CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	Using Gaussian processes as surrogate models for the CMA evolution strategy
Student:	Bc. Nikita Orekhov
Supervisor:	doc. Ing. RNDr. Martin Holeňa, CSc.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2017/18

Instructions

Black-box optimization is increasingly important in industrial applications. The state-of–the-art approach to continuous black-box optimization is the covariance matrix adaptation evolution strategy (CMA-ES). The disadvantage of evolutionary optimization, frequent evaluations of the objective, is alleviated using surrogate models.
Get familiar with evolutionary black-box optimization, in particular with CMA-ES [3].
Get familiar with the multimodal benchmark functions described in [5] and incorporate their Matlab implementation into the BBOB framework employed in [1].
Get familiar with the methods for using Gaussian processes as surrogate models for CMA-ES proposed in [1,2,4,6-9].
Implement the methods proposed in [2,4,9] using Matlab.
Compare the methods proposed in [1,2,4,6-9] on the benchmark functions described in [5] within the BBOB framework, using authors' implementations of the methods proposed in [1,6-8] and your own implementations of the approaches proposed in [2,4,9].

References

[1] L. Bajer et al. Benchmarking Gaussian Processes and Random Forests Surrogate Models on the BBOB Noiseless Testbed. In Companion of GECCO 2015.[2] D.Buche et al. Accelerating evolutionary algorithms with Gaussian process fitness function models. IEEE Transactions on Systems, Man, and Cybernetics, Part C 35 (2005).[3] N. Hansen. The CMA Evolution Strategy: A Tutorial. INRIA, 2011.[4] J.W. Kruisselbrink et al. A robust optimization approach using Kriging metamodels for robustness approximation in the CMA-ES. In CEC 2010.[5] X. Li et al. Benchmark Functions for CEC'2013 Special Session ...[6] I. Loshchilov et al. Intensive Surrogate Model Exploitation in Self-adaptive Surrogate-assisted CMA-ES (saACM-ES). In GECCO 2013.[7] J. Lu, et al. An Evolution Strategy Assisted ... In GECCO 2013.[8] H. Mohammadi et al. EGO and CMA-ES Complementary for Global Optimization. In Learning and Intelligent Optimization, 2015.[9] H. Ulmer et al. Evolution strategies assisted .... In CEC 2003.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 25, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Using Gaussian processes as surrogate models for the CMA evolution strategy

Bc. Nikita Orekhov

Supervisor: doc. Ing. RNDr. Martin Holeňa, CSc.

10th May 2016

Acknowledgements

First and foremost I thank my supervisor doc. Ing. RNDr. Martin Holeňa, CSc. for his great support and guidance during the entire period of writing this thesis. Further, I would like to express my appreciation to Ing. Lukáš Bajer for his technical assistance and for many valuable suggestions.

Also, access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

I am especially grateful to Bc. Ksenia Shakurova for providing an inexhaustible source of confidence without which this work would not be possible. Finally, I thank my parents for the exclusive opportunity to learn so many things about this world.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 10th May 2016 ....

Citation of this thesis

Orekhov, Nikita. *Using Gaussian processes as surrogate models for the CMA evolution strategy.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato práce zkoumá efektivitu metod založených na Gaussovských procesech v oblasti spojité black-box optimalizace. Tyto metody slouží jako náhradní modely pro CMA evoluční strategii. Práce popisuje několik nejmodernějších metod a pak srovnává jejích výkon na souboru funkcí z CEC'2013.

Klíčová slova black-box optimalizace, CMA-ES, Gaussovské procesy, náhradní modely, srovnávání.

Abstract

This thesis focuses on performance gain investigation for Gaussian process-based surrogate modeling techniques in the field of continuous black-box optimization by means of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We outline several state-of-the-art techniques and experimentally evaluate them within the optimization framework using recently proposed CEC'2013 benchmarking testbed.

Keywords black-box optimization, CMA-ES, Gaussian processes, surrogate models, benchmarking.

Contents

List of Figures

List of Tables

Introduction

Motivation

Evolutionary computation has been successfully applied to a wide spectrum of engineering problems. The randomized exploration guided by a set of candidate solutions (population) was found resistant against most optimization obstacles such as noise or multi-modality. On the other hand, the universality of the evolutionary computation methods allows us to adapt to almost any optimization problem, e.g. high- or low-dimensional, with or without knowing prior information.

All these features make evolutionary algorithms (EAs) efficient for black-box optimization, where we do not know any specific analytic form of the objective function. This functions can be thought of to be wrapped in a black box, thereby having unknown properties. For this cases an evaluation of the objective function remains the only way to optimize the function.

While talking about the black-box optimization it is often implied that optimization of some objective function can be costly, i.e. the evaluation of the function takes a significant amount of time and/or money. Usually, it comes to functions which should be evaluated physically during some experiment: gas turbine profiles optimization [1] or protein folding stability optimization [2].

But despite its attractiveness, the evolutionary computation has one distinctive feature: it takes quite much evaluations to explore the function, which can be infeasible if there is an upper bound to the evaluations amount. This can be avoided by the introduction of so-called surrogate models, which serve as a prototype, providing regression-based predictions of the expensive objective function values. ESs here use the surrogate model as a cheap replacement to the real function occasionally correcting it as the new evaluations arrive. Thereby, the main target of modern surrogate-assisted strategies is to fit the learning process into a given evaluation budget preserving comparable solution quality.

Context

Today, one of the most promising methods in the field of black-box evolutionary optimization is the Covariance Matrix Adaptation evolution strategy (CMA-ES), which was initially introduced in [3]. But as any other ES, CMA-ES may suffer from insufficient convergence speed where the budget of objective function evaluations is limited.

Past works on surrogate-assisted optimization showed that utilizing models based on Gaussian Processes (GP) can lead to a significant acceleration, compared to the other models, e.g. random forests[4]. The other advantage of such approaches is the fact that GPs are more stable to over-fitting and don't have many hyper-parameters that should be additionally optimized[5].

It is not surprising that recently there were introduced quite a lot of techniques based on Gaussian processes. Hence, the main goal of this thesis is to compare several surrogate-modeling methods together in the same benchmarking framework by utilizing a set of test functions from the CEC'2013 Special Session for multi-modal function optimization [6].

The thesis is divided into three chapters. Chapter 1 introduces the concept of the Covariance Matrix Adaptation ES and Gaussian processes in the context of black-box optimization. Chapter 2 describes several existing surrogate models which have been chosen for benchmarking. Finally, chapter 3 describes the benchmarking procedure and testbed functions which were chosen for experiments, and discuss the obtained results followed by conclusion about a performance of the chosen methods.

Chapter 1

Background review

This part describes basic concepts which became de-facto standard in the field of continuous evolutionary optimization. The chapter also defines the most important objectives which must be taken into account before we can delve into the topic.

1.1 Continuous black-box optimization

During a continuous optimization process our objective is to optimize (minimize or maximize) function:

$$f : \mathbb{R}^n \mapsto \mathbb{R}.$$

The function is called *black-box* when there is no any assumption about it's analytical form. The only things that are typically known about these functions are dimensionality and variable boundaries. The objective of *continuous black-box optimization* is thus to find a solution $\boldsymbol{x} \in \mathbb{R}^n$ which has $f(\boldsymbol{x})$ value as good as possible. From this point we consider that the optimization problem refers to a minimization problem, until stated otherwise.

When assessing optimizer's performance on some benchmark function it comes reasonable to use the number of function evaluations as the only cost criterion and therefore keep this metric as low as possible. Evolution algorithms, however, may slow down the optimization process because of inability to handle to handle some property of the function to be optimized. This usually results in a larger amount of objective function evaluations. Here we describe some of these properties (see [7] for details).

Multi-modality

A *local minimum* of an unconstrained function f is a real vector $\boldsymbol{x}_l \in \mathbb{R}^n$ such that there is a neighborhood ε with a center in $\boldsymbol{x}_l$ and there is no point $\boldsymbol{x} \in \mathbb{R}^n$ with objective value smaller than $\boldsymbol{x}_l$:

$$\forall \boldsymbol{x} \in \varepsilon, f(\boldsymbol{x}) \geq f(\boldsymbol{x}_l).$$

A *global minimum* is thus a minimum x_g on the whole universe:

$$\forall \boldsymbol{x} \in \mathbb{R}^n, f(\boldsymbol{x}) \geq f(\boldsymbol{x}_g).$$

A function f called *multi-modal* if it has more than one (local) optimum. So in this way the optimization process does not guarantee that the found optimum matches to a global one. Multi-modal functions have thus much more complicated landscapes than unimodal functions.

Exponential scaling

This problem is also known in machine learning as the *curse of dimensionality.* This phenomena is related to the fact that the amount of time or memory scales exponentially with the growing problem dimensionality. It means that for highly-dimensional problems ($\gg 100$), a lot of optimization techniques, especially based on full search space coverage, will become unusable due to extreme time/memory demands.

Separability

Separability is a feature meaning that the n-dimensional optimum can be obtained by performing n independent one-dimensional optimizations along each coordinate. Hence, the curse of dimensionality has no influence on separable functions because the volume of the search space will grow linearly with n. *Non-separable* functions finding the optimum by one-dimensional searches, and *partially-separable* functions allows such searches only for a certain dimension subset. It is quite obvious that most state-of-the-art approaches exploit the separability only if it was detected, i.e. such techniques must employ some mechanism to identify separate problems.

Noisiness
Function $\tilde{f}$ is called *noisy* if its value is perturbed by some random variable ξ, called noise. It can be imagined in two ways: multiplicative noise: $\tilde{f}(x) = f(x)(1+\xi)$ or additive noise: $\tilde{f}(x) = f(x) + \xi$. Optimization of such functions seems more problematic because the information obtained from evaluation of the noisy function is less representative than in the *noiseless* case.

Ill-conditioning
Briefly speaking, the term *ill-conditioning* refers to a situation where different variables, or even directions in a search space show a completely different sensitivity on their contribution to the resulting function value. In other words, we call a function ill-conditioned if for points with similar objective values the minimal displacement that provides a certain value improvement differs by orders of magnitude. Ill-conditioned problems often lead to premature convergence.

Dynamic
A function $f(x,t)$ is called *dynamic* if the obtained objective value is dependent on the time-step t. It means that such functions are changing in time: it may be a simple shift or rotation or a complete function replacement. For the sake of simplicity we do not cover dynamic functions concentrating only on stationary ones.

Multi-objectiveness
A problem is called *multi-objective* if there are m objectives $f_i(x),\ i \in 1 \dots m$, which should be optimized simultaneously. Same as dynamic, multi-objective functions will not be covered by this thesis.

Expensiveness
A function f is called *expensive* if each evaluation of the candidate solution x costs certain price in terms of chosen metric (time or money). The optimization of such functions becomes quite complicated because the optimizer is being constrained by the relatively small evaluation budget.

1.2 Evolution strategies and CMA-ES

Evolution strategy is a randomized optimization technique which was initially introduced by Ingo Rechenberg and Hans-Paul Schwefel in the 1960-s and

1970-s [7] was based on ideas of biological adaptation and evolution. Modern ES uses primarily *selection* and *mutation* as search operators. The latter is usually performed by sampling a *normally-distributed* random variable to each candidate solution. This can be explained in the way that having defined all variances and covariances, normal distribution has the largest entropy among all distributions in $\mathbb{R}^n$ [8].

The evolution strategies can be divided into several categories according to its vision of generational change. For example, in a $(\mu + \lambda)$ strategy there will be μ best out of total $\mu + \lambda$ individuals selected as a parents for the next generation and in a $((\mu, \lambda)$ - ES) strategy only $\mu < \lambda$ offsprings are passed to the next iteration, i.e. parents never survive [7].

The covariance matrix adaptation evolution strategy (CMA-ES) has very quickly become a standard in the field of evolution strategies generally and in the continuous black-box optimization in particular. So this section aims to describe several basic concepts of the CMA-ES algorithm and to provide reasons which stand for its impressive performance on continuous optimization benchmarks. For the full description refer to [8] and [9].

1.2.1 Candidate sampling

As in the classic ES, the CMA strategy makes up a new generation of the candidate solutions by sampling from a multivariate normal distribution. This can be expressed by the following equation:

$$\boldsymbol{x}_k^{(g+1)} \quad \sim \quad \boldsymbol{m}^{(g)} + \ \sigma^{(g)} \mathcal{N}\Big(\boldsymbol{0}, \boldsymbol{C}^{(g)}\Big) \qquad \text{for } k = 1, \ldots, \lambda, \tag{1.1}$$

where

$\lambda \geq 2$ is a population size.

$\boldsymbol{x}_k^{(g+1)} \in \mathbb{R}^n$ is the k-th offspring in $g + 1$-th generation.

Symbol $\sim$ denotes the equality between the probability distributions.

$\boldsymbol{m}^{(g)} \in \mathbb{R}^n$ is the mean of the search distribution at g-th iteration (generation).

$\sigma^{(g)} \in \mathbb{R}_+$ is a step-size (standard deviation) at g-th generation.

$\boldsymbol{C}^{(g)} \in \mathbb{R}_{n \times n}$, covariance matrix at generation g.

To express the whole iteration sequence it remains to define how to update $\boldsymbol{m}^{(g+1)}$, $\sigma^{(g+1)}$ and $\boldsymbol{C}^{(g+1)}$ having all candidate solutions sampled.

1.2.2 Selection and recombination

Mean of the new search distribution within a (μ, λ)-CMA-ES can be calculated as a *weighted average* of μ best candidate solutions:

$$\boldsymbol{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(g+1)}, \tag{1.2}$$

while

$$\sum_{i=1}^{\mu} w_i = 1, \qquad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0, \tag{1.3}$$

where

$\mu < \lambda$ is the parents amount, i.e. the population size for the ongoing generation.

$w_i \in \mathbb{R}_+$ is the weight coefficients for *recombination.* For all $w_i = 1/\mu$ equation 1.2 calculates the mean of the μ selected points.

$\boldsymbol{x}_{i:\lambda}^{(g+1)}$ denotes i-th best solution point out of $\boldsymbol{x}_1^{(g+1)}, \ldots, \boldsymbol{x}_\lambda^{(g+1)}$. The index $i : \lambda$ describes the i-th ranked individual and implicates the inequality $f(\boldsymbol{x}_{1:\lambda}^{(g+1)}) \leq f(\boldsymbol{x}_{2:\lambda}^{(g+1)}) \leq \ldots \leq f(\boldsymbol{x}_{\lambda:\lambda}^{(g+1)})$, where f is the objective function.

Equation 1.2 implements a selection mechanism called *truncation selection.* In addition, a metric called *variance effective selection mass* will be widely used in the following sections. This can be calculated with the following equation:

$$\mu_{\text{eff}} = \left(\frac{\|\boldsymbol{w}\|_1}{\|\boldsymbol{w}\|_2} \right)^2 = \frac{\|\boldsymbol{w}\|_1^2}{\|\boldsymbol{w}\|_2^2} = \frac{1}{\|\boldsymbol{w}\|_2^2} = \left(\sum_{i=1}^{\mu} w_i^2 \right)^{-1}. \tag{1.4}$$

From definition of w_i in the equation 1.3 we can conclude that $1 \leq \mu_{\text{eff}} \leq \mu$ and $\mu_{\text{eff}} = \mu$ when the recombination weights was chosen identically, i.e. $w_i = 1/\mu$ for all $i = 1 \ldots \mu$. Usually, $\mu_{\text{eff}} \approx \lambda/4$ indicates a reasonable setting of w_i. A typical setting of w_i could be proportional to $\mu - i + 1$, and $\mu \approx \lambda/2$.

1.2.3 Covariance matrix adaptation

This subsection describes the way how to use the covariance matrix inside CMA-ES. This involves the method of the initial covariance matrix estimation as well as adaptation techniques called *rank-one-update* and *rank-μ-update* that are suitable for different population sizes. The adaptation procedure itself can be also enhanced by utilizing dependencies between accomplished steps as described in 1.2.3.4. In the end, all of those strategies will be combined together in order to obtain a universally-behaving algorithm.

1.2.3.1 Covariance matrix estimation

For the sake of simplicity here we will set up a step size $\sigma_{(g)} = 1$. This insignificant assumption will shift the expression only by a constant factor.

The covariance matrix can be obtained by the following equation:

$$\boldsymbol{C}_{\lambda}^{(g+1)} = \frac{1}{\lambda}\sum_{i=1}^{\lambda}\left(\boldsymbol{x}_i^{(g+1)} - \boldsymbol{m}^{(g)}\right)\left(\boldsymbol{x}_i^{(g+1)} - \boldsymbol{m}^{(g)}\right)^{\mathrm{T}}. \tag{1.5}$$

The matrix $\boldsymbol{C}_{\lambda}^{(g+1)}$ is thus an unbiased estimator of $\boldsymbol{C}^{(g)}$ (i.e. we have that $\mathbb{E}\left(\boldsymbol{C}_{\lambda}^{(g+1)} | \ \boldsymbol{C}^{(g)}\right) = \boldsymbol{C}^{(g)}$). Note that in this equation $\boldsymbol{C}_{\lambda}^{(g+1)}$ estimates variances of **all** sampled points $(\boldsymbol{x}_i^{(g+1)} - \boldsymbol{m}^{(g)})$. In addition, to obtain "better" covariance matrix we can employ a *weighted average mechanism* similar to one described in 1.2.2. Thereby, the equation 1.5 now looks like:

$$\boldsymbol{C}_{\mu}^{(g+1)} = \sum_{i=1}^{\mu} w_i\left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\right)\left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\right)^{\mathrm{T}}. \tag{1.6}$$

The main difference here is that the matrix $\boldsymbol{C}_{\mu}^{(g+1)}$ estimates only **selected** (successful) steps while matrix $\boldsymbol{C}_{\lambda}^{(g+1)}$ estimates variances for **all** steps before selection. In other words sampling from a $\boldsymbol{C}_{\mu}^{(g+1)}$ will tend to reproduce candidates based on successful steps which can potentially improve the estimation quality.

Figure 1.1 demonstrates results of the covariance matrix estimation by the use of equation 1.6 in the context of minimizing *linear* function for $\lambda = 150$, $\mu = 50$ and $w_i = 1/\mu$. First picture visualizes a sampling of $\lambda = 150$ $\mathcal{N}(\mathbf{0}, \mathbf{I})$ points. The second one: $\mu = 50$ best points interconnected with the mean value. And the last picture denotes search distribution estimated for the next generation. Note that the estimation via $\boldsymbol{C}_{\mu}^{(g+1)}$ increases expected variance in a gradient direction.

