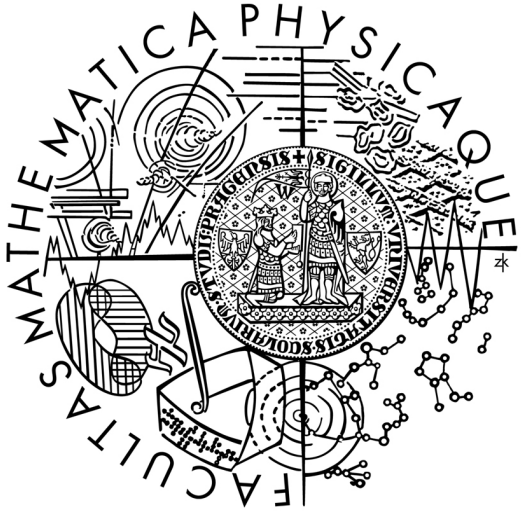
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Kateřina Márová

Optimization using derivative-free and metaheuristic methods

Department of Numerical Mathematics

Supervisor of the master thesis: Petr Tichý

Study programme: Numerical and Computational Mathematics

Study branch: Matrix Computations

Prague 2016

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague on May 1, 2016 signature of the author

Title: Optimization using derivative-free and metaheuristic methods

Author: Kateřina Márová

Department: Department of Numerical Mathematics

Supervisor: Petr Tichý, Department of Numerical Mathematics

Abstract: Evolutionary algorithms have proved to be useful for tackling many practical black-box optimization problems. In this thesis, we describe one of the most powerful evolutionary algorithms of today, CMA-ES, and apply it in a novel way to solve the problem of tuning multiple coupled PID controllers in combustion engine models.

Keywords: black-box optimization, evolutionary algorithms, tuning of PID controllers

I am very grateful to all colleagues at Ricardo. To Steve, who came up with the project idea. To managers Jiří and Lukáš who approved the whole experiment and provided much encouragement. To Michal, Adam and Bohumil, who mentored me on controllers. To WAVE support team Martin and Jiří, who answered hundreds of my questions. To Vladimír, who gave me valuable advice on programming in Python and, sitting next to me, had to hear all my swearing. And to all others in the Software department for being a great team.

I would also like to thank my supervisor Petr Tichý, who was willing to share with me the adventure of discovering new horizons.

Last but not least, I want to thank my boyfriend, who was willing to listen to hours of my jibber-jabber regarding work and did not go crazy.

Contents

Introduction

Black-box optimization problems are fairly common in engineering. In such cases, there is no information about the objective function provided (like smoothness, convexity or number of local minima), nor is it explicitly given, for example when taking the form of a finite-element model. Traditional optimization methods cannot be used here and so-called evolutionary methods are applied instead. Introduction to evolutionary computing is the objective of chapter one.

Then we focus on one of the most powerful evolutionary algorithms of the day: the Covariance Matrix Adaptation Evolution strategy. Its design is described in detail in chapter two.

In chapter three, the reader is presented with a real-world hard optimization problem – tuning multiple coupled PID controllers within combustion engine models. To the author's knowledge, it seems that this problem, on this level of complexity, has not been resolved yet, despite the practical need. We show how this problem may be formulated mathematically and how we apply and tune the algorithm described in the previous chapter. We conclude with the method's verification on real engine models.

1. Hard optimization problems

In this work, we focus on single-objective global numerical optimization of a function $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}$, further called the objective function [5]. Without loss of generality, under optimization we shall always understand minimization.

In addition, the only requirement upon the objective function is that we can evaluate the objective function $f = f(x)$ in any point of its domain $x \in \mathcal{D}$. This allows us to deal with functions that appear in many real-world problems, typically as numerical simulations. Clearly, in such cases derivatives do not exist, are too costly to compute or their numerical computation is of little value (e.g. when the objective function is noisy). Either way, we are forced to deal with a problem hidden in a black box that we cannot further analyze – the so called black-box optimization problem.

To complicate the task even further, we are tight on the time budget. That is, the number of evaluations of the objective function is limited and usually required to be as little as possible. This is a very pragmatic requirement, since it is not a rare case when the objective function is represented by a numerical model that takes significant time to be computed. Therefore, when it comes to comparing algorithms, the most important measure of its quality is the number of function evaluations required to reach given precision. Compared to that, computational costs of the algorithm itself are usually negligible.

1.1 The optimization goal

The functions that arise in applications may have very unpleasant properties. Usually, they are not separable, i.e. the optimization problem cannot be transformed into a series of one-dimensional subproblems. Multimodality, i.e. having multiple local or even global optima, is also a common feature. They may be ill-conditioned or noisy or even have singularities at some points.

As a consequence, it makes little sense to use the classical definition of a global minimum when setting the optimization goal. The minimum may not exist (e.g. $f(x) = x$ for $x \in \mathcal{D} = (0, 1)$), it may not be unique (e.g. $f(x) = \sin(x)$ for $x \in \mathcal{D} = \mathbb{R}$) or it may be impossible to approach in practice (e.g. $f(x) = x^2$ for $x \in \mathbb{R} \setminus \{1\}$ and $f(1) = -1$).

Therefore, we use the approach common in the measure theory and consider classes of functions (rather than functions themselves). Two functions f and g belong to the same class if and only if they are equal everywhere except on a set of zero measure, i.e. $\mu\big(\{x; f(x) \neq g(x)\}\big) = 0$, for μ Lebesgue measure on $\mathcal{D}$. Hereafter, we shall not distinguish between a function and its class $[f]$. Also, by minimum we shall further understand the essential infimum, which is generalization of the term to the class of functions as defined above. More precisely, we shall not distinguish between the minimum of the function class $[f]$ and the essential infimum of any function of this class $g \in [f]$.

Definition. *Essential infimum.*

$$\operatorname{essinf} f = \sup \{b \in [-\infty, \infty], \mu\big(\{x \in \mathcal{D} : f(x) < b\}\big) = 0\},$$

where μ is the Lebesgue measure on $\mathcal{D}$.

The optimization goal is to approach the point of global minimizer

$$x^* \in \overline{\mathcal{D}},\ f(x^*) = \operatorname{essinf}_{x\in\mathcal{D}} f(x).$$

The necessary condition is that there is nonzero probability of reaching points arbitrarily close to x^*, whose function values realize arbitrary closeup to the value of essential infimum [5].

For simplicity, we further assume uniqueness of the global minimizer. Having multiple global optima, we may simply divide $\mathcal{D}$ into subdomains, each one corresponding to a single global optimum. In practice, we usually do not care which one of multiple global optima we reach, as long as it is feasible.

Let us stress out that by global convergence we will always mean convergence to a global optimum (as opposed to the definition used in the context of deterministic optimization: convergence to a local optimum independently of the starting point).

1.2 General approach to the problem

Having stated the problem, we can see that the traditional approach to optimization, also called local search, fails here. First and foremost, looking for a global optimum requires different tools than searching for a local optimum. Second, there are no derivatives available and neither do we approximate them.

Therefore, algorithms discussed in this thesis are derivative-free and make use of randomization. The range of possibilities is vast and the number of algorithms countless, see e.g. [7].

For example, rigorous mathematical approach lead professor Powell to create the interpolation-based class of algorithms (e.g. NEWUOA [47]). Other algorithms, so called metaheuristic or evolutionary, are often inspired by nature. Even though they are perfectly functional, there is little theory to support them. And there are many other options, e.g. purely stochastic methods or neural networks.

1.3 Evolutionary computing

When we look at the world around us, we realize that all objects, living or not, optimize. Atoms and molecules tend to organize themselves to reach the state with minimal energy – they form crystals. Ants developed a smart strategy to look for food using pheromones, eventually finding the optimal path to the source. Animals choose mates to maximize their offspring's chances for survival. There are very many optimization algorithms to be found in nature and they are an endless source of inspiration.

The most obvious optimization method found in nature is the principle of evolution. Generation after generation, individuals are tested by their environment. The fittest ones (the most stable in case of inanimate objects) reproduce (or replicate), forwarding and spreading the successful traits, while the failed

individuals are forgotten. Diversity, which is crucial for optimizing globally, is ensured by mutations (i.e. small random perturbations) and recombination (i.e. various combinations of parents' genetic information or its equivalent).

In 1950s, these ideas gave rise to evolutionary computing that has branched and developed greatly, being extremely successful when solving hard optimization problems.

1.3.1 Vocabulary

First of all, let us go over the basic vocabulary of evolutionary computing. The terms are usually intuitive and correspond with their real-world meanings.

- An *individual* is a candidate vector of variables.
- *Population* is the set of all individuals handled by an algorithm at a time.
- A *generation* is the population at a particular time. Generations can be numbered and all individuals at a given time belong to the same generation.
- *Parents* are those individuals that are allowed to procreate and produce (in some way or another) the *offspring* – the new generation. If a parent belongs to the n-th generation, its offspring belongs to the $(n+1)$-th generation.
- The *fitness function* is a different name for the objective function within the evolutionary computing framework. It evaluates the fitness of individuals within a population. The fittest individuals become parents. Let us emphasize that in the optimization literature the fitness function is often same as the error function but in the control theory application described in the latter sections, the error function has an entirely different meaning. In that context, we shall not use these two terms interchangeably.

Many evolutionary algorithms use stochastic tools. If the reader is not familiar with basic probability theory, they are strongly encouraged to see Appendix A before continuing.

1.3.2 General scheme of randomized black box search

The general scheme of a randomized black-box search, where f is the fitness function can be described by the following pseudocode. The probability distribution P defines the evolution method (e.g. in the case of CMA-ES, the main algorithm described in this thesis, P is the multivariate normal distribution). Function F_θ describes adjustment of the algorithm's parameters.

```
Initialize distribution parameters θ^(0)
For generation g = 0, 1, 2, ...
  Sample λ independent points from distribution P(x|θ^(g)) → x_1, ..., x_λ
  Evaluate sample x_1, ..., x_λ by objective function f
  Update parameters θ^(g+1) = F_θ(θ^(g), (x_1, f(x_1)), ..., (x_λ, f(x_λ)))
  Stop, if termination criterion is met
```

1.3.3 Overview of some algorithms

There are great many algorithms that can be classified as evolutionary. Many of them closely follow its biological inspiration and many others employ only the basic idea of an evolving population and combine it with advanced stochastic tools. The goal of this section is to provide the reader with a brief overview of different approaches taken by evolutionary algorithms without the ambition of being exhaustive. A very extensive overview of evolutionary algorithms including their codes a and further references can be found in [7] and more information can be found e.g. in [13, 10].

Genetic algorithm (GA)

The genetic algorithm is a search heuristic that parallels the principles of Darwinian evolution and Mendelian principles of genetics. It mimics how genes are reproduced based on the fitness of individuals that carry them, and how chromosomes are crossed over and randomly mutated in an attempt to produce yet better-fitting offspring.

Since its beginning in 1960s, the basic GA has branched into numerous versions. For more information, see for example [29], [15] or [14].

Differential evolution (DE)

This technique from 1990s [53, 52] reminds us of GA but the gene recombination process differs significantly.

For each individual x in a population, three other individuals a, b, c are chosen and combined to get the vector $y = a + W(b - c)$, where parameter W is called the differential weight. With a given crossover probability P, each element of x is replaced (or not) with the corresponding element of y. If the resulting individual supersedes x, it takes its place in the new generation.

As a result, we can observe each individual moving over the generations, hopefully converging to the global optimum in the end. For further information on DE, see for example [48].

Swarm intelligence

The swarm intelligence can be observed in nature in behavior of bird flocks, fish schools, ant colonies, honey bees foraging or bacterial growth. Each individual moves by itself, yet the whole self-organized system acts as a single organism.

There are several algorithms based on swarm intelligence (see for example [11, 9, 45, 41]) but lets briefly describe probably the most popular one for global optimization: the particle swarm optimization (PSO).

Each particle within a swarm moves in the search space as it is assigned a different "velocity" vector in each generation. This vector is defined as a combination of the previous "velocity", the individual's best known position so far and the swarm's (or sub-swarm's) best known position. This way, the whole

swarm moves towards historically best areas. Tuning the combination's weights is a highly non-trivial matter that greatly affects the algorithm's performance. Further information on PSO may be found for example in [34, 46].

Estimation of Distribution Algorithms (EDA)

This family of algorithms builds explicit probabilistic models of the problem being solved. The model is continually being improved using information mined in every generation and it is used to guide further search for the optimum.

For example, the Bayesian optimization algorithm (BOA) uses Bayesian networks. Its purpose is to capture stochastic relationships between the components of individuals within the search space. It is represented by a directed acyclic graph. For more information, see e.g. [43, 44, 42].

Evolution strategies (ES)

In evolution strategies, whose origins date back to 1960s, the idea of evolution is revisited from a different perspective than in GA. The mechanism of genetics is omitted, and the focus is on the fitness-based selection process, mutation and recombination.

Individuals within a generation are sampled stochastically, usually from a multivariate normal distribution. The fittest become parents of the new generation by influencing parameters of the search algorithm (for example by shifting the mean of the new distribution).

The most important design principles for evolution strategies is unbiasedness (i.e. advancement towards better solutions is left *only* to the selection process) and adaptive control of search parameters.

Despite its early origins, the field of ES is busy with ongoing research. The current best representative of this family is also one of the most powerful algorithms of today. It is called the Covariance Matrix Adaptation Evolution Strategy and it is described in detail in the following chapter. For latest information on ES and its theoretical background, see e.g. [5, 20].

2. CMA-ES algorithm

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) uses tools of probability theory and linear algebra to define optimally diverse population in an area that seems to be most promising. The size of the area and its location are determined based on the algorithm's previous experience with the objective function.

More precisely, CMA-ES is a stochastic method that samples new candidate solutions from a multivariate normal distribution, whose mean and covariance matrix are adapted in each generation.

The following picture is an illustration of an actual CMA-ES optimization run on a simple 2D problem. The spherical optimization landscape is depicted with solid isolines. The population (dots) is much larger than necessary, but clearly shows how the distribution of the population (dashed line) changes during the optimization. On this simple problem, the population concentrates over the global optimum within a few generations.

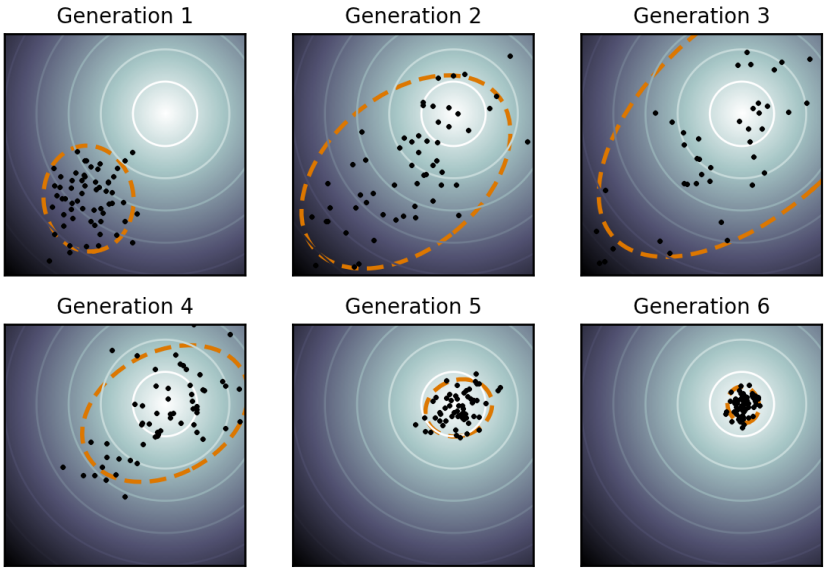


Figure 2.1: Illustration of a CMA-ES run. [Source: Wikipedia]

2.1 Derivation of the basic method

The CMA Evolution Strategy uses the multivariate normal distribution to populate a generation. Its parameters – the mean and the covariance matrix – evolve based on the method's progress.

This section is a combination of sources [19] and [22].

2.1.1 Sampling

The basic equation for generating new search points (i.e. sampling) is:

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)} \mathcal{N}(0, C^{(g)}), \tag{2.1}$$

for $k = 1, \ldots, \lambda$ within each generation $g = 0, 1, \ldots$, and where:

$\sim$ denotes the equality of distributions.

$\mathcal{N}(0, C^{(g)})$ is a multivariate normal distribution with zero mean and covariance matrix $C^{(g)}$. It can be translated and scaled – it holds that:

$$m^{(g)} + \sigma^{(g)} \mathcal{N}(0, C^{(g)}) \sim \mathcal{N}(m^{(g)}, (\sigma^{(g)})^2 C^{(g)}).$$

$x_k^{(g+1)} \in \mathbb{R}^n$ is the k-th individual in the $(g+1)$-th generation.

$m^{(g)} \in \mathbb{R}^n$ is the mean value of the distribution in g-th generation – location of the probability peak and the center of the probability isolines.

$\sigma^{(g)} \in \mathbb{R}^+$ is the "overall" standard deviation and also the step size in the g-th generation. It characterizes the overall size of the sampled area.

$C^{(g)} \in \mathbb{R}$ is the covariance matrix in the g-th generation. Its eigenvectors define the principal axes of the ellipses of the probability isolines.

$\lambda \geq 2$ is the population size – the number of samples.

In the following sections, we derive how the mean $m^{(g)}$, the covariance matrix $C^{(g+1)}$ and the step length $\sigma^{(g+1)}$ are to be updated.

2.1.2 Updating the mean

The mean in $(g+1)$-th generation is defined as weighted average of μ best-ranking individuals (the parent set) from $x_1^{(g+1)}, \ldots, x_\lambda^{(g+1)}$:

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)}, \qquad \sum_{i=1}^{\mu} w_i = 1, \tag{2.2}$$

where $w_1 \geq w_2 \geq \ldots \geq w_\mu > 0$ are weight coefficients and $x_{i:\lambda}^{(g+1)}$ is the i-th best (i.e. best ranking) individual among λ-sized population of the $(g+1)$-th generation. Larger weights always correspond with better ranking individuals, but the relative values may differ. According to N. Hansen, the author of CMA-ES, it is typical to set w_i proportional to $\mu - i + 1$ and μ to be approximately $\frac{\lambda}{2}$.

