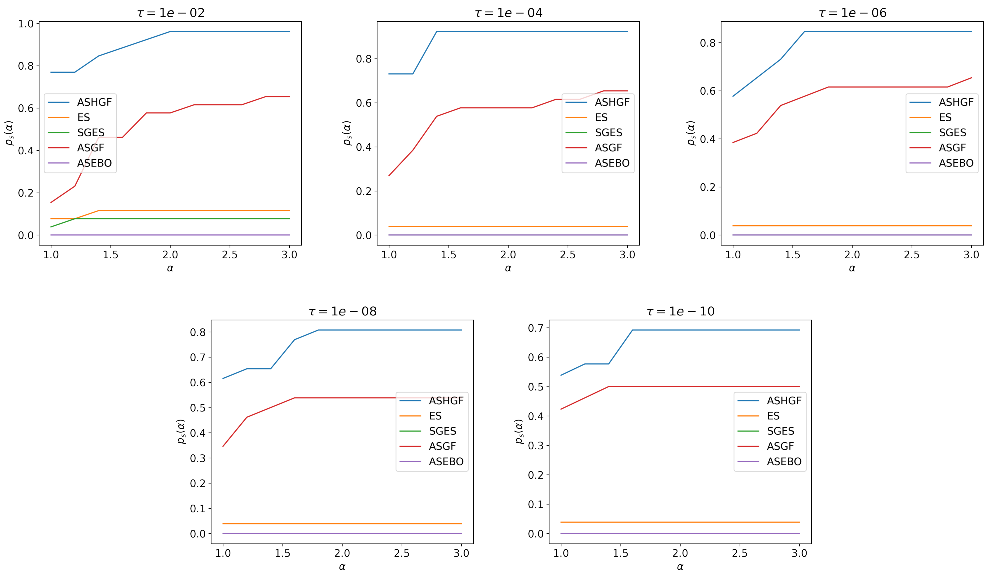

Figure 4.8: Performance profiles with $\mu_L = 10^6$

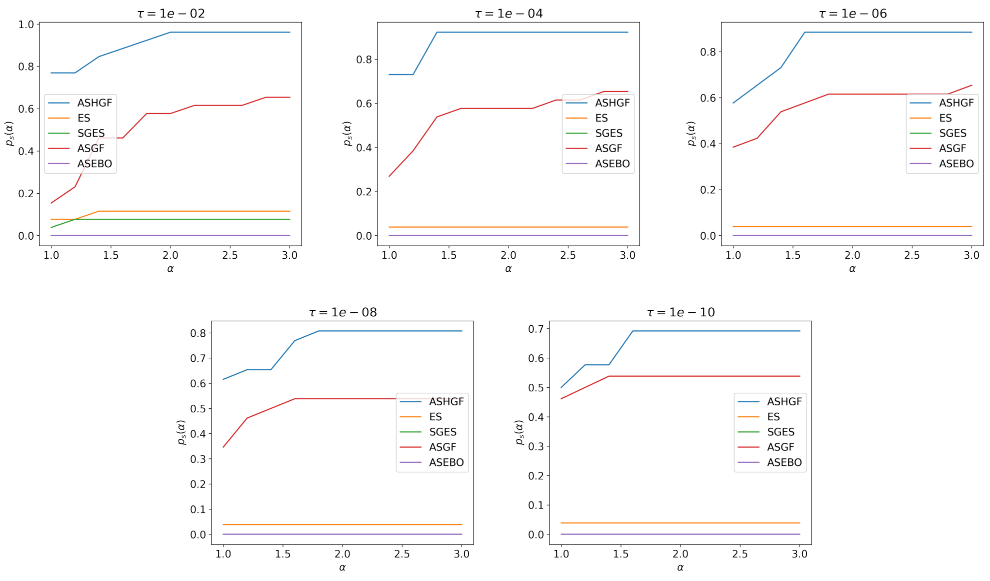

Figure 4.9: Performance profiles with $\mu_L = 10^7$

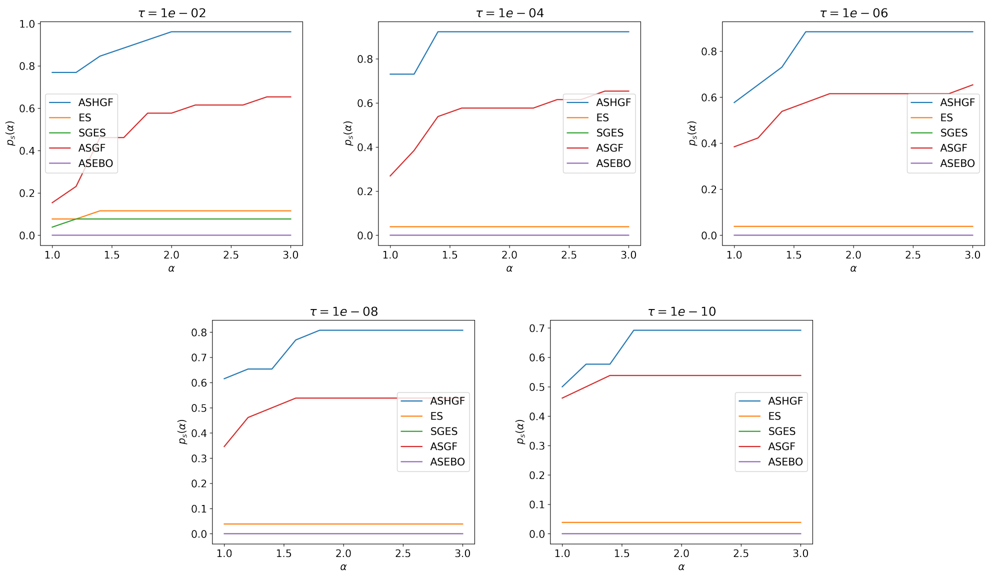

Figure 4.10: Performance profiles with $\mu_L = 10^8$

Dimension 1000

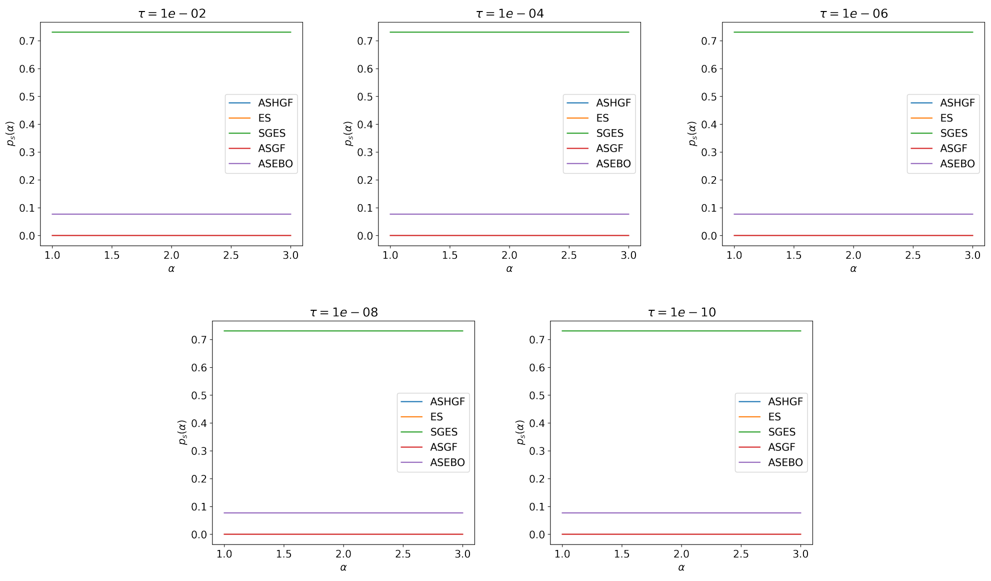

Figure 4.11: Performance profiles with $\mu_L = 10^4$

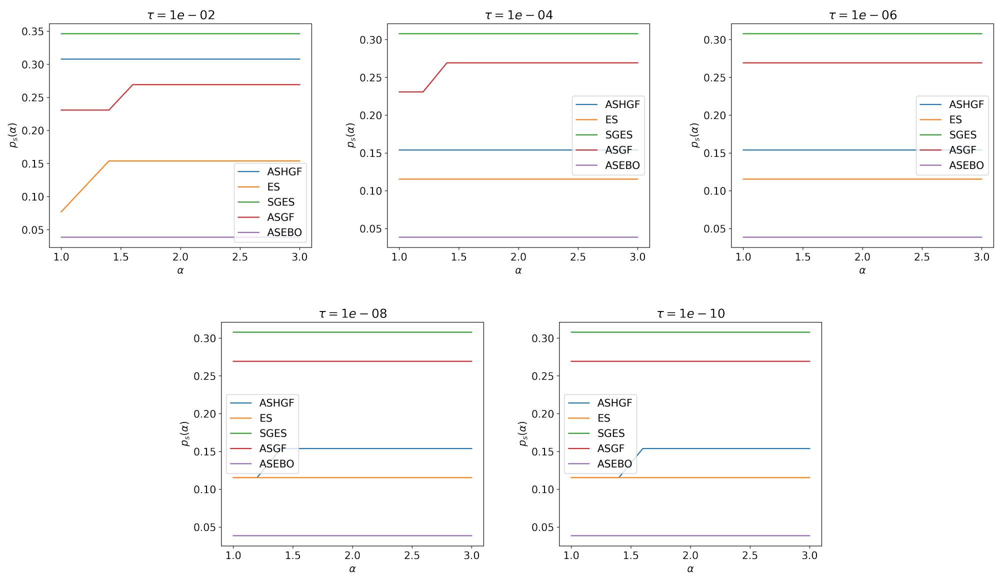

Figure 4.12: Performance profiles with $\mu_L = 10^5$