1.2.3.2 Rank-μ-update

According to [8], to achieve a fast *local* search (e.g. for competitive performance on f_{sphere}), the population size must be small. But then in becomes impossible to estimate a *reliable* covariance matrix from 1.6. As a workaround to this, the information from previous generations can be used as well. For example, after some number of generations, a covariance matrix $\mathbf{C}$ can be estimated

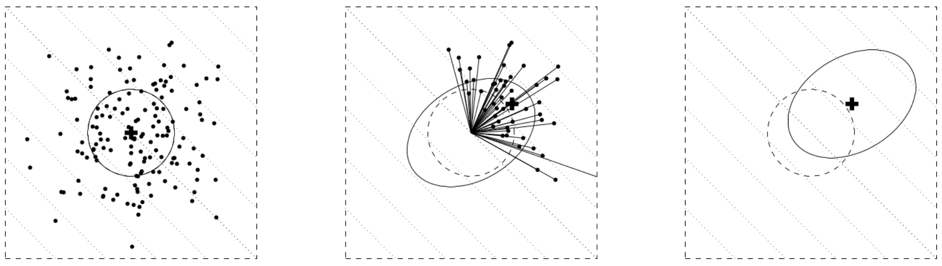


Figure 1.1: Estimation of the $\boldsymbol{C}_{\mu}^{(g+1)}$ covariance matrix on $f_{linear}(x) = -\sum_{i=1}^{2} x_i$. *Dotted* lines indicate that the strategy should move towards the upper right corner.

from the estimated covariance matrices over all previous iterations [8]. This can be expressed as:

$$\boldsymbol{C}^{(g+1)} = \frac{1}{g+1} \sum_{i=0}^{g} \frac{1}{{\sigma^{(i)}}^2} \boldsymbol{C}_{\mu}^{(i+1)}. \tag{1.7}$$

Such method can already be treated as a reliable estimator for the selected steps. Obviously, all generations in equation 1.7 have the same weights. To provide recent generations with extra weight, a technique called *exponential smoothing* was introduced. Assuming $\boldsymbol{C}^{(0)}$ to be the identity matrix and a *learning rate* $0 < c_{\mu} \leq 1$, $\boldsymbol{C}^{(g+1)}$ becomes

$$\boldsymbol{C}^{(g+1)} = (1 - c_{\mu})\boldsymbol{C}^{(g)} + c_{\mu} \frac{1}{{\sigma^{(i)}}^2} \boldsymbol{C}_{\mu}^{(g+1)} \tag{1.8}$$

$$= (1 - c_{\mu})\boldsymbol{C}^{(g)} + c_{\mu} \sum_{i=1}^{\mu} w_i \boldsymbol{y}_{i:\lambda}^{(g+1)} {\boldsymbol{y}_{i:\lambda}^{(g+1)}}^{\mathrm{T}}, \tag{1.9}$$

where

c_{μ} is the learning rate for the covariance matrix update. For $c_{\mu} = 1$ no prior information is adopted and $\boldsymbol{C}^{(g+1)} = \frac{1}{{\sigma^{(i)}}^2} \boldsymbol{C}_{\mu}^{(g+1)}$. For $c_{\mu} = 0$ no learning takes place and $\boldsymbol{C}^{(g+1)} = \mathbf{I}$. Here $c_{\mu} \approx \min(1, \mu_{\text{eff}}/n^2)$ is a reasonable choice.

$\boldsymbol{y}_{i:\lambda}^{(g+1)} = \left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\right)/\sigma^{(g)}$.

This method of the covariance matrix update is called rank-μ-update, as the sum of the products is of rank $\min(\mu, n)$. The sum can even consist of a single term if $\mu = 1$.

The choice of c_{μ} is important. Values close to zero result in a slow learning tempo. Otherwise, too large values fail because of covariance matrix degeneration. Fortunately, an empirical approximation of $c_\mu \approx \mu_{\text{eff}}/n^2$ was found independent to the function to be optimized.

In the end, supposing a small population size λ and fixed evaluation budget (amount of objective function evaluations) will result in a larger number of generations and, usually, in a better adaptation of the covariance matrix.

1.2.3.3 Rank-one-update

The approach of the initial covariance matrix estimation was described in (1.2.3.1). Now it's time to review the technique where the covariance matrix is being updated subsequently as of generations change. This section introduces the method called *rank-one-update*, which updates the covariance matrix using a single selected step for the latest generation.

Consider a method to sample from normal distribution with zero mean. Then, let vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{g_0} \in \mathbb{R}^n$, $g_0 \geq n$ and let $\mathcal{N}(0,1)$ denote independent normally distributed random numbers. In such case we get that

$$\mathcal{N}(0,1)\boldsymbol{y}_1 + \ldots + \mathcal{N}(0,1)\boldsymbol{y}_{g_0} \quad \sim \quad \mathcal{N}\Big(\mathbf{0}, \sum_{i=1}^{g_0} \boldsymbol{y}_i\ \boldsymbol{y}_i^{\mathrm{T}}\Big) \tag{1.10}$$

represents a normally distributed random vector with zero mean and covariance matrix $\sum_{i=1}^{g_0} \boldsymbol{y}_i\ \boldsymbol{y}_i^{\mathrm{T}}$. Such vector is obtained by adding *line distributions* $\mathcal{N}(0,1)\boldsymbol{y}_i$. The singular distribution $\mathcal{N}(0,1)\boldsymbol{y}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{y}_i\ \boldsymbol{y}_i^{\mathrm{T}})$ generates vector $\boldsymbol{y}_i$ with maximum likelihood among all normal distributions with zero mean.

Now using the conjunction of 1.10 and slightly shortened 1.9 with $\mu = 1$ as the only selected step, we obtain a rank-one-update rule with the following equation:

$$\boldsymbol{C}^{(g+1)} = (1 - c_1)\boldsymbol{C}^{(g)} + c_1 \boldsymbol{y}^{(g+1)} {\boldsymbol{y}^{(g+1)}}^{\mathrm{T}}, \tag{1.11}$$

where

$$\boldsymbol{y}^{(g+1)} = \Big(\boldsymbol{x}_{1:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\Big)/\sigma^{(g)}.$$

Note that in 1.11 the right summand is of rank 1 and the maximum likelihood term for $\boldsymbol{y}^{(g+1)}$ to the covariance matrix $\boldsymbol{C}^{(g)}$. Thereby the probability of obtaining the right $\boldsymbol{y}^{(g+1)}$ in next generation increases.

1.2.3.4 Cumulation strategy

Embedding an expression $\boldsymbol{y}^{(g+1)} = \Big(\boldsymbol{x}_{1:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\Big)/\sigma^{(g)}$ into the covariance matrix update we discard an information about a sign of the selected steps because $\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}$ equals to $-\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}$. To reuse the sign information a special technique called *cumulation* needs to be described [3].

Consider a sequence of successive steps over a number of generations which is called *evolution path*. This can be expressed as a sum of the consecutive steps which in its turn referred to as *cumulation.* The example below illustrates an evolution path of the distribution mean m constructed by the summation of three steps:

$$\boldsymbol{p}_c = \frac{\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}} + \frac{\boldsymbol{m}^{(g)} - \boldsymbol{m}^{(g-1)}}{\sigma^{(g-1)}} + \frac{\boldsymbol{m}^{(g-1)} - \boldsymbol{m}^{(g-2)}}{\sigma^{(g-2)}}.$$

Applying an exponential smoothing from 1.9 and generalizing the evolution path for the generation $(g+1)$ with given $\boldsymbol{p}_c^{(0)} = \mathbf{0}$ we obtain

$$\boldsymbol{p}_c^{(g+1)} = (1 - c_c)\boldsymbol{p}_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}}\,\frac{\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}}, \tag{1.12}$$

where

$\boldsymbol{p}_c^{(g)} \in \mathbb{R}^n$ is the evolution path at generation g.

c_c is the learning rate. According to [8], $1/n \leq c_c \leq 1/\sqrt{n}$ is a reasonable setting.

The factor $\sqrt{c_c(2 - c_c)\mu_{\text{eff}}}$ was chosen as a normalization constant in order to achieve

$$\boldsymbol{p}_c^{(g+1)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C}) \tag{1.13}$$

if

$$\forall i = 1, \ldots, \mu, \quad \boldsymbol{p}_c^{(g)} \sim \frac{\boldsymbol{x}_{i:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C}). \tag{1.14}$$

In the end we rewrite rank-one-update of the covariance matrix using the evolution path $\boldsymbol{p}_c^{(g+1)}$ as follows:

$$\boldsymbol{C}^{(g+1)} = (1 - c_1)\boldsymbol{C}^{(g)} + c_1 \boldsymbol{p}_c^{(g+1)} \boldsymbol{p}_c^{(g+1)^{\mathrm{T}}}. \tag{1.15}$$

Setting the $c_1 \approx 2/n^2$ was empirically validated and acknowledged as reasonable in the most cases. Using the evolution path for the update of $\boldsymbol{C}$ shows significant performance improvement for small μ_{eff} because of exploiting the sign information between consecutive steps.

1.2.3.5 The final procedure

Finally, the covariance matrix update technique is built by combining 1.9 and 1.15.

$$\boldsymbol{C}^{(g+1)} = (1-c_1-c_\mu)\boldsymbol{C}^{(g)} + c_1 \underbrace{\boldsymbol{p}_c^{(g+1)} \boldsymbol{p}_c^{(g+1)^{\mathrm{T}}}}_{\text{rank-one update}} + c_\mu \underbrace{\sum_{i=1}^{\mu} w_i \, \boldsymbol{y}_{i:\lambda}^{(g+1)} \boldsymbol{y}_{i:\lambda}^{(g+1)^{\mathrm{T}}}}_{\text{rank-}\mu\text{ update}}, \quad (1.16)$$

where

$c_1 \approx 2/n^2$.

$c_\mu \approx \min(\mu_{\text{eff}}/n^2, 1 - c_1)$.

$\boldsymbol{y}_{i:\lambda}^{(g+1)} = \left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \boldsymbol{m}^{(g)}\right)/\sigma^{(g)}$.

Note that 1.16 can be converted to 1.9 if $c_1 = 0$ and to 1.15 if $c_\mu = 0$. Thereby, the joint equation combines advantages of both 1.9 and 1.15. On the one hand, the information within single generation is utilized by the rank-μ-update, which is useful for larger populations. On the other hand, the information of correlations between generations is exploited by using the evolution path for the rank-one update. This is important if the population size is small.

1.2.4 Step-size adaptation

The covariance matrix adaptation method described in 1.2.3 does not handle step size (i.e. the scale of the distribution) explicitly. The update process increases (implicitly) step size only in one direction for each selected step and decreases this step only by discarding old information over exponential smoothing, $1 - c_1 - c_\mu$. This ensures that the adaptation of σ is very slow "as is" and additional technique must be introduced.

To better control the step size it is rationally to incorporate already known to us evolution path technique. The step size adjustment is applied independently of the covariance matrix update and therefore is mentioned in the literature as *cumulative step length adaptation* (CSA).

The step size is being modified according to the information provided by the evolution path length. The concept of this method is well explained in figure 1.2. If the evolution path is quite short, i.e. the particular steps cancel each other, it suggests us to decrease the current step size σ. Such situation is shown on the left image. Otherwise, whenever the evolution path is very long, as shown on the right picture, the same distance can be covered in fewer amount of prolongated steps. As a consequence to this, the step size should be increased. Note that figure 1.2 also depicts a situation where particular

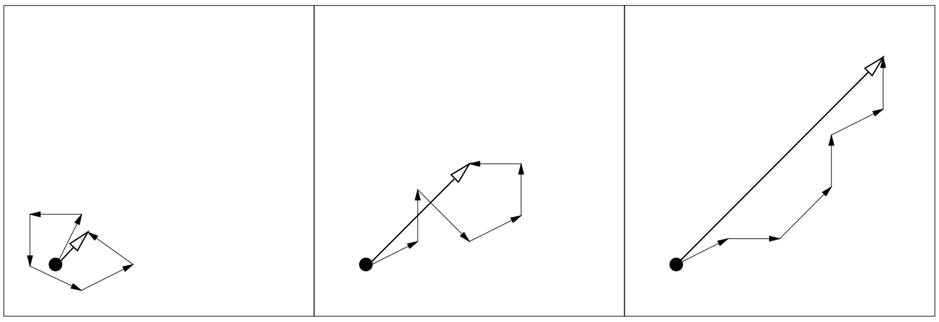


Figure 1.2: Three different evolution paths constructed of six steps from different optimization situations. The lengths of particular steps are comparable. The lengths of evolution paths are different in order to underline the dynamic step size adjustment strategy.

steps are *uncorrelated* (picture in the middle). This indicates that the step size length is being chosen adequately.

In order to keep the step size at a reasonable length we need to know whether and how much it should be changed. To achieve this we compare the current step size with its *expected length under random selection.* Under random selection particular steps are performed independently on their effectiveness and are therefore uncorrelated. So, if the evolution path is happened to be loner than expected, the step size needs to be increased. Analogically, σ will be decreased if the evolution path is shorter than expected.

Unlike the equation 1.12, a *conjugate* evolution path must be built here, because the expected length of the evolution path $\boldsymbol{p}_c$ would be dependent on its direction (especially in case of ill-conditioned objective functions). The conjugate evolution path thus (having defined $\boldsymbol{p}_\sigma^{(0)} = 0$) reads as follows

$$\boldsymbol{p}_\sigma^{(g+1)} = (1 - c_\sigma)\boldsymbol{p}_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\, \boldsymbol{C}^{(g)^{-\frac{1}{2}}} \frac{\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}}, \tag{1.17}$$

where

$\boldsymbol{p}_\sigma^{(g)} \in \mathbb{R}^n$ is the conjugate evolution path at iteration g.

$c_\sigma \leq 1$ is the learning rate.

$\sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}$ is a normalization constant (as defined in 1.2.3.4).

$\boldsymbol{C}^{(g)^{-\frac{1}{2}}} \overset{\text{def}}{=} \boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)^{-1}}\boldsymbol{B}^{(g)^{\mathrm{T}}}$, where $\boldsymbol{C}^{(g)} = \boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)^2}\boldsymbol{B}^{(g)^{\mathrm{T}}}$ is an eigendecomposition of $\boldsymbol{C}^{(g)}$, where $\boldsymbol{B}^{(g)}$ is an orthonormal basis of eigenvectors, and the diagonal elements of the diagonal matrix $\boldsymbol{D}^{(g)}$ are square roots of the corresponding positive eigenvalues [8].

For $\boldsymbol{C}^{(g)} = \mathbf{I}$, we have $\boldsymbol{C}^{(g)^{-\frac{1}{2}}} = \mathbf{I}$ which converts 1.17 to 1.12. The transformation $\boldsymbol{C}^{(g)^{-\frac{1}{2}}}$ re-scales the step $\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}$ within the coordinate system given by $\boldsymbol{B}^{(g)}$. Note that the eigendecomposition applies a rescaling so that all axes becomes equally sized without being rotated by the overall transformation which preserves the directions of consecutive steps [8]. All the above makes the expected length of $\boldsymbol{p}_\sigma^{(g+1)}$ independent to its direction and for any sequence of realized covariance matrices $\boldsymbol{p}_\sigma^{(g+1)} \sim \mathcal{N}(0, \mathbf{I})$.

In order to update step size, the actual length of evolution path ($\|\boldsymbol{p}_\sigma^{(g+1)}\|$) should be compared to its expected length (expectation of the Euclidean norm of a $\mathcal{N}(\mathbf{0}, \mathbf{I})$ distributed random vector) $\mathbb{E}\ \|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$:

$$\ln\ \sigma^{(g+1)} = \ln\ \sigma^{(g)} + \frac{c_\sigma}{d_\sigma\ \mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} \Big(\|\boldsymbol{p}_\sigma^{(g+1)}\| - \mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|\Big), \tag{1.18}$$

where

$d_\sigma \approx 1$ is a *damping* parameter which scales the change magnitude of $\ln \sigma^{(g)}$. Note that this parameter can be used to control the step size learning speed independently from covariance matrix.

It should be mentioned that for equality of $\|\boldsymbol{p}_\sigma^{(g+1)}\|$ and $\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$ a second summand of 1.18 is zero which results in an unchanged $\sigma^{(g)}$. Obviously, if the real evolution path length is bigger than expected: $\sigma^{(g)}$ will be increased and vice versa. Note also that the step size change is unbiased on the log scale, because $\mathbb{E}\ (\ln\sigma^{(g+1)} |\ \sigma^{(g)}) = \ln\sigma^{(g)}$.

Assuming that $\sigma^{(g)} > 0$, we can rewrite 1.18 to

$$\sigma^{(g+1)} = \sigma^{(g)}\exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma^{(g+1)}\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1\right)\right). \tag{1.19}$$

1.2.5 Discussion

We widely use CMA-ES as a primary optimization algorithm in several use-cases (e.g. for performance comparison or as a basement for surrogate modeling) in this thesis. So this subsection aims to sum up several properties pointing out that this algorithm is a good candidate to lead the optimization process.

A big advantage of the CMA-ES is the ability to overcome problems typical for most evolutionary algorithms:

- Inability to properly optimize badly scaled and non-separable problems. Equation 1.16 adaptively changes the search distribution in order to handle such functions.

- Need for large population sizes. Using 1.16, population size can vary without fear of population degeneration. Small population size leads to faster convergence rate while huge population helps to escape from local optima.

- Premature convergence. Presence of adaptive step size control in 1.19 prevents the algorithm from being stuck at local optima and saves budget by dynamically adjusting the step size σ.

Here we also outline some principles which formed the basis of the algorithm:

Change rates
This term can be referred to as a speed of parameter change per sampled point. An algorithm has the ability to independently control this rates for the mean $\boldsymbol{m}$ of the distribution, the covariance matrix $\boldsymbol{C}$ and the step size σ.

Invariance
Invariance should be considered as a fundamental criterion for any optimization technique. The algorithm invariant to some property of the search space is characterized by an identical performance on a set of objective functions defined by this property. For example, this frees the algorithm hyper-parameters from being manually tuned according to some unknown scale of the optimized function.

CMA-ES employs the following sources of invariance:

- Invariance w.r.t. order preserving transformations of the objective function. CMA-ES in this case relies on the ranking of the solution candidates.

- Scale invariance if the initial step size $\sigma^{(0)}$ and mean $\boldsymbol{m}^{(0)}$ are chosen accordingly.