Let us emphasize that the individuals are assigned rank based on their relative fitness but the actual values of the objective function are not important and are not used here or after. As a result, the whole method will be invariant to strictly monotonous transformations of the objective function, i.e. such transformations that do not change the relative ranking of the individuals.

For future use, we denote

$$\mu_{\text{eff}} \;=\; \Big(\sum_{i=1}^{\mu} w_i^2\Big)^{-1}. \tag{2.3}$$

This constant effects the change rate (expected parameter change per sampled search point) of the mean m. The larger is μ_{eff} the smaller is the possible change rate [19].

Here, we use notation from [19] but in some important sources (e.g. [26]), μ_{eff} is denoted as μ_W.

2.1.3 Updating the covariance matrix

Smart updates of the covariance matrix is what makes this method effective. The updates must reflect what was learned about the underlying objective function. There are two fundamental principles upon which CMA-ES is built.

- We want to maximize the probability of successful search steps and candidate solutions. Therefore, the first cornerstone is the maximum-likelihood principle. Being given some data, we assume that it is normally distributed with an unknown mean and variance (or covariance matrix). Then we look for these parameters' values, so the corresponding normal distribution not only fits the given data, but it is also the distribution that would generate this data with the highest probability of all normal distributions. Formally, this (wanted) set of parameters maximizes the likelihood function, which is essentially the probability density function but with the parameters perceived as variables and vice versa.

- The other fundamental idea is the so-called evolution path. It reflects the progress across the generations. It is used for updating the covariance matrix and also for step size control.

Recall, how a covariance matrix is computed: the element on the ij-th position is the covariance of the i-th and the j-th random variable

$$\operatorname{cov}(X_i, X_j) \;=\; \mathrm{E}[(X_i - \mathrm{E}[X_i])(X_j - \mathrm{E}[X_j])]. \tag{2.4}$$

This formula provides us with framework for covariance matrix estimates.

Basic estimates

We shall first describe how a covariance matrix can be estimated using only information from a population, assuming this information is complete enough.

We shall use the notation introduced in the above sections. Also, for now, let us assume that $\sigma^{(g)} = 1$ (for $\sigma^{(g)} \neq 1$ all formulas hold but for a constant factor).

When we have a population (of distinct individuals) $x_1^{(g+1)}, \ldots, x_\lambda^{(g+1)}$, we can (re)estimate the matrix $C^{(g)}$ by $C_{\text{emp}}^{(g+1)}$:

$$C_{\text{emp}}^{(g+1)} \;=\; \frac{1}{\lambda - 1} \sum_{i=1}^{\lambda} \left(x_i^{(g+1)} - \frac{1}{\lambda} \sum_{j=1}^{\lambda} x_j^{(g+1)} \right) \left(x_i^{(g+1)} - \frac{1}{\lambda} \sum_{j=1}^{\lambda} x_j^{(g+1)} \right)^T. \quad (2.5)$$

Here, the mean of an actually realized sample $\frac{1}{\lambda} \sum_{j=1}^{\lambda} x_j^{(g+1)}$ (the sample mean) approximates the mean value denoted $\mathrm{E}[X_i]$ in (2.4). Therefore $C_{\text{emp}}^{(g+1)}$ estimates the covariance matrix within the sampled *points*.

Now, let's use the true mean value of the sampled distribution at generation g as the reference mean value. We get another unbiased estimate of $C^{(g)}$ by $C_\lambda^{(g+1)}$:

$$C_\lambda^{(g+1)} \;=\; \frac{1}{\lambda} \sum_{i=1}^{\lambda} \left(x_i^{(g+1)} - m^{(g)} \right) \left(x_i^{(g+1)} - m^{(g)} \right)^T. \quad (2.6)$$

The difference from the previous approach is that now we estimate variances of the sampled *steps* (from the mean of the old generation to the sampled points of the new one: step vectors $x_i^{(g+1)} - m^{(g)}$). As these vectors are column vectors, matrix $C_\lambda^{(g+1)}$ can be perceived as a linear combination of matrices.

We can improve the estimate by considering only μ best samples and also by weighting the step vectors (changing the uniform weight of $\frac{1}{\lambda}$ to more general w_i):

$$C_\mu^{(g+1)} \;=\; \sum_{i=1}^{\mu} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(g)} \right) \left(x_{i:\lambda}^{(g+1)} - m^{(g)} \right)^T. \quad (2.7)$$

Then $C_\mu^{(g+1)}$ shall tend to reproduce successful (selected) steps, so we can call it a "better" estimate. Given $w_i = \frac{1}{\mu}$, the expected variance increases in the gradient direction (for all $\mu < \frac{\lambda}{2}$), see figure 2.2.

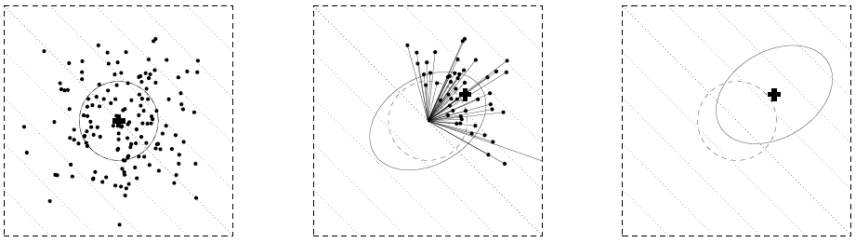


Figure 2.2: Left: sampling of $\lambda = 150$ $\mathcal{N}(0, I)$-distributed points. Middle: estimation of $C_\mu^{(g+1)}, \mu = 50$. Right: outlining the new distribution (the solid ellipsoid). The dotted diagonal lines are the contour lines of the objective function, indicating that the strategy should move to toward the upper right corner. Source: [19].

Rank-μ update

The approach used in the previous section needs large enough λ to be sensible (details in [19]). However, we want to achieve a fast/cheap search, i.e. to use as few function evaluations as possible. We need to balance the opposing requirements. We use memory to speed up the search.

Let

$$C^{(g+1)} \;=\; \frac{1}{g+1} \sum_{i=1}^{g} \frac{1}{\sigma^{(i)^2}} \, C_\mu^{(i+1)} \tag{2.8}$$

to be the mean of the estimated covariance matrices from all generations, where they are "normalized", i.e. the step length is disregarded. After a sufficient number of generations, it becomes an estimator of the selected steps.

In (2.8), all generations have the same weight. However, it is to be expected that recent generations contain more up-to-date information and should therefore be preferred. So, we modify the formula accordingly:

$$C^{(g+1)} \;=\; (1 - c_\mu) \, C^{(g)} \;+\; c_\mu \, \frac{1}{\sigma^{(g)^2}} \, C_\mu^{(g+1)}, \tag{2.9}$$

where we choose $C^{(0)} = I$ to be the identity matrix and $c_\mu \in (0, 1]$ is the learning rate for updating the covariance matrix.

For $c_\mu = 1$, no information of previous generations is used and we obtain $C^{(g+1)}$ by simple normalization of $C_\mu^{(g+1)}$. If it were $c_\mu = 0$, no learning would take place and $C^{(g+1)} = C^{(0)}$.

The choice of the learning rate is crucial. The matrix degenerates when too large values are used and the method fails. On the other hand, small values result in too slow learning. The authors of CMA-ES state that a good setting of the learning rate seems to be independent of the objective function. They have found the value of c_μ empirically and it has been applicable to all functions tested so

far. They assign

$$c_\mu \;\approx\; \min\left\{1;\; \frac{\mu_{\text{eff}}}{n^2}\right\},$$

where μ_{eff} was defined in (2.3) [19].

In formula (2.9), we can see that the effect of each generation diminishes in accordance with its "age". Thus, assuming fixed search costs, a small population size is actually an advantage. It allows for a larger number of generations and therefore (usually) faster adaptation – the covariance matrix is "younger" and it (usually) reflects the current trends better. We shall see this effect later on in the practical application.

Rank-one update

Previously, we used all samples from a single generation to estimate the covariance matrix. Now, we shall look at the generation sequence and use only a single selected step to update the covariance matrix. We shall apply the maximum likelihood principle.

Let's have vectors $y_1, \ldots, y_{g_0} \in \mathbb{R}^n,\ g_0 \geq n$, spanning $\mathbb{R}^n$. Then, thanks to properties of normal distribution [51], we have:

$$\mathcal{N}(0,1)y_1 + \ldots + \mathcal{N}(0,1)y_{g_0} \sim \mathcal{N}(0, \sum_{i=1}^{g_0}\; y_i y_i^T).$$

The left hand side is a sum of g_0 normal "line" distributions

$$\mathcal{N}(0,1)y_i \sim \mathcal{N}(0, y_i y_i^T)$$

with zero mean and covariance matrix $y_i y_i^T$. Each one is the distribution that, of all normal distributions with zero mean, generates the vector y_i with maximal probability. Therefore it holds that $\mathcal{N}(0,1)\sigma y_i \sim \mathcal{N}(0, \sigma^2 y_i y_i^T)$. Any other distribution cannot generate y_i at all. Since $y_i^T y_i = \|y_i\|^2$, choosing the variance σ is trivial: $\sigma = 1$. The matrix $y_i y_i^T$ has rank one, its only eigenvalue is $\|y_i\|^2$ and the corresponding eigenvector is a nontrivial multiple of y_i.

On the right hand side there is a normally distributed random vector with zero mean and covariance matrix $\sum_{i=1}^{g_0} y_i y_i^T$, which is a sum of rank-one matrices $y_i y_i^T$.

Now, we shall combine these rank-one updates with the rank-μ updates (2.9), having $C_\mu^{(g+1)}$ as in (2.7).
Let's denote

$$y_i^{(g+1)} \;:=\; \frac{x_{i:\lambda}^{(g+1)} - m^{(g)}}{\sigma^{(g)}}, \tag{2.10}$$

where $m^{(g)}$ is the mean and $\sigma^{(g)}$ is the variance of the distribution of g-th gener-

ation. When combined, we get

$$C^{(g+1)} \;=\; (1-c_\mu)C^{(g)} \;+\; c_\mu \sum_{i=1}^{\mu} w_i y_i^{(g+1)} y_i^{(g+1)^T}. \tag{2.11}$$

When we set $\mu = 1$, we obtain

$$C^{(g+1)} \;=\; (1-c_1)C^{(g)} \;+\; c_1 y_i^{(g+1)} y_i^{(g+1)^T}. \tag{2.12}$$

Apparently, the $c_1 y_i^{(g+1)} y_i^{(g+1)^T}$ summand (of rank one) adds the maximum likelihood term for $y_i^{(g+1)}$. As a result, it increases the probability that $y_i^{(g+1)}$ is generated in the next generation.

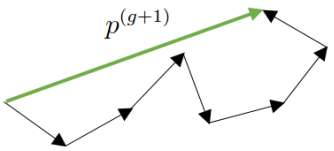
We can iterate this process, always adding a rank-one matrix. Starting with $\mathcal{N}(0, I)$ in the zeroth generation, the distribution of the first generation $\mathcal{N}(0, C^{(1)})$ will tend to reproduce y_1 with greater probability than the initial distribution.

Then distribution $\mathcal{N}(0, C^{(2)})$ will tend to produce y_2 with greater probability than $\mathcal{N}(0, C^{(1)})$ and so on. In general, $\mathcal{N}(0, C^{(g)})$ will tend to reproduce the previously selected (successful) steps $y_1, \ldots, y_g$.

Evolution path

Let's note that $yy^T = (-y)(-y)^T$. Therefore, there is no sign information contained in the updates as derived in the previous sections. To exploit it, the progress of the strategy is recorded in so-called evolution path.

It is a sequence of successive steps that the strategy takes over a number of generations. Because we are interested in directions alone, the influence of step size is eliminated by normalization. The evolution path is then cumulation of (normalized) consecutive steps from $m^{(i)}$ to $m^{(i+1)}$. The naïve evolution path $p^{(g+1)} \in \mathbb{R}^n$ in generation $g+1$ over s steps is



Evolution path

$$p^{(g+1)} \;=\; \sum_{i=g+1-s}^{g} \frac{m^{(i+1)} - m^{(i)}}{\sigma^{(i)}}$$

We can further improve it by using smoothing in the same way as when we introduced the learning rate in (2.9). We start with $p_c^{(0)} = 0$ and then we define

$$p_c^{(g+1)} \;=\; (1-c_c)p_c^{(g)} \;+\; F_c \,\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}, \tag{2.13}$$

where $p_c^{(g)} \in \mathbb{R}^n$ is again the evolution path at generation g, $c_c \in (0, 1]$ is the smoothing constant, and the factor

$$F_c \;=\; \sqrt{c_c(2-c_c)\mu_{\text{eff}}} \tag{2.14}$$

is the normalization constant for p_c. It is chosen such that in every generation $p_c^{(g+1)} \sim \mathcal{N}(0, C)$ (i.e. $p_c^{(g+1)}$ is normally distributed with mean 0 and covariance matrix C). That is, if $p_c^{(g)} \sim \mathcal{N}(0, C)$, then the smoothing in (2.13) preserves this property for$(g+1)$-th generation.

The rank-one update of the covariance matrix (2.12) using the evolution path (2.13) reads

$$C^{(g+1)} \;=\; (1 - c_1)C^{(g)} \;+\; c_1 p_c^{(g+1)} p_c^{(g+1)^T}. \tag{2.15}$$

The author of CMA-ES notes that an empirically validated choice for the learning rate in this formula is

$$c_1 = \frac{2}{n^2}.$$

For $c_c = 1$ and $\mu = 1$, this formula is identical with the formulae (2.11) and (2.12).

Combining rank-μ update and cumulation

Finally, we simply combine the advantages of rank-μ update (2.11) and rank-one update that uses the evolution path (2.15).

As in (2.9), we write the matrix $C^{(g+1)}$ as a weighted sum of its predecessors, but this time we use $c_1 + c_\mu$ as the weight:

$$C^{(g+1)} \;=\; \big(1 - (c_1 + c_\mu)\big)C^{(g)} \;+\; (c_1 + c_\mu)\, \frac{1}{\sigma^{(g)^2}}\, C_\mu^{(g+1)}.$$

As when deriving (2.11), we get

$$C^{(g+1)} \;=\; \big(1 - c_1 - c_\mu\big)C^{(g)} \;+\; (c_1 + c_\mu) \sum_{i=1}^{\mu} w_i y_i^{(g+1)} y_i^{(g+1)^T}.$$

Then we use the evolution path to transform

$$c_1 \sum_{i=1}^{\mu} w_i y_i^{(g+1)} y_i^{(g+1)^T} \;\longrightarrow\; c_1 p_c^{(g+1)} p_c^{(g+1)^T}$$

just as was done to obtain (2.15). In the end, we have the following formula:

$$C^{(g+1)} \;=\; (1 - c_1 - c_\mu)C^{(g)} + c_1 p_c^{(g+1)} p_c^{(g+1)^T} + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} y_{i:\lambda}^{(g+1)^T}, \tag{2.16}$$

where

$$y_{i:\lambda}^{(g+1)} = \frac{x_{i:\lambda}^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$$

and

$$c_1 \approx \frac{2}{n^2}, \qquad c_\mu \approx \min\left\{\frac{\mu_{\text{eff}}}{n^2};\ 1 - c_1\right\}.$$

The learning rates c_1 and c_μ generally do not depend on the population size or the parent set size (even though these can be and are calculated from the problem dimension too). It is easy to see that the learning rates explicitly influence the change rate of the covariance matrix.

The rank-μ update exploits the information within the population of one generation. It is especially important when handling large populations. The information of correlation between generations is stored in the evolution path rank-one updates. Its significance rises when the populations are small and thus the number of generations increases. The learning rates c_1 and c_μ are to prevent the degeneration even when small populations are used. Small populations usually cause the algorithm to converge faster, but larger populations are better for avoiding local optima. The adaptation strategy following this equation should be able to deal with badly scaled and non-separable functions.

2.1.4 Step length control

The matrix adaptation described above changes the shape of the distribution's contour lines. These ellipses are rotated and the relative lengths of their axes are modified as a preferred direction is favored. Also, their center is moved according to the mean updating rule. However, their overall scale – the step length – is being changed only a little. The optimal step length cannot be well approximated by formula (2.16) and even the largest reliable learning rate for the update in (2.16) is too slow to adjust the overall step length at a competitive rate. Therefore, we need an additional strategy to cope with the step length control.

The cumulative step length adaptation method utilizes the evolution path:

- When the evolution path is long, the single steps point to similar directions (correlation). Prolong the step.
- When the evolution path is short, the single steps cancel each other out (anti-correlation). Shorten the step.
- Desired situation: the individual steps are (approximately) mutually perpendicular and therefore uncorrelated.

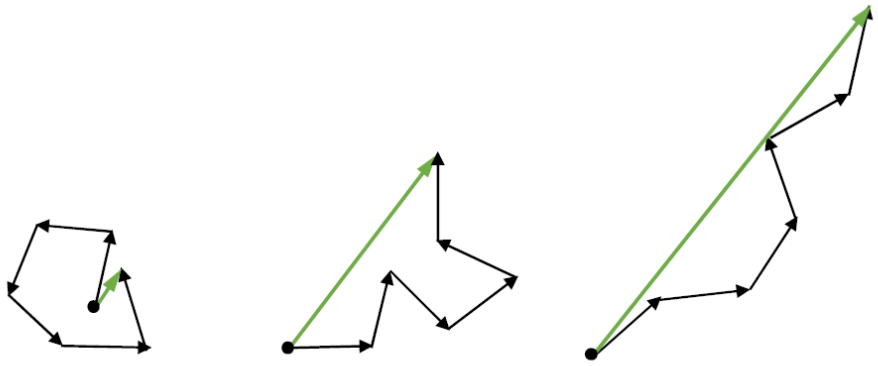


Figure 2.4: Three evolution paths of six steps in different selection situations (idealized). The lengths of the individual steps are all comparable, but their sums (the evolution paths) differ greatly. On the left, the evolution path is short and thus the step size should be decreased. In the middle, we have (approximately) the desired situation. On the right, all steps point in approximately the same direction, making the evolution path very long — the step size should be increased. Source: [19]

How do we decide, whether an evolution path is long or short? What does it mean? We shall compare the actual evolution path with its expected length (under random selection — the mean value of corresponding random walk). In the optimal situation, the evolution path length is (approximately) as long as expected. If it is biased to be longer than expected, the step size should be increased and vice versa.