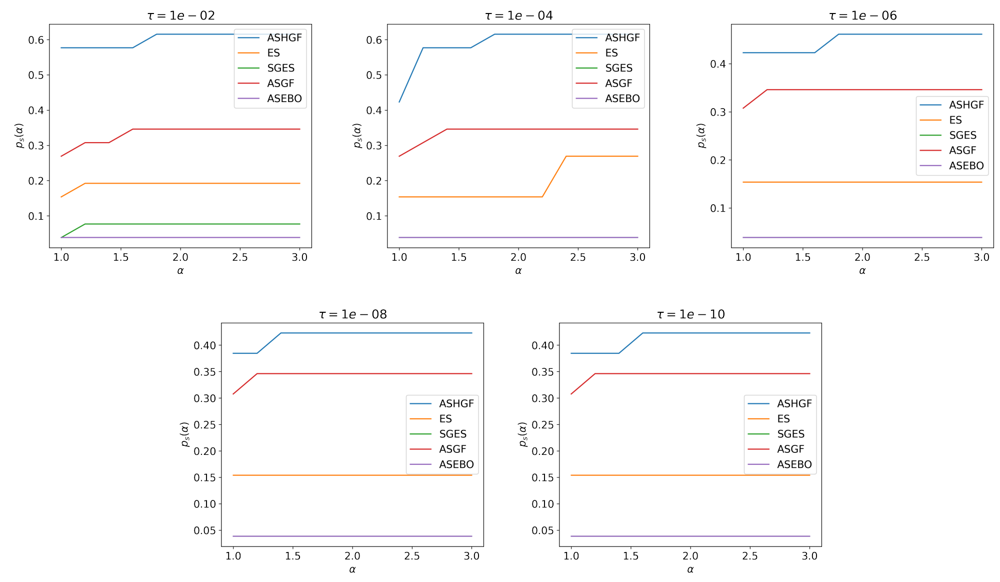

Figure 4.13: Performance profiles with $\mu_L = 10^6$

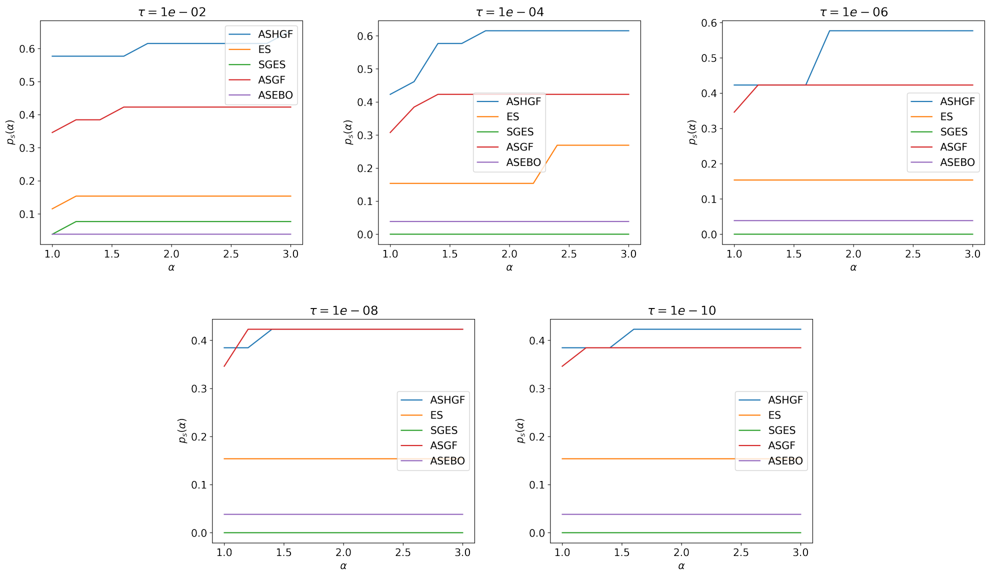

Figure 4.14: Performance profiles with $\mu_L = 10^7$

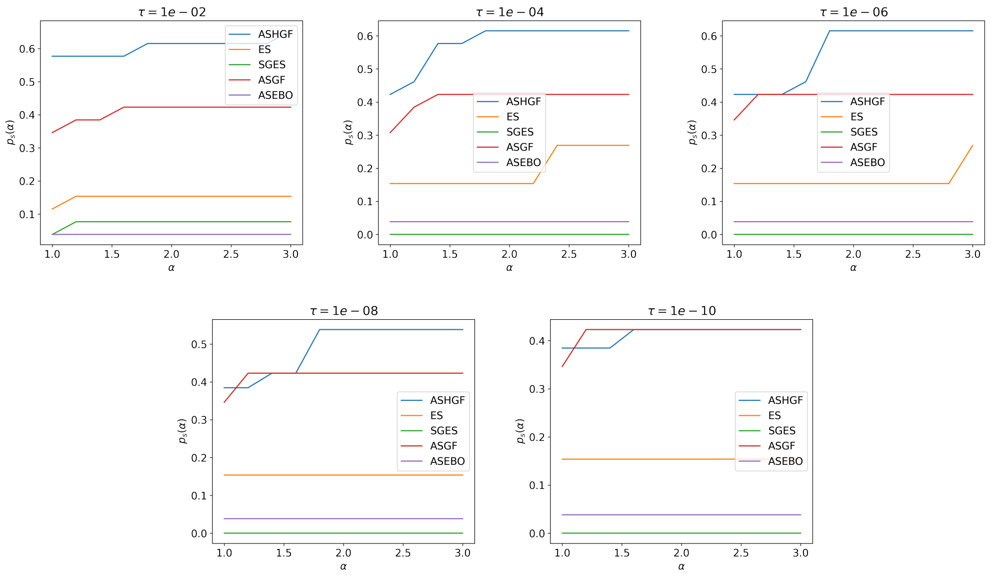

Figure 4.15: Performance profiles with $\mu_L = 10^8$

4.1.2 Data profiles

With this quantity is measured the percentage of problems solved within a tolerance τ for a certain number of function evaluations.