- Invariance w.r.t. angle preserving transformations (rotation, reflection, translation) if the initial mean value $\boldsymbol{m}^{(0)}$ is chosen accordingly.

All of the properties described above shows that CMA-ES can be highly competitive on a wide range of benchmark functions. Nevertheless, the algorithm also suffers from several disadvantages:

Computational complexity
This problem is reflected in both time and space complexity. In order to learn the covariance matrix CMA-ES needs to store $(n^2 + n)/2$ real-valued numbers. The algorithm also performs an eigendecomposition of the covariance matrix which time complexity can be roughly expressed as $\mathcal{O}(n^3)$. All this limitations may become a challenge even for n starting from 1000.

Premature convergence
Even with adaptively changing parameters, the CMA-ES sometimes tends to converge prematurely on multi-modal functions. But even more interesting the fact that it fails to find the minimizer on a set of unimodal functions called *HappyCat* [10]. The behavior of CMA-ES on such class is similar to one observed because of exponentially fast decreasing mutation strength of the step length adaptation.

High FE consumption
The problem can be treated as characteristic of every evolutionary algorithm. This limitation is especially important when the function to be optimized is costly. Note that this thesis aims to investigate methods capable of decreasing such a high consumption of function evaluations.

1.3 CMA-ES restart strategies

The CMA-ES is considered as a *local* optimizer [8]. So for the case of multi-modal functions it may end up in a local optima. In order to reduce the influence of such functions on the resulting solution several CMA-ES restart strategies were introduced.

1.3.1 IPOP-CMA-ES

In [11] a restart strategy for the CMA-ES called IPOP-CMA-ES with increasing population size was introduced. The resulting algorithm is similar to (μ, λ)-CMA-ES and therefore uses standard settings except for the population size. This parameter is left default for the first run but then becomes repeatedly increased. The algorithm is interrupted when the stopping criterion is met and then the independent restart takes place with the population size increased by the factor of 2. This relation can be described for the given initial population size λ_{default} as follows:

$$\lambda_{i_{\text{restart}}} = 2^{i_{\text{restart}}} \lambda_{\text{default}}, \tag{1.20}$$

where

i_{restart} denotes the current restart number.

1.3.2 BIPOP-CMA-ES

Unlike the previous, a BIPOP-CMA-ES employs two different restart strategies with increasing and decreasing population size [12]. Each restart strategy takes into account the remaining budget of the function evaluations and a budget from the last run (see below). The choice of the strategy is dependent on which evaluation budget is smaller.

The first scheme is equal to the one described in 1.3.1, where the population size is being doubled. However, the author introduces several distinctions such as the increasing scenario can be performed at most nine times (so the largest population size just not cross the $2^9\lambda_{\text{default}}$ score). The initial step size σ^0 is also set to 2 (i.e. 1/5 of the domain width).

The second scheme represents algorithm's runs with *small* population size. It's update is determined by the equation below:

$$\lambda_{\text{small}} = \lfloor \lambda_{\text{default}} \left(\frac{\lambda_{\text{large}}}{2\lambda_{\text{default}}} \right)^{\mathcal{U}[0,1]^2} \rfloor, \tag{1.21}$$

where

$\mathcal{U}[0,1]$ denotes uniformly distributed numbers in range [0,1],

λ_{large} denotes the population size from the last increasing restart scheme.

In addition, the initial step size σ^0 is set to $2 \times 10^{-2\mathcal{U}[0,1]}$.

The restart with small population size is performed only if the remaining FE budget is smaller than one spent in the last scheme with increasing populations. The budget of the initial increasing restart is obtained from the first restart, so the first single run with population size λ_{default} is disregarded.

1.4 Surrogate-assisted optimization

In the context of black-box optimization the only source of information comes from the objective function evaluations. In addition, it often comes that the object of interest can not be obtained instantly, or can be obtained only limited number of times. The function representing such problem is called costly while the problem itself is characterized by available evaluation budget.

In order to save the evaluation budget, the real objective function f can be replaced by its image, an artificial *approximation model* $\hat{f}$. These models also referred to as the *surrogate models* or *meta-models.* Incorporating the surrogate modeling technique can be used to get a hint where the promising solutions can be found. But the learning process of such models is still determined by the real function evaluations. Furthermore, it often comes that the optimization algorithm (i.e. CMA-ES) "discards" some records about solutions evaluated some time ago. Of course, there are several reasons to do so, but it is obvious that in such cases the algorithm loses a part of important information.

There are several major categories of surrogate models based on the incorporated regression techniques. *Polynomial models*, *Artificial neural networks*, *Support Vector Machines*, *Gaussian processes* and *Radial Basis Function*-based methods can be listed among the best known [13]. However, this thesis is aimed at the investigation of Gaussian processes-based surrogate modeling techniques only.

In a contrast to pure function optimization, the whole set of evaluated points (individuals) can be used in order to train a regression model. Nevertheless, every regression model $\hat{f}$ often takes time to learn the real function adequately. And still, an approximation can be erroneous and differ from a real function value. So to prevent the optimizer from being confused by outdated or imprecise surrogate model, it still has to be occasionally corrected[1] on the real objective function during the whole optimization process.

1.4.1 Interacting with ESs

A correction of the surrogate model prediction is performed during the core algorithm's iterations and on solutions sampled by this algorithm. Generally, there are two main ways to integrate surrogate model adjustment within ES. One can proclaim the model as accurate enough and replace the real function with this model, estimate the global model's optimum and evaluate in on the true function. Such method is then called *surrogate approach* [1]. The other way consists in a selection of a fraction of individuals at each generation (in fact, it can be done less frequently). This *controlled* subset is evaluated on the real objective function, while the rest is evaluated on its model. This approach is thus called *evolution control* (EC) [13].

1.4.1.1 Evolution control

In evolution control a selected set of individuals are evaluated on the true objective function, the rest on the surrogate model. The usage of the method grants a certain reduction in cost of the objective function proportional to the

[1]For example, the correction of the surrogate model can be done even by re-learning the model with new solutions added at each iteration.

amount of uncontrolled individuals [2]. [13] suggested to split this category into the following sub-categories: *individual-based* and *generation-based* EC.

By the generation-based it is meant that the entire population is evaluated on the original objective function only at certain iterations (such iterations are marked as g_o), while for the other (marked as g_m) the surrogate model evaluation is retained. Note that such kind is especially efficient for parallelization, as the individuals can be evaluated simultaneously. As an example of generation-based EC application, the following practice can be observed: the real objective evaluations are performed until the model error drops under a certain threshold. The population then is left uncontrolled for several iterations [1].

For a population-based strategy the main question stands for: how to select an appropriate set of candidates to be evaluated on a true objective function. In case of easy problems some simple control scheme (e.g. randomly selected individuals or best ones [1]) may be sufficient. However, for multi-modal functions such techniques may get stuck and bypass the whole accelerating potential. This involves a trade-off between so-called *exploration* and *exploitation* model aspects. This can be explained in a way that the model *exploits* information about the function in order to determine the most promising candidates. At the same time this function needs to be thoroughly *explored* to prevent premature convergence. A technique called *probability of improvement* can be a good example of the usage of such trade-off [14].

1.4.1.2 Surrogate approach

In the surrogate approach an optimization algorithm uses the surrogate model to predict the global optimum. The predicted optimum thus represents an ideal candidate to be evaluated on the real function. The result is the incorporated to the model resulting in an improved approximation ability. This kind of surrogate model is incorporated e.g. in *efficient global optimization* (EGO) [15]. Note that the surrogate approach is also a subject to exploration-exploitation trade-off. The performance of surrogate approach methods may overcome evolution controlled ones, especially if the model already possesses all necessary points to provide accurate estimations near the global optimum. On the other hand, such approaches is difficult to parallelize as it proceeds sequentially from one estimated optimum to another.

1.5 Gaussian processes

Gaussian process appears to be quite interesting technique which can be employed in the context of costly black-box optimization as the objective function

[2]Of course, this assumption is true only if the model approximation is accurate enough.

approximation model. The popularity of such models has increased significantly in the recent years. So in this section there we describe the main features of the Gaussian process, its pros and cons and, in particular, the GP based approximation technique, so-called Gaussian process regression. In this thesis we use the interpretation of the Gaussian processes, as defined in [5].

1.5.1 Description

A Gaussian process is a generalization of the Gaussian probability distribution. But whereas the probability distribution is aimed to describe variables or vectors, a random process describes functions. For example, writing $X \sim \mathcal{GP}(m, k)$ means that the random function X is distributed over the Gaussian distribution with mean m and covariance function k. Informally, this can be imagined in a way that the GP represents distribution over all suitable functions. The properties of such functions is determined by the covariance function. And, as many covariance functions are possible, the Gaussian process behavior can be flexibly set up.

The other way is to interpret an objective function as an unknown scalar function of a point $\boldsymbol{x} \in \mathbb{R}^n$ in a n-dimensional space. Then, evaluating of a such function on a set of solution points $\mathbf{X}_N = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ will result in the set of function values $\boldsymbol{t}_N = \{t_1, \dots, t_N\}$, where $\forall i = 1, \dots, N, \quad t_i = f(\boldsymbol{x}_i)$. In other words, an input for the Gaussian processes is a training set $\mathcal{D}$ of N data points with corresponding function values at those points. The training set thus can be read as follows:

$$\mathcal{D} = \{(\boldsymbol{x}_i, t_i) \mid i = 1, \dots, n\} = (\boldsymbol{X}, \boldsymbol{t}).$$

An introduction of point x and corresponding function value $f(x)$ can be treated as a point of certainty to the distribution. In other words, this will eliminate functions which properties are inconsistent with the structure given by evaluations. So the problem of learning in the context of Gaussian processes is a problem of finding a suitable combination of properties for the chosen covariance function. Note that the resulting model and its characteristics can be well interpreted.

1.5.2 Gaussian process regression

Unlike the classification, the main objective of the regression is to provide predictions for function values (which is of continuous quantities) at a given point. So the objective of a Gaussian process regression or *Kriging* is to predict function value t_{N+1} at a new point $\boldsymbol{x}_{N+1}$.

Gaussian processes use a probabilistic model where the given vector of function values $\boldsymbol{t}_N$ is a sample from a joint multivariate Gaussian distribution with probability density $p(\boldsymbol{t}_N|\boldsymbol{X}_N)$. Analogically, a sought-for vector $\boldsymbol{t}_{N+1}$ is assumed to be a sample from the distribution with probability density

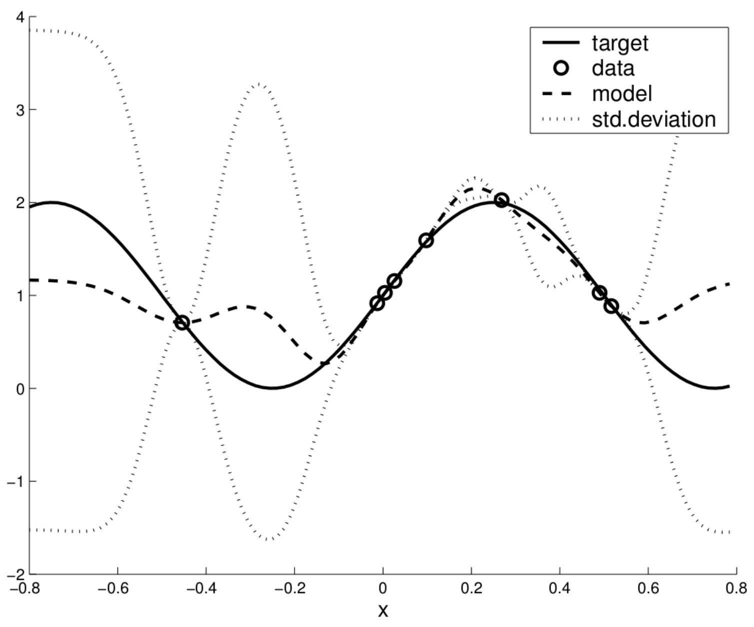


Figure 1.3: A GP model approximation of the *sinus* function $f(x) = sin(x)$ using a training set of 8 data points.

$p(\boldsymbol{t}_N, t_{N+1} \mid \boldsymbol{X}_N, \boldsymbol{x}_{N+1})$. Thus, the dimensionality of the probability density $(N + 1)$ here equals to the amount of points and is independent from the dimensionality of the search space (n in this case).

A figure 1.3 demonstrates how an example 1-dimensional function $f(x) = sin(x)$ can be optimized by using Gaussian process with a given training set of points. *Weakly-dotted lines* show a standard deviation for the approximated graph areas. Note that the scantily explored segments have higher standard deviation than well explored ones [14].

Applying a conditional probability rule we get that the probability density for t_{N+1}, given $\boldsymbol{X}_{N+1}$ data points and $\boldsymbol{t}_N$ function values, stands for [1]:

$$p(t_{N+1} \mid \boldsymbol{X}_{N+1}, \boldsymbol{t}_N) = \frac{p(\boldsymbol{t}_{N+1}|\boldsymbol{X}_{N+1})}{p(\boldsymbol{t}_N|\boldsymbol{X}_N)}. \tag{1.22}$$

According to the equation above the probability density for the new function value comes from a univariate Gaussian distribution [1].

The next step is to express the equation 1.22 using a distribution mean and standard deviation. A denominator (and similarly a numerator) reads as:

$$p(\boldsymbol{t}_N|\boldsymbol{X}_N) = \frac{\exp\big(-\frac{1}{2}\boldsymbol{t}_N^{\mathrm{T}}\boldsymbol{C}_N^{-1}\boldsymbol{t}_N\big)}{\sqrt{(2\pi)^N \det(\boldsymbol{C}_N)}}, \tag{1.23}$$

where

$\boldsymbol{C}_N$ is the covariance matrix of the Gaussian distribution for N solution points.

The covariance matrix represents our prior assumptions about the objective function. This matrix is constructed through the covariance function k for each pair of solution points. The covariance function represents the covariance between pairs of random variables:

$$cov\big(f(\boldsymbol{x}_p), f(\boldsymbol{x}_q)\big) = k(\boldsymbol{x}_p, \boldsymbol{x}_q).$$

Note that the covariance between outputs is characterized by the function of inputs.

Here we list examples of the covariance functions. The first one, so-called *squared exponential* (SE) function can be expressed as follows:

$$k_{\mathrm{SE}}(\boldsymbol{x}_p, \boldsymbol{x}_q) = \exp\Big(-\frac{1}{2}\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2\Big). \tag{1.24}$$

In [4], the isotropic *Matérn* covariance function $k^{\nu}_{\mathrm{Matérn}}(r)$ is introduced with parameter ν set to 5/2. Thus, the resulting function has the following form:

$$k^{\nu=5/2}_{\mathrm{Matérn}}(r) = \sigma_f^2\left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right)\exp\left(-\frac{\sqrt{5}r}{l}\right), \tag{1.25}$$

where

$r = |\boldsymbol{x}_p - \boldsymbol{x}_q|$ is the maximal size of neighborhood.

Having defined the covariance matrix for N points we can extend $\boldsymbol{C}$ for $(N+1)$-th point by the following expression:

$$\boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^{\mathrm{T}} & \kappa \end{pmatrix}, \tag{1.26}$$

where

$\boldsymbol{k} = k(\boldsymbol{x}_{N+1}, \boldsymbol{X}_N)$ is an N-dimensional real vector of covariances between the new point and already known from $\mathcal{D}$,

$\kappa = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1})$ is a variance of the new point.

Note that the exact definitions of $\boldsymbol{k}$ and κ depends on the covariance function to be chosen.

In the end, by combining all those equations together we obtain a final expression for the desired function value [1]:

$$p(t_{N+1} \mid \boldsymbol{X}_{N+1}, \boldsymbol{t}_N) \;\propto\; \exp\left(-\frac{(t_{N+1} - \mu_{N+1})^2}{2\sigma^2_{t_{N+1}}} \right), \tag{1.27}$$

where

$\mu_{N+1} \;= \boldsymbol{k}^{\mathrm{T}} \boldsymbol{C}_N^{-1} \boldsymbol{t}_N$ is the predictive mean.

$\sigma^2_{t_{N+1}} = \kappa - \boldsymbol{k}^{\mathrm{T}} \boldsymbol{C}_N^{-1} \boldsymbol{k}$ is the predictive variance.

1.5.3 Hyper-parameter estimation

The Gaussian processes are able to specify prior information (such as smoothness or characteristic length-scale, etc.) about the function to be modeled into the covariance matrix structure. This can be done by incorporating the hyper-parameters into the covariance function. Thus, the described above squared exponential covariance function can be rewritten as follows:

$$k_{\mathrm{SE}}(\boldsymbol{x}_p, \boldsymbol{x}_q, \boldsymbol{\theta}) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2}{2l^2} \right) + \sigma_n^2 \delta_{pq}, \tag{1.28}$$

where

$\boldsymbol{\theta} = \{\sigma_f^2, l, \sigma_n^2\}$ is the input set of hyper-parameters.

σ_f^2 is the signal variance.

l is the characteristic length-scale.

σ_n^2 is the noise variance.

Varying this parameters we can affect the approximation behavior. For example, considering the objective function to be *noisy*, we can model the covariance function with certain (increased) value of σ_n^2. We can also model the difference between strength of noise and signal by manipulating σ_f^2 and σ_n^2. The σ_f^2 parameter also scales the exponent output. In the end, changing the length-scale parameter l will result in a scaled "x-axis".

The covariance function is often defined by some set of hyper-parameters which should be properly estimated in order to obtain a well-behaving model. Of course, the set can be specified by the user but a more robust way is to estimate it via the maximum likelihood approach or to cross-validate those parameters. Below we briefly review both methods.

1.5.3.1 Maximum likelihood

This method aims to maximize the given function values $\boldsymbol{t}_N$ under a multivariate Gaussian with zero mean and covariance matrix $\boldsymbol{C}_N$, $\{\boldsymbol{C}_N\}_{i,j} = k_{\mathrm{SE}}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{\theta})$. For the computation of this metric we use so-called log-likelihood instead. The term and it's derivatives w.r.t. $\boldsymbol{\theta}$ can be expressed as [1]:

$$\begin{aligned}\mathcal{L} &= \log p(\boldsymbol{t}_N|\boldsymbol{X}_N, \theta) \\ &= -\frac{1}{2}\Big(\log \det \boldsymbol{C}_N + \boldsymbol{t}_N^{\mathrm{T}}\boldsymbol{C}_N^{-1}\boldsymbol{t}_N + N\log 2\pi\Big) \quad \text{and} \\ \frac{\partial\mathcal{L}}{\partial\theta} &= \frac{1}{2}\Big(\boldsymbol{t}_N^{\mathrm{T}}\Gamma_N\boldsymbol{C}_N^{-1}\boldsymbol{t}_N - \mathrm{trace}(\Gamma_N)\Big),\end{aligned}$$

where

$$\Gamma_N \equiv \boldsymbol{C}_N^{-1}\tfrac{\partial \boldsymbol{C}_N}{\partial\theta}.$$

Note that this approach requires time $\mathcal{O}(n^3)$ in order to compute the inversion of the covariance matrix.