Before we can compare the evolution path length with its expected value, we must construct it in a way that it is comparable. We use the same reasoning and technique as when constructing $p_c^{(g+1)}$ to get the formula:

$$p_\sigma^{(g+1)} \;=\; (1-c_\sigma)p_\sigma^{(g)} \;+\; F_\sigma C^{(g)^{-\frac{1}{2}}} \frac{m^{(g+1)}-m^{(g)}}{\sigma^{(g)}},$$

where $p_\sigma^{(g)} \in \mathbb{R}^n$ is the conjugate evolution path at g-th generation, $c_\sigma \in (0,1)$ is the smoothing constant and

$$F_\sigma \;=\; \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}$$

is the normalization constant similar to F_c in (2.14).

The transformation represented by $C^{(g)^{-\frac{1}{2}}}$ rescales the step $m^{(g+1)} - m^{(g)}$ within the corresponding coordinate system given by the eigenvectors of $C^{(g)}$. As a (desired) consequence, the expected length of $p_\sigma^{(g+1)}$ does not depend on its direction. Also, given that $p_\sigma^{(0)} \sim \mathcal{N}(0, I)$, we get $p_\sigma^{(g+1)} \sim \mathcal{N}(0, I)$ for any sequence of realized covariance matrices up to g-th generation [19].

The realized path has the length $\|p_\sigma^{(g+1)}\|$. The expected length, i.e. the mean

value of random walk [51], is given by

$$\mathrm{E}\left[\|\mathcal{N}(0,I)\|\right] \;\approx\; \sqrt{n} \;+\; \mathcal{O}\Big(\frac{1}{n}\Big).$$

To update the step size $\sigma^{(g)}$, we compare the realized path length and the expected path length. If the realized path is longer than expected, we want to shorten the step. If it is shorter, we prolong the step. Therefore we multiply $\sigma^{(g)}$ by a factor either smaller or greater than one, respectively. This factor shall depend exponentially on the ratio of the realized and expected path lengths.

$$\sigma^{(g+1)} \;=\; \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(g+1)}\|}{\mathrm{E}\left[\|\mathcal{N}(0,I)\|\right]} - 1\right)\right) \tag{2.17}$$

where $d_\sigma \approx 1$ is the damping parameter and the factor $\frac{c_\sigma}{d_\sigma}$ is based on deeper investigation of the algorithm. Let us emphasize that the change rate of the step size σ, explicitly controlled by d_σ, is independent of the change rate of the covariance matrix.

The exponential factor enables the step size to adapt very quickly. On the other hand, due to its fast convergence, the starting step size $\sigma^{(0)}$ limits the area that can be searched by the algorithm.

Choosing the step length based on the evolution path length is rather heuristic, but it has been validated empirically. The step size control prevents premature convergence of the population. However, it cannot prevent the algorithm from ending up in a local optimum.

2.1.5 Stopping criteria

There are eight stopping criteria for the algorithm. Even though the corresponding variables are assigned default values, it might be desirable to change their setting to fit the problem at hand.

- `EqualFunValues` — range of the best objective function values of last $10 + \lceil 30n/\lambda \rceil$ generations is zero, i.e. less than `EqualFunValues`.
- `TolFun` — range of the best objective function values of last $10 + \lceil 30n/\lambda \rceil$ generations and also of all function values of the current generation is almost zero, i.e. less than `TolFun`. Value of this parameter depends on the problem but the default is 10^{-12}).
- `TolX` — the standard deviation σ of the normal distribution $\mathcal{N}(\mu, \sigma^2 C)$ is less than `TolX` in every coordinate and also the evolution path vector σp_c is less than `TolX` in all its components. Default value is $10^{-12}\sigma^{(0)}$.
- `NoEffectAxis` — the mean value ($m^{(g)}$, given by equation (2.2)) remains numerically constant when a small (0.1) standard deviation is added to it in

the principal axis direction of the current covariance matrix. That is, when $m^{(g)} = m^{(g)} + 0.1\sigma^{(g)}\sqrt{\lambda_i}u_i$, where λ_i and u_i are the i-th eigenvalue and unit eigenvector of the covariance matrix $C^{(g)}$, respectively, $i = 1 + g \mathrm{mod} n$, n being the dimension.

- `NoEffectCoor` — the mean value does not change (numerically) when 0.2-standard deviation is added to each coordinate (similar to `NoEffectAxis`).
- `ConditionCov` — the condition number of the covariance matrix exceeds the limit of 10^{14} (larger conditioning leads to numerical errors [22].
- `TolXUp` — stopping if $\sigma \max(diag(D))$ increases by more than `TolXUp`. It indicates divergent behaviour or too small initial σ_0. Default value is 10^4.
- `Stagnation` — we remember the history of the best and the median fitness in each iteration over the last 20% generations (however, at least $120+30n/\lambda$ and at most 20000 iterations). If the median of the most recent 30% values is not better than the median of the oldest 30% in both histories, we stop.

2.2 Summary

In this section, we summarize the whole algorithm and point out its most important properties.

Further theory of CMA-ES and evolution strategies can be found, for example in [5, 20, 6]. However, even though many empirical results show that the performance of CMA-ES-based algorithms is overall very good [22, 21, 3], there are no convergence proofs available yet. According to Hansen et al. [20], they can be expected in the upcoming decade or so.

2.2.1 Algorithm summary

There are many constants that control the algorithm's behavior. Their default settings are not recommended to be changed, except for λ. They are discussed in detail in [22]. On the other hand, choice of the initial distribution mean $m \in \mathbb{R}^n$ and step size $\sigma \in \mathbb{R}^+$ depends on the particular problem. For a good performance, we want to bracket the optimum within the initial cube $m \pm 3\sigma(1, \ldots, 1)^T$.

Set constants (default setting):

$$\lambda = 4 + \lfloor 3 \ln N \rfloor \quad \ldots \text{population size}$$

$$\mu = \left\lfloor \frac{1}{2}\lambda \right\rfloor \quad \ldots \text{parent set size}$$

$$w_i = \frac{\ln(\mu+1) - \ln i}{\sum_{j=1}^{\mu}(\ln(\mu+1) - \ln j)}, \quad i = 1, \ldots, \mu, \quad \text{where } \sum_{i=1}^{\mu} w_i = 1,\ w_i > 0$$

$\ldots$ recombination weights

$$\mu_{\text{eff}} = \left(\sum_{i=1}^{\mu} w_i^2\right)^{-1} \quad \ldots \text{variance effective selection mass; } 1 \le \mu_{\text{eff}} \le \mu$$

$$c_\sigma = \frac{\mu_{\text{eff}} + 2}{N + \mu_{\text{eff}} + 5} \quad \ldots \text{smoothing for step size control}$$

$$d_\sigma = 1 + c_\sigma + 2 \max\left\{0, \sqrt{\frac{\mu_{\text{eff}} - 1}{N + 1}} - 1\right\} \quad \ldots \text{damping for s. s. control}$$

$$c_c = \frac{4 + \mu_{\text{eff}}/N}{4 + N + 2\mu_{\text{eff}}/N} \quad \ldots \text{smoothing in covariance matrix adaptation}$$

$$c_1 = \frac{2}{(N + 1,3)^2 + \mu_{\text{eff}}} \quad \ldots \text{rank-one learning rate}$$

$$c_\mu = \min\left\{1 - c_1, 2\frac{\mu_{\text{eff}} - 2 + 1/\mu_{\text{eff}}}{(N + 2)^2 + 2\mu_{\text{eff}}/2}\right\} \quad \ldots \text{rank-}\mu \text{ learning rate}$$

Initialize:

$$p_\sigma = 0 \ \ldots \text{conjugate evolution path for step size control}$$

$$p_c = 0 \ \ldots \text{evolution path for covariance matrix adaptation}$$

$$C = I \ \ldots \text{initialize the covariance matrix as the identity matrix}$$

$$g = 0 \ \ldots \text{generation number}$$

Choose:

$$m \in \mathbb{R}^n \ \ldots \text{distribution mean}$$

$$\sigma \in \mathbb{R}^+ \ \ldots \text{step size}$$

Until a termination criterion is met, $g \leftarrow g+1$, **for every generation do:**

• Sample new population. For $k = 1, \ldots, \lambda$:

$$z_k \sim \mathcal{N}(0, I)$$

$$y_k = BDz_k \sim \mathcal{N}(0, C), \quad \text{where } C = BD^2B^T \text{ is eigendecomposition}$$

$$x_k = m + \sigma y_k \sim \mathcal{N}(m, \sigma^2 C)$$

• Select and recombine:

$$\langle y \rangle_w \;=\; \sum_{i=1}^{\mu} w_i y_{i:\lambda} \qquad \ldots \text{distribution mean step disregarding } \sigma$$

$$m \;\leftarrow\; m + \sigma \langle y \rangle_w \;=\; \sum_{i=1}^{\mu} w_i x_{i:\lambda}$$

$$\ldots \text{it is } y_{i:\lambda} = \frac{x_{i:\lambda} - m}{\sigma}, \text{ where } x_{i:\lambda} \text{ is } i\text{-th best point out of } x_1, \ldots, x_\lambda$$

• Step size control:

$$p_\sigma \;\leftarrow\; (1 - c_\sigma) p_\sigma + \sqrt{c_\sigma (2 - c_\sigma) \mu_{\text{eff}}}\; C^{-\frac{1}{2}} \langle y \rangle_w$$

$$\ldots \text{it is } \; C^{-\frac{1}{2}} \overset{def.}{=} B D^{-1} B^T, \text{ so we use } \; C^{-\frac{1}{2}} \langle y \rangle_w \;=\; B \sum_{i=1}^{\mu} w_i z_{i:\lambda}$$

$$\sigma \;\leftarrow\; \sigma \exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{\|p_\sigma\|}{\mathrm{E}\left[\|\mathcal{N}(0, I)\| \right]} - 1 \right) \right)$$

• Covariance matrix adaptation:

$$p_c \;\leftarrow\; (1 - c_c) p_c + h_\sigma \sqrt{c_c (2 - c_c) \mu_{\text{eff}}}\; \langle y \rangle_w$$

$$\ldots \text{where the step function } h_\sigma \text{ is defined further in (2.18)}$$

$$C \;\leftarrow\; (1 - c_1 - c_\mu) C + c_1 \left(p_c p_c^T + (1 - h_\sigma) c_c (2 - c_c) C \right) + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T$$

$$C \;=\; B D^2 B^T \quad \ldots \text{perform eigendecomposition}$$

In their implementation, Hansen [19, 26] uses a minor improvement in updating p_c. The idea is to prevent too fast increase of axes of C when the step size is too small. That is, we limit the update of p_c if $\|p_\sigma\|$ is too large. For that, the step function h_σ (2.18) is used.

$$h_\sigma \;=\; \begin{cases} 1 & \text{if } \; \frac{\|p_\sigma\|}{\sqrt{1 - (1 - c_\sigma)^{2(g+1)}}} \;<\; (1{,}4 + \frac{2}{n+1}) \mathrm{E}\left[\|\mathcal{N}(0, I)\| \right] \\ 0 & \text{otherwise} \end{cases}, \qquad (2.18)$$

where g is the generation number.

Usually [19], it is $h_\sigma = 1$, which leaves the update of p_c (and then also C) as

they were in (2.13) and (2.16).

2.2.2 Invariance properties

Invariance means that the algorithm exhibits identical behavior and thus also identical performance on classes of objective functions. Such properties are highly desirable as they enable us to generalize empirical results.

It is its invariance properties that make CMA-ES so effective optimizing objective functions that are ill-conditioned, noisy or non-smooth, where many other algorithms fail [21, 3].

The most important property is invariance to strictly monotonic transformations of the objective function [19]. This is implied by the fact that the method does not use actual objective function values for anything but to assign relative fitness ranking. Transformations of the objective function that have no effect upon this relative ranking of individuals thus do not effect the method's performance.

Further, the method exhibits invariance to rigid transformations of the search space (i.e. rotation, reflection and translation) and, in general, it is invariant to invertible linear transformations of the search space. It is especially worthy to notice that the method is invariant to scaling of variables (coordinate axes).

Of course, even though the performance remains intact, it is crucial that with the transformations of either the search space or the objective function, the initial parameters $(\sigma^{(0)}, C^{(0)}, m^{(0)})$ or stopping criteria are adjusted accordingly.

2.3 Upgrades of CMA-ES

In the previous sections, we have described the basic algorithm of CMA-ES, which itself is a fast and robust method. However, there are many upgrades and extensions that significantly improve its performance, especially when dealing with global optimization. In this section, we shall look at the most important ones. Let us point out that none of them spoil the important properties of CMA-ES.

2.3.1 Restart strategies

When optimizing globally, it is essential to use restarts. The algorithm might converge to a local minimum and by restarting it, we increase the chance of finding the global minimum.

We can use a simple restart strategy, when the algorithm is always restarted as it is, with no adjustment. Or we can learn from the experience and change the algorithm's parameters in the attempt to increase the probability of success.

The following restart strategies change, among other things, the size of the initial population. There exist several more restart strategies for CMA-ES [35], which can further somewhat improve the performance but the difference is not great.

Increasing population size (IPOP)

The default population size for CMA-ES is given by formula

$$\lambda_{\text{def}} \;=\; 4 + \lfloor 3 \log(N) \rfloor \,,$$

where N is the problem dimension [22]. However, the optimal population size may be much larger.

Therefore, the IPOP strategy gradually increases the population size [4]. Every time an independent restart is launched, the population doubles in size. (The factor of 2 is empirical.) Other parameters remain unchanged. In [4], superiority of IPOP restart strategy over pure restarts is demonstrated.

The restart criteria are inherited from CMA-ES (see section 2.1.5), except for `Stagnation` and `TolXUp` [4].

Bi-population multistart scheme (BIPOP)

The motivation for this improvement is better balance of exploration (exploring as much of the vector space as possible) and exploitation (thorough investigation of promising areas). To achieve that, two interlacing approaches are applied. One regime uses increasing population size (as in the IPOP restart strategy described above in section 2.3.1), while the other uses varying small populations [26].

In the beginning, a single run with default population size λ_{def} is performed. Then, a run under the first regime is started. After this run and every one afterwards, it is decided which of two regimes is to be applied next depending on whose count of conducted function evaluations is lower (the initial run budget does not count). Number of restarts under the first regime is limited, cutting

the maximal population size and also possibly ending the restarts. If not terminated earlier, the last restart is conducted under the first regime with maximal population size.

Every time the first regime is run, the population size is doubled. By default, the largest population size is $\lambda = 2^9\lambda_{\text{def}}$. Lets denote the latest (large) population size from the first regime λ_l.

The second regime uses small population size

$$\lambda_s = \left\lfloor \lambda_{\text{def}} \left(\frac{1}{2} \frac{\lambda_l}{\lambda_{\text{def}}} \right)^{\mathcal{U}[0,1]^2} \right\rfloor,$$

where $\mathcal{U}[0,1]$ denotes independently uniformly distributed numbers in $[0,1]$. It is clearly $\lambda_s \in [\lambda_{\text{def}}, \frac{\lambda}{2}]$.

Apart from the inherited stopping criteria `NoEffectAxis`, `NoEffectCoor`, `ConditionCov`, `TolFun`, `TolX` (see section 2.1.5), we have several more [26].

- `MaxIter` — the maximal number of generations (= iterations) in each run of CMA-ES. By default, `MaxIter` $= 100 + 50(N+3)^2/\sqrt{\lambda}$, where N is the dimension and λ is the population size in the current restart. This criterion is included to prevent excessively long runs.

- `EqualFunVals` — in more than a third of the last N generations, the objective function values of the best and the k-th best solution are equal, $k = 1 + \lceil 0, 1 + \lambda/4 \rceil$.

- `TolUpSigma` — when $\sigma/\sigma^0 >$ `TolUpSigma`$\sqrt{l}$ (where σ is the current step length and l is the largest eigenvalue of the covariance matrix at a given generation), it indicates "creeping" behavior. Default value is 10^{20}.

- `TolStagnation` — when no improvement is observed over several (i.e. $100 + 100N^{1.5}/\lambda$ by default) generations. That is, taking the Hansen's new code [18] as the source.

Apart from these criteria discussed in [26], the code [18] contains a few additional, possibly experimental, ones. However, in experiments described in chapter 3, they were never observed to have any effect.

2.3.2 Elitist selection

When choosing the parents, we have multiple options [20]. The basic algorithm selects them from the current generation. When we want to speed up the convergence, we can choose to select the parent set from the individuals of the current generation and also its parents. That helps to preserve the exceptionally good individuals until they are superseded and thus amplify their influence. The downside of this approach is that it may lead to premature convergence to a local optimum.

2.3.3 Active covariance matrix adaptation

In the original algorithm, we use the information of the successful individuals to adapt the covariance matrix. We increase the variance in directions that have proven to be beneficial. As a result, these directions are preferred when sampling next generation.

The idea of active updates is simple: we shall also exploit the information hidden in the unsuccessful individuals [30]. As opposed to passive decay over time, we actively decrease the variance in such directions. In other words, besides telling the method where to go, we also tell it where *not* to go.

Equation (2.16) in section 2.1.3 gives us the update formula for the covariance matrix:

$$C^{(g+1)} \;=\; (1 - c_1 - c_\mu)C^{(g)} + c_1 p_c^{(g+1)} p_c^{(g+1)^T} + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} y_{i:\lambda}^{(g+1)^T},$$

where the rank-μ update is given by the third term. Now, we replace this term by

$$c_\mu \left(\sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} y_{i:\lambda}^{(g+1)^T} - \sum_{i=\lambda+1-\mu}^{\lambda} w_{\lambda+1-i}\; y_{i:\lambda}^{(g+1)} y_{i:\lambda}^{(g+1)^T} \right).$$

This way, μ worst solutions are assigned corresponding weights and used for the negative update. Jastrebski and Arnold [30] use weights $w_i = 1/\mu$ for all $i = 1, \ldots, \mu$ and $\mu = \lambda/4$.