As it can be seen with the performance profiles, again thanks to the data profiles it can be seen that increasing the dimensionality of the problem, ASHGF requires increased budget. Given that, the algorithm shows itself to be more efficient than others given a bigger budget.

Dimension 10

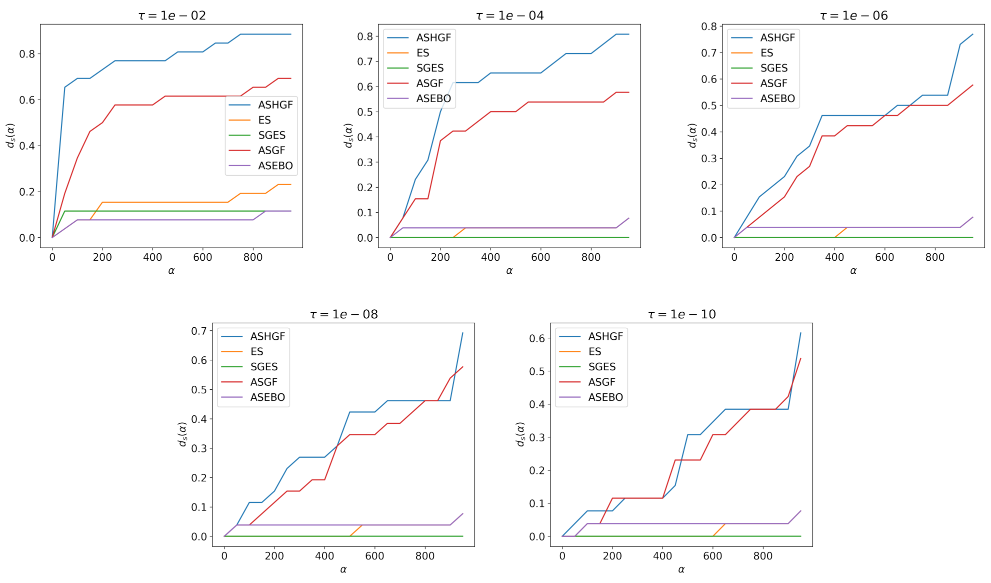

Figure 4.16: Data profiles with $\mu_L = 10^4$

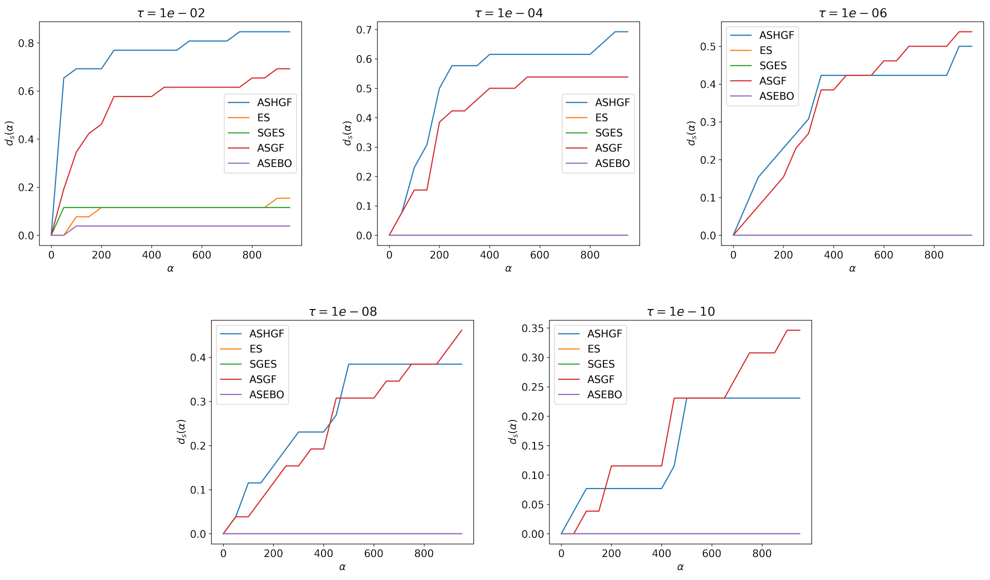