There is no guarantee that the landscape of the likelihood function is unimodal. Otherwise, it was shown to be multi-modal in case of GP with several covariation functions [16]. This means that an employment of the global optimization technique may yield better results. However, local maxims here interpret the data in a different way. In case of the model employing SE covariance function the possible solutions can be viewed as:

- a function having complicated landscape but a low or zero noise, or
- a function with a simple structure and a severe noise.

So in case of multiple optima one would consider comparing several solutions according to their posterior probabilities. However, for a training set of sufficient size, certain optimum will tend to have much bigger probability than the others and a comparison may not be necessary. But in this case one should care not to end up in a local optima.

1.5.3.2 Cross-validation

Using the equation 1.22 the predictive log probability for the extracted training sample i and the hyper-parameter set $\boldsymbol{\theta}$ can be expressed as follows [5]:

$$\log p(t_i \mid \boldsymbol{X}, \boldsymbol{t}_{-i}, \boldsymbol{\theta}) = -\frac{1}{2}\log \sigma_i^2 - \frac{(t_i-\mu_i)^2}{2\sigma_i^2} - \frac{1}{2}\log 2\pi, \tag{1.29}$$

where

$\boldsymbol{t}_i$ denotes all points except t_i. Note that mean μ_i and variance σ_i^2 are also computed w.r.t. the i-extracted training set.

The equation above can then be generalized to compute *leave-one-out* (LOO) probability:

$$\mathcal{L}_{\mathrm{LOO}}(\boldsymbol{X}, \boldsymbol{t}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \log p(t_i \mid \boldsymbol{X}, \boldsymbol{t}_{-i}, \boldsymbol{\theta}). \tag{1.30}$$

Having μ_i and σ_i^2 computed, we obtain a performance estimation which can then be optimized w.r.t. model hyper-parameters. This can be performed by computing the $\mathcal{L}_{\mathrm{LOO}}$ partial derivatives and applying the conjugate gradient optimization [5]. The computational expense of the cross validation method thus stands for $\mathcal{O}(n^3)$ which is similar to likelihood approach.

It is interesting to note that the likelihood method tells the probability given by *the assumptions of the model*, while the CV approach estimates the predictive probability regardless of the model assumptions were fulfilled [5].

1.5.4 Merit functions

Using the surrogate model to find the global optimum one *exploits* model's knowledge regarding the unknown objective function. However, it can potentially prevent the optimization algorithm from convergence to an optimal solution due to unexplored regions that were incorrectly modeled. To protect the model from making such predictions there is a need for some mechanism which cares about the *exploration* of new regions in the solution space. Hence, the *merit functions* were introduced precisely for this reason: they incorporate both exploration and exploitation patterns into the model's decision-making mechanism. In the context of Gaussian processes the merit functions f_{M} often exploit the model's standard deviation as a measure of uncertainty about its predictions[14]. An example merit function may have the following form:

$$f_{\mathrm{M}}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}) - \alpha\sigma_t(\boldsymbol{x}), \tag{1.31}$$

where

$\alpha \geq 0$ balances between the exploration and exploitation.

Note that this notation is designed for the minimization problems, so in case of maximization α must be negative.

1.5.5 Summary

The main advantages of the Gaussian Process ability to approximate function can be outlined as follows [1]:

- it does not require any hard-to-estimate predefined structure,
- it approximates rather complicated function landscapes such as discontinuity, multi-modality or non-smoothness,

- it provides set of meaningful hyper-parameters and includes a theoretical framework to efficiently optimize them,
- it can utilize some important information from the supervisor optimization algorithm.

As for drawbacks, using the GP assumes quite high computational demands. The time complexity scales $\mathcal{O}(n^3)$ with the training set[3] and only $\mathcal{O}(n)$ with the problem dimension [1].

The described method has a big potential, especially in its applicability to work in a conjunction with the CMA-ES as a surrogate model. In the next chapter we will pay more attention to employment of different GP derivatives into the CMA evolution strategy.

1.6 Self-adaptive Surrogate-assisted CMA-ES

In this section a we describe a method called self-adaptive surrogate-assisted CMA-ES (^{s*}ACM-ES-k), which is, however, not based on the Gaussian processes. Nevertheless, this approach is present among the list of methods that should be investigated, so we give a brief method description here and then move on to the next chapter.

The technique is based on the early approach to make a surrogate-assisted CMA-ES invariant to both rank-preserving transformations of f and to orthogonal transformations of the search space, called ACM-ES [17]. This method employs the concept of ordinal regression models based on SVM which provides only rankings of the points evaluated on the model in order to achieve the mentioned invariance properties[7]. This method has been later improved in [18] enabling to adapt the model hyper-parameters in an on-line manner. The described technique brings further improvements by means of more intensive exploitation of the model, using much larger population size while operating on surrogate and preserving original population size on the real objective function. The strategy's fundamentals can be found in [19].

The ^{s*}ACM-ES-k algorithm combines two following optimization procedures:

1. Optimization of the objective function f using so-called CMA-ES #1, assisted by a surrogate model $\hat{f}$ with a $\boldsymbol{\theta}$ set of hyper-parameters.
2. Optimization of the surrogate model error $\mathrm{Err}(\boldsymbol{\theta})$ given by $\boldsymbol{\theta}$ set of hyper-parameters and using CMA-ES #2.

In both cases the original CMA-ES operations to update the variables remained unchanged. However, every iteration now consists of the following steps. First,

[3]The most demanding operation here is caused by the computations of inverse $\boldsymbol{C}_N$.

the surrogate model $\hat{f}$ is built and optimized for $\hat{n}$ iterations. Then, the real objective function f is optimized for 1 iteration. After that, the model error $\mathrm{Err}(\boldsymbol{\theta})$ is estimated and the number of generations $\hat{n}$ is adjusted. In the end, the model hyper-parameters are optimized and the $\boldsymbol{\theta}$ is chosen for the next iteration.

1.6.1 Objective function optimization procedure

During this procedure the method employs a generation-based EC scheme and builds a model $\hat{f}$ after a sufficient g_{start} number of generations. The surrogate employed here is built using Ranking SVM so that the model predicts only the ordering of the test points, providing an invariance w.r.t. the rank-preserving transformations of the objective function. In addition, the second invariance property (invariance w.r.t. orthogonal transformations) is preserved using the transformation applied to every (training and so testing) point $\boldsymbol{x}$:

$$\boldsymbol{x}' = \boldsymbol{C}^{(g)^{-1/2}}(\boldsymbol{x} - \boldsymbol{m}^{(g)}). \tag{1.32}$$

When the surrogate model is built, the second optimization procedure takes place. In this case the CMA-ES optimizes the model for a given number of generations $\hat{n}$. According to [18], $\hat{n}$ is adjusted by a linear function inversely proportional to a *global* model error $\mathrm{Err}(\boldsymbol{\theta})$. The global error is constructed (with a certain relaxation) from *local* error on λ most recent points from an archive $\mathcal{D}$ by the following way:

$$\mathrm{Err}(\boldsymbol{\theta}) = \frac{2}{|\mathcal{D}|(|\mathcal{D}|-1)} \sum_{i=1}^{|\mathcal{D}|} \sum_{j=i+1}^{|\mathcal{D}|} w_{i,j} \cdot 1_{\hat{f}_{\boldsymbol{\theta},i,j}}, \tag{1.33}$$

where

$1_{\hat{f}_{\boldsymbol{\theta},i,j}}$ is true if $\hat{f}$ violates the ordering of pair (i, j) given by the real objective function f.

$w_{i,j}$ defines the weights of such violations.

1.6.2 Surrogate error optimization procedure

This procedure is done in the end of every main CMA-ES iteration, where an additional CMA-ES #2 performs one optimization iteration for the model hyper-parameters. Here, the algorithm samples λ_{hyp} different points in a space of hyper-parameters and builds λ_{hyp} surrogate models. Then, those models are evaluated using the $\mathrm{Err}(\boldsymbol{\theta})$ metric and $\mu_{\text{hyp}} = \lfloor \lambda_{\text{hyp}}/2 \rfloor$ best performing are used to update the CMA-ES #2 internal variables. The resulting mean of the hyper-parameter distribution is used to obtain $\boldsymbol{\theta}$ for the next iteration of ^{s*}ACM-ES-k.

All the above makes the algorithm handle its internals almost independently, which enables the user to specify nothing except the range of hyperparameters.

1.6.3 Model exploitation

To intensify the model exploitation (in case if the model provides reasonably good approximations, i.e. when the $\mathrm{Err}(\boldsymbol{\theta}) \approx 0$) the author suggests to estimate a separate local covariance matrix $\boldsymbol{C}_{\mathrm{loc}}$ constructed from a much larger number of points (in order of 10^5). Using $\boldsymbol{C}_{\mathrm{loc}}$ instead of the original $\boldsymbol{C}$ from CMA-ES usually results in a quicker learning of the appropriate function landscape which leads to a faster convergence. However, if it is determined that the model is not accurate enough, the influence of the $\boldsymbol{C}_{\mathrm{loc}}$ is limited by the learning rate in order to prevent a possible degradation of $\boldsymbol{C}$.

In this case, the optimization of $\hat{f}$ is done with the population size $\lambda = k_\lambda \lambda_{\mathrm{default}}$ and the number of parents is chosen as $\mu = k_\mu \mu_{\mathrm{default}}$, where $k_\lambda \geq 1$ and $k_\mu \geq 1$ are adjusted w.r.t. the model error.

CHAPTER 2

GP-based surrogates in the context of CMA-ES

This chapter aims to describe several state-of-the-art surrogate modeling techniques based on the Gaussian process regression. Those techniques are used in the context of CMA-ES. This can be for example the exploitation of the uncertainty measure in order to maintain a global model behavior[14] or an attempt to the decrease the computational complexity of training Gaussian process by employing an ensemble of simpler models instead of the single global model[20]. In this way, we describe such techniques in this chapter.

2.1 Gaussian Process Optimization Procedure

The technique presented here uses the methodology of *merit* functions to address the problem of *exploration vs. exploitation.* The *Gaussian Process Optimization Procedure* described below may perform quite efficient on the standard test functions. It's ability to solve complex optimization problems was also shown on a real-world problem of optimization of compressor profile (for the detailed description of both method and experiment refer to [1]).

2.1.1 Model settings

This technique uses already described, but slightly modified SE covariance function from equation 1.28. The local version of such function reads:

$$k_{\mathrm{SE}}(\boldsymbol{x}_p, \boldsymbol{x}_q) = \theta_1 \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \frac{(x_{p,i} - x_{q,i})^2}{l_i^2} \right) + \theta_2 + \sigma_{pq}\theta_3, \tag{2.1}$$

where

hyper-parameter θ_2 introduces an additional offset of the resulting values from zero.

In addition, the variance vector κ can now be expressed as:

$$\kappa = \theta_1 + \theta_2 + \theta_3.$$

2.1.2 Hyper-parameter estimation

The author of the technique introduces a separate method for the log-likelihood maximization. This is in fact a combination of the CMA-ES and a quasi-Newton gradient (BFGS) method. The CMA strategy is always used for the first computation of the likelihood in order to obtain a global optimum. Then, after additional points is added to the training set, a fast local method (in this case BFGS coupled with line search by golden section) is applied to keep settings up-to-date. To prevent being trapped in a possible local optima, every 10 runs of the local method was followed by one run of the corrective CMA-ES.

In addition, the author suggests several modifications to be put on the data: function values in the training set are normalized to be in range [0, 1] and decision variables to be in [-1, 1]. The hyper-parameters are in turn being constrained to the following bounds:

$$\begin{aligned} \theta_1 &\in [10^{-3}, 1] \quad , \\ \theta_2 &\in [10^{-3}, 1] \quad , \\ \theta_3 &\in [10^{-9}, 10^{-2}] \quad , \\ l_i &\in [10^{-2}, 10], \quad \text{for } i = 1, \ldots, n. \end{aligned}$$

As the ratios of the parameter bounds are quite big, an operation with log-values instead of a real hyper-parameters should be considered. Note also that if the computation of the inverse covariance matrix fails too often, a rise of the θ_3 lower bound can resolve the issue.

2.1.3 Procedure description

The optimization procedure described here is initially based on the surrogate approach. It considers the presence of initial set of training points. Every iteration the GP model is constructed using those points and the hyper-parameters are optimized. Then, an optimization algorithm searches for the model's global minims using several merit functions (discussed in 1.5.4). In the end, the obtained optima are evaluated on the real function and added to the archive.

Author suggests to equip the model with 4 merit functions with α values set to 0,1,2,4 resp. Here, setting α to 0 enforces the complete exploitation of the model knowledge. By contrast, setting $\alpha = 4$ pushes the optimizer onto the unexplored regions.

Next, as the Gaussian processes require quite a lot of computations to train, it may be beneficial to restrict several points from being included in the training set. However, in this case one should not forget about the presentability of the particular training points. In order to keep a reasonable precision, the author suggests to form $\mathcal{D}$ with N_{C} points closest to $\boldsymbol{x}^{\mathrm{best}}$ in the feature space and with N_{R} most recently evaluated points. In addition, to reflect the spread of points one should restrict searching space near the $\boldsymbol{x}^{\mathrm{best}}$. It can be done by specifying the diagonal $\boldsymbol{d}$ of the search hypercube.

2.2 POI MAES

The GP surrogate models are known to provide a variance values as an uncertainty measure for selecting promising individuals. However, it is not always clear when the exploration- or exploitation-oriented behavior should be preferred. Hence, the authors of this technique refined the selection process and introduced a metric that is able to assess the individual according to his ability to improve currently the best solution. The authors refer to this technique as Probability of improvement model-assisted evolution strategy (POI MAES).

The full description of the technique presented here can be found in [14]. Here, the authors use a standard GP framework, particularly the SE covariance function with $n + 2$ hyper-parameters optimized by the likelihood maximization:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q, \boldsymbol{\theta}) = \theta_1 \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \frac{(x_{p_i} - x_{q_i})^2}{r_i^2} \right) + \delta_{pq}\theta_2. \tag{2.2}$$

The authors incorporate the model into the (μ, λ)-CMA scheme, which is done via so-called *pre-selection* method. Hence, the initialization phase goes as usual: population creation, archive creation and the initial training of the model. The concept of the approach requires the algorithm to be modified at the generation loop. At the beginning of each iteration a $\lambda_{\mathrm{pre}} > \lambda$ new individuals are sampled from μ parents and evaluated on the surrogate model by a certain evaluation criterion. Then, λ best ranked offsprings are selected, evaluated on the objective function and added to the Archive. In the end, the surrogate becomes updated. The key aspect of the method is that the preselection criterion must represent the quality of individual w.r.t. a exploration-exploitation trade-off (i.e. has the same purpose as the merit functions from 1.5.4).

The authors name 2 pre-selection criteria used in his paper:

- Mean of model prediction (MMP).
- Probability of improvement (POI).

2.2.0.1 Mean of model prediction

The idea of this metric is rather straightforward. The λ_{pre} individuals are being evaluated using the mean of GP prediction distribution. Then, the λ best candidates are selected according to their values. The surrogate model is thus used to directly replace the real objective function. However, such criterion have very serious disadvantage: in case of multi-modal functions points with high predicted values may mislead the optimizer to false optima, neglecting the whole effort.

2.2.0.2 Probability of improvement

This metric does not require to specify any additional hyper-parameter and hence, is considered to be more promising. The idea of the POI metric is clearly shown in the figure 2.1. For any given solution vector $\boldsymbol{x}$ the model gives the uncertainty as a normally-distributed random variable $Y(\boldsymbol{x}$ with mean $\hat{t}(\boldsymbol{x})$ and standard deviation $\sigma(\boldsymbol{x})$. Then, having $f_{\text{min}} = \min(t_1, \ldots, t_N$ as the current best obtained function value we define some number T so that $T \leq f_{\text{min}}$. Hence, the probability of improvement simply stands for the probability that $Y(\boldsymbol{x}) \leq T$ and can be computed as follows:

$$\text{POI}(\boldsymbol{x}) = \Phi\left(\frac{T - \hat{t}(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \tag{2.3}$$

where

Φ is the normal cumulative distribution function.

Areas with high POI value have big probability to sample a point with value better than f_{min}. Having the standard deviation in the denominator means that the model tends to explore areas with relatively small number of evaluated points. Also, regions with model prediction $\hat{t}(\boldsymbol{x}) \ll f_{\text{min}}$ will have the POI metric close to zero, which encourages the model to search somewhere else. Therefore, the incorporation of the POI metric makes the model to prefer unexplored regions which may be useful while dealing with multi-modal functions.

The assumption above was also experimentally confirmed by the author of the technique. There, the use of POI led to the best results among the criteria to be investigated on the multi-modal functions. However, as the POI tends to sample preferably at unexplored regions, it was outperformed by the MMP

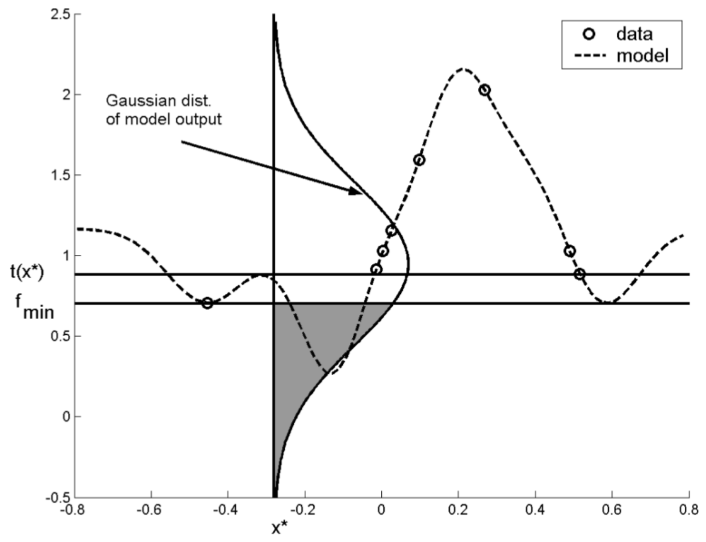


Figure 2.1: Representation of the POI concept applied to the Gaussian process approximation model. The gray area on the graph demonstrates the probability that the objective value sampled at point $\boldsymbol{x}^*$ is smaller than $f_{\min}$.

in case of unimodal functions. Both methods, however, performed better than a standalone CMA-ES.