As a result of this change, the related constants must be adjusted in order to maintain stationarity of the expectation of the covariance matrix. Detailed description is to be found in [30]. This paper and [23] then contain experimental comparisons.

3. Real-world application: automated tuning of PID controllers in an engine model

This chapter is based on the author's independent work for company Ricardo Prague s.r.o., which supported this applied research. The goal was to investigate possible improvements regarding automated PID controller tuning to program WAVE, 1D engine and gas dynamics simulation software package by Ricardo Software [49]. It enables engineers to model complex systems of car engines using the finite element method.

I would like to give special thanks to those who counseled me all along the way. First and foremost, I would like to thank Steve Amphlett, for it was him who came up with the task and provided me with much support. Jiří Navrátil, the product manager of WAVE, gave his blessing to this adventure and together with Martin Horáček of the WAVE customer support team initiated me to the world of WAVE users and were willing to answer countless practical questions. Last but not least, many thanks are to be given to control engineers Michal Vinklář, Adam Kouba and Bohumil Hnilička, who mentored me on control theory and its application in engine design.

3.1 The task

In a running engine, so called controllers ensure that certain quantities (such as intake pressure, exhaust gas temperature and many others) remain constant or within given range. WAVE models often include one or several PID (P – proportional, I – integral, D – derivative) controllers as simple, yet powerful tools of system control. Each PID controller has three parameters that need to be "tuned" (i.e. set to near-optimal values) in order to have a well-controlled system. In most cases, the "D" parameter is set to zero, leaving only two parameters to be tuned. In general, this is an optimization problem, with the model being the heart of the objective function.

The following picture shows a simple WAVE model of a six-cylinder engine with a single PID controller. It controls pressure in the rail (measured by a sensor) by controlling (the speed of) the turbine powered by exhaust gases.

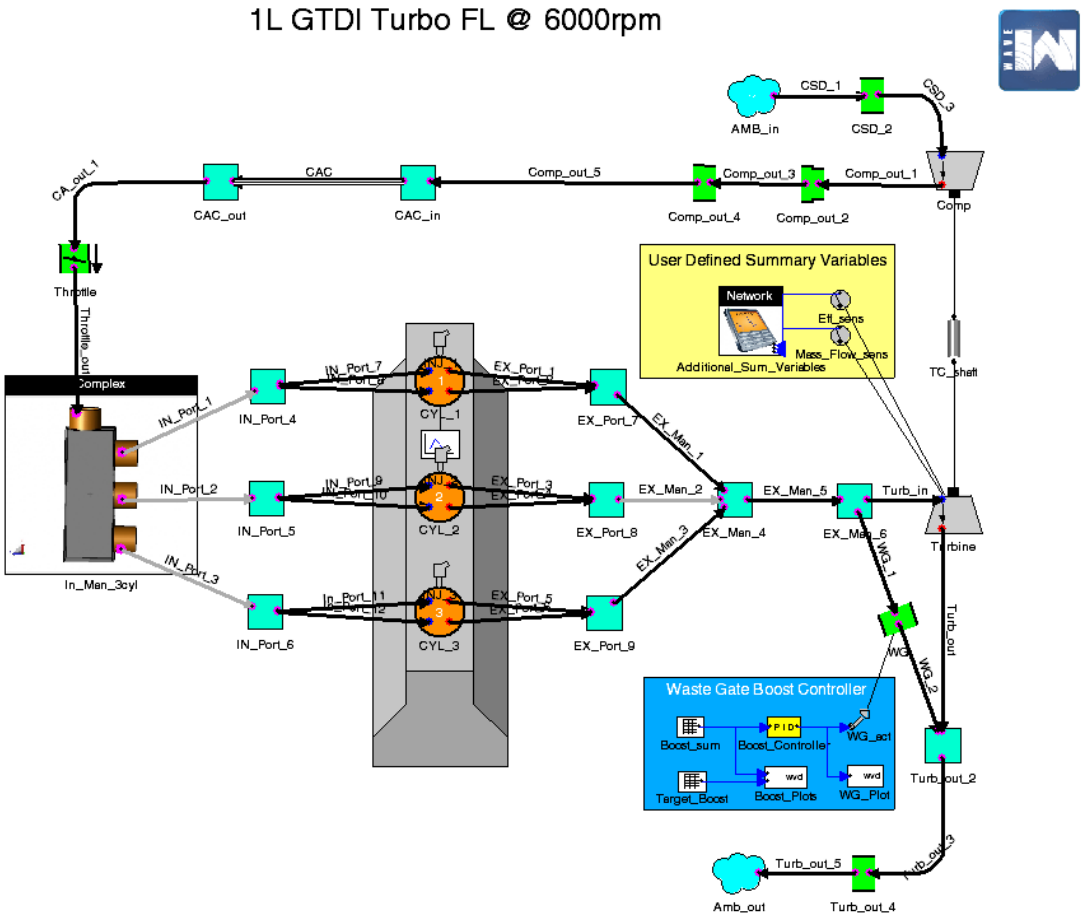


Figure 3.1: A simple WAVE model of a three cylinder engine with a single controller (yellow "PID" box in the blue field).

3.2 Motivation

Besides posing an interesting challenge, there is strong practical motivation to solve this task.

3.2.1 The current state

Currently, controllers are tuned manually (i.e. by trial and error) and using two simple tools which are implemented in WAVE: the well-known Ziegler-Nichols rule and Ricardo No Lag method (developed by Ricardo). The first one is essentially an empirical rule of thumb, while the other takes an analytical approach and often performs better. Simplifying assumptions are made for the problem to be solvable analytically, possibly loosing essential information along the way. Solution found this way cannot, in general, be the optimal one. Nonetheless, thanks to robustness of PID controllers, this tuning usually provides good results that may be further refined manually.

However, by these tools, only one controller at a time can be tuned. When having a model with multiple coupled controllers, both these tools fail. Usually an engineer seeks inspiration in library of older models to get an sensible first

estimate and continues with manual tuning. This is a lengthy and tedious task that requires a knowledgeable, experienced user.

3.2.2 Commercial motivation

In practice, not only a single controller but also two coupled controllers need to be tuned relatively often and sometimes three or even more are needed. Ten controllers within one system is maximum seen so far in our application. Depending on the user's expertise and experience, the tuning process can take up to several days of work. More than three interacting controllers are not to be seen very often and it becomes virtually impossible to tune them by hand when no additional information (e.g. inspiration by other models) is given. Therefore, engineers could use a feature in WAVE that would take care of this chore. Moreover, among WAVE users there are engineers that have little or no working knowledge of controllers but have to have them tuned before they can carry on with their own work.

On October 3, 2014, Steve Amphlett, one of the fathers of WAVE, wrote in an email:

> [It would be nice to have] "...some kind of clever/novel/interesting system identification for control system design and tuning. Once we've given our customers the ability to add controllers to their simulations (e.g. the best timing of some device to get a target output of some measure), the first thing they ask us is how to set it up and choose the right parameters to make it work."

Also according to Ricardo WAVE customer support, there is demand from the customers to have such a controller tuning tool. None of our competitors have, despite their effort, reached a satisfactory solution of this problem, so cracking it would imply a significant competitive advantage.

3.3 Formulation of the problem

This section is to provide a rather compact formulation of the problem from the global point of view. Reader who is not familiar with basics of control theory is strongly encouraged to read the Appendix B first before continuing with this chapter.

3.3.1 Main objective summary

PID controllers are simple, yet powerful tools that control behavior of an engine or, in our case, a model of an engine ("the system"). The aim is to set up the controllers, i.e. tune their gains ("the parameters" or "the variables") such that the system's response to the control fits the requirements.

3.3.2 Assumptions

Theoretically, to choose the right type of a controller (and to tune it), we need to know the properties of the system that we want to control — the order and

stability of the system, its transport delay etc. PID controllers might not even stand a chance to fully control a high-order system. However, in practice, we are not given the luxury of this knowledge. On the other hand, according to WAVE support team, we can safely assume that a solution (i.e. controllers' setup such that the controlled quantities sooner or later converge to the target values) exists, if our model in WAVE is set up correctly (from the engineering point of view). We know typical uses of PID regulators in an engine model and we know that it is possible, though sometimes very hard, to tune them. At the beginning of the tuning process, the controllers are already a part of the model.

3.3.3 Basic requirements

First, we have to know how to observe controllers' influence upon the system's behavior.

Step response

The system's response to control is "measured" as so-called step response. Theoretically, it is the evolution of the system's output when its control input changes immediately from zero to one. When the response function converges, the corresponding controller is stable.

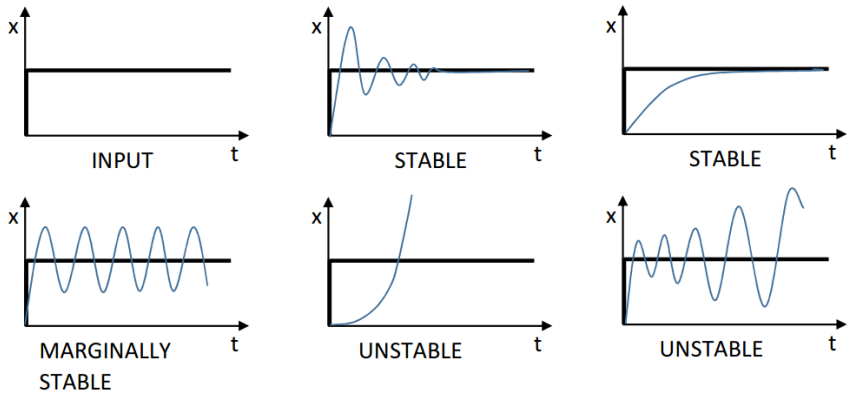


Figure 3.2: Examples of stable and unstable step response.

Requirements

Our requirements for the response of a tuned system are straightforward. Each of the controllers is to be stable and the step responses of the controlled quantities (e.g. rail pressure) are to converge to the required value. In our case, the number of controllers usually equals the number of controlled quantities.

3.3.4 MIMO problems

A multiple input multiple output (MIMO) system is characterized by equation

$$y = Hx + n$$

Where x is the input vector, y is the output vector, H is the matrix of functions characterizing the system and n is the noise vector. In our case: x is the vector of controllers' parameters and y is the vector, whose elements correspond (in some way or another) to the system's responses to control. Each element of matrix H, $h_{ij} = h_{ij}(x_i, y_j)$, is a function that describes the relationships between i-th input and j-th output. Dealing with a noiseless model, we set $n = 0$.

When each output is determined by one input variable only, matrix H is diagonal (when arranged appropriately) and the MIMO system is essentially only a set of independent single input single output (SISO) problems that are easier to solve.

Similarly, when the matrix H can be rearranged to block diagonal form, the original, big MIMO problem can be broken down to smaller problems.

When there is only one variable on the output, i.e. matrix H is just a column vector, it is called multiple input single output (MISO) problem.

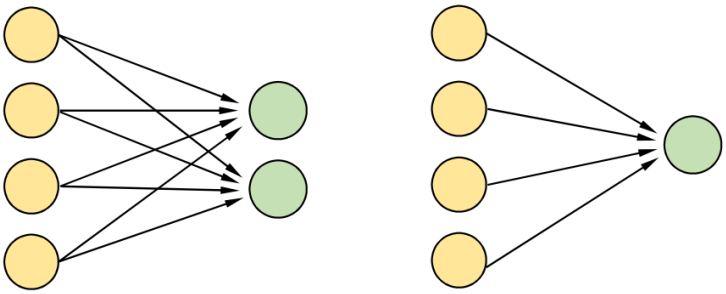


Figure 3.3: A 4×2 MIMO system (left) and a 4×1 MISO system (right).

Tuning one PID controller is a SISO problem (lets consider all three gains of a controller to be a single input). Tuning multiple coupled PID controllers is, in general, a MIMO problem. When the controllers are coupled, it cannot be transformed to a series of independent SISO problems, one for each controller with three parameters, as the problem is non-separable.

Without loss of generality, we will further assume that the problem at hand is a full MIMO problem. If it is not, i.e. some of the controllers to be tuned are independents of others, the whole problem can be broken down into smaller subproblems, each one to be solved separately. I.e. if the matrix H is block diagonal, we take each of the blocks as a separate problem.

The question remains how to recognize (detect) which controllers are coupled, which is not to be discussed in this thesis and might be subject of further research. Now, we do not even attempt to break the problem into such subproblems – instead, we take any given system as a MIMO system.

There of course is the problem how to deal with multiple outputs – multiple objectives of optimization. There are tools that could be used but, in general, multi-objective optimization is more (computationally) costly than an "ordinary" single-objective optimization. Therefore, we want to transform the MIMO problem into a MISO problem. Due to specifics of our task, as it is useless to tune only some controllers, while others diverge, it is only natural to try to describe the whole task by a single objective.

3.4 Objective function

The most difficult task is to find out what it is we want. That is, we need to define the objective function that describes our requirements and ranks the fitness of candidate solutions.

3.4.1 Controlling multiple objectives

Lets have a model with k controlled quantities (and also k controllers). Lets assume for a moment that quality of the step response of i-th controller can be described by objective function F_i. That provides us with k objectives that we want to minimize. However, we want to use tools for single-objective optimization, so the question remains how to combine these objectives into a single one.

The answer is rather simple. We shall define, and justify experimentally later, the objective function describing the response of the whole model as

$$F(t) \;=\; \sum_{i=1}^{k} n_i \; F_i(t), \tag{3.1}$$

where n_i is the "normalization" constant corresponding to the i-th controlled quantity. Purpose of the constants n_i is to make all the objective functions F_i comparable, as they may significantly differ in range depending on the corresponding units. Therefore, we set

$$n_i \;=\; \frac{1}{|target_i|},$$

where $target_i$ is the target value of the i-th controlled quantity (e.g. i-th target value $780°K$ gives us $n_i = \frac{1}{780}$).

There is also the option of prioritizing the individual objective functions F_i – assigning different weights to them. Considering all the objectives to be equally important, we shall set all the priorities equal to one, getting precisely formula (3.1).

3.4.2 Step response based objective function

General framework for the definition of F_i, the objective function corresponding to the i-th controller, is based on how the step response looks like (see figure 3.2) and thus is naturally given by formula

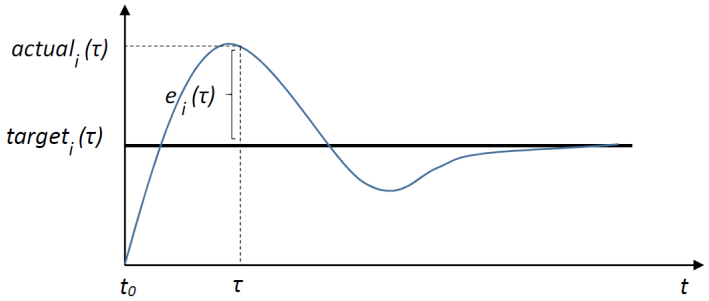
$$F_i(t) \;=\; \int_{t_0}^{t} T(\tau) \; E(e_i(\tau)) \; d\tau, t_0 \geq 0, \tag{3.2}$$

where t_0 is the initial time, $T(\tau)$ is a function of time and $E(e_i(\tau))$ is a function of error

$$e_i(\tau) \;=\; |actual_i(\tau) - target_i(\tau)|.$$

Here, $target_i$ is again the target value and $actual_i$ is the actual value (the value measured by a sensor) of the i-th controlled quantity.

We could further add a weight function of time to the integral (e.g. $w(t) = 1/t$ to obtain average value), but we shall not use it now.



Function $E(e_i(\tau))$ describes our concern over the actual error *size*. On the other hand, the function $T(\tau)$ characterizes how much we care about *when* the particular error occurs. Its value at a given time can also be perceived as weight assigned to the corresponding error function value.

Common criteria

There are multiple commonly used criteria of step response quality: IAE – Integral of Absolute magnitude of Error, ITAE –Integral of Time times Absolute magnitude of Error, ITSE – Integral of Time times Squared Error, and ISE – Integral of Squared Error. They all fit the framework described above, giving us common choices of functions $E(e_i(\tau))$ and $T(\tau)$:

$$IAE(t) = \int_0^t |e(\tau)|\, d\tau,$$

$$ITAE(t) = \int_0^t \tau|e(\tau)|\, d\tau,$$

$$ISE(t) = \int_0^t e^2(\tau)\, d\tau,$$

$$ITSE(t) = \int_0^t \tau e^2(\tau)\, d\tau,$$

The ITAE criterion has been shown to be superior to the other criteria in [17]. It also fits our purpose the best. We need to tune the controllers to be stable but we are not particularly concerned with other possible aspects (e.g. magnitude of overshoot over the setpoint value). In the beginning of the time interval, we do not really care about the error. As the time progresses, we want to place increasing weight upon the error.

Modifications of ITAE

We shall further modify the ITAE criterion to suit our needs better, but the idea remains just the same. We shall discuss several objective functions in the form of (3.2).

First, we consider the fact that we measure the step response over time interval longer than one second. We do not want to diminish the error that happens when $t < 1s$, while enlarging those at time $t > 1s$. For that reason (and simplicity of notation), we slightly redefine the ITAE criterion:

$$ITAE(t) \;=\; \int_0^t (\tau + 1)|e(\tau)|\; d\tau. \tag{3.3}$$

We might want to disregard the beginning of the time interval, as the step response may behave wildly in the beginning, regardless of its neat convergence later. We shall see from the experiments, that this approach leads to better-suited description of the optimization problem by the objective function as it removes possible source of confusing information.

$$SITAE(t) \;=\; \int_{t_0}^t (\tau + 1)|e(\tau)|\; d\tau, \tag{3.4}$$

where $t_0 > 0$ marks the shifted beginning of measurement.