Figure 4.17: Data profiles with $\mu_L = 10^5$

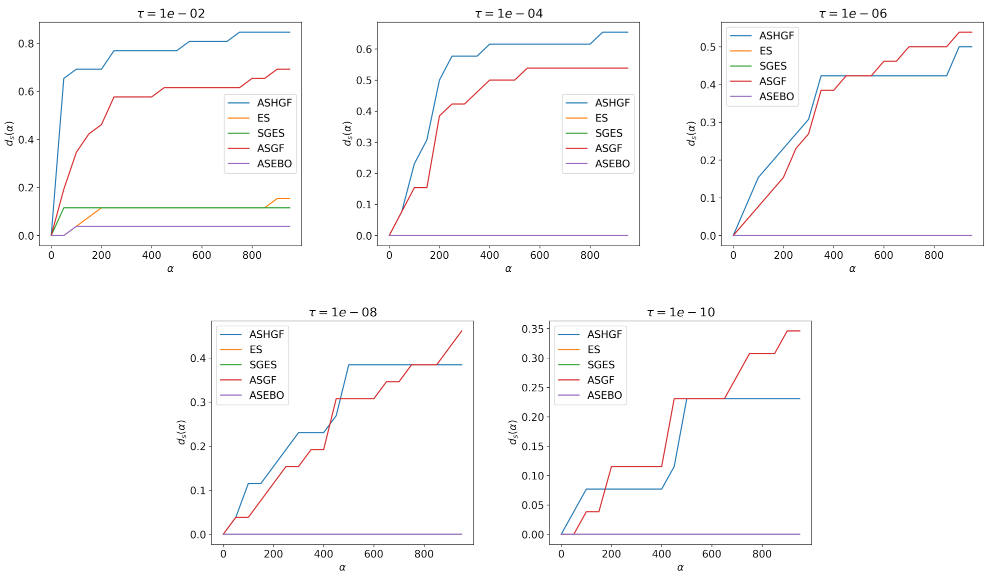

Figure 4.18: Data profiles with $\mu_L = 10^6$

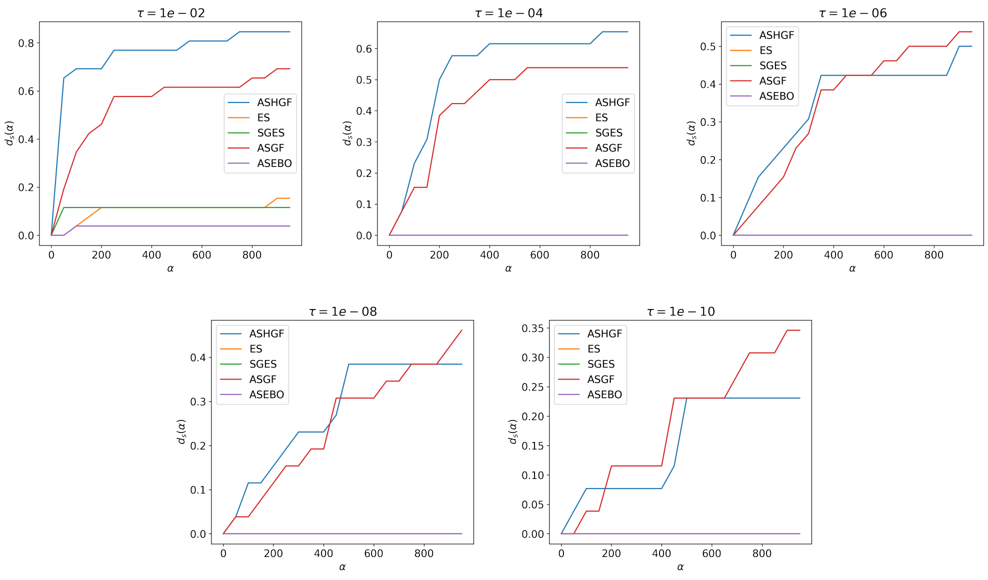

Figure 4.19: Data profiles with $\mu_L = 10^7$

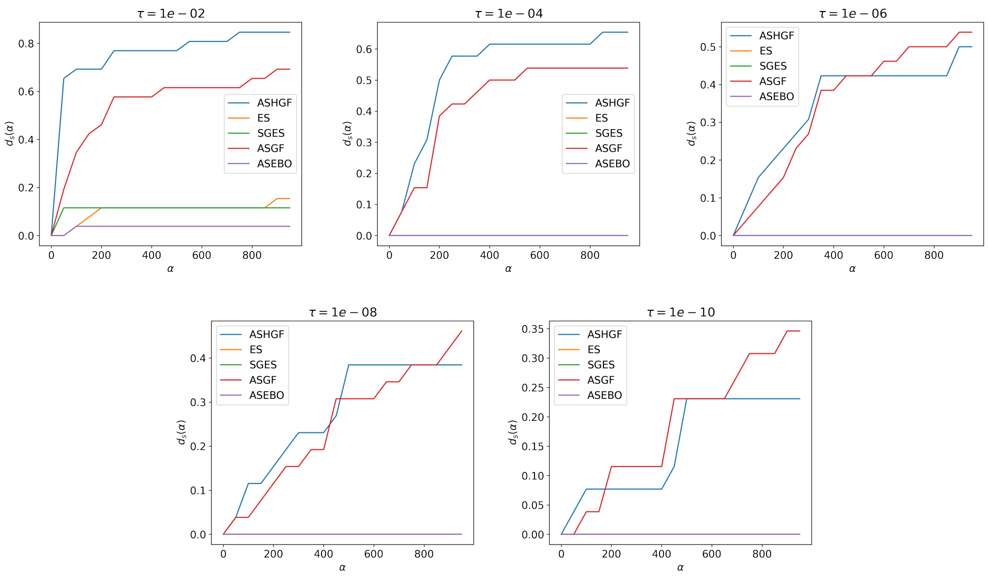

Figure 4.20: Data profiles with $\mu_L = 10^8$

Dimension 100

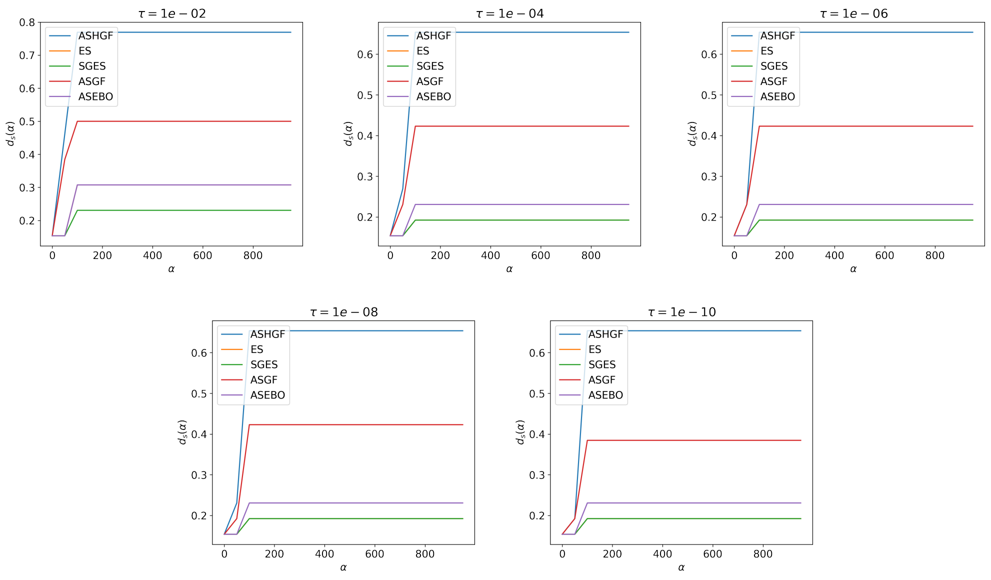

Figure 4.21: Data profiles with $\mu_L = 10^4$

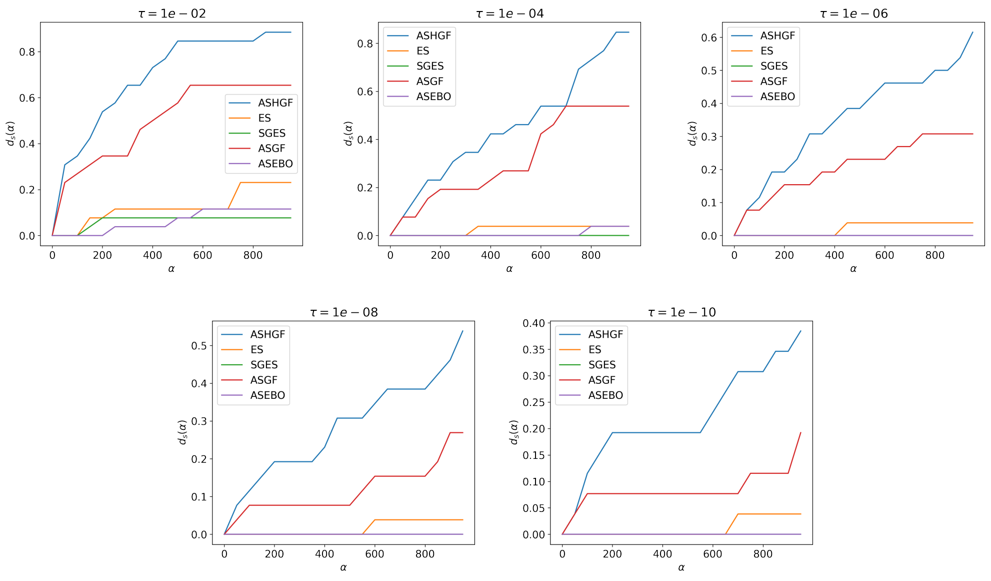