2.3 Robustness Approximation in GP

The authors of this technique addresses the problem of interconnecting CMA-ES with GP meta-models to find robust solutions. The approach described below directly utilizes the covariance matrix from the CMA evolution strategy in a way that does improve the prediction accuracy without much additional computational cost. It also introduces a special way to select promising solutions from the archive. More details can be found in [21].

2.3.1 Robustness approximations

Finding robust solutions is especially important in the context of costly optimization. One can imagine a robustness approximation as an attempt to approximate function values within noisy or imprecise environment, e.g. due to readout imprecision or perturbations in design variables. The difficulty in such situation is the fact that noisy approximation require an additional neighboring points to be evaluated as well. Considering surrogate-assisted optimization it becomes extremely important to make precise predictions for noisy problems as it requires a certain trade-off between limiting the noise and reducing the evaluations amount.

2.3.2 Surrogate specification

Following the most straightforward way to obtain a required covariance matrix the author uses an isotropic SE covariance function variant from the equation 1.28:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q, \theta) = \exp\left(-\theta \, \|\boldsymbol{x}_p - \boldsymbol{x}_q\|\right). \tag{2.4}$$

The use of function which have only one inner hyper-parameter θ is dictated by high computational demand as the robustness approximations here require construction of several local meta-models (see below). For the same reason the author persuades to use grid search using logarithmically scaled grid on the interval $[10^{-50}, 10^{5}]$ as a method to tune the hyper-parameter. The author also claims that such settings were found to perform reasonably well.

2.3.3 Integration with CMA-ES

The integration process is again performed in every CMA iteration. The robustness approximation is obtained in the way that the local (i.e. model trained only on certain training points) GP model $\hat{f}_k$ is trained for every fresh individual $\boldsymbol{x}_k$. Representative training sets for such models are often within the hypercube $[\boldsymbol{x}_k - \boldsymbol{\sigma}_\xi, \boldsymbol{x}_k + \boldsymbol{\sigma}_\xi]$ in order to handle an uncertainty of the input space.

2.3.3.1 Representatives selection

To fulfill the model with certain training points a selection mechanism must be introduced. The method below attempts to choose n_{krig} pairs $(\boldsymbol{x}, t)$ from the archive $\mathcal{A}$ to shape a training set $\mathcal{D}$. If the sufficient amount of points is not found, the method also returns a set $\boldsymbol{X}_{\text{cand}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$ of length $l = n_{\text{krig}} - |\mathcal{D}|$. The points from $\boldsymbol{X}_{\text{cand}}$ are then evaluated on the real objective function and added to both $\mathcal{A}$ and $\mathcal{D}$.

The difficulty of selecting points from the archive is that a set of points needs to be well spread (e.g. to avoid numerical problems with $\boldsymbol{C}$ creation) on the region of interest (hypercube). In addition, the hypercube selection may be infeasible for highly-dimensional problems as the required amount of points will be unavailable in most regions.

The author of this technique proposes to use a Latin hypercube sampling (LHS) method to obtain the required dataset. First of all, the required n_{krig} *reference* points are generated via the LHS and stored in the reference set $\mathcal{R}$. Then, for every point in $\mathcal{R}$, the closest point from $\mathcal{A}$ is assigned. An important note here is that this point from $\mathcal{R}$ must be the closest one to the other point as well. If this is the case, the the archive point is added to the training set, otherwise such point is admitted a good candidate for sampling and added to $\boldsymbol{X}_{\text{cand}}$. When all points from the archive are assigned, we select an assigned pair (archive point, reference point) of points which are the most distant to each other. Such behavior ensures that the archive is always updated at the region which needs extra representatives the most.

2.3.3.2 Exploitation of the the supervisor covariance matrix

The author also suggests to exploit the covariance matrix adapted by the supervisor optimizer to improve the meta-models approximation quality. As the matrix learned by the CMA-ES represents the local quadratic approximation of the objective function, it seems reasonable to equip local models with such information.

The supervised matrix can be utilized in a way that substitutes distance metric inside the covariance function k with the Mahalanobis distance. So the covariance function now has the following form:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q, \theta) = \exp\,(-\theta\ (\boldsymbol{x}_p - \boldsymbol{x}_q)^{\mathrm{T}} \boldsymbol{C}^{(g)^{-1/2}} (\boldsymbol{x}_p - \boldsymbol{x}_q)). \tag{2.5}$$

The experiments conducted by the authors of the approach showed that the algorithms using Mahalanobis distance metric may provide better more accurate approximations of the local landscape than the Euclidean distance. However, the difference in metrics did not noticeably improved the overall performance.

2.4 GP ensembling

The approach described below aims to cover one of the most disappointing problems of the Gaussian processes - high computational demand - by the means of ensembling techniques. The proposed solution adopts an ensemble of local Gaussian process models sharing the same input parameters. Afterwards, the author ensures that such technique can provide reliable fitness prediction and uncertainty estimation [20].

The usage of the GP framework in this method is quite standard. The only difference is that the author introduces squared exponential covariance function with a vector $\boldsymbol{\theta} \in \mathbb{R}^n$ of hyper-parameters, where n is the dimensionality of points to be sampled. The modified function reads:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q, \boldsymbol{\theta}) = \exp\,(-\sum_{i=1}^{n} \theta_i\ \|x_{p_i} - x_{ki}\|). \tag{2.6}$$

Here, the specified hyper-parameters are optimized by the log-likelihood maximization.

2.4.1 Ensembling scheme

It was demonstrated that the uncertainty information exploitation can maintain the ability to globally optimize the objective functions. However, the main obstacle for training GP models which show global search ability is the need for computing the inverse covariance matrix which restricts one to unpleasant $\mathcal{O}(N^3)$ scaling rate, where N stands for the amount of points in the training set.

To resolve this problem, [20] has proposed to use an ensemble of local GP models. Those models are constructed using N_c closest points to every individual in the current iteration. But in a contrary to local models from 2.3 here the ensemble shares the same deterministic function (usually, constant β) and the set of hyper-parameters.

To be able to use shared parameters properly, several terms must be generalized for multiple GP models. Hence, the changes affect the log-likelihood computing and the estimated values of β and σ. Those variables (except the likelihood) now have the arithmetical mean form, as shown below:

$$\mathcal{L} = \sum_{i=1}^{\lambda} \mathcal{L}_i, \tag{2.7}$$

$$\hat{\beta} = \sum_{i=1}^{\lambda} \frac{\beta_i}{\lambda}, \tag{2.8}$$

$$\hat{\sigma}^2 = \sum_{i=1}^{\lambda} \frac{\hat{\sigma}_i^2}{\lambda}, \tag{2.9}$$

where

λ is the number of offsprings in the current generation.

Consider a computational effort of training standard GP model to be $\mathcal{O}(N^3)$. Hence, the cost of training of such ensembles becomes $\mathcal{O}(N_c^3\ \lambda)$. Note that in this case $N = N_c \times \lambda$, which gives us a quite significant speed-up (in terms of model-caused computations). Generally, the described approach can be treated as simplified global GP model, considering though the correlation effect only of N_c closest points for each offspring.

2.4.2 Model incorporation

The incorporation process is rather straightforward. The initial setting done at the beginning of the algorithm are set by the following way:

- Sample and evaluate initial archive of $11 \times D$ training points using LHS method.
- Initialize $\mu = 5$, $\lambda = 25$, $C_n = 6$ and search from the current best solution.

The other settings of the algorithm were left default.

2.4.3 Prescreen strategies

Unlike the strategies described in 1.4 the author uses several so-called *prescreen* strategies to effectively select and evaluate promising individuals. Below we briefly describe those.

Best selection

The *best selection* (BS) strategy chooses C_n individuals with the best predicted fitness value and evaluates them on the real objective function.

Clustering technique

The *clustering technique* (CT) splits the whole offsprings into C_n clusters. Then, it selects individuals closest to each cluster centroid and evaluates them.

Lower confidence bound

The *lower confidence bound* (LCB) strategy computes a separate metric called confidence bound for every individual as $t_{lb} = \hat{t} - 2\hat{s}$. The individuals marked for costly evaluation are selected from ones having the best LCB value.

The remaining 2 strategies are made using combinations of the techniques described above. Hence, in CTBS strategy the individuals with best fitness predictions in appropriate clusters are evaluated. Analogically, in CTLCB strategy we select an individual with the best LCB value for each cluster.

In the end, experiments on author's benchmark have shown that the best results (especially for multi-modal functions) were achieved by the CTLCB strategy.

2.5 EGO-CMA

In this section, an attempt to combine the CMA-ES and an Efficient Global Optimization is made. The key aspect of this approach is to exploit EGO's high initial performance and, consequently, switch to more robust (and slow) CMA-ES at the region found by EGO. So in this section the both strategies will be briefly specified as well as the steps required to combine both strategies and metrics used to detect the switch point. We also give a brief description of the EGO algorithm here. To find out more about the EGO approach mentioned here, refer to [15]. Also, refer to [22] for the complete description of the surrogate.

2.5.1 Efficient global optimization

The efficient global optimization approach also uses the concept of surrogate-assisted optimization to deal with expensive multi-modal problems. The principle of such algorithm is (compared to 1.4) to use the uncertainty information $v(\boldsymbol{x})$ obtained from the model to balance visiting either attractive (low $\hat{f}(\boldsymbol{x})$) or unexplored (large $v(\boldsymbol{x})$ value) regions. One can conclude that using Gaussian process meta-models is a good candidate to participate in EGO as it directly provides an uncertainty measure by means of the predicted variance. To keep the surrogate-model up-to-date it is being retrained as the new "balanced" point becomes evaluated on the real objective function.

2.5.2 Combining EGO and CMA

The optimization algorithm described here is quite unique as it does not affect the insides of the CMA-ES iterations but provides advantageous starting parameters so that the convergence rate is (hopefully) being accelerated. The motivation is explained in a way that EGO becomes especially efficient in the early stage (diversification) of the optimization process but loses the pace in the later phases. The CMA strategy is expected to steadily converge (however, with a lower speed) to the optimum and hence is efficient during the intensification phase.

2.5.2.1 Turning point selection

The provided EGO-CMA algorithm initially explores the search space with EGO, then, as the switch occurs, it is being replaced by the CMA-ES in order to converge to optimum. The switch point is suggested to perform if the best observed function value f^{best} did not improved for at least $0.1 \times$ *budget* of function evaluations and if one of the following conditions is met:

- $0.5 \times$ *budget* evaluations has been spent, or
- $\overline{EI} < 0.01 \times (f_{\text{DoE}}^{\text{best}} - f^{\text{best}})$, where $\overline{EI}$ stands for the average of the maximum expected improvement over the last 5 iterations, while $f_{\text{DoE}}^{\text{best}}$ and f^{best} are the best function values obtained in the initial design of experiments (DoE) phase and the current best value respectively.

When the switch point detected, a point with the best function value becomes a starting point for the CMA-ES, in other words $\boldsymbol{m}^{(0)} = \boldsymbol{x}^{\text{best}}$. In addition, the CMA strategy obtains the trained Kriging model as an approximation to the true objective function.

2.5.2.2 The transfer of control

Now we set up the covariance matrix $\boldsymbol{C}^{(0)}$ and the step size $\sigma^{(0)}$. First of all the second order Taylor expansion is performed on the GP objective function approximation at point $\boldsymbol{x}^{\mathrm{best}}$ to obtain the function representation:

$$f_{\boldsymbol{H}}(\boldsymbol{x}) = m(\boldsymbol{x}^{\mathrm{best}}) + \nabla m(\boldsymbol{x}^{\mathrm{best}})^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^{\mathrm{best}}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{\mathrm{best}})\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{\mathrm{best}}). \quad (2.10)$$

The initial covariance matrix for the CMA strategy is then set to the inverse of the Hessian of the Kriging mean at point $\boldsymbol{x}^{\mathrm{best}}$:

$$\boldsymbol{C}^{(0)} = \boldsymbol{H}^{-1}. \quad (2.11)$$

Note that if the $\boldsymbol{H}$ matrix is not positively definite we need to force the convexification of $f_{\boldsymbol{H}}$. Hence, the eigendecomposition ($\boldsymbol{H} = \boldsymbol{B}\boldsymbol{D}^2\boldsymbol{B}^{\mathrm{T}}$) is performed so that the negative eigenvalues in $\boldsymbol{D}^2$ are substituted by the 10^{-6}. In addition, the author suggests to improve the condition number of the resulting matrix by adding a positive value to the main diagonal of the Hessian matrix.

Considering that CMA-ES optimizes a spherical function $g(\boldsymbol{t})$ inside the t-space ($\boldsymbol{t} = \boldsymbol{D}\boldsymbol{B}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*_{\boldsymbol{H}})$), we express the average initial step length as a distance to optimum $\boldsymbol{x}^*_{\boldsymbol{H}}$:

$$\sigma\sqrt{n - 0.5} = \|\boldsymbol{D}\boldsymbol{B}^{\mathrm{T}}(\boldsymbol{m} - \boldsymbol{x}^*_{\boldsymbol{H}})\|, \quad (2.12)$$

where

$\sqrt{n - 0.5}$ is an expectation of the χ^2_n random variable, n stands for dimensionality of the input vector.

Finally, by the minimization of $f_{\boldsymbol{H}}$ we obtain an optimum approximation $\boldsymbol{x}^*_{\mathrm{H}}$ and hence, using the equation 2.12, an initial step length as:

$$\sigma^{(0)} = \frac{\|\boldsymbol{D}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{H}^{-1}(\boldsymbol{x}^{\mathrm{best}})\nabla m(\boldsymbol{x}^{\mathrm{best}})\ \|}{\sqrt{n - 0.5}}. \quad (2.13)$$

2.6 Surrogate CMA-ES

In [4], several methods were proposed in order to optimize costly black-box functions. The paper included methods based on Random forests and Gaussian processes. However, we omit methods based on the Random forests as it is beyond the scope of this thesis and focus only on the GP-based approach.

2.6.1 Covariance matrix for GP

The authors define a matrix $\boldsymbol{K}_N \in \mathcal{R}^{N \times N}$ of N training points which contains covariances for all pairs of points in training set:

$$\{\boldsymbol{K}_N\}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{\theta}), \quad for\ i, j = 1, \ldots, N$$

This approach employs the Matérn covariance function, as described in 1.5.2.

The covariance matrix $\boldsymbol{C}_N$ is constructed as:

$$\boldsymbol{C}_N = \boldsymbol{K}_N + \sigma_N^2 \boldsymbol{I}. \tag{2.14}$$

The remainder parameters are quite similar to ones explained in the equation 1.28. The described model obtains the hyper-parameter set $\boldsymbol{\theta} = \{\sigma_f^2, l, \sigma_n^2\}$ tuned by using the maximum-likelihood method.

2.6.2 Model employment

The described method uses generation-based evolution control strategy to incorporate the model into the CMA-ES. The author also reported that the experiments with analogous method with generation-based EC were not successful.

The integration process of the GP is quite straightforward. The only changed point of the original CMA-ES is the sampling and fitness-evaluation part at every generation. Whenever the ongoing generation is marked as original-evaluated and the whole population is evaluated on the expensive objective function. The pairs $(\boldsymbol{x}, y)$ are then added to so-called *archive* $\mathcal{A}$, containing only points evaluated on the real function. The archive is used to select (if possible) n_{REQ} *promising* solutions to shape a training set $\mathcal{D}$. Here, promising means that those points are not farther from the CMA mean value $\boldsymbol{m}^{(g)}$ than r. The training set selection process can be defined as follows:

$$\mathcal{D} \leftarrow \{(\boldsymbol{x},\ y) \in \mathcal{A} \mid (\boldsymbol{m}^{(g)} - \boldsymbol{x})^{\mathrm{T}} \sigma^{(g)} {\boldsymbol{C}^{(g)}}^{-1/2} (\boldsymbol{m}^{(g)} - \boldsymbol{x}) \leq r\}, \tag{2.15}$$

where

$\boldsymbol{C}^{(g)}$ is the CMA-ES covariance matrix at generation g.

When there is enough points to create a training set, the GP model is trained on $\mathcal{D}$ and the next generation is marked as model-evaluated and every fitness evaluation during such generation is simply made on the model. An important notice here is that there is no model-provided fitness evaluation smaller (better) than the best from $\mathcal{A}$. And in case that there is not enough points, the process is repeated at the next generation, which is, in its turn, marked as original-evaluated.

Chapter 3

Performance analysis

Here we analyze the performance of the models described in the previous chapter. First of all, we describe the framework created to perform surrogate model benchmarking and discuss our contribution: embedding of the new benchmarking testbed. In the end we outline the experimental setup for all methods to be investigated and discuss the obtained results.

3.1 Existing framework

In this section we describe a framework used to benchmark the surrogate models. This is in fact a combination of two smaller sub-frameworks from different sources ensuring algorithms integration, flexible experiment initialization, CPU time allocation and handling the whole benchmarking procedure including some post-processing tools. Below we briefly describe both frameworks.

3.1.1 Surrogate CMA-ES framework

This part of the global framework was created in order to provide a composition interface to the domain of the surrogate-assisted black-box optimization. It is implemented using the *Matlab* ecosystem and the *Statistics and Machine Learning* toolbox [23]. The framework uses CMA-ES as a base optimizer and several kinds of surrogate models. The CMA-ES source code was originally obtained from [8] and then transformed to the S-CMA-ES version from [4], allowing easy modification and further development of the mentioned technique. However, this framework also supports an integration of the surrogate techniques based on the original CMA-ES and written in Matlab. For details concerning operation with this software refer to [24]. An important notice here is that the framework uses the supplied benchmarking testbed to assess the desired methods. The testbed and its content will be outlined later in 3.1.2.

3.1.1.1 Optimizer structure

In order to run experiments we need to create an optimizer (optimization algorithm with integral surrogate model) object with the following structure:

1. `fitfun`: a fitness function used to evaluate the algorithm. This function must provide the following interface:

   ```
   y = fitness(x, varargin)
   ```

2. `xstart`: an algorithm initial point. This point also determines the problem dimension n.
3. `inopts, SurrogateOptions`: structures with configuration variables passed directly to the algorithms. `inopts` adjusts the CMA-ES settings while the `SurrogateOptions` is an optional structure which determines initial settings for the desired surrogate model.