However, we may also take the opposite approach and instead of disregarding the beginning of the interval, we may penalize the nonconvergence.

$$ITAEP(t) \;=\; \int_0^t (\tau + 1)|e(\tau)|\; d\tau \;+\; \int_{t_p}^t P\big(\tau, e(\tau)\big)(\tau + 1)d\tau, \tag{3.5}$$

where $P\big(\tau, e(\tau)\big)$ is the penalization function applied at the interval $[t_p, t]$. We shall set

$$P(\tau) \;=\; p\;(\tau + 1)\;\big(|e(\tau)| - tol\big)_+, \tag{3.6}$$

where p is the penalty constant, tol is the defined tolerance and $(\ldots)_+$ is the positive part (i.e. the penalty is not applied, if the error is smaller than tolerance). This criterion too fits the form (3.2) as it can be rewritten using characteristic function of interval.

The penalty function causes the objective function to be steep at some points, making it ill-conditioned (compared to the previous criteria). As verified experimentally, this slows down the optimization and magnifies the cost differences among the individual runs (e.g. within ten runs of the same test, half of the trials takes ten times longer to complete than the other half).

We can also want to try higher-polynomial function of time to weight the

error. Hence

$$IQTAE(t) \;=\; \int_0^t (\tau+1)^2 |e(\tau)|\; d\tau, \tag{3.7}$$

and

$$SIQTAE(t) \;=\; \int_{t_0}^t (\tau+1)^2 |e(\tau)|\; d\tau, \tag{3.8}$$

The particular form of the penalty function is, of course, matter of one's opinion and taste. We try out this one and see that from the above reasons, other objective functions perform much better. While there is still a possibility of finding a better-suited penalty function, we do not need to search for it as we already have a good objective function. For the same reason, we do not need to experiment with higher-polynomial functions of time.

3.5 Choosing the right optimization method

According to "no free lunch" theorems [54], there is no algorithm that would perform better than any other on all classes of problems. To choose the right optimization method, characteristics of the particular problem must be considered.

3.5.1 Problem characteristics

Given the problem described above, there are many natural constraints.

Black box

First and foremost, we cannot disassemble the model to look inside and analyze what processes define its behavior, as it is too complex. We must take the model as a black box. All we can do is to evaluate it when a set of controllers' parameters is given and see how the result fits our idea of "good" behavior.

No pretty properties

We know very little about the function we want to optimize, so any simplifying assumptions could be misleading. It is certainly non-convex, non-linear, non-quadratic and (highly) multimodal – there may be multiple local optima, where we do not want to get stuck when searching for a global optimum. The function is probably continuous, but possibly ill-conditioned (i.e. very steep at some points – this happens especially when penalties are used). There are no derivatives available and we can hardly assume that the function is smooth, so neither the derivative approximations would necessarily be sensible. We do not expect the objective function to be noisy. However, this is just guessing, we know nothing in advance.

Non-separability (coupling)

Definition. *Separable function. A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is separable if*

$$\arg\min_{(x_1,\ldots,x_n)} f(x_1,\ldots,x_n) \;=\; \Big(\arg\min_{x_1} f(x_1,\ldots),\ldots,\arg\min_{x_n} f(\ldots,x_n)\Big).$$

Our objective function is non-separable. We cannot tune one parameter independently of the others and expect that all will be well in the end. One controller's parameter's value strongly affects other parameters. Further, one controller's setting may affect the behavior of other controllers. This property is called coupling and it is the reason why tuning multiple controllers is hard. However, relationships between the variables are far from random. We would like the algorithm to mine and use this information to improve its search for an optimum.

When the coupling is weak, it can be expected that "pre-tuning" the controllers separately (i.e. using the simple tuning tools to one controller at a time, while the others are fixed) can give sensible results. The stronger the coupling is, the more useless is the pre-tuning.

Expensive evaluations

The function evaluations are expensive – they take a significant amount of computational time (minutes) while the time budget is limited. Usually, we want an overnight computation. Therefore, we must judge carefully when and where to evaluate the function.

Robustness

Last but not least, the algorithm we will use must be robust. In our application, we cannot tolerate an algorithm that would not provide us with any result in the end of its computations. If unavoidable, it must not happen very often, since every such case irritates the user – our customer.

3.5.2 Looking for a suitable method

Having considered the limitations and requirements described above, it is clear that we need a very robust black-box derivative-free global optimization algorithm. These requirements leads us to the large family of evolutionary methods. However, how do we decide, which particular algorithm best fits our problem?

Black-box optimization benchmarking

In order to systematically compare global optimization algorithms, a platform called COCO (COmparing Continuous Optimizers) has been developed [27]. It provides benchmark functions and experimentation templates that are easy and free to use for anybody who wants to plug in and test their algorithm.

There are 24 test noiseless and 30 noisy test functions provided. We are more interested in the noiseless test suite. These functions are divided into five groups:

1. separable functions (e.g. sphere or Rastrigin functions)

2. functions with low or moderate conditioning (e.g. Rosenbrock function)

3. functions with high conditioning and unimodal (e.g. Sharp Ridge function)

4. multimodal functions with adequate global structure (e.g. Weierstrass function)

5. multimodal functions with weak global structure (e.g. Schwefel function)

Definitions of the test functions along with illustrations are available at the website of COCO [28]. More details and explanation are provided in [25].

The COCO platform has been used for BBOB (Black Box Optimization Benchmarking) workshops that have been a part of the GECCO (Genetic and Evolutionary Computation) conference since 2009.

The results of BBOB 2009 provide us with valuable comparison of many well known algorithms. Later BBOBs provide further comparisons of dozens of methods, including many variants of CMA-ES, which was the general "winner" of BBOB 2009 [21, 2].

For the sake of self-containedness, we shall review the most important results of BBOB 2009 that led the author to believe that CMA-ES is well suited for this particular application. The following graphs are directly from [21]. List of all tested algorithms can be found in Appendix C.

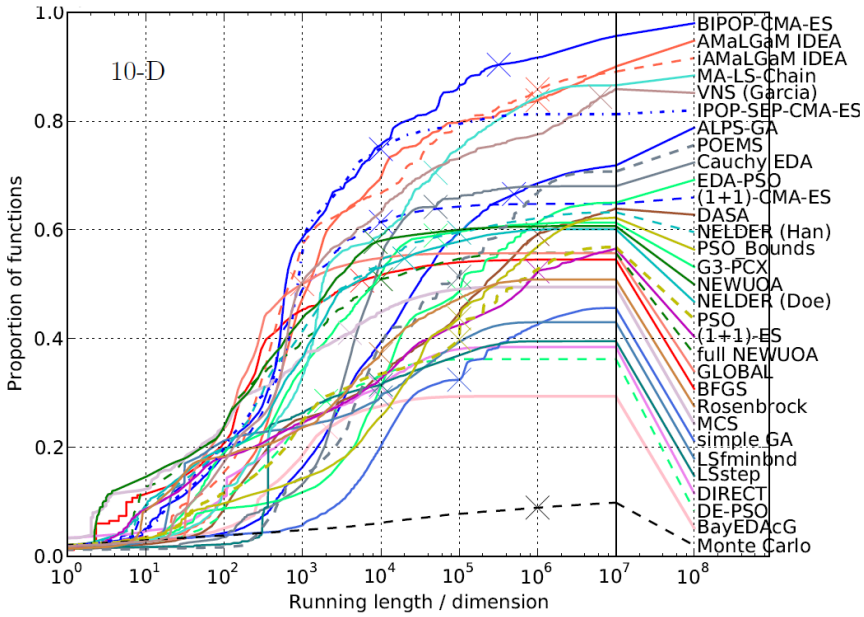


Figure 3.4: BBOB 2009: all noiseless test functions, dimension 10, stopping tolerance within 100 and 10^{-8}.

Figure 3.4 shows the performance of the methods on all the test functions in dimension 10. Pure random search method (Monte Carlo) is provided for comparison. The algorithms' run time is measured in number of function evaluations per dimension. The vertical axis shows the success rate that characterizes the

robustness – proportion of problems an algorithm is able to solve with given budget. We can see that, given budget of at least $10 \times 10^3 = 10^4$ function evaluations, BIPOP CMA-ES trumps the other algorithms.

Differences between the methods grow proportionately with dimension of the problem. For small dimensions, CMA-ES does not perform best of all, though it never fails (i.e. it always provides a solution, albeit later than other methods) and the success rate is in general very high. Also the required precision affects the success rate.

However, we are interested in multimodal functions. Figures 3.5 and 3.6 show that BIPOP-CMA-ES wins this category.

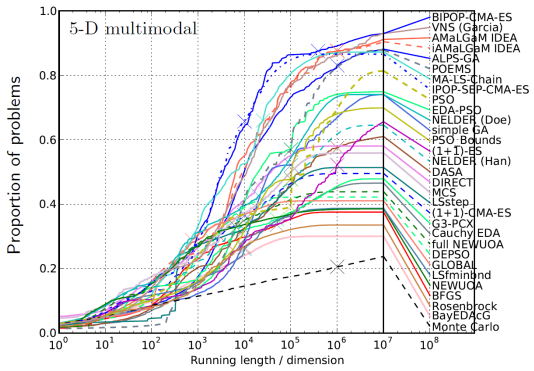


Figure 3.5: Multimodal functions in dimension 5.

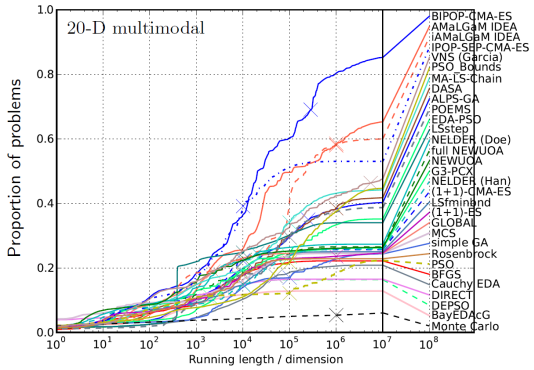


Figure 3.6: Multimodal functions in dimension 20.

Now we can look at the algorithms' performance respective to the function classes.

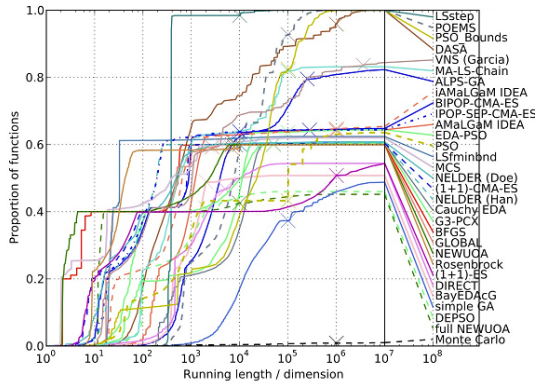


Figure 3.7: Separable functions $f_1 - f_5$.

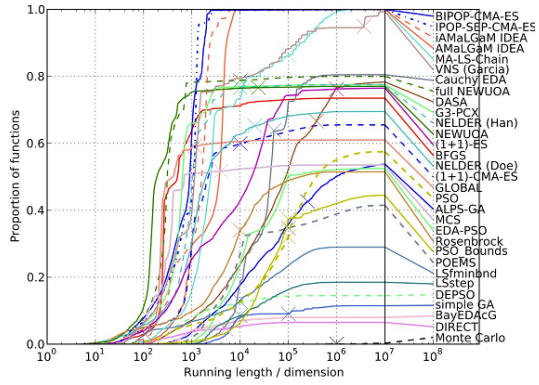


Figure 3.8: Functions with low or moderate conditioning $f_6 - f_9$.

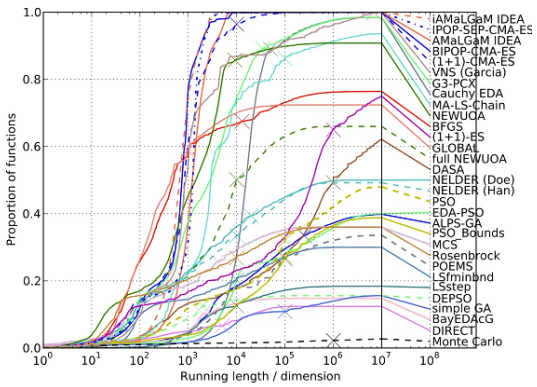


Figure 3.9: Ill-conditioned functions $f_{10} - f_{14}$.

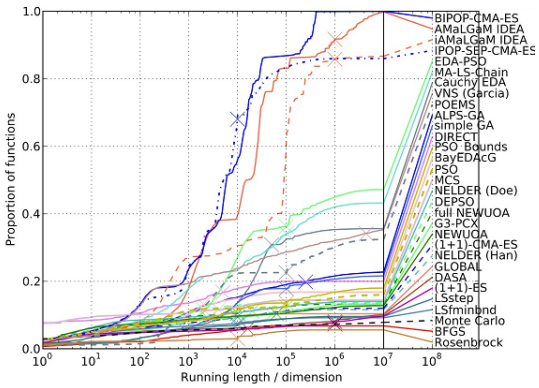


Figure 3.10: Multimodal structured functions $f_{15} - f_{19}$.

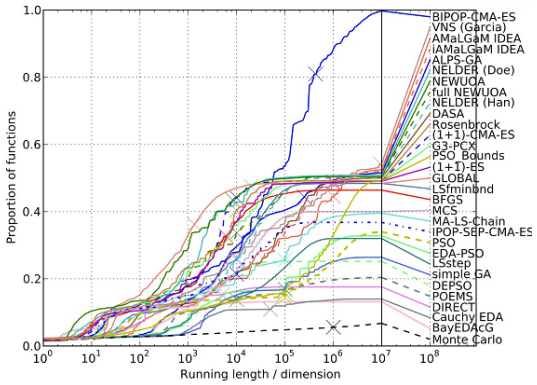


Figure 3.11: Multimodal weakly structured functions $f_{20} - f_{24}$.

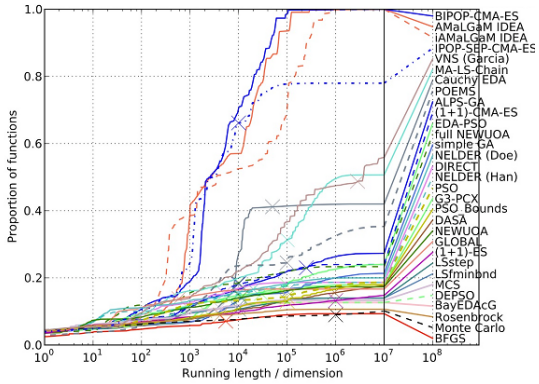


Figure 3.12: Non-smooth functions f_7, f_{16}, f_{23}.

Conclusion

Following the results above, it is clear that (BIPOP-)CMA-ES is a good choice, even though it can be outperformed for some sorts of problems (complete results for this algorithm can be found in [26]).

Also in newer BBOBs, e.g. [3] (exhaustive list of all papers is available at COCO's website [27]), dozens of CMA-ES variants are compared and are highly successful.

We will use the variant named elitist BIPOP-aCMA-ES-CSA, which was described earlier. Its use (as opposed to using other variant) is based purely on trial-and-error experiments and observations of the models' behavior described further in section 3.7. However, we shall see that its performance is excellent.

Of course, we would like to compare it with other methods' performance. However, tuning the CMA-ES method to fit our problem was a nontrivial task that took several months. Implementing another method would too require significant effort and, at the end, the comparison would only show how well we are able to tune both methods and nothing more.

Moreover, our task is to find a working solution to the problem, if it exists, and to implement it in shortest time possible. Therefore it is not desirable to spend months trying out and tuning other methods, once a satisfactory working solution has been found and can be implemented.

Nevertheless, further research and improvements are possible and the possibilities are foreshadowed in section 3.9.

3.6 Prototype implementation and testing models

Once we have the tools, we need to test if the suggested solution to our problem is viable. We do this by implementing and testing a prototype.

3.6.1 Prototype implementation using Python

For convenient prototype implementation, we use scripting language Python that is becoming a great tool for numerical computations. The necessary numerical libraries are provided by SciPy ecosystem [32] and the most advanced CMA-ES routine too is provided by Hansen as a Python script [18]. It also is a natural way how to connect WAVE with external programs. Last but not least, this language and its libraries are free and commonly used, also within the company, so the prototype shall be easy to understand when implementing the final version as a part of WAVE (planned for the autumn release in 2016).

3.6.2 Basic testing model

Models of real engines are very complicated and they take several minutes to run. Therefore, we need a simple, quick model for testing purposes. However, it must be complicated enough to be a good representative.

Our problem is primarily about tuning multiple controllers. Therefore, the basic testing model we use has three coupled controllers. It takes about 11 seconds to run, which is very fast compared to usual run time in minutes.

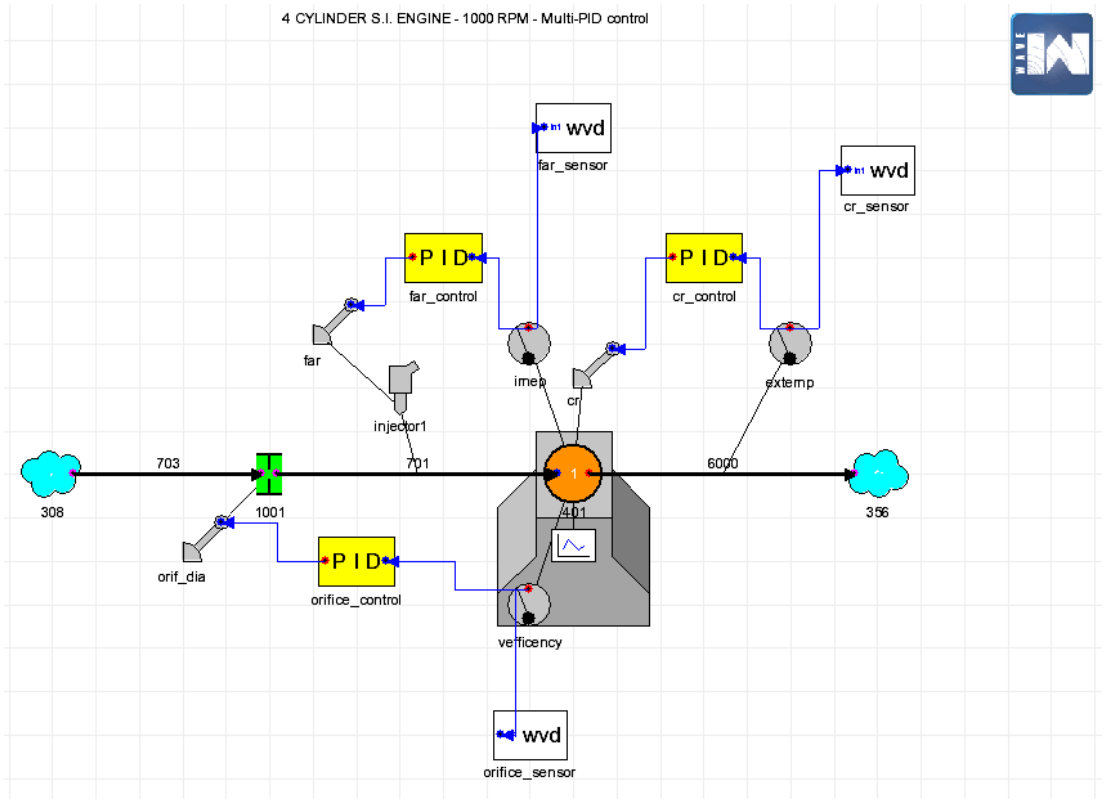


Figure 3.13: Basic testing model in WAVE.