Figure 4.22: Data profiles with $\mu_L = 10^5$

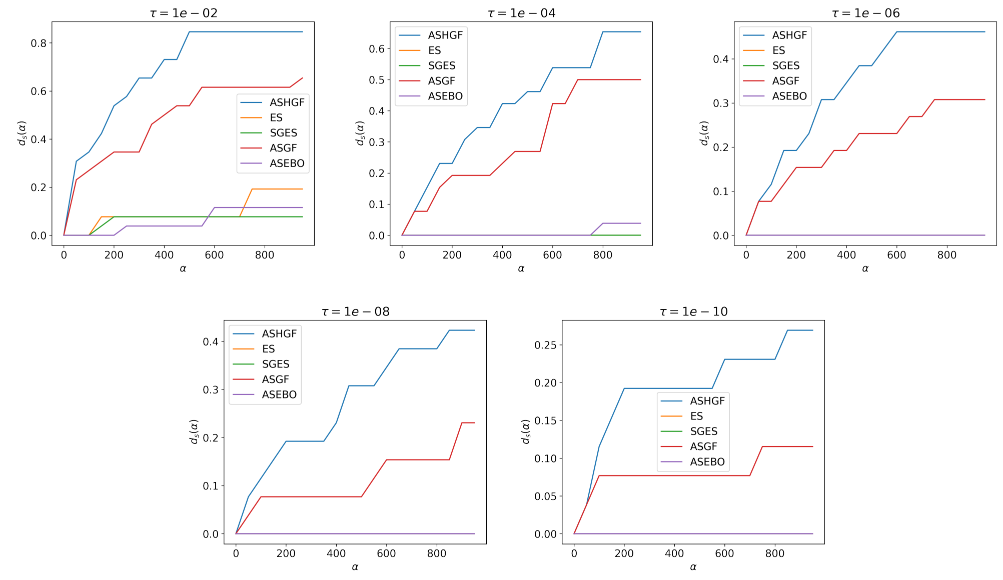

Figure 4.23: Data profiles with $\mu_L = 10^6$

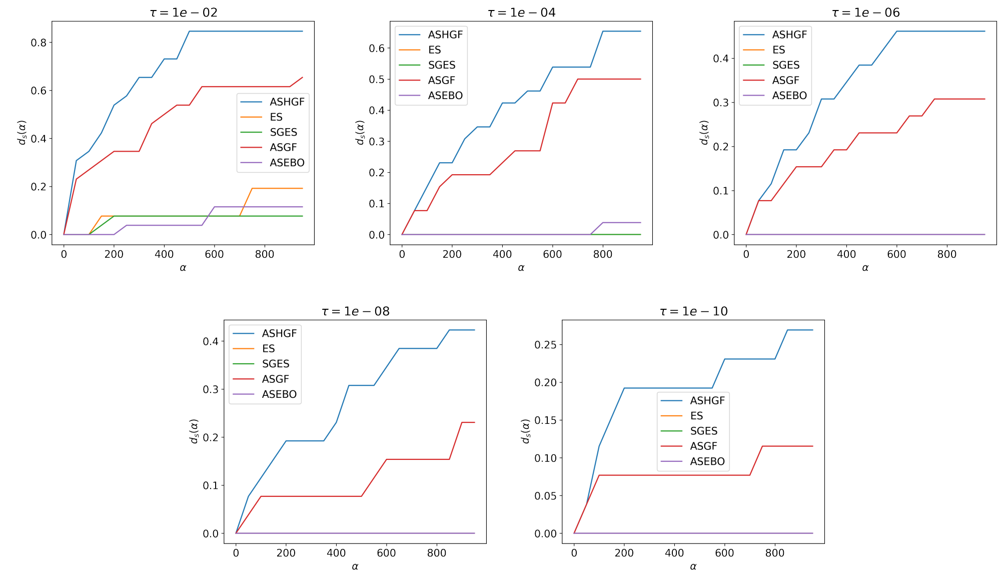

Figure 4.24: Data profiles with $\mu_L = 10^7$

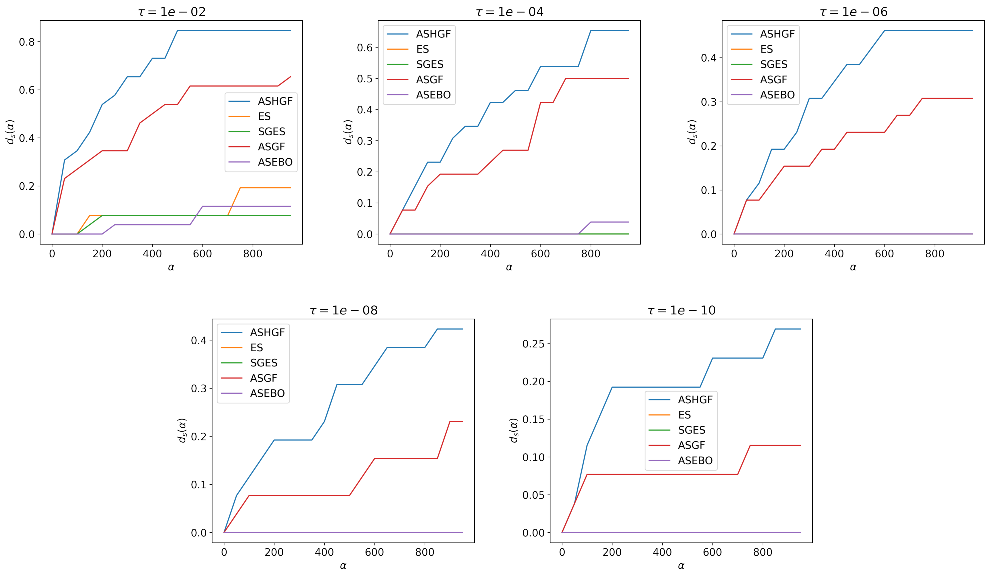
Figure 4.25: Data profiles with $\mu_L = 10^8$

Dimension 1000

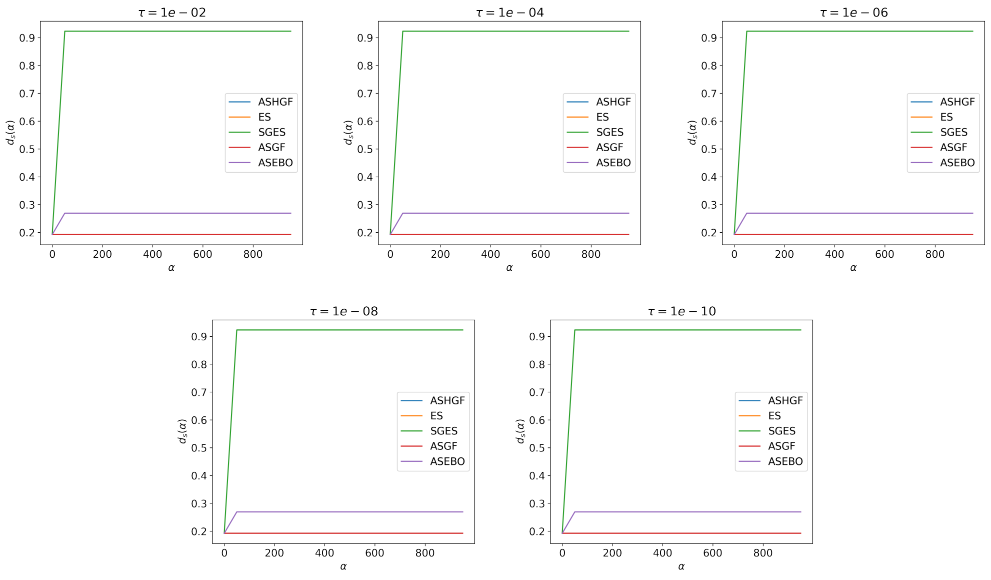
Figure 4.26: Data profiles with $\mu_L = 10^4$