In the end of the experiment the following parameters can be obtained from the framework:

- `xmin`: a search point with the minimum obtained fitness value.
- `fmin`: a function value of `xmin`, i.e. `fitfun(xmin) = fmin`.
- `counteval`: the number of function evaluations spent by the algorithm run.
- `y_eval`: numeric matrix with the best objective values at the first column and the respective function evaluation number in the second column.

3.1.1.2 Running the experiment

The framework provides a functionality to initialize experiments, which resulting produces a group of tasks determined by the objective function, dimensionality and other possible options (e.g. surrogate model distance metric). Combinations of those parameters are then encoded as unique `task_id` and the tasks can then be run as a separate Matlab process on the local machine or be submitted to the Czech national computational cluster *Metacentrum*.

In order to run some task on the local machine (or on the Metacentrum front-end node) one can use the function `metacentrum_task_matlab()` which has the following syntax:

```
metacentrum_task_matlab(exp_id, exppath_short, task_id, varargin)
```

where

`exp_id` denotes the experiment unique name.

`exppath_short` specifies the path to the experiment directory.

`task_id` is the mentioned unique number representing desired function and dimensionality.

`varargin` is the optional argument passed to the optimizer instance.

In order to use the computational power of the Metacentrum and to gain the benefit of parallel computation one can call script `metacentrum_master_template.sh` from shell with the following parameters:

```
metacentrum_master_template.sh EXPID META_QUEUE [ID1] [ID2]...
```

where

`EXPID` is the unique experiment name (analogous to `exp_id` above).

`META_QUEUE` is the name of the Metacentrum queue with possible values as `2h`, `4h`, `1d`, `2d`, etc. The meaning of this parameter is the upper bound to the algorithm's expected computational time.

`[ID1] [ID2]` are the optional arguments representing a certain `task_id` to be run. If there are no id given, the whole set of tasks will be submitted for computation.

Note that tasks submitted for Metacentrum do not require the Matlab license because of being compiled using the Matlab Compiler Runtime (MCR).

3.1.2 BBOB framework

To run experiments within the surrogate CMA-ES framework one should also ensure the presence of the black-box optimization benchmarking (BBOB) testbed. This testbed is a source of benchmarking functions of many kinds. It handles the whole interaction between the optimizer and the black-box function including logging of the results. However, the aim of this thesis is to measure the performance on the other benchmark testbed. Thus, we describe only general ideas of the BBOB framework, as the desired functions still need to be introduced into the BBOB framework. One can obtain the full description of the framework itself and functions related to it from [25] or `http://coco.gforge.inria.fr/`.

3.1.2.1 Concepts of the framework

The core concepts of the BBOB framework can be outlined as follows:

- An algorithm operating in the search space of dimensionality 2, 3, 5, 10, 20 or 40. However, the dimensionality 40 is optional and can be omitted.
- The search domain is defined everywhere in $\mathbb{R}^n$ and have the global optimum in $[-5, 5]^n$. In addition, most functions present in the framework have their global optima in $[-4, 4]^n$, which can be considered a reasonable setting for the initial search point.
- Indication of the testbed to be used. Different algorithms may perform well on a certain kind of benchmark functions and fail on the others. The whole testbed is divided into 2 main categories containing either *noisy* and *noiseless* functions. The noiseless testbed has the 24 functions divided into the following subcategories: separable functions, functions with low or moderate conditioning, functions with high conditioning and unimodal, multi-modal functions with adequate global structure and, in the end, multi-modal functions with weak global structure. The noisy testbed contains 30 functions divided into the following subcategories: functions with moderate noise, functions with severe noise, highly multi-modal functions with severe noise. Note here that most functions in the BBOB framework are in fact unimodal.
- The final function precision ($\Delta f = 10^{-8}$) in order to effectively detect global optimum breakthrough and to set up the termination criteria.
- The algorithm's performance is determined over a several trials (i.e. trials with fixed function and dimensionality) in order to obtain a statistically significant measures. The performance is evaluated over the $Ntrials = 15$ trials. Note, that the number of trial is also used to initialize the seed for the random number generator determining some special benchmark parameters.
- Different measurement scenarios are possible in order to evaluate the algorithm performance. Two basic scenarios are shown in the figure 3.1. Results obtained by the fix-cost (vertical) view can be interpreted in a way that the algorithm converges faster/slower for the certain budget of function evaluations. The results obtained by the fixed-target (horizontal) view can be used to conclude how much one algorithm is faster/slower than the other in terms of time required to converge to a certain distance from the optimum.

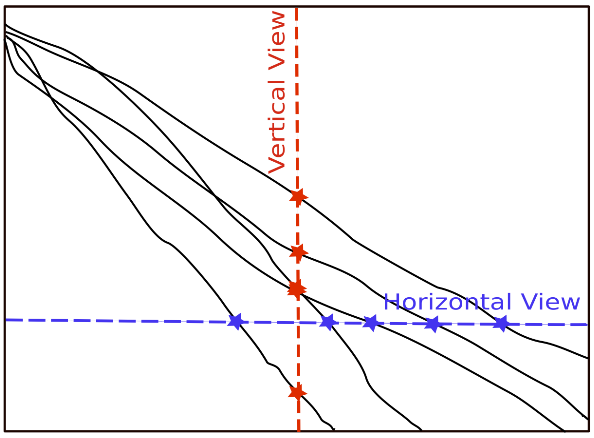


Figure 3.1: Two different scenarios of the performance representation drawn for several algorithm runs.

3.2 CEC functions

In this section we give a brief description of the functions which are used to measure the performance of the methods described in chapter 2. Unlike the testbed mentioned in 3.1.2, we conduct our experiments on a different set of benchmark functions, introduced on the special session for multi-modal function optimization during the Congress of Evolutionary Computation (CEC) in 2013. In order to define an explicit name we refer to those functions as CEC functions. To obtain a complete description of the functions being used and overall benchmark capabilities refer to [6].

In the end of the section we also specify necessary actions, which should be performed in order to incorporate desired functions into the BBOB framework. The resulting implementation of the CEC functions is compatible with an existing BBOB test suite and so can be subsequently treated as an integral part of this framework.

3.2.1 Testbed description

The mentioned benchmarking testbed includes totally 12 different functions. Note also that all the functions listed here are, unlike the BBOB, multi-modal. However, the testbed is designed in a way that the dimensionality and variable ranges are specific for a particular function. The dimensionality is always a subset of (1, 2, 3, 5, 10, 20), while the variable ranges can widely differ. Also, all the functions are formulated as the maximization problems.

An important note here is the fact that the existing BBOB framework does not accept 1D problems, which prevents several CEC functions from participation in the benchmarking procedure. Hence, we will not describe such functions in the future, focusing only on 9 multi-dimensional variants.

These are 5 *standard* functions (f_4 to f_8) and 4 so-called *composition* functions ($f_9 - f_{12}$).

The following part aims to review standard functions only. However, after the description of the composition procedure in 3.2.1.2, we return to the remaining 4 functions in 3.2.1.3.

3.2.1.1 Standard functions

The basic CEC functions are described in the following way:

- f_4: **Himmelblau (2D)** - Fig. 3.2a

  $$f_4(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2.$$

  The Himmelblau function, which is in fact an inverted Himmelblau[6], has 4 global optima with no local optima, as shown in the figure 3.2a. The input variables x and y are in range $[-6, 6]$.

- f_5: **Six-Hump Camel Back (2D)** - Fig. 3.2b

  $$f_5(x, y) = -4\Big(\Big(4 - 2.1x^2 + \frac{x^4}{3}\Big)x^2 + xy + (4y^2 - 4)y^2\Big).$$

  The function has both 2 global and 2 local optima, while the input variables are bounded to: $x \in [-1.9, 1.9]$, $y \in [-1.1, 1.1]$.

- f_6: **Shubert (2D, 3D)** - Fig. 3.2c

  $$f_6(\boldsymbol{x}) = -\prod_{i=i}^{n}\sum_{j=1}^{5} j\ cos\Big((j + 1)x_i + j\Big).$$

  This function is an inverted Shubert function, where there are $N3^n$ global optima divided into 3^n groups. The variables are constrained to: $x_i \in [-10, 10], \quad \text{for } i = 1, \ldots, n.$

- f_7: **Vincent (2D, 3D)** - Fig. 3.2d

  $$f_7(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} sin\Big(10\ log(x_i)\Big).$$

  The inverted Vincent function has 6^n global optima, but, unlike the regular distances between optima in f_6, f_7 employs vastly different spacings between them. In addition, it does not have any local optima. The input variables varies in range $[0.25, 10]$ for every dimension.

Function instance		
Function number	Dimensionality	Optimum value
f4	2D	200.0
f5	2D	1.03163
f6	2D	186.731
	3D	2709.0935
f7	2D	1.0
	3D	
f8	2D	-2.0

Table 3.1: Estimated global optima values for 5 simple CEC functions.

- f_8: **Modified Rastrigin - all global optima (2D)** - Fig. 3.2e

  $$f_8(\boldsymbol{x}) = -\sum_{i=1}^{n}\Big(10 + 9\ cos(2\pi k_i x_i)\Big).$$

  This version of the Rastrigin function is modified in a way that it has $\prod_{i=1}^{n} k_i$ global optima without any local one. The figure shows an f_8 example given by $\boldsymbol{k} = (3, 4)$, which results in 12 global optima. All the input variables are in range $[0, 1]$.

In addition, the authors estimated global optima values for every simple benchmark function. These values can be obtained from the table 3.1:

3.2.1.2 Composition process

Here we present the general idea of constructing the multi-modal composition functions. A n-dimensional composition function can be obtained as a combination of simple functions f from some pool of basic composition units. All the functions taken from this pool can be either shifted in space, transformed by some linear transformation matrix or used as it is. The authors propose the basic pool to be consisted of the following functions [4]:

- Sphere function,
- Griewank function,
- Rastrigin function,
- Weierstrass function,
- Expanded Griewank plus Rosenbrock function (EF8F2).

[4]For the sake of brevity we do not give exact definitions of those functions, since the complete description is available in [6]

Note that the pool of basic used to obtain a composite may be of the same type but with different properties and parameters.

Now we define a composition function CF_j as a weighted aggregation of m functions as follows:

$$\mathrm{CF}_j(\boldsymbol{x}) = \sum_{i=1}^{m} w_i \Big(\hat{f}_i((\boldsymbol{x} - \boldsymbol{o}_i)/\lambda_i \cdot \boldsymbol{M}_i) + \mathrm{bias}_i \Big) + f^j_{\mathrm{bias}}, \tag{3.1}$$

where

$\hat{f}_i$ denotes the normalization of the i-th basic function.

w_i is the corresponding function weight.

$\boldsymbol{o}_i$ is the new shifted optimum.

λ_i is a parameter used to stretch ($\lambda_i > 1$) or to compress ($\lambda_i < 1$) the function.

$\mathbf{M}_i$ is the linear transformation (rotation) matrix.

Each composition function has two bias parameters. The former, bias_i, defines the function value offset for each pool function and thus denotes which optimum is the global one. The later, f^j_{bias}, determines the global optimum function value. The authors set both parameters to zero in order to obtain a function with several global optima all with the fitness values equal to zero. The function weights w_i are obtained by the three-stage computation procedure defined as:

$$w_i = \exp\Big(- \frac{\sum_{k=i}^{n} (x_k - o_{i,k})^2}{2n\sigma_i^2} \Big), \tag{3.2}$$

$$w_i = \begin{cases} w_i, & \text{if } w_i = \max(w_i) \\ w_i(1 - \max(w_i)^{10}) & \text{otherwise} \end{cases} \tag{3.3}$$

$$w_i = w_i / \Big(\sum_{j=1}^{m} w_j \Big), \tag{3.4}$$

where

σ_i^2 controls the coverage range of each pool function. The authors recommend to use small values to provide narrow coverage ranges.

In the end, to obtain a better mixture of the pool functions the authors define the following normalization scheme:

$$\hat{f}_i(\boldsymbol{x}) = C \, \frac{f_i(\boldsymbol{x})}{|f^i_{\max}|}, \tag{3.5}$$

where

$C = 2000$ is a predefined constant.

$f^i_{\max}$ is estimated using $f^i_{\max}=f_i((\boldsymbol{x}^*/\lambda_i)\boldsymbol{M}_i)$ with $\boldsymbol{x}^* = [5, 5, \dots, 5]$.

3.2.1.3 Composition functions

Finally, we define remaining 4 composition functions as follows:

- f_9: **CF1 (2D)** - Fig. 3.2f

  The composition function 1 uses 6 pool functions (2 different versions of each Griewank, Weierstrass and Sphere functions) resulting in a 6 global optima bounded to the optimization hypercube [-5,5]. During the composition following parameter set was used:

$$\begin{aligned} \boldsymbol{\sigma} &= \mathbf{1}, \quad \text{where } \mathbf{1} \text{ is a unit vector,} \\ \boldsymbol{\lambda} &= [1,\ 1,\ 8,\ 8,\ 1/5,\ 1/5], \\ \boldsymbol{M}_i &= \mathbf{I}, \quad \forall i = 1, \dots, m. \end{aligned}$$

- f_{10}: **CF2 (2D)** - Fig. 3.2g

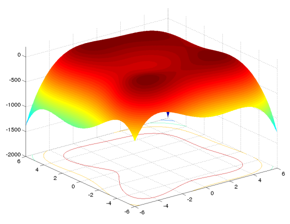
  The composition function 2 consists of 8 pool functions combined in the way similar to CF1, but using Rastrigin, Weierstrass, Griewank and Sphere functions. The input variable bounds and special parameters were left unchanged, except the $\boldsymbol{\lambda}$ adjusted to [1, 1, 10, 10, 1/10, 1/10, 1/7, 1/7].

- f_{11}: **CF3 (2D, 3D, 5D, 10D)** - Fig. 3.2h

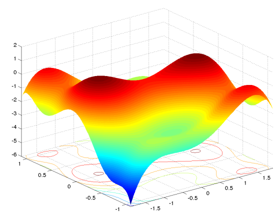
  The CF 3 consists of 6 pool functions ($\times 2$ EF8F2, $\times 2$ Weierstrass, $\times 2$ Griewank) in the optimization same box. The parameter $\boldsymbol{\sigma}$ was set to [1, 1, 2, 2, 2, 2] and $\boldsymbol{\lambda}$ to [1/4, 1/10, 2, 1, 2, 5]. As $\boldsymbol{M}_i$ were chosen different linear transformation matrices with condition number one.

- f_{12}: **CF4 (3D, 5D, 10D, 20D)** - Fig. 3.2i

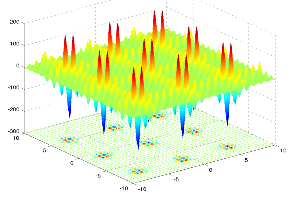
  The CF 4 is based on the combination of 8 pool functions ($\times 2$ Rastrigin, $\times 2$ EF8F2, $\times 2$ Weierstrass, $\times 2$ Griewank) in the same optimization box. The parameter $\boldsymbol{\sigma}$ was set to [1, 1, 1, 1, 1, 2, 2, 2] and $\boldsymbol{\lambda}$ to [4, 1, 4, 1, 1/10, 1/5, 1/10, 1/40]. As $\boldsymbol{M}_i$ were chosen different linear transformation matrices with condition number one. Note that the 2D version of this function from figure 3.2i is only referential as we do not measure performance on this function for dimensions lower than 3 [6].



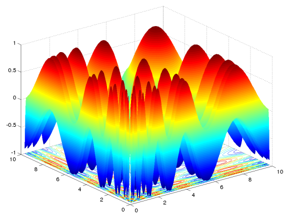
(a) Himmelblau



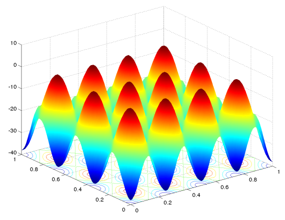
(b) Six-Hump Camel Back



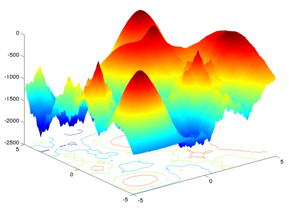
(c) Shubert



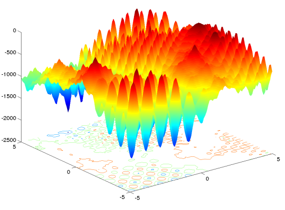
(d) Vincent



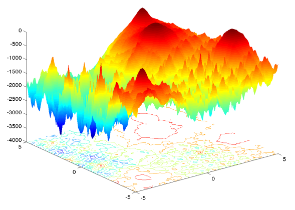
(e) Modified Rastrigin



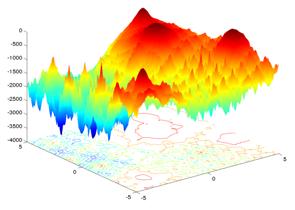
(f) CF 1



(g) CF 2



(h) CF 3



(i) CF 4

Figure 3.2: Plots of 9 multi-dimensional CEC functions used to extend the BBOB framework. All the functions are drawn in 3D so that each plot represents the function landscape of 2 input variables. Composition functions $1-4$ are shown under the *CF* abbreviation.

3.3 Implementation notes

In this section we give several brief notes to our contributions to the frameworks mentioned above. We integrated 9 mentioned above CEC functions into the BBOB framework. We based our implementation on the source code obtained from the authors of the paper. Hence, the original source code is available at `https://github.com/mikeagn/CEC2013`. Several key changes have been made in order to satisfy the BBOB requirements. Below we briefly describe our contribution to this field.

The most obvious correction concerns the fact that the CEC functions were designed as maximization problems. To handle this we transform the maximization task to the minimization using the following equation:

$$\max(f(\boldsymbol{x})) = -\min(-f(\boldsymbol{x})). \tag{3.6}$$

Hence, every value obtained by the evaluation of the target CEC function is then multiplied by the -1.