In this model, we can see a single cylinder (orange circle) engine. The blue "clouds" on the left and on the right are called ambients and they contain information about the surroundings (e.g. ambient pressure and temperature or initial fluid composition). The thick black lines connecting the ambients with other elements are ducts, where the fluids flow. The green element is an orifice, opening of variable diameter. The yellow PID elements are the controllers, whose gains we want to tune. The inside of the element is shown in figure 3.14. The "calibrate gains" button enables the user to use two simple tuning methods (fit for one-controller systems only).

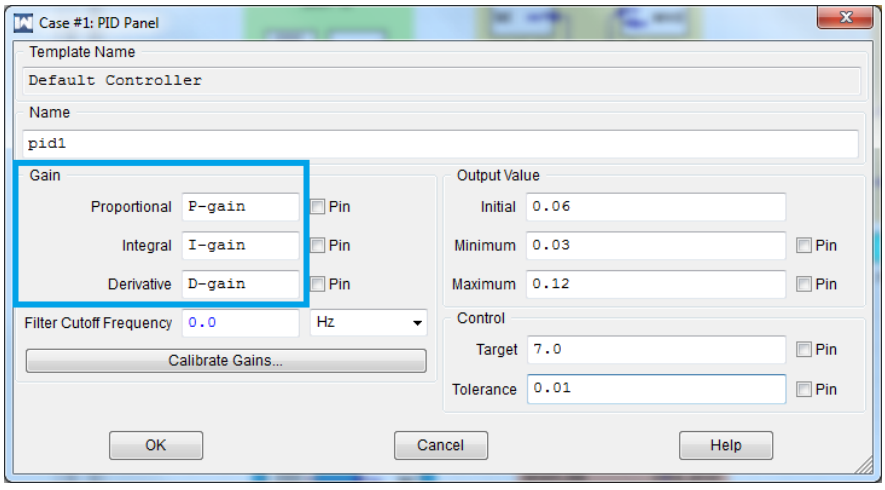


Figure 3.14: The PID element panel.

The arrow-like elements are actuators that perform the actual mechanical control based on the control signal outputted by the corresponding PID controllers. The first one controls the fuel injector, the second affects the compression ratio (the ratio of the maximum to minimum volume in the cylinder and the third controls the orifice diameter.

The controlled quantities are:

- Indicated mean effective pressure (IMEP) — the average pressure acting upon the piston during its cycle. It is controlled by adjusting the fuel-air ratio, i.e. how much fuel we inject into the rail.
- Exhaust gas temperature. We control the compression ratio, i.e. the ratio of largest and smallest possible capacity of the combustion chamber.
- Volumetric efficiency — the ratio of the volume of fluid actually displaced by a piston. We adjust the diameter (opening) of the orifice.

Clearly, they have nontrivial influence on each other, so the controllers are coupled.

The actual values of these quantities during the process are measured by sensors, which are depicted as gray circles. The sensors' outputs are then forwarded to the corresponding controllers and they are also plotted out by the "wvd" elements. These are the step responses that we want to observe and evaluate by the objective function.

In figure 3.15, we can see two step responses. Controllers tuned by an engineer (hereafter called the baseline solution) result in the dark blue step response. Solution found by our method is in light blue. Clearly, the cyan solution is much better in two cases (i.e. the convergence to the target value is faster) and slightly worse in one.

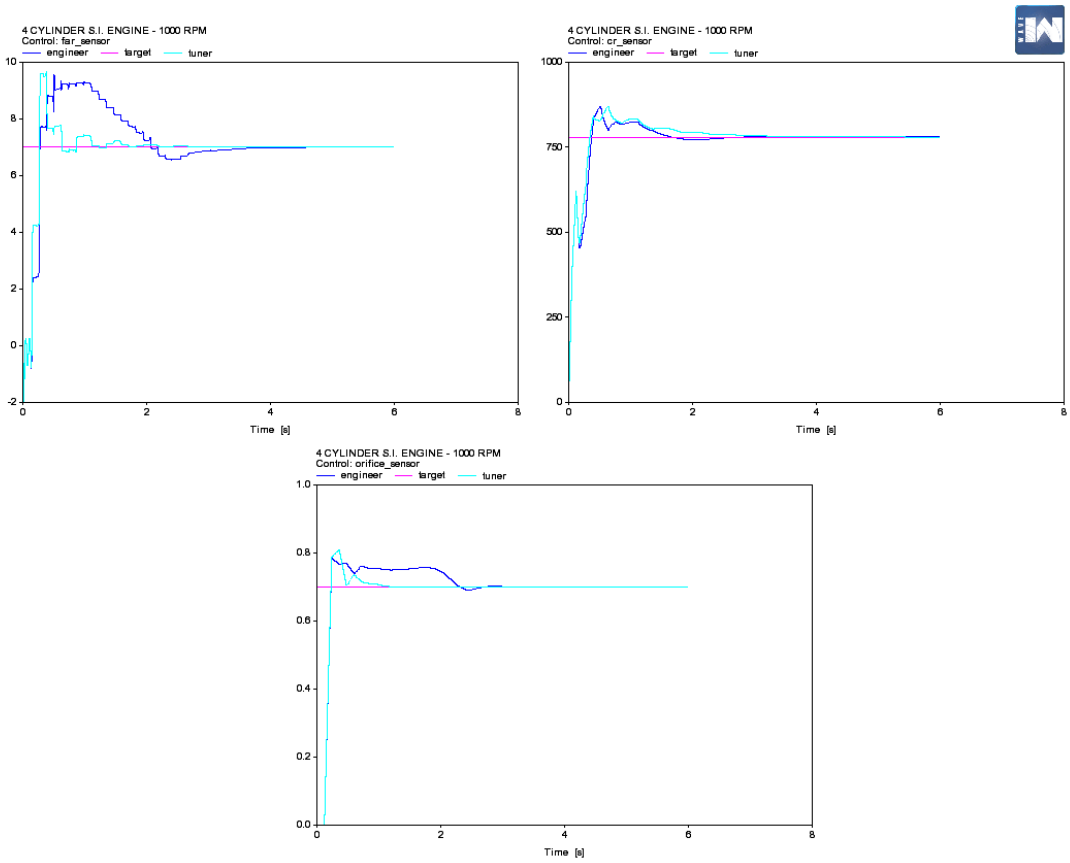


Figure 3.15: Basic testing WAVE model: comparison of good solutions. Pink: the target value, dark blue: tuned by engineer, light blue: tuned by the algorithm.

On the other hand, most of candidate solutions in the tuning process are not acceptable. They may look for example like those in figure 3.16. The dark blue solution converges but to a different value that is wanted. The light blue solution oscillates. The green solution looks good for the fuel-air ratio and volumetric efficiency step responses but it is unsatisfactory for the compression ratio.

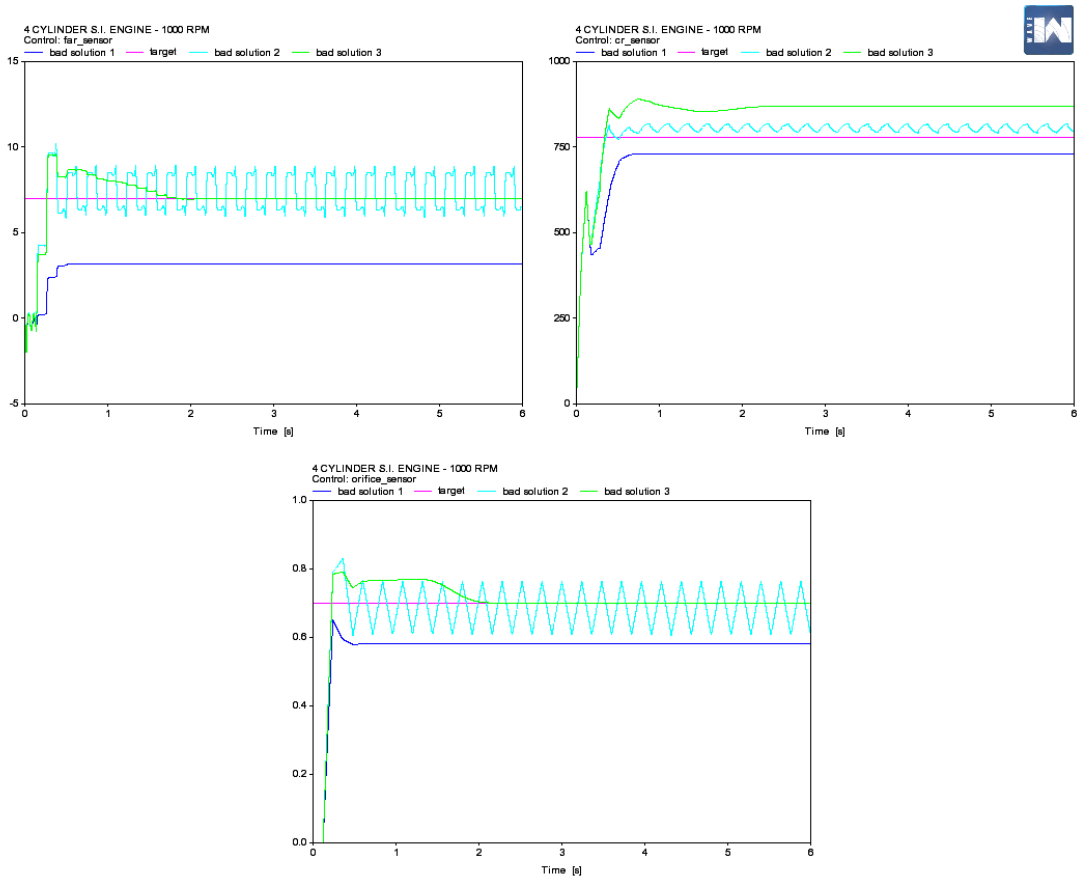


Figure 3.16: Basic testing WAVE model: unsatisfactory solutions.

3.6.3 Other models

We use more complicated and real world models for the final verification of the method. However, we do not have very many real models with multiple controllers available. Also, these tests are very time consuming. For these reasons we leave thorough testing for the beta version where parallelization as well as more benchmarks will be available.

Unfortunately, we cannot show the layout of the real world models as they are confidential.

We will also use two WAVE example models to verify that (and how well) the method works for models with a single controller in its typical and important use in a turbocharger. One of them is shown in figure 3.17. This is a twin turbocharger model, where the controller controls boost pressure (and thus the engine's efficiency and power output) by adjusting two turbines. These turbines then force more or less air (and proportionately more fuel) into the combustion chamber than atmospheric pressure alone.

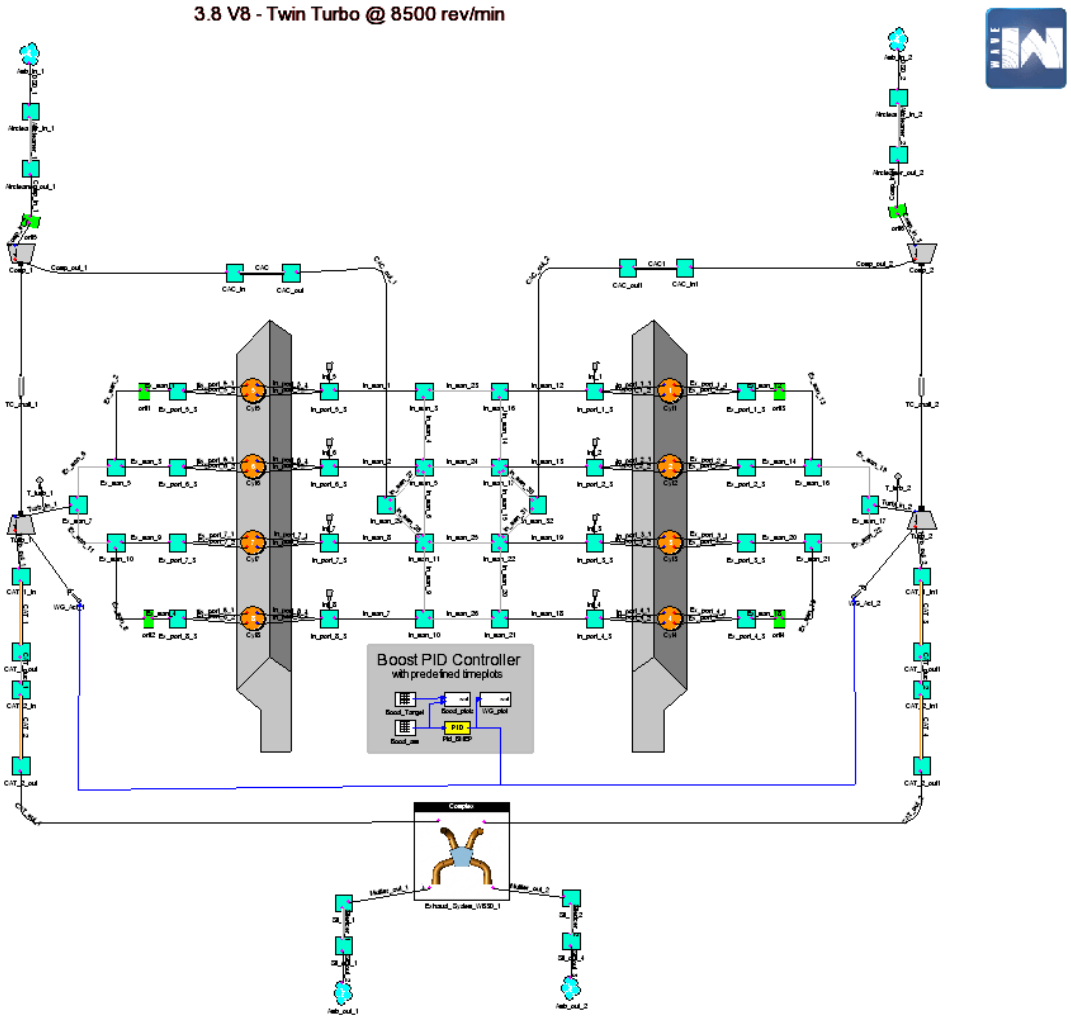


Figure 3.17: Twin turbocharger model with a single controller.

3.7 Experimental calibration of the method

Having settled what we want to do and how, the greatest amount of work is yet awaiting us. We have to try out different objective functions and algorithm variants and settings.

This section is about the process of experimental calibration of parameters of both the algorithm and the objective function. Over the span of seven months, hundreds of tests were run in search of a good setting and its verification on real-world models. We will use selected data of the actual experiments to illustrate gradual improvements of the method.

As the method uses randomness, each run is different and takes different amount of time. It is therefore necessary to run the same test several times to get a trustworthy result. In our experiments, we will always use ten runs of the same test (unless stated otherwise) and show the minimal, maximal and average run time until a target function value was hit. This value was set empirically to 0.5 based on what a good solution looks like for the given model.

3.7.1 The goal of the experiments

As stated earlier, we search for a usable solution. That is, we do not need a perfect solution, nor is it desirable to search for it once a satisfactory one has been found, should it mean significant delay.

Regarding the performance/run time, it is highly desirable to have overnight computation. There might be an exception for extra large models but roughly 16 hours is our limit. That is, having a one-minute model, we can perform 960 evaluations. For two-minute or three-minute models, the number decreases to 480 and 320 function evaluations, respectively. Longer models are not exceptional.

On the other hand, the algorithm is naturally parallelizable. All individuals within one generation can be evaluated independently of each other. For two-dimensional problem (i.e. one PI controller), there are at least 6 individuals in the population (the number increases after restarts). Therefore, as model evaluations take up most of the computation time making costs of the core algorithm negligible, we can speed up the whole process six times. Larger models may require more restarts with larger populations and/or the dimension is higher, magnifying the effect of parallelization as well.

Considering one, two and three-minute models once again and using parallelization of factor 6, we are provided with budget 5760, 2880 and 1920 function evaluations, respectively. Of course, these numbers give us only very rough approximation, but lets consider 3000 function evaluations as the borderline of usability for regular, small and medium-sized, models – the value that we must reach, though it is still highly desirable to reach even smaller numbers.

3.7.2 Verification of the idea

At first, we did not know whether the suggested approach would work and be practically usable. This was a serious threat since nobody had succeeded yet, especially none of our competitors (according to WAVE support team and project management). Therefore, the first step was to verify the whole idea, find out whether it was viable, regardless of the run time.

For that purpose, the basic testing model was set up with a very good starting point very close to the baseline solution. The ITAEP criterion (3.5) was chosen as the objective function. For optimization, BIPOP-CMA-ES routine of software package DEAP [12] was used.

After little adjusting (e.g. larger penalty constants seemed to produce better results), these first trials proved that the strategy may work. However, the number of function evaluations was huge considering the excellent starting position: about six thousand.

3.7.3 Tuning the algorithm

Over time, many improvements were made based on experiments with the basic testing model. We will look how some of the parameters that were tuned and calibrated affect the overall performance of the method. In the following tables, comparisons are always made with the best method found (i.e. when all parameters of the algorithm are calibrated and so is the objective function), the difference being just the particular parameter.