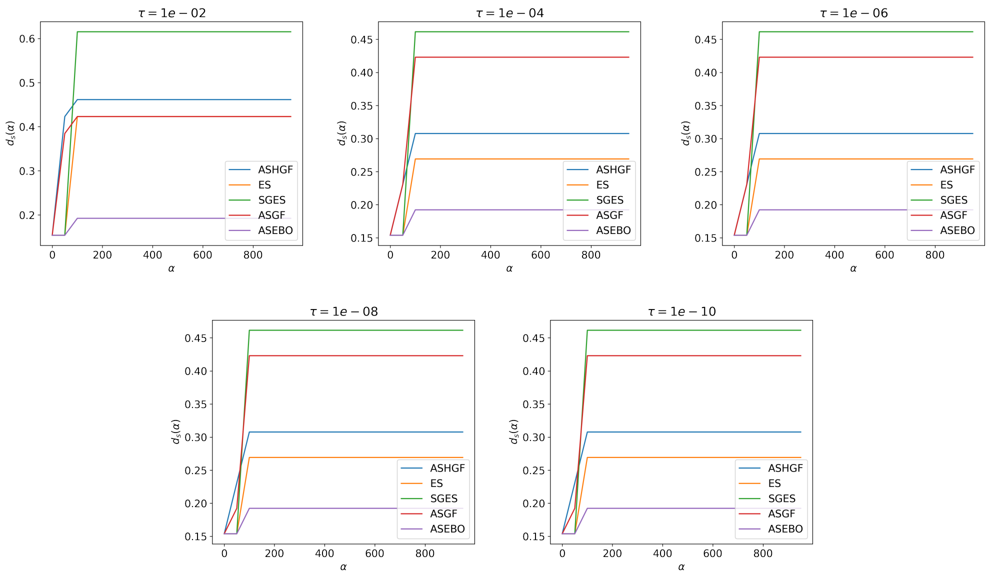

Figure 4.27: Data profiles with $\mu_L = 10^5$

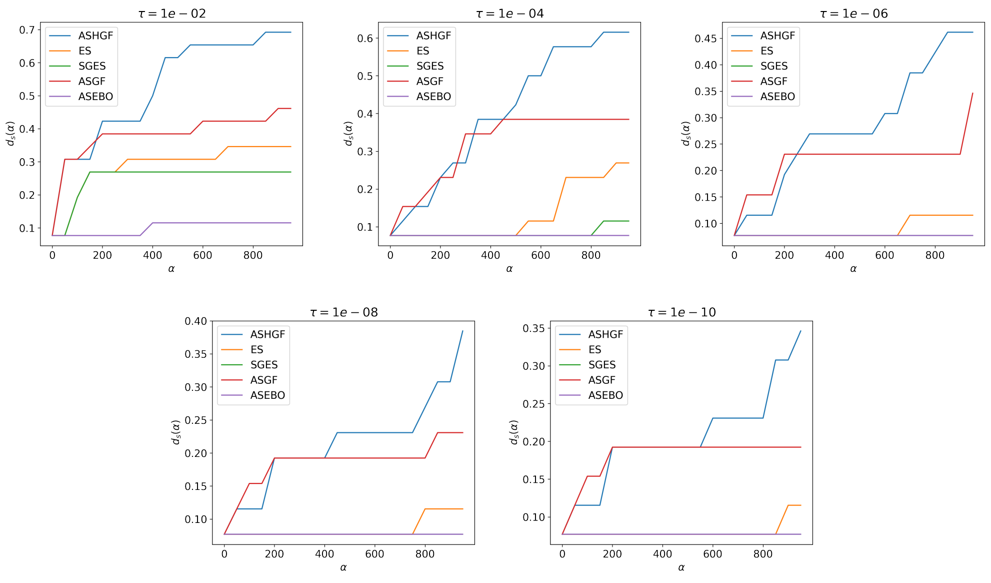

Figure 4.28: Data profiles with $\mu_L = 10^6$

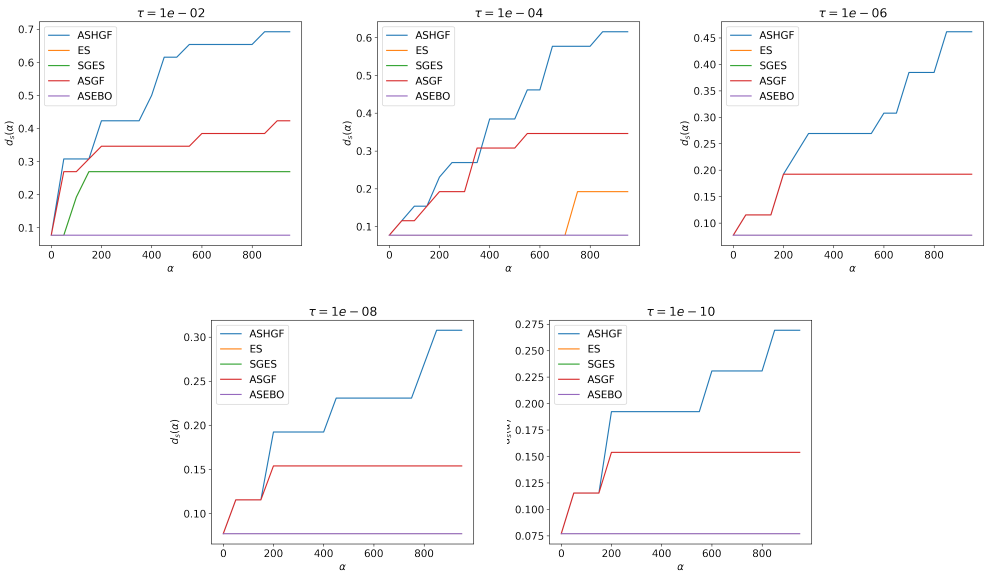

Figure 4.29: Data profiles with $\mu_L = 10^7$

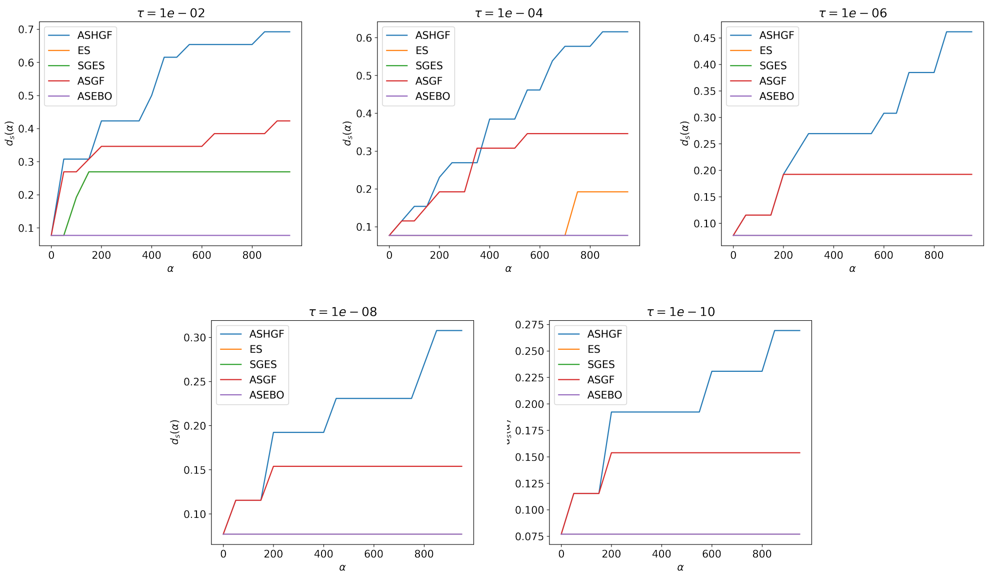

Figure 4.30: Data profiles with $\mu_L = 10^8$

4.1.3 Statistical analysis of the convergence

In order to understand the robustness of the algorithm varying the random seed, four of the previously defined functions were selected, namely: *Sphere*, *Rastrigin*, *Levy* and *Ackley*; they were optimized using ASHGF with one hundred different random seeds, the same for each function.

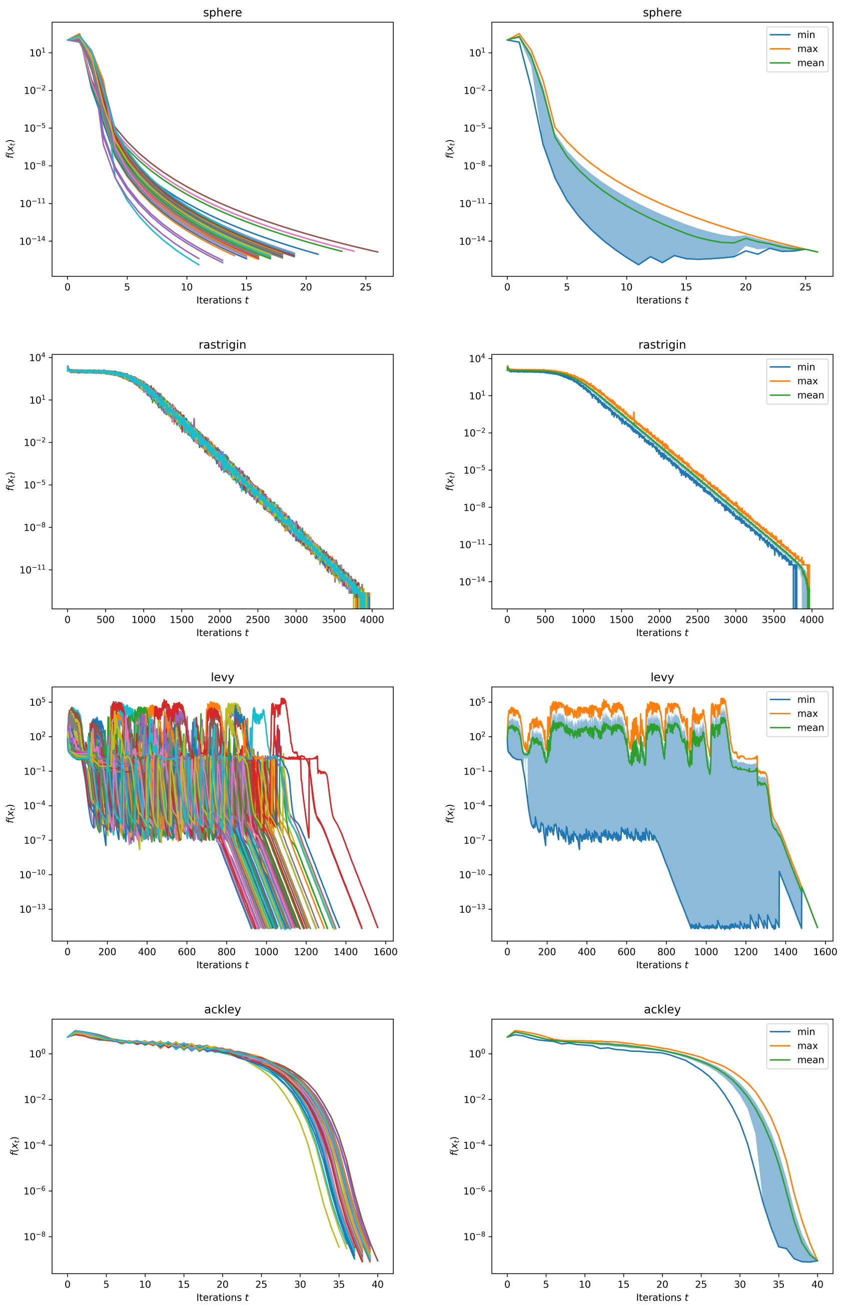

Figure 4.31: Results of the optimization of the functions *Sphere*, *Rastrigin*, *Levy* and *Ackley* on 100 different random seeds. The figures on the left