Next, the variable bounds vary for different functions and are not generally consistent with the framework structure. The BBOB constrains the optimizer in range $[-5, 5]^n$ [25], hence, to overcome this issue we rescale the former search space so it agrees with the given function constraints. Consider the original domain is constrained to the range $[\boldsymbol{l},\ \boldsymbol{u}]$, $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$ and the function expects the candidate solution $\boldsymbol{x}$ to be in the new range $[\boldsymbol{l}_{\text{new}},\ \boldsymbol{u}_{\text{new}}]$, $\boldsymbol{l}_{\text{new}}, \boldsymbol{u}_{\text{new}} \in \mathbb{R}^n$. Then, every solution $\boldsymbol{x}$ is rescaled according to the following equation:

$$x_{\text{new}i} = (x_i - l_i)\frac{u_{\text{new}i} - l_{\text{new}i}}{u_i - l_i}, \quad \forall i = 1, \ldots, n. \tag{3.7}$$

In our case $\boldsymbol{l} = \boldsymbol{-5}$ and $\boldsymbol{u} = \boldsymbol{5}$, so the equation 3.7 can be rewritten as:

$$x_{\text{new}i} = \frac{1}{10}(x_i - 5)(u_{\text{new}i} - l_{\text{new}i}), \quad \forall i = 1, \ldots, n. \tag{3.8}$$

We also implemented several GP-based surrogate methods using Matlab. Our implementations are based on the original CMA-ES source code (Matlab version 3.62) obtained from `https://www.lri.fr/~hansen/cmaes_inmatlab.html`. We also used the Matlab's `fitrgp` framework in order to train Gaussian process regression models. During the implementation of the Robustness CMA-ES described in 2.3 we partially reused the source codes from the author's website (`http://natcomp.liacs.nl`).

3.4 Experimental setup

In this section we define the way how we conduct the experiments. But first of all, there is a trouble that must be outlined before we proceed. By the time of acquaintance to this work the whole set of surrogate-modeling techniques was expected to be investigated and benchmarked. However, it turned to be impossible since the implementations of most methods did not simply exist while the others approaches had certain troubles with the stability[5]. After several unsuccessful attempts to fix the issues it was decided to exclude unstable methods and concentrate on the implementation of the claimed ones. Unfortunately, this idea turned out to be a challenge too. Due to a time consuming understanding of the domain we haven't managed to finish the implementation of the GPOP approach. Finally, we selected all the reliably performing

[5]By the stability we mean the presence of critical errors in the obtained code.

methods and compared them between each other and, in addition, with the surrogate-free CMA-ES. As a consequence, we examined the following set of methods: S-CMA-ES[4], ^{s*}ACM-ES-k[19], POI MAES[14], Robust CMA[21] and the classic CMA-ES[8].

Below we outline settings used to measure the performance of the mentioned techniques. Since every surrogate approach performs within the CMA-ES environment, it is reasonable to unify it's settings for all methods to achieve more accurate results. The final settings used for the benchmarking reads as:

- **CMA-ES** (described in 1.2)

  As a consequence to the integration into the framework described in 3.1 we follow the CMA-ES settings taken from [4]. More precisely, we use the Matlab version of the IPOP-CMA-ES with the number of restarts $= 4$, $IncPopSize = 2$, $\sigma_{\text{start}} = \frac{8}{3}$, $\lambda = 4 + \lfloor 3\ logn \rfloor$ and setting the remaining parameters to it's defaults[25].

- **S-CMA-ES** (described in 2.6)

  In our experiments we used the S-CMA-ES with the following parameters: the distance $r = 8$, the covariance function $k_{\text{Matérn}}^{\nu=5/2}$ was used with starting values $(\sigma_n^2, l, \sigma_f^2) = \log\ (0.01, 2, 0.5)$, n_{orig} was set to $\lceil 0.1\lambda \rceil$.

- **^{s*}ACM-ES-k** (described in 1.6)

  The mentioned algorithm does not require to explicitly set any special hyper-parameter.

- **POI MAES** (described in 2.2)

  We set the $\lambda_{\text{pre}} = 3\lambda$ and the amount of points to train the model to 2λ.

- **Robust CMA** (described in 2.3)

  For the Robust CMA we left the n_{krig} to be equal to $2n$ while the m parameter was set to 10.

3.5 Results assessment

In this thesis we compared the performance of two recently proposed surrogate-modeling techniques against two different existing approaches. All the methods have been described in 2. The experiments were performed on the whole set of benchmarking functions from [6], employed by the author within the larger BBOB framework. In this thesis we use the idea from [26][6] to obtain aggregated results suitable for comparison. We first describe this concept and then return to the assessment.

[6]This paper was submitted to the PPSN 2016 and was in fact unpublished during the whole period of writing this thesis.

3.5.1 Representation tools

In order to get insights of the actual algorithm convergence speed we exploit the algorithm's distance from the global optimum given by the budget of available function evaluations. The use of such representation enables us to determine if the optimum was reached and how far from this was the algorithm in the opposite case. The simple distance Δ_f from the optimum given by the fixed FEs budget $\mathfrak{b}$ may be explained in terms of BBOB notation [25] as:

$$\Delta_f^{\mathfrak{b}} = f_{\text{opt}} - f_{\text{best}}^{\mathfrak{b}}, \tag{3.9}$$

where

f_{opt} is the function's global optimum.

$f_{\text{best}}^{\mathfrak{b}}$ depicts the best objective value found during $\mathfrak{b}$ evaluations.

Example outputs obtained by application of the described method are shown in the figure 3.3 and 3.4. According to the BBOB specification we rerun every method for $Ntrials = 15$ times in order to obtain statistically significant results [25]. Hence, every technique is identified by the empirical median Δ_f^{med} (solid lines) and the space between the first and third quartiles (translucent area). Note that the figures mentioned above use logarithmic y-axis to distinguish the orders of magnitude while converging to the optimum. This simple method lets us register whether the algorithm has reached the optimum within the specified precision.

The method described above, however, does not allow to average results over a certain combination of functions, dimensions, etc. To allow better comparisons we employ the scaled logarithm of the Δ_f, where the budget $\mathfrak{b}$ is normalized by the dimensionality of the search space n. Note also that for the sake of convenience we reassign the term D (which can be sometimes treated as training set size) to be equal to n. So, we compute the desired logarithm Δ_f^{log} from medians Δ_f^{med} over $Ntrials$ trials:

$$\Delta_f^{\text{log}} = \frac{\log\Delta_f^{\text{med}} - \Delta_f^{\text{MIN}}}{\Delta_f^{\text{MAX}} - \Delta_f^{\text{MIN}}} \log_{10}(1/10^{-8}) + \log_{10}10^{-8}, \tag{3.10}$$

where

Δ_f^{MIN} and Δ_f^{MAX} are the lowest and the biggest log Δ_f^{med} values obtained by the methods being studied for the particular benchmark function f, dimension D and normalized budget $\mathfrak{b} = 250$ FEs$/D$.

The described method can then be used to obtain aggregated performance measure for any combination of function and dimensionality. However, we cannot directly determine if the algorithm has actually reached the optimum. Plots from figures 3.5, 3.6, 3.7 describe the mentioned metric. Note that we employ only medians here.

3.5.2 Discussion

We have compared performance of the CMA-ES, S-CMA-ES, ^{s*}ACM-ES-k with the POI MAES and Robust CMA-ES. The comparison is done in a two-stage manner. During the first phase we analyze the algorithm's convergence rates and outline speed-up potential w.r.t. a standalone CMA-ES. Note that in the first stage we analyze the results obtained from independent experiments on our benchmarks. In the second phase we concentrate on the comparison of surrogate modeling techniques by aggregating the results only over functions or dimensions. We also present the CMA-ES results for convenience. In the end we discuss the global performance by aggregating over the entire CEC testbed.

3.5.2.1 Comparison using independent experiments

The performance shown on the particular CEC functions and dimensions is caught by the figures 3.3 and 3.4. Note that we referred to the ^{s*}ACM-ES-k as s*-ACM-ES for all figures. In can be seen that the methods from [4] and [19] show the best performance on the majority of benchmark functions. Those methods converge steadily, reaching the global optimum in most cases and providing a significantly accelerate the classic CMA-ES. However, for f_7 in 3D (^{s*}ACM-ES) or f_{12} in 5D the models may stumble and mislead the optimizer converging to the false optima [7]. The methods does not noticeably speed up the CMA-ES showing comparable performance for f_6 in both 2D and 3D. From the graphs mentioned above we conclude that the S-CMA-ES and ^{s*}ACM-ES methods can be considered as direct competitors. However, we cannot currently nominate the winner because the both methods often end up in a global optima and show comparable convergence curves. Nevertheless, the S-CMA-ES converge in the earlier stages of the experiment and often has lower interquartile ranges (colored areas) which may indicate a more stable behavior.

The methods implemented by the author of the thesis haven't shown such high performance rates as the algorithms above. Unfortunately, these techniques led to a deceleration of the CMA-ES convergence speed in most cases. In addition, the global optimum remains undiscovered in most cases for Robust CMA and (except f_7, f_4 and f_5) for POI MAES. However, in case of more complicated f_6 where the other approaches get stuck, POI MAES clearly becomes

[7] According to the first quartiles we can still observe the occasional upturns

the best performing method. Also, this method was shown to outperform the classic CMA-ES on 20 dimensional f_{12} which may indicate the ability to effectively replace CMA-ES for higher dimensions.

The Robust CMA method was recognized as the worst performing algorithm. It has so low convergence speed that it has never reached the f_{target} during the experiments. This can be interpreted in a way that the benchmark functions were initially designed to be noiseless. On the other hand, the described method expects the objective function to be noisy and hence, provides every fitness prediction with multiple objective evaluations within the range close to the desired point. However, while the most methods become confused on the f_6 and degrade at the local optima the Robust CMA-ES converges faster and, which is more interesting, with an invariable speed. We hence treat such behavior as a sign of robustness appeared even on noiseless problems. However, to make an unambiguous assertion one should additionally investigate the robustness approximations on the benchmark of noisy functions.

3.5.2.2 Experiments aggregation

In this section we have combined the results from our experiments through the certain function or/and dimension. Hence, the figure 3.5 depicts the results aggregated over different dimensions for every suitable function. Note that description of the averaging technique is given earlier in subsection 3.5.1). Next, we averaged the results of the methods for every dimension over all functions. The aggregated graphs can be obtained from the figure 3.6. In the end we have combined all the experiments conducted on the CEC testbed together in the single graph shown in the figure 3.7.

Using the combined results we can conclude that S-CMA-ES and ^{s*}ACM-ES have the fastest convergence rates for the majority of functions and dimensions. However, it should be noted that the S-CMA-ES is the fastest at the beginning of the experiment (till approx. 75FEs/D). In the middle phase (approx. $75 - 125$FEs/D) the algorithm slows down (e.g. for functions f_{11}, f_{12} and dimensions higher than 2) and often gets stuck at some false optimum for a while. But, despite that, the algorithm often converges to a global optimum clearly outperforming a classic CMA-ES.

According to the aggregated results it becomes clear that the ^{s*}ACM-ES performs slightly worse, but still at the comparable level. It converges fast at the beginning (however, not so fast) but does not have such strongly marked stagnation period (especially for f_{11} and for dimensions 5 and 10) which makes the algorithm the fastest at the middle phase. However, the algorithm reaches the global optimum less frequently, which is relevant for functions f_7, f_{12}.

In reference to POI MAES we say that it shows comparable (but always slower convergence) performance to the classic CMA-ES. We can see that for low-dimensional problems POI MAES finally converges (at 250 FEs/D) close to the other algorithms (outperforming CMA-ES and ^{s*}ACM-ES for 3D). We

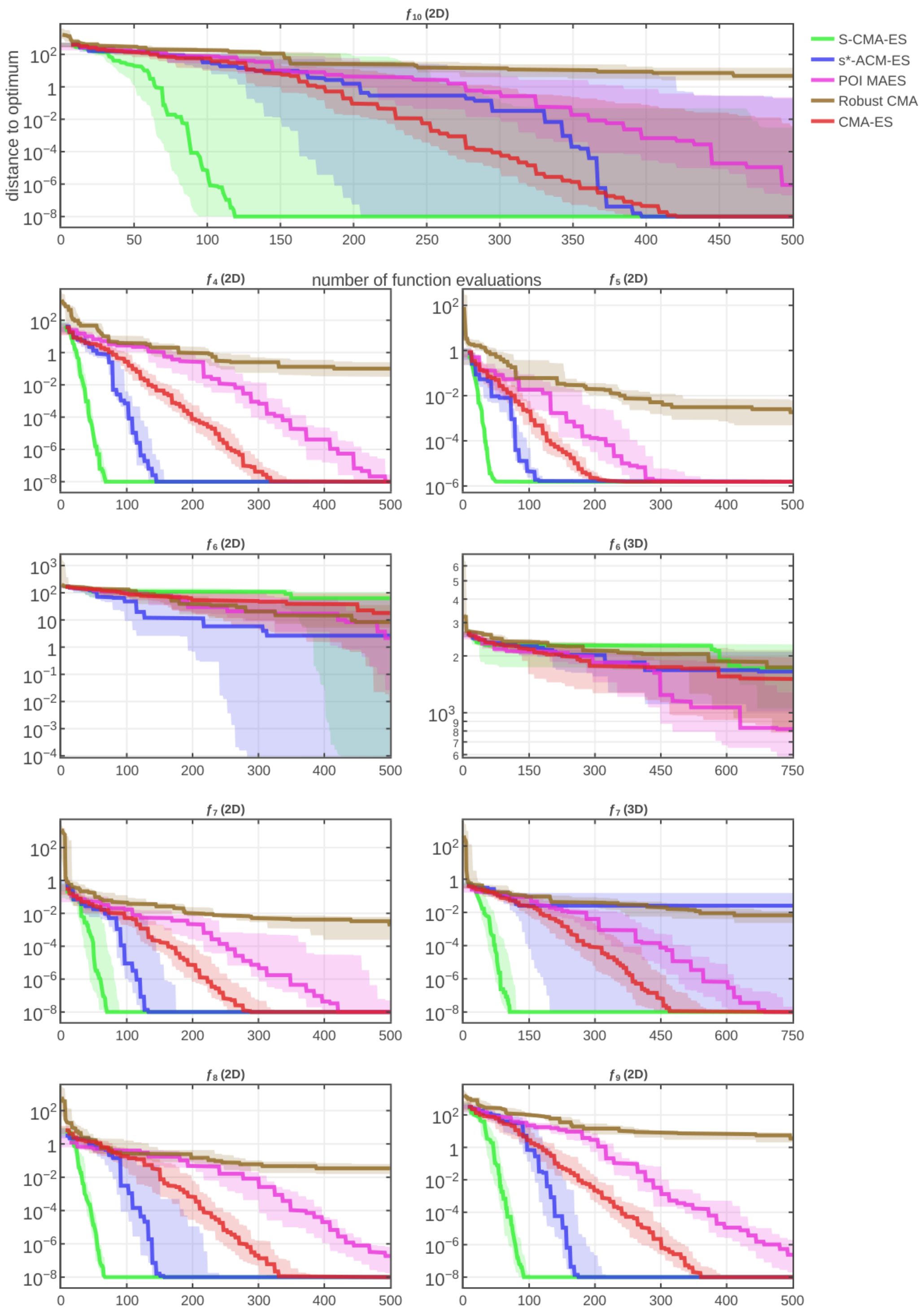


Figure 3.3: Convergence curves (median, first and third quartiles) computed from the particular algorithm runs for all feasible combinations of CEC functions $f_4 - f_{10}$ and dimensions $n = \{2, 3, 5, 10, 20\}$.

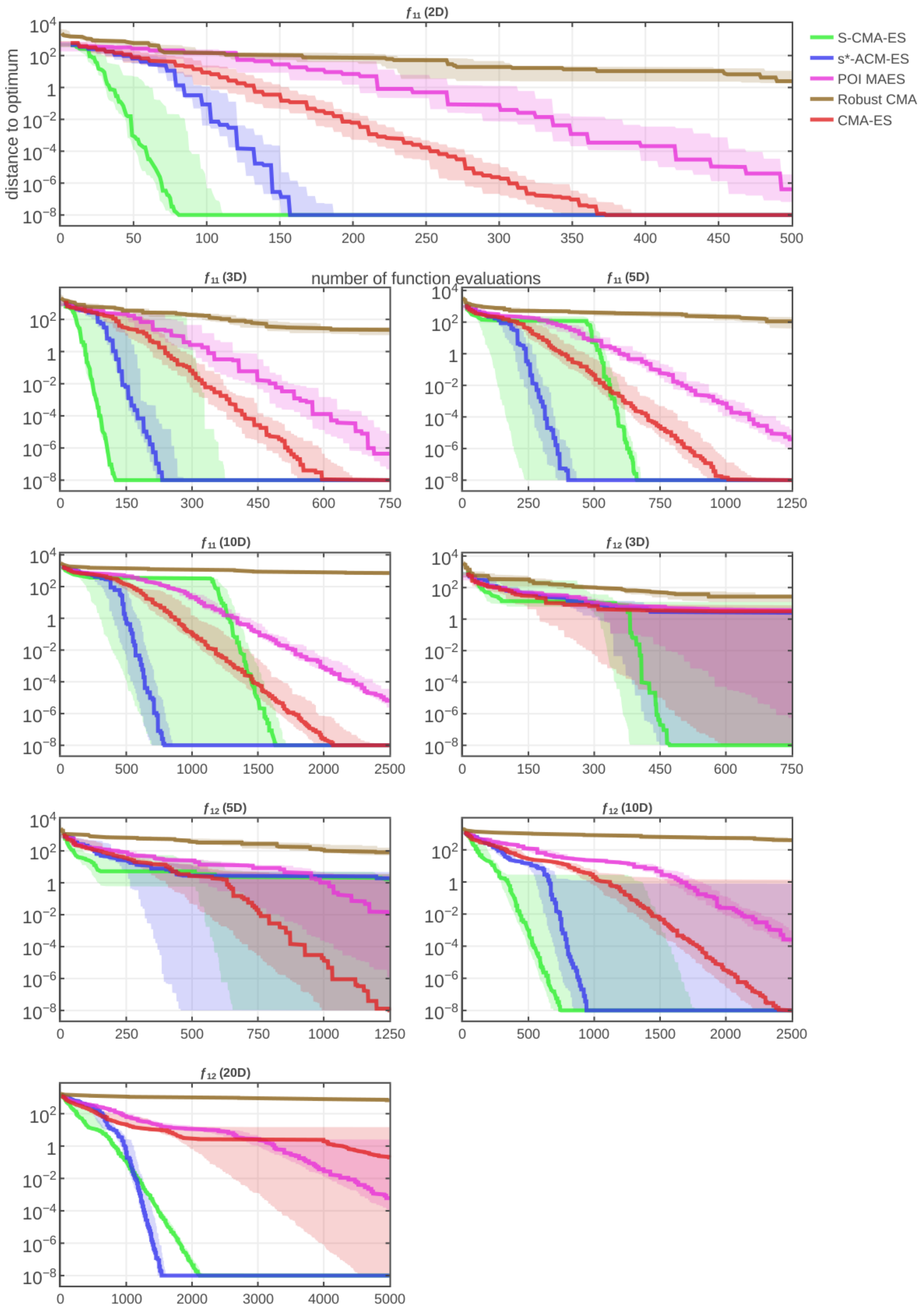


Figure 3.4: Convergence curves (median, first and third quartiles) computed from the particular algorithm runs for all feasible combinations of CEC functions f_{11}, f_{12} and dimensions $n = \{2, 3, 5, 10, 20\}$.

also observed an interesting case when the POI MAES outperformed CMA-ES on 20-dimensional f_{12}. We address such behavior to the fact that POI-based approach tends to explore areas with high model uncertainties which is useful for multi-modal problems[14], especially in case of high dimensionality, where the classic CMA-ES is not capable to learn the sufficient landscape area. Also, slower convergence rates may be explained by the fact that the model requires more time to move from exploration phase to exploitation.