There are many more parameters that can be, and some of which have to be, adjusted. However, we leave this part confidential.

Hansen's version of CMA-ES source code [18] is further used. It is slightly more complicated than DEAP [12] but it provides better tools and gives the user much more control over the algorithm.

Scaling

First and foremost, scaling of variables has to be introduced. The initial step length is set to 1.0 and, as signs of the variables have a huge effect, the starting point is fixed to always be the zero vector. Then the approximation of solution that used to be the starting point becomes the vector of scale coefficients.

Example. Let the initial approximation of solution be:

$$(15.0,\ -0.24,\ 123.3,\ -0.0008).$$

By removing the signs, we create the vector of scales:

$$s = (15.0,\ 0.24,\ 123.3,\ 0.0008).$$

Then, when the algorithm uses (wants to evaluate) for example vector

$$(1.0,\ 0.1,\ -1.0,\ -20.0),$$

it is scaled element-wise by vector s to become vector of actual gains for the model:

$$(15.0,\ 0.024,\ -123.3,\ -0.016).$$

In terminology of evolutionary computing, we thus have the genotype (what numbers the algorithm sees and computes with) and the phenotype (the actual parameters for the WAVE model).

Hereafter, we shall still call the initial approximation solution the starting point, even though the starting point is actually the zero vector.

Unless stated otherwise, in the tests of the basic testing model in this chapter, we use the same starting point/scales that is given by a simple one-controller tuning method and is further discussed in section 3.7.4.

Elitist selection

Using non-elitist parent selection scheme was observed not to exploit the advantage of a very good solution, if it is found early in the search. It would be used as a parent within that one generation, which produced it and then be lost.

Elitist selection, as desribed in section 2.3.2, enables much faster convergence, as the parent set is chosen from the current generation and also its parents.

On the other hand, it may and does sometimes lead to premature convergence. However, this is compensated for by execution of more restarts.

Active search

We further improve the search by using negative updates of the covariance matrix, an upgrade described in section 2.3.3.

Table 3.1 summarizes the improvements of performance.

combination	min	max	average
elitist & active	268	2269	1098
no upgrade	812	> 12000*	> 5334
elitist only	327	5361	2184
active only	644	10877	4145

Table 3.1: *) Number of evaluations exceeded the maximum number of iterations allowed. This affects the corresponding average value.

It is important to note that there are many satisfactory solutions. That is, not every successful run ends with the same good solution.

Step size adaptation method

There are two options how to adapt the step size. Again, the default version CSA (Cumulative Step-size Adaptation; described in section 2.1.4) gives better results than the alternative TPA (Two-Point step-size Adaptation, [24], which, roughly, implements a line search along the direction of the latest mean shift).

method	min	max	average
CSA	268	2269	1098
TPA	926	> 12000	> 4724

Table 3.2

Population size

Experimenting with the initial population size showed that small populations fit our problem better. They cause faster adaptation, as there are more generations for the same cost than with a large population. The default size of the initial population is

$$\lambda_0 \;=\; \lambda_{\text{def}} \;=\; 4 + \lfloor 3 \log(N) \rfloor \,,$$

where N is the problem dimension. Using smaller populations is discouraged by Hansen [19].

Number of parents is by default given as

$$\mu_0 = 1/2\lambda_0.$$

In table 3.3, we compare results of tests with double initial population size. The parent set size μ_0 is either doubled too (i.e. it is half of the new initial population) or it stays the same (i.e. it becomes a quarter of the new initial population).

λ_0	μ_0	min	max	average
λ_{def}	$1/2\ \lambda_{\text{def}}$	268	2269	1098
$2\times\ \lambda_{\text{def}}$	λ_{def}	1533	8812	3273
$2\times\ \lambda_{\text{def}}$	$1/2\ \lambda_{\text{def}}$	697	10973	4181

Table 3.3

Restart criteria

An absolutely vital step is to tune the restart parameters. As this in particular is know-how that is to be kept confidential (and it is not particularly interesting from the mathematical point of view), we will not discuss it. However, lets illustrate the point by table 3.4. It shows how slight changes of one restart parameter (its best setting that was found) affect the number of evaluations.

parameter value	min	max	average
best	268	2267	1098
0.8× best	987	8980	3147
1.2× best	674	4990	2058
2.0× best	871	4960	2837

Table 3.4

Termination criteria

Every model being different, it is impossible to set one objective function value as the universal termination criterion. User's judgment is necessary.

3.7.4 Calibrating the objective function

The choice of the objective function is not as straightforward as it may seem. Different numerical values must be reflected by different restart and stopping criteria

of the algorithm, which themselves affect the total run time very much. Therefore, though striving to create equal conditions for all the objective functions, it is impossible to provide rigorous comparison.

Comparison of function values in three points

To have get some intuitive insight regarding the objective functions, lets look at the outputs of the basic testing model with parameters given by the baseline solution (tuned by an engineer), rough starting approximation (i.e. not very good but also not too bad a solution) and a good solution found by the method.

The baseline solution of the basic testing model is

$$baseline = (-0.0001,\ 0.04,\ -0.2,\ -0.01,\ 0.1,\ -20.0). \tag{3.9}$$

The first three elements are proportional gains and the other three integral gains; the derivative gains are set to zero. For the testing purposes, "starting point" given by Ricardo No Lag method was used. Its value is

$$RNL = (-0.557179,\ 0.134853,\ 231.682,\ -4.78055,\ 1.40074,\ 1287.91). \tag{3.10}$$

To have also a comparison with a very good solution (there are many of them, though), we use the vector

$$good = (-0.00736,\ 0.00365,\ -5.08174369,\ -0.04944,\ 0.06595,\ -37.36658). \tag{3.11}$$

The corresponding outputs are shown in figure 3.18.

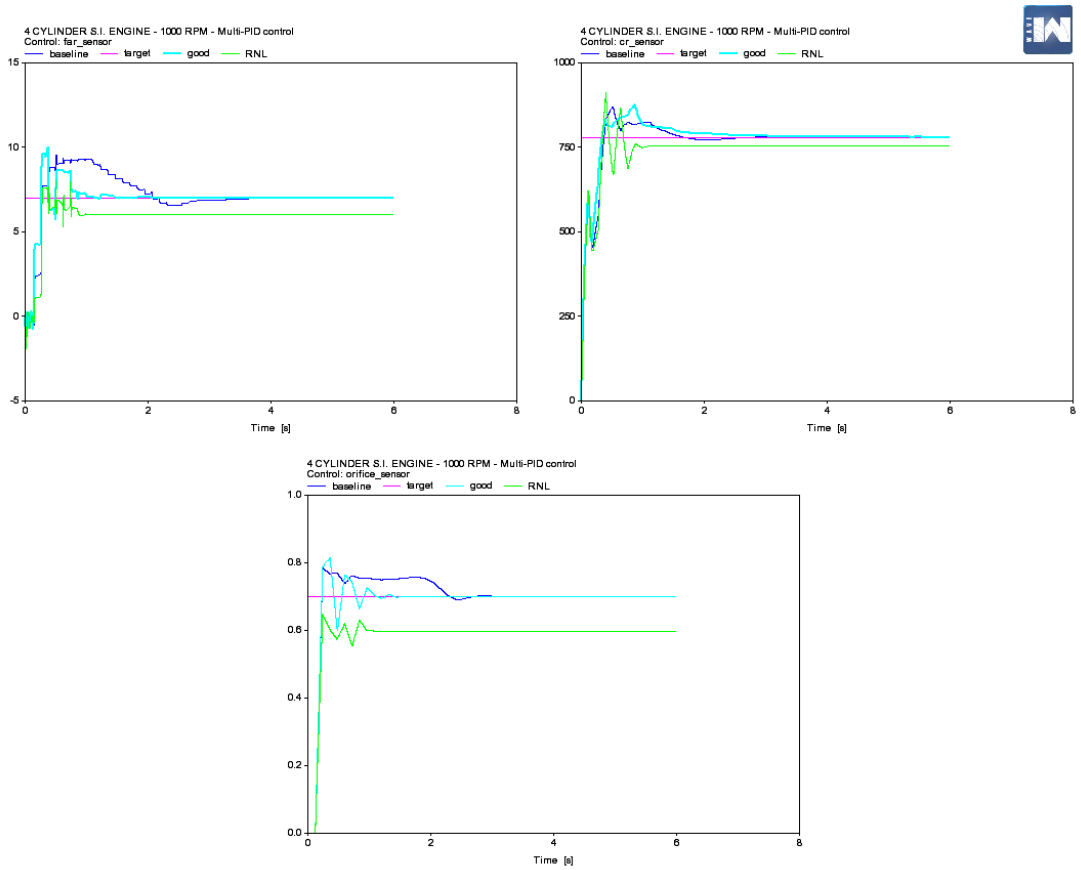


Figure 3.18: Basic testing WAVE model: comparison of three solutions of different quality. Pink: the target value, dark blue: baseline solution (tuned by an engineer), light blue: good solution, green: RNL starting point – poor solution.

Table 3.5 summarizes the corresponding values of the objective functions described in section 3.4.2. We can see what effect a large penalty has.

objective		baseline	RNL	good
ITAE	(3.3)	2.01847	8.13594	1.02137
SITAE #1	(3.4)	1.22473	7.33359	0.30324
SITAE #2	(3.4)	0.73805	6.96473	0.21377
ITAEP	(3.5)	8.69438	2294.18	3.06164
IQTAE	(3.7)	4.10874	36.75238	1.56846
SIQTAE #2	(3.8)	2.21393	35.06491	0.39313

Table 3.5

Comparison of performance

As reasoned above, results in table 3.6 illustrate rather than prove the comparisons of the objective functions and back up the resulting choice of the SITAE

objective function (version #2, i.e. with greater shift than #1, for this particular model).

objective	min	max	average
ITAE	919	9567	3687
SITAE #1	354	11169	2963
SITAE #2	268	2267	1098
ITAEP	2661	> 12000	> 6449
IQTAE	1173	10685	4354
SIQTAE #2	703	4930	2485

Table 3.6

Unfortunately, choosing the best shift highly depends on the model.

3.7.5 Testing robustness

Now, having tuned both the algorithm and the objective function, we shall look at the method's robustness. By robustness, we understand how sensitive the run time is to various starting points (i.e. various scaling of coordinates).

In the previous section, we used the RNL starting point (3.10) that was a good representative of a poor, yet not utterly terrible, starting point. When we compare orders of magnitude of the elements of the baseline solution 3.9 to RNL, we get:

1. P element ...+3 orders of magnitude,
2. P element ...+1 orders of magnitude,
3. P element ...+4 orders of magnitude,
1. I element ...+2 orders of magnitude,
2. I element ...+1 orders of magnitude,
3. I element ...+2 orders of magnitude.

When we compare RNL to the good solution (3.11), we see that all the elements are off by two orders of magnitude. This motivates our probes of robustness.

Robustness of PI controllers

We will take the baseline solution, multiply *each* of its elements by factors $10^{-3}, 10^{-2}, 10^{-1}, 10^{1}, 10^{2}$, and 10^{3} and observe the changes in run time. Table 3.7 summarizes the results.

starting point (scaling)	min	max	average
baseline	12	168	76
10^1 baseline	540	2064	1049
10^2 baseline	821	5700	2061
10^3 baseline	11857	> 18000	> 17386**
10^{-1} baseline	61	816	317
10^{-2} baseline	222	1334	592
10^{-3} baseline	259	1617	812

Table 3.7: **) Nine out of ten runs did not finish within the budget of 18000 function evaluations.

We are interested in the shortest average run time and the changes of the average run time with respect to the starting point.

We can see that for this model, the run times are good except for scaling by 10^3. Closer look at the behavior of the algorithm suggests that too large search area causes it to look for a needle in a haystack and "get lost" very far from the optimum.

Shorter run times when multiplying the baseline by small factors are also caused by the fact that there are good solutions at the "10^{-1} level" too, while there were none found at the "10^1 level" or "higher".

We can conclude that when the starting point (scaling) is within a reasonable range, the average run times are very usable. There was a single case of 5700 function evaluations but other than that, there was no run exceeding 3000 function evaluations.

Robustness of PID controllers

Now, we allow the solver to use PID instead of PI controllers, i.e. dimension of the problem becomes 9 instead of 6. As we do not have a PID baseline solution with nonzero D parameters, we estimate them based on the corresponding magnitudes of P and I parameters. We set

$$PID\_baseline \;=\; (-0.0001,\; 0.04,\; -0.2,\; -0.01,\; 0.1,\; -20.0,\; 0.001,\; 0.1,\; 1.0).$$

Based on the above results, this time we will use only factors $10^{-3}, 10^{-2}, 10^{-1}$, 10^1, and 10^2. The results are summarized in table 3.8.

starting point (scaling)	min	max	average
PID baseline	32	256	67
10^1 PID baseline	62	1462	1102
10^2 PID baseline	953	4130	2022
10^{-1} PID baseline	141	782	343
10^{-2} PID baseline	202	941	580
10^{-3} PID baseline	416	2324	1138

Table 3.8

Surprisingly enough, we get results that are very similar to the above table. We would expect the run time to prolong as the dimension of the problem increases but this happens only at the "10^{-3}" level. Other average values show no significant difference. It seems that the PID problem complexity is made up for by enabling overall better control of the system and thus also having more good solutions available.

As in the PI control experiments, we get a single case of an very long run of 4130 function evaluations. All other runs finished within budget of 3000 function evaluations.

Finding a good starting point

The question remains how to obtain a good starting point. In WAVE, we have typical uses of controllers and we also know that the gains' magnitudes are related to the units used by the model. Given that, we can set up a database that can further grow as more knowledge is added to it. Its implementation is now under construction. Also, we can use the simple tuning tools to get a rough estimate, especially when dealing with a few weakly-coupled controllers.

3.8 Testing real-world models

We conclude the testing phase with verifying the method's usability on real-world models. Our goal is not to perform thorough testing. That would require much larger suit of real data that we do not have at hand and it would also be extremely time consuming. Further experiments are left to the users of the beta version.

Our set of models consists of

- two models with 1 controller labeled M1.1, M1.2,
- three models with 2 controllers labeled M2.1, M2.2, M2.3,
- one model with 3 controllers labeled M3.1.

Even though our primary objective is to tune multiple controllers, we include the single-controller models as they are very important in practical use and we

want our method to work for them as well (as an upgrade of or alternative to the current methods).

For all these models we have a baseline setting of controllers. Unless stated otherwise, these baseline settings are always PI controllers, i.e. the derivative gains are set to zero.

We will try five starting points (scaling) for the single-controller models: the baseline solution and its element-wise multiples by factors $10^{-2}, 10^{-1}, 10^{1}$ and 10^{2}. For the larger, multi-controller models, we will try only three starting points: the baseline solution and its multiples of 10^{-1} and 10^{1}.

Also, we will run each test only five times. Even though it is too little for serious statistics, it is enough to have an approximate idea about the algorithm's performance.

The main termination criterion is hitting the target objective function value. It was set empirically based on what a good solution looks like for each of the models (i.e. it is a different value for each model). The backup criterion is the maximum number of function evaluations.

3.8.1 Single-controller models

Model M1.1 is a turbocharger very similar to the one in figure 3.1 and M1.2 is the twin turbocharger shown in figure 3.17. They represent the typical use of a PI controller. Following tables summarize results of these models' tests.

MODEL M1.1			
starting point (scaling)	min	max	average
baseline	2	68	28
10^{1} baseline	35	153	79
10^{2} baseline	95	519	225
10^{-1} baseline	20	120	66
10^{-2} baseline	49	296	123

MODEL M1.2			
starting point (scaling)	min	max	average
baseline	1	22	9
10^1 baseline	4	28	11
10^2 baseline	80	225	187
10^{-1} baseline	34	100	51
10^{-2} baseline	57	181	94

All the run times are very short as the model dimension is only 2, especially when starting with a fairly accurate baseline solution.

We can see that sensitivity to the starting point corresponds with table 3.7. Here too we can see how enlarging the search area (by using larger scaling) effects the performance much more negatively than "small" starting points – observe the ratio between the average values for each model.

This behavior is especially apparent in model M1.2. The run time with the baseline and 10^1 baseline starting points are essentially the same (because there are good solutions at both these "levels") but when moving to factor 10^2, the number of function evaluations increases rapidly.

3.8.2 Multi-controller models

Models M2.1 and M2.2

A quick look at the first two two-controller models reveals that they do not pose a harder challenge than the single-controller models.

MODEL M2.1			
starting point (scaling)	min	max	average
baseline	11	66	35
10^1 baseline	244	280	255
10^{-1} baseline	4	32	21

MODEL M2.2			
starting point (scaling)	min	max	average
baseline	8	98	29
10^1 baseline	60	770	364
10^{-1} baseline	44	107	64

Here too, we can see how smaller scaling produces shorter run times. It is caused by the distribution of good solutions as well as the fact that larger area takes longer to search.

Comparing PI and PID control on a two-controller model M2.3

For model M2.3, we shall compare PI and PID control. The baseline value for PI control is given, and the D gains estimates are added to it to provide the starting point "PID baseline". By comparing the "baseline" run times, we can see that the D gains were chosen well.

MODEL M2.3			
starting point (scaling)	min	max	average
PI baseline	9	78	32
10^1 PI baseline	250	757	629
10^{-1} PI baseline	49	1188	347
PID baseline	10	91	57
10^1 PID baseline	274	857	522
10^{-1} PID baseline	82	1576	749

The extreme differences in minimal and maximal values of "10^{-1} PI baseline" and "10^{-1} PID baseline" tests are caused by unsatisfactory setting of restart parameter described in section 3.7.3. However, we can see that the same setting of the parameter is fine in other cases and models. This is the reason why it is so hard to estimate the restart parameters' values that would work without user interaction.

When adjusting the restart parameter to a more suitable value, the "10^{-1} PI baseline" and "10^{-1} PID baseline" run times drop.

starting point (scaling)	min	max	average
10^{-1} PI baseline	49	149	80
10^{-1} PID baseline	51	597	234

With this setting, the results of PI control resemble those of M2.2. In this case, the PID control does not improve the solution of PI control and its costs are acceptable, but significantly higher. However, we have only five runs of each so further testing might yet change the figures.