show all the function decay. The figures on the right show the slowest, fastest and average decay, with the highlighted area being the standard deviation from the mean.

The algorithm converged for each seed and function. The dimension of choice was 100, being both high enough while still being feasible in terms of computational times. As expected, with *Sphere*, a strongly convex function, the algorithm converged in a very small number of iterations.

In the following figures the presented algorithm is compared in its average run on 100 random seeds, on the same functions as in 4.31, against 2.2, 3, 5 and 6.

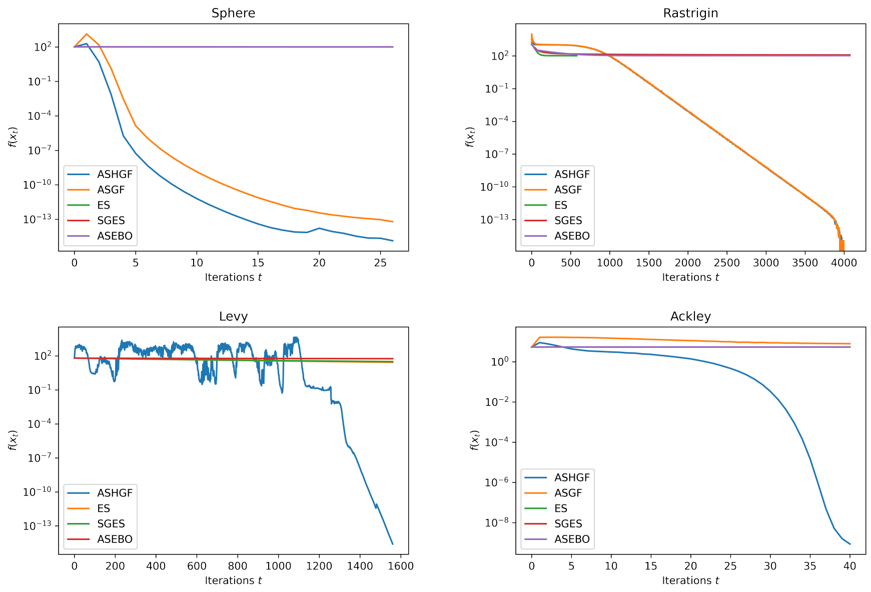

Figure 4.32: Results of the optimization of the functions *Sphere*, *Rastrigin*, *Levy* and *Ackley* on 100 different random seeds for each of the algorithms in 2. In the figures are reported the average function decay for each algorithm.