We also believe that the POI metric itself can be used to speed up the CMA-ES technique, because of a reasonable utilization of the uncertainty criterion without introducing any new hyper-parameters [14]. However, one can consider the method described in 2.2 to be not flexible enough. The issue is that the model selects 2λ most recent evaluated points for training. A better strategy for the selection of a reasonable training set may be required in order to obtain better performing algorithm.

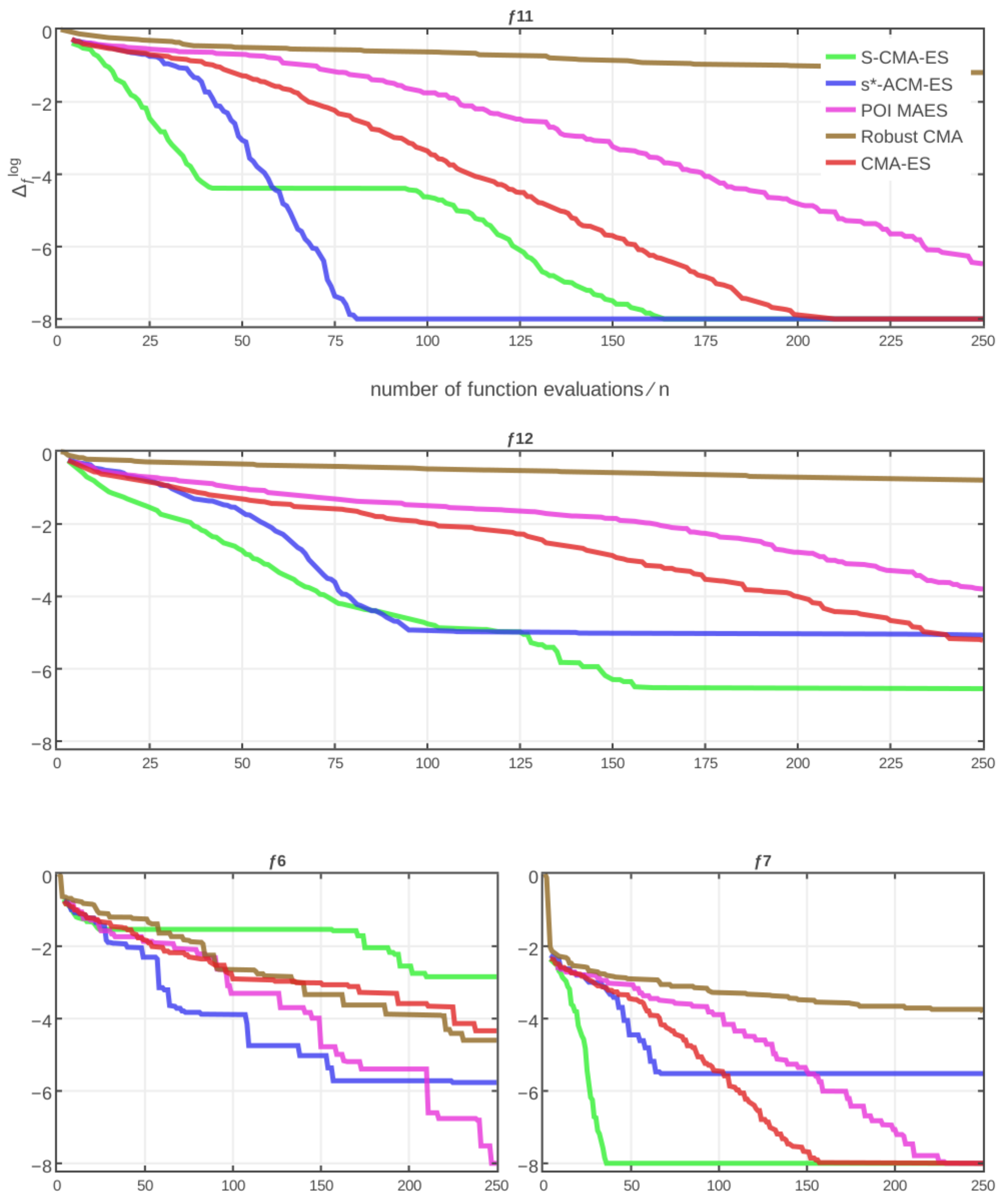


Figure 3.5: Scaled logarithms of the empirical medians (Δ_f^{med}) depending on FEs/D. The graphs show the benchmark results on f_{11}, f_{12}, f_6 and f_7 averaged over all feasible dimensions.

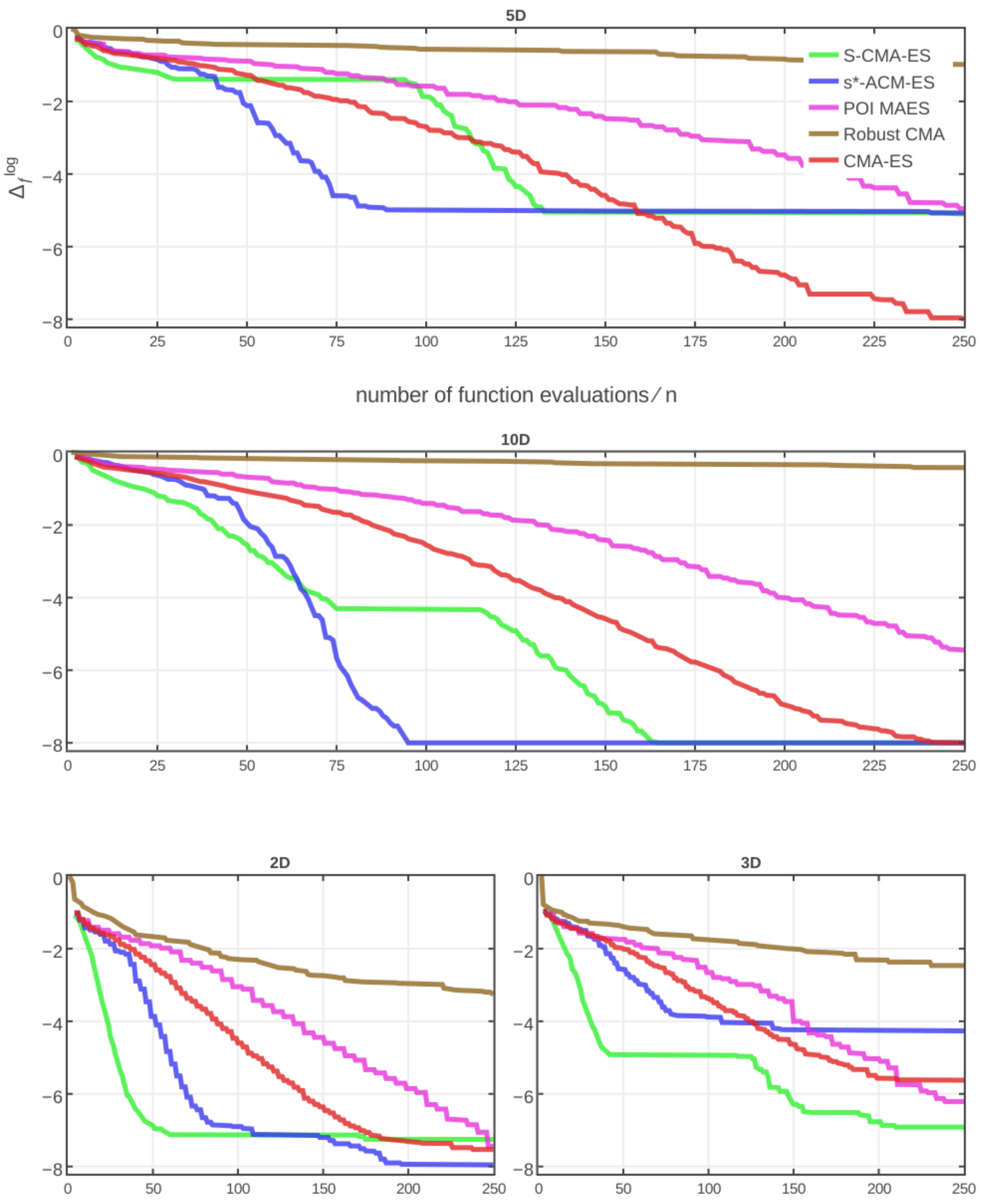


Figure 3.6: Scaled logarithms of the empirical medians (Δ_f^{med}) depending on FEs/D. The graphs show the benchmark results achieved by averaging all functions defined in 5D, 10D, 2D and 3D.

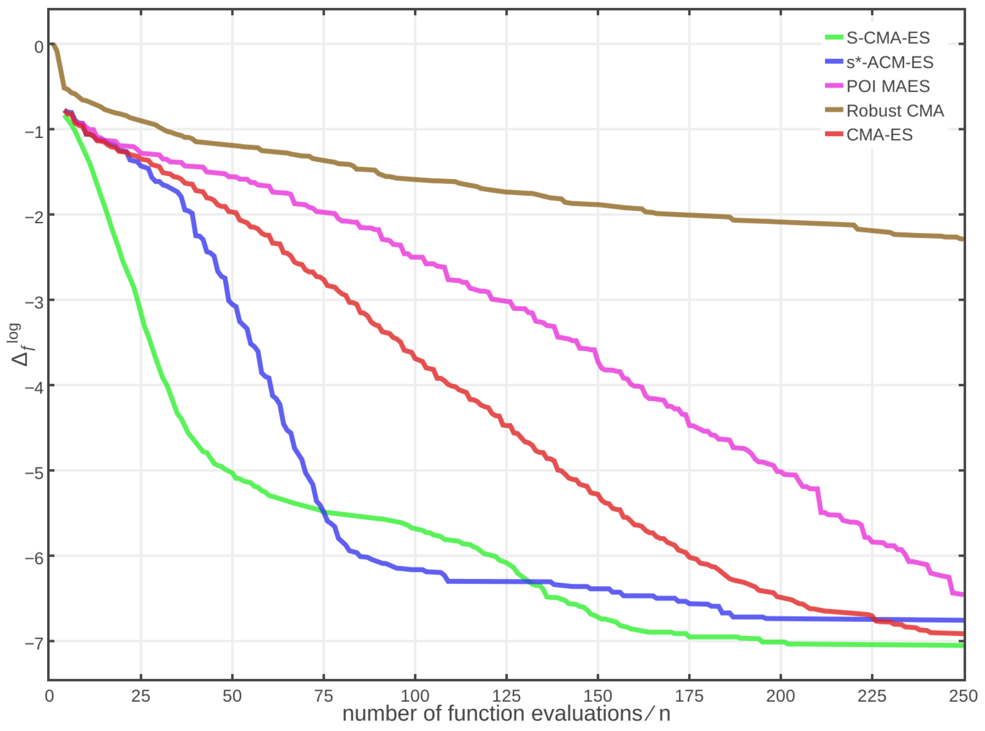


Figure 3.7: Scaled logarithms of the empirical medians (Δ_f^{med}) depending on FEs/D. The results are summarized over all CEC functions at all dimensions.

Conclusion

In this thesis we have reviewed several state-of-the-art surrogate modeling approaches to continuous black box optimization. All those approaches, based on the concept of Gaussian processes, were used to replace the expensive objective function evaluations into the CMA evolution strategy. Subsequently, we have selected several existing methods for the performance comparison. Unfortunately, we haven't managed to employ all the existing methods because of insufficient implementation quality. In addition, we have also implemented several methods from the list mentioned above and compared them to the already existing ones.

For the benchmarking purpose we have implemented the testbed from [6] within the surrogate-modeling [4] and benchmarking [25] frameworks. As a result, we have compared 5 approaches: primary CMA-ES [8] without any surrogate model, S-CMA-ES [4], ^{s*}ACM-ES [19], POI MAES [14] and Robustness approximation CMA-ES [21].

During the experiments it was shown that the best performing approach for small evaluation budgets ($\sim$75FEs/D) appears to be S-CMA-ES, clearly outperforming the standard CMA-ES in most cases. However, it is being outperformed by the ^{s*}ACM-ES-k for budgets of size $75-125$FEs/D. For the larger evaluation budgets the former approach again performed the best. The POI MAES technique has shown a slightly worse performance comparing to CMA-ES with rare improvements. However, we believe that the method can be further enhanced by introducing a new selection strategy. The last method has shown inadequate performance in our experiments. We also remark that the method was designed to perform in noisy environment. However, as the experiments on noisy benchmarks were beyond the scope of this thesis, we cannot uniquely state that the method is not suitable for surrogate-assisted black box optimization.

Bibliography

[1] Büche, D.; Schraudolph, N. N.; Koumoutsakos, P. Accelerating evolutionary algorithms with Gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, volume 35, no. 2, 2005: pp. 183–194.

[2] Chaput, J. C.; Szostak, J. W. Evolutionary optimization of a nonbiological ATP binding protein for improved folding stability. *Chemistry & biology*, volume 11, no. 6, 2004: pp. 865–874.

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

[4] Bajer, L.; Pitra, Z.; Holeňa, M. Benchmarking gaussian processes and random forests surrogate models on the BBOB noiseless testbed. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2015, pp. 1143–1150.

[5] Rasmussen, C. E.; Williams, C. K. *Gaussian Processes for Machine Learning*. the MIT Press, 2006, ISBN 026218253X.

[6] Li, X.; Engelbrecht, A.; Epitropakis, M. G. Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization.

[7] LOSHCHILOV, I. G. *Surrogate-Assisted Evolutionary Algorithms*. Dissertation thesis, Ecole Doctorale d'Informatique, ED 427, Université Paris Sud 11, 2013.

[8] Hansen, N. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, Springer, 2006, pp. 75–102.

[9] Hansen, N.; Ros, R.; Mauny, N.; et al. PSO Facing Non-Separable and Ill-Conditioned Problems. [Research Report], 2008.

[10] Beyer, H.-G.; Finck, S. HappyCat–A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail. In *Parallel Problem Solving from Nature-PPSN XII*, Springer, 2012, pp. 367–376.

[11] Auger, A.; Hansen, N. A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, IEEE, 2005, pp. 1769–1776.

[12] Hansen, N. Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, ACM, 2009, pp. 2389–2396.

[13] Jin, Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, volume 9, no. 1, 2005: pp. 3–12.

[14] Ulmer, H.; Streichert, F.; Zell, A. Evolution strategies assisted by Gaussian processes with improved preselection criterion. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 1, IEEE, 2003, pp. 692–699.

[15] Jones, D. R.; Schonlau, M.; Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, volume 13, no. 4, 1998: pp. 455–492.

[16] MacKay, D. J. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, volume 168, 1998: pp. 133–166.

[17] Loshchilov, I.; Schoenauer, M.; Sebag, M. Comparison-based optimizers need comparison-based surrogates. In *Parallel Problem Solving from Nature, PPSN XI*, Springer, 2010, pp. 364–373.

[18] Loshchilov, I.; Schoenauer, M.; Sebag, M. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, ACM, 2012, pp. 321–328.

[19] Loshchilov, I.; Schoenauer, M.; Sebag, M. Intensive surrogate model exploitation in self-adaptive surrogate-assisted cma-es (saacm-es). In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, ACM, 2013, pp. 439–446.

[20] Lu, J.; Li, B.; Jin, Y. An evolution strategy assisted by an ensemble of local gaussian process models. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, ACM, 2013, pp. 447–454.

[21] Kruisselbrink, J. W.; Emmerich, M.; Deutz, A. H.; et al. A robust optimization approach using Kriging metamodels for robustness approximation in the CMA-ES. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, IEEE, 2010, pp. 1–8.

[22] Mohammadi, H.; Le Riche, R.; Touboul, E. Making EGO and CMA-ES Complementary for Global Optimization. In *Learning and Intelligent Optimization*, volume 8994, Springer, 2015, pp. 287–292.

[23] The Mathworks Inc., Natick, Massachusetts. *MATLAB version 8.6.0.267246 (R2015b)*. 2015.

[24] Bajer, Lukáš and Pitra, Zbyněk. S-CMA-ES framework tutorial. [Cited 2016-5-5]. Available from: `https://github.com/bajeluk/surrogate-cmaes`

[25] Hansen, N.; Auger, A.; Finck, S.; et al. Real-parameter black-box optimization benchmarking 2010: Experimental setup. *INRIA*, 2010, <inria-00462481>.

[26] Bajer, L.; Pitra, Z.; Holeňa, M. Doubly Trained Evolution Control for the Surrogate CMA-ES, submitted to PPSN 2016.

Appendix A
Acronyms

BBOB Black box optimization benchmarking

CEC Congress on evolutionary computation

CF Composition function

CMA-ES Covariance matrix adaptation evolution strategy

CSA Cumulative step length adaptation

EC Evolution control

ES Evolutionary strategy

EGO Efficient global optimization

FE (Objective) function evaluation

GP Gaussian process

GPR Gaussian process regression

LHE Latin hypercube sampling

LCB Lower confidence bound

MAES Model-assisted evolution strategy

MCR Matlab compiler runtime

MMP Mean of model prediction

POI Probability of improvement

SE Squared exponential (covariance function)

SVM Support vector machines

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ....................................... the directory of source codes
  surrogate-cmaes ......... the directory of the framework source codes
  thesis ............... the directory of LaTeX source codes of the thesis
text ........................................ the thesis text directory
  thesis.pdf .......................... the thesis text in PDF format
  thesis.ps ............................ the thesis text in PS format
```