For this model, a different shift (described in section ss:objfun, equation (3.4)) was used. Choosing the right shift depends on the properties (speed of response) of the model.

Large model M3.1 with three PID controllers

PI control may not be able to fully control the system, as it is not sufficient for model M3.1. Here, one of the quantities does not converge – neither with the baseline values, nor could be a better PI solution found by the method. When we change the controller type to PID, getting a good solution is not a problem. This effect is apparent in the first graph of figure 3.19.

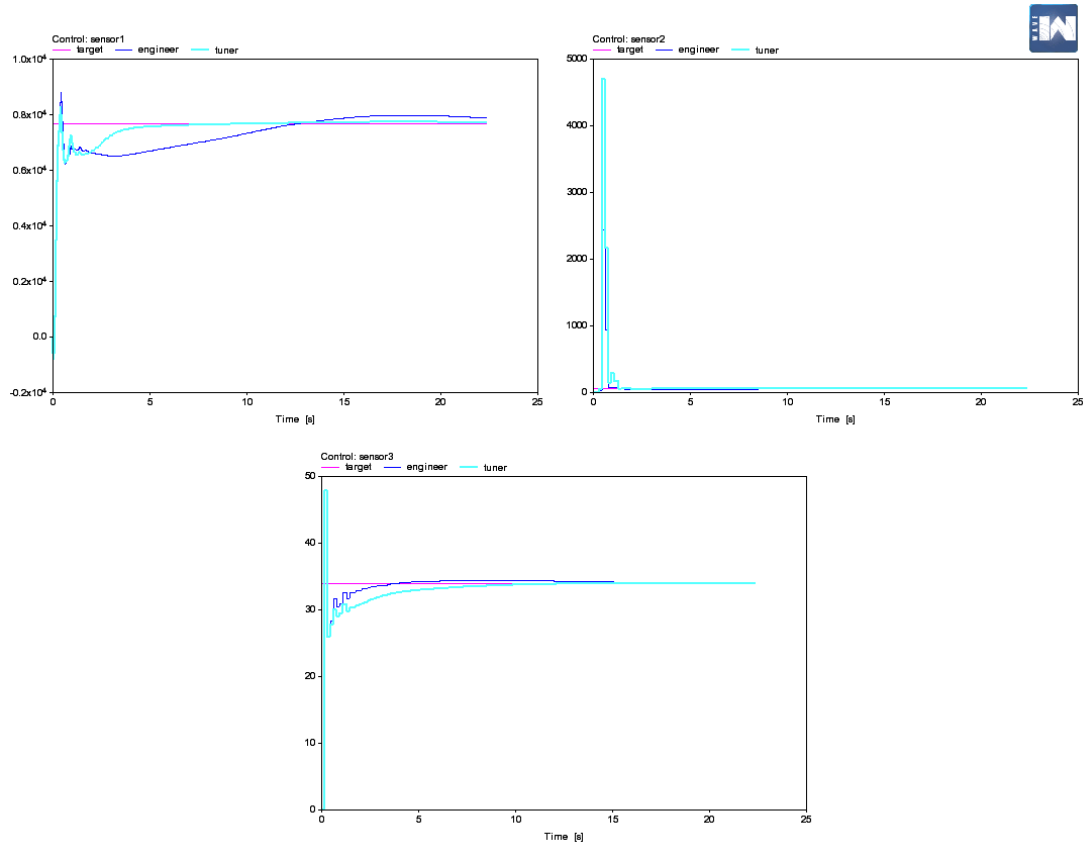


Figure 3.19: PI (dark blue) vs. PID (light blue) controlled system of $M3.1$. Pink: target value.

The following table summarizes the test results on M3.1. The starting point was obtained by estimating the D gains and adding them to the given PI baseline.

MODEL M3.1			
starting point (scaling)	min	max	average
PID baseline	41	331	152
10^1 PID baseline	827	1763	1268
10^{-1} PID baseline	179	3867	2476

It can be seen that this model is considerably harder to tune than the previous models, possibly due to its complexity. Contrary to the previous models, it is significantly easier to find an optimum with large scaling than small. Close look at the program's output reveals that this is due to problems of getting stuck in local optima, whose objective values are nearly globally-optimal. In such cases, farther solutions are not noticeably better and so the step size σ does not get large enough to skip "over the hills". While the algorithm has no trouble finding engineer-like solutions depicted in figure 3.19, it is hard to find a better one, where all three controllers converge.

Three out of five runs with the "10^{-1} PID baseline" starting point are longer than the desired 3000 evaluations. The shortest run time can be considered a happy coincidence. Another trial that had to be interrupted due to technical problems reached over 5000 evaluations. However, we must take into account the fact that this model is very large and an engineer was not able to tune it at all, so sensibly longer waiting time is a small price. In that light, we may still consider the algorithm successful.

3.8.3 Conclusion

The above results indicate that the method is usable for real-world models. For a good run time it is essential to have a good starting point, i.e. coordinate scaling. When each coordinate is off by one order, overall very good run time can be expected. For easier models, two orders do not pose a problem either.

Under such conditions, it seems that using PID instead of PI control does not make the problem much harder to solve. PID controllers may be able to control systems that PI controllers cannot. The difference, as well as the total run time, highly depends on the model.

3.9 Further improvements

There are multiple possible improvements that can be further investigated.

First and foremost, we can try out surrogate-assisted versions of CMA-ES. One is described e.g. in [38, 39, 36] and tested in [37]. Another one is fairly recent and much more promising combination of EGO (efficient global optimization of expensive black-box functions using Gaussian processes, [33]) and CMA-ES

explored and first tested in [40]. There is hope that the EGO part can be later plugged in to increase the performance.

Secondly, there is room for improvement regarding restart and stopping criteria and automatic detection of shift.

Conclusion

In this work, we have looked at a real-world problem of hard optimization: tuning of multiple coupled PID controllers within combustion engine models. We have formulated the problem, identified its properties, chosen an appropriate algorithm and tuned it to comply with strict practical limitations. In order to prove its workability, the whole method was verified on real models. The resulting method is now to be programmed as a part of WAVE simulation software by Ricardo Prague, s.r.o.

Appendix A: Basic vocabulary of probability theory

The aim of this appendix is not to provide extensive overview or rigorous definitions. It is included purely for the convenience of an educated reader, who is not an expert on probability theory. It is to remind them of some basic tools that are used in the description of CMA-ES. For more details see the textbook [51], for nice intuitive explanations see [31].

A.1 Elementary terms

In this section, we shall introduce the random variable and other terms closely related to it.

Random variable

A random variable or a stochastic variable is a variable, whose value is subject to variations due to chance (randomness). It can take on a set of possible different values, each with an associated probability.

Probability distribution

The probability distribution $F_X(x)$ is a function that associates the possible values of a random variable X with their probabilities:

$$F_X(x) \;=\; Pr(X \leq x).$$

It is characterized by its probability density function f:

$$F_X(x) \;=\; \int_{-\infty}^{x} f(t)\; dt.$$

Generalization for multiple dimensions is straightforward. For example, in two dimensions, it is

$$F_{XY}(x, y) \;=\; \int_{-\infty}^{x} \int_{-\infty}^{y} f(t, s)\; dtds.$$

Expected value

The expected value or mean $\mathrm{E}[X]$ (also μ or μ_X) of a random variable X is the probability-weighted average of all its possible values. The definition for a discrete random variable taking values $x_1, x_2, \ldots$ with probabilities $p_1, p_2, \ldots$ is

$$\mathrm{E}[X] \;=\; \mu_X \;=\; \sum_{i=1}^{\infty} x_i p_i.$$

The definition for a continuous random variable is

$$\mathrm{E}[X] \;=\; \mu_X \;=\; \int_{-\infty}^{\infty} x f(x)\; dx,$$

where $f(x)$ is the probability density function that characterizes the probability distribution.

Having $g(X)$ a measurable function of a continuous random variable X, its expected value is

$$\mathrm{E}\big[g(X)\big] \;=\; \int_{-\infty}^{\infty} g(x) f(x)\; dx,$$

where $f(x)$ is again the probability density function of X.

Generalization for random vectors, whose elements each are a random variable, is again straightforward. For example, for two-dimensional random vector (X, Y), both X and Y are random variables, whose expected values can be computed as above, for example:

$$\mathrm{E}\big[X\big] \;=\; \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x f(x,y)\; dxdy,$$

where $f(x, y)$ is the corresponding probability density function of two (random) variables. In general, expected value of any function $g(X, Y)$ is

$$\mathrm{E}\big[g(X,Y)\big] \;=\; \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y) f(x,y)\; dxdy.$$

Variance

The variance $\mathrm{var}(X)$ (also σ^2 or σ_X^2) is the expected value of the squared deviation from the mean:

$$\mathrm{var}(X) \;=\; \sigma_X^2 \;=\; \mathrm{E}\big[(X - \mathrm{E}[X])^2\big].$$

I.e. for a continuous distribution, it is

$$\sigma_X^2 \;=\; \int_{-\infty}^{\infty} (x - \mu_X)^2 f(x)\; dx.$$

It measures how much are the variable's values spread out. Small variance indicates that the data are clustered close to the mean (and thus to each other). Variance is always non-negative. The square root of the variance, σ, is called the standard deviation.

Covariance

Covariance $\operatorname{cov}(X, Y)$ or σ_{XY} is a measure of how much two random variables depend on each other:

$$\operatorname{cov}(X, Y) \;=\; \sigma_{XY} \;=\; \mathrm{E}\big[(X - \mathrm{E}[X])(Y - \mathrm{E}[Y])\big].$$

I.e. for a continuous distribution, it is

$$\sigma_{XY} \;=\; \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \Big((x - \mu_X)(y - \mu_Y) \Big) f(x, y)\; dxdy.$$

Covariance of a variable with itself is, by definition, its variance, i.e. $\sigma_{XX} = \sigma_X^2$. The sign of covariance shows the tendency in their linear relationship. If the variables tend to show similar behavior (greater values of one correspond with greater values of the other), the covariance is positive. In the opposite case, when greater values of one variable correspond with smaller values of the other variable, the covariance is negative. The magnitude of covariance is usually not interpreted, the correlation coefficient is referred to instead.

Correlation

Correlation or the correlation coefficient $\operatorname{corr}(X, Y)$ or ρ_{XY} can be perceived as the normalized version of covariance

$$\operatorname{corr}(X, Y) \;=\; \frac{\operatorname{cov}(X, Y)}{\sigma_X \sigma_Y}.$$

It describes the strength of the linear relation between two random variables. If X and Y are independent, then $\operatorname{cov}(X, Y) = \operatorname{corr}(X, Y) = 0$. The converse implication does not hold [51].

Sample mean and sample variance

When the true mean and variance of a distribution are unknown, we may need to compute them from an actually realized sample of that distribution. Such approximations are called the sample mean and sample variance [51].

Let $X_1, \ldots, X_n$ be independent and identically distributed random variables with mean μ and variance σ^2. Then their sample mean is

$$\overline{X} \;=\; \frac{1}{n} \sum_{i=1}^{n} X_i$$

and it is again a random variable. Its mean is equal to μ and its variance is σ^2/n.

The average square deviations of the sampled data is

$$\frac{1}{n} \sum_{i=1}^{n} \big(X_i - \overline{X}\big)^2$$

and its expected value is not σ^2 but rather

$$\frac{n-1}{n}\,\sigma^2.$$

Therefore, the (unbiased) sample variance is

$$s^2 \;=\; \frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n}\left(X_i-\overline{X}\right)^2\right) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i-\overline{X}\right)^2.$$

A.2 Normal distribution

The normal distribution is a continuous probability distribution very common in nature and it describes many phenomena. Due to this and its convenient properties (and the central limit theorem, see [51] or [31]), it is often used to represent real-valued random variable whose distribution is unknown.

Normal distribution in one dimension

The normal distribution is denoted by $\mathcal{N}(\mu,\sigma^2)$, where mean $\mu\in\mathbb{R}$ and variance $\sigma^2>0$ are its parameters. Its (probability) distribution function is defined as

$$F(t) \;=\; \int_{-\infty}^{t}\varphi_{\mu,\sigma^2}(x)\;dx, \quad t\in\mathbb{R},$$

where the integrand is the probability density function that defines the normal distribution:

$$\varphi_{\mu,\sigma^2}(x) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}}\;e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty<x<\infty.$$

The standard (normalized) normal distribution is centered at zero and its variance is equal to one: $\mathcal{N}(0,1)$.

A random variable X with normal probability distribution is called normally distributed (denoted $X\sim\mathcal{N}(\mu,\sigma^2)$).

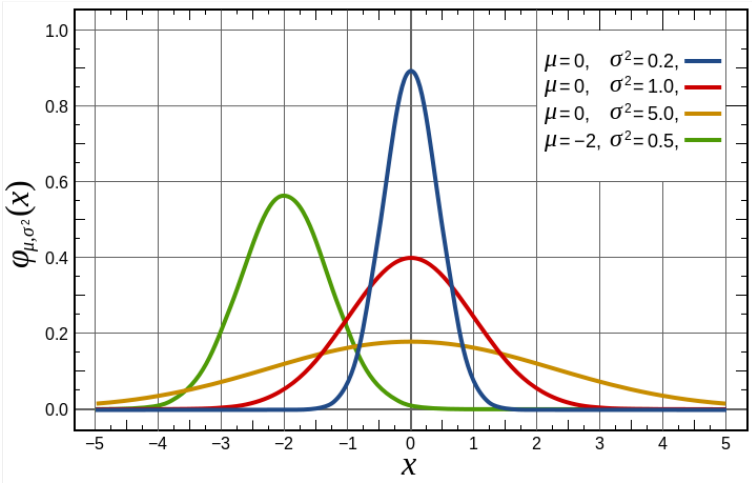


Figure A.1: The probability density function of normal distribution with various parameters. Source: Wikipedia.

Multivariate normal distribution

The generalization to n-dimensional space is denoted $\mathcal{N}(\mu, C)$, where

$$\mu = (\mu_{X_1}, \ldots, \mu_{X_n}) \quad \in \mathbb{R}^n$$

is the mean vector and C is the covariance matrix:

$$C = \begin{pmatrix} \sigma_{X_1X_1} & \cdots & \sigma_{X_1X_n} \\ \vdots & \ddots & \vdots \\ \sigma_{X_nX_1} & \cdots & \sigma_{X_nX_n} \end{pmatrix} \in \mathbb{R}^{n\times n},$$

where $\sigma_{X_iX_j}$ is the covariance of i-th and j-th random variables, $i, j = 1, \ldots, n$.

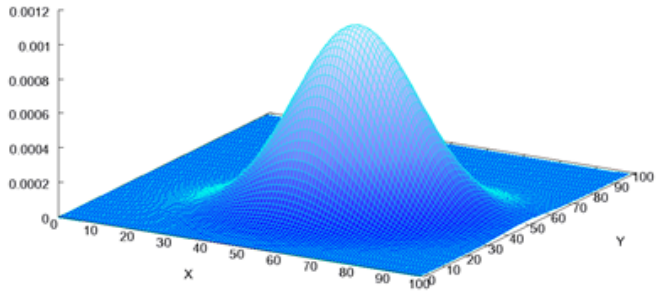


Figure A.2: The probability density function of multivariate (2D) normal distribution. Source: Wikipedia.

An important property of normal distribution that we shall exploit later is the following theorem [51].

Theorem. *Let X, Y be two random, normally distributed variables such that $X \sim \mathcal{N}(\mu_X, C_X)$ and $Y \sim \mathcal{N}(\mu_Y, C_Y)$. Then*

$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, C_X + C_Y).$$

This is also commonly denoted as

$$\mathcal{N}(\mu_X, C_X) + \mathcal{N}(\mu_Y, C_Y) \sim \mathcal{N}(\mu_X + \mu_Y, C_X + C_Y).$$

A.3 Link to linear algebra

The covariance matrix $C \in \mathbb{R}^{n \times n}$ is positive definite, i.e.

$$x^T C x > 0, \ \forall x \in \mathbb{R}^n \setminus \{0\}.$$

For the eigenvectors $v_i \in \mathbb{R}^n$ and positive eigenvalues $\theta_i \in \mathbb{R}^+, i = 1, \ldots, n$ of the matrix C it holds

$$C v_i = \theta_i v_i,$$

i.e. the eigenvectors are such vectors that are not rotated by the transformation represented by matrix C.

Lets make a simple, yet essential observation. There exists an orthogonal basis of $\mathbb{R}^n$ defined by eigenvectors of C. In other words, they define a new orthogonal coordinate system of the n-dimensional space. If we drew the probability density isolines of the 2-dimensional function plotted in figure A.2, we would get circles. In n dimensions, the isolines are n-dimensional balls: all n of its (coordinate) axes are mutually perpendicular and all have the same length. That means there is no mutual dependence of the individual variables, no covariance. The covariance matrix C is an identity matrix and all its eigenvalues are identical. When scaling its diagonal entries (while all others are still zeros), we scale the lengths of the axes, stretching the ball to become ellipsoid. By filling in the non-diagonal entries, the random variables become mutually dependent (correlated). The ellipsoid is rotated in space as its axes are aligned with the (mutually orthogonal) eigenvectors of the covariance matrix and the axes' lengths are determined by its eigenvalues.

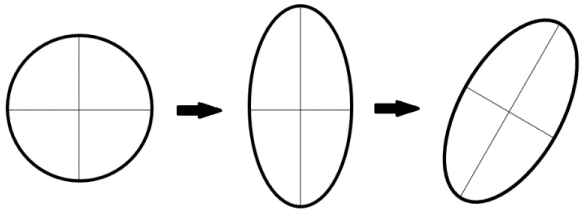


Figure A.3: Left: C is an identity matrix; middle: C is a diagonal matrix; right: C is a general positive definite matrix.

The normal distribution is by default isotropic (i.e. it is not biased towards any direction) and it is centered around the mean value. However, the probability density isolines are defined by the covariance matrix as described above – they form concentric ellipses. Using non-identity covariance matrices is equivalent to coordinate system transformations. Picture A.4 shows what this means in practice.