Again, like with the results of the performance and data profiles, the proposed algorithm shows to be faster with respect to the competitors, as it can be seen in 4.32. It is to be noted that in the case of *Rastrigin*, *Levy* and *Ackley* only algorithm 8 converged in the given number of iterations, 10000; furthermore, contrarily to what shown in the paper of 6, in these tests the algorithm never decreased the value of the function in the case of Levy (this is the reason why we removed it from the plot).

4.2 Reinforcement learning problems

The Algorithm 8 was tested on two Reinforcement Learning problems against the other algorithms presented in 2. The two problems are `CartPole-v1` and `Pendulum-v0`. Those were implemented using the Python library OpenAI gym [17]. As defined in (1.2), the algorithms are asked to maximise the sum of the reward of the episodes. In both graphs the horizontal axis is the number of iterations, while the vertical axis is the total reward of the point reached by the algorithm at that given iteration. Furthermore the displayed data are, for each algorithm, the average results among the same 10 random seed.

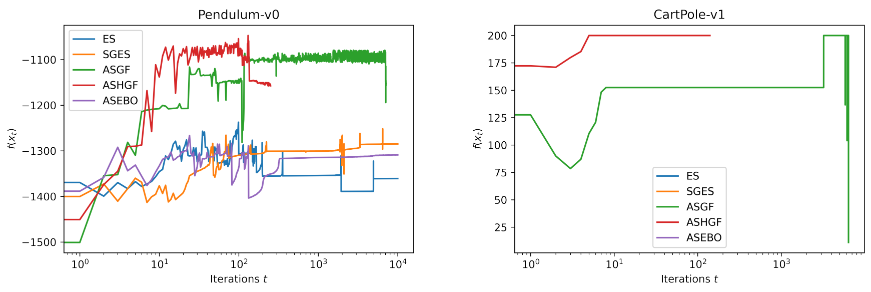

Figure 4.33: The figures show the average run of the algorithms in two different RL problems with 10 different random seeds.

As it can be seen, on average the presented algorithm achieves higher rewards and does so faster with respect to the considered state-of-the-art algorithms.

5 Conclusion

Starting with Chapter 2, this thesis presented and studied current state-of-the-art algorithms proposed and used in the field of black-box optimization. The path follows the historical progress and evolution of the field showing ideas and techniques used by authors to improve the fundamental idea of Nesterov [4], to enhance performances in optimization of non-convex functions and Reinforcement Learning problems. The main basic ideas are the following:

- to parallelize the computations made by the algorithm with the use of multi-core CPUs;
- to specialize the choice of random directions in order to take into account the exploration-exploitation trade-off, view taken from the Reinforcement Learning framework;
- to implement a novel way to estimate the gradient and adaptivity of the parameters of the algorithm itself.

In Chapter 3 a novel algorithm was presented inspired by the ones presented in the literature. Its adaptivity and use of previous gradients information allow to improve the performance with respect to state-of-the-art methods. In ASHGF the adaptivity of the parameters and the novel way of estimating the gradient, was adapted to the handling of the trade-off between exploration and exploitation.

Chapter 4 shows numerical results of ASHGF and other state-of-the-art algorithms on a widely used testbed and two Reinforcement Learning problems. The presented algorithm outperforms the others in all benchmarks: in terms of data and performance profiles, it achieves a better accuracy most of the time while requiring less function evaluations; in terms of its average behaviour, it is the one that performs better on average, outperforming all the algorithms in some functions; solving Reinforcement Learning problems it achieves higher average rewards faster than the competitors.

Further work should focus on rewriting the algorithm, and the others, in order to parallelize its execution and then testing them in a greater pool of functions and, more importantly, reinforcement learning problems, which in this case were few due to technical limitations.

References

[1] E. A. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[2] T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber, "Exploring Parameter Space in Reinforcement Learning," *Paladyn, Journal of Behavioral Robotics*, vol. 1, no. 1, 2010.

[3] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 01 1965. [Online]. Available: https://doi.org/10.1093/comjnl/7.4.308

[4] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, no. 2, pp. 527–566, 2017.

[5] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[6] S. Liu, B. Kailkhura, P. Y. Chen, P. Ting, S. Chang, and L. Amini, "Zeroth-order stochastic variance reduction for nonconvex optimization," *Advances in Neural Information Processing Systems*, vol. 2018-December, pp. 3727–3737, 2018.

[7] X. Chen, S. Liu, K. Xu, X. Li, X. Lin, M. Hong, and D. Cox, "ZO-AdaMM: Zeroth-order adaptive momentum method for black-box optimization," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, pp. 1–30, 2019.

[8] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 312–317.

[9] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," *Acta Numerica*, vol. 28, pp. 287–404, 2019.

[10] A. L. Custódio, J. Dennis, J. E., and L. N. Vicente, "Using simplex gradients of nonsmooth functions in direct search methods," *IMA Journal of Numerical Analysis*, vol. 28, no. 4, pp. 770–784, 10 2008. [Online]. Available: https://doi.org/10.1093/imanum/drn045

[11] A. L. Custódio and L. N. Vicente, "Using sampling and simplex derivatives in pattern search methods," *SIAM Journal on Optimization*, vol. 18, no. 2, p. 537–555, 2007.

[12] F. Y. Liu, Z. N. Li, and C. Qian, "Self-guided evolution strategies with historical estimated gradients," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2021-January, no. 14380004, pp. 1474–1480, 2020.

[13] A. Dereventsov, C. G. Webster, and J. D. Daws, "An adaptive stochastic gradient-free approach for high-dimensional blackbox optimization," 2020. [Online]. Available: http://arxiv.org/abs/2006.10887

[14] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[15] S. Surjanovic and D. Bingham, "Virtual library of simulation experiments: Test functions and datasets," Retrieved August 1, 2021, from http://www.sfu.ca/~ssurjano.

[16] N. Andrei, "An unconstrained optimization test functions collection," *Adv. Model. Optim*, vol. 10, no. 1, pp. 147–161, 2008.

[17] OpenAI, "A toolkit for developing and comparing reinforcement learning algorithms." [Online]. Available: https://gym.openai.com/

[18] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv*, pp. 1–13, 2017.

[19] K. Choromanski, A. Pacchiano, J. Parker-Holder, and Y. Tang, "From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization," *arXiv*, no. Mc, 2019.

[20] P. G. Constantine, *Active subspaces: Emerging ideas for dimension reduction in parameter studies*. SIAM, 2015.

[21] J. Zhang, H. Tran, D. Lu, and G. Zhang, "A novel evolution strategy with directional gaussian smoothing for blackbox optimization," *arXiv preprint arXiv:2002.03001*, 2020.

[22] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st ed. Springer Publishing Company, Incorporated, 2014.

[23] J. J. Moré and S. M. Wild, "Benchmarking derivative-free optimization algorithms," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 172–191, 2009.

Acknowledgments

I would like to thank my supervisor, prof. Francesco Rinaldi, for giving me the opportunity to learn and dedicate myself to this branch of mathematics, being a boundless source of knowledge.
A special thank to Anna for always being by my side through the years. Thanks to my family who supported me and made it possible for me to be here.
Thanks to all my friends, the ones who helped me relieve the stress, the ones who helped me with my questions and supported me.
To everyone, thank you very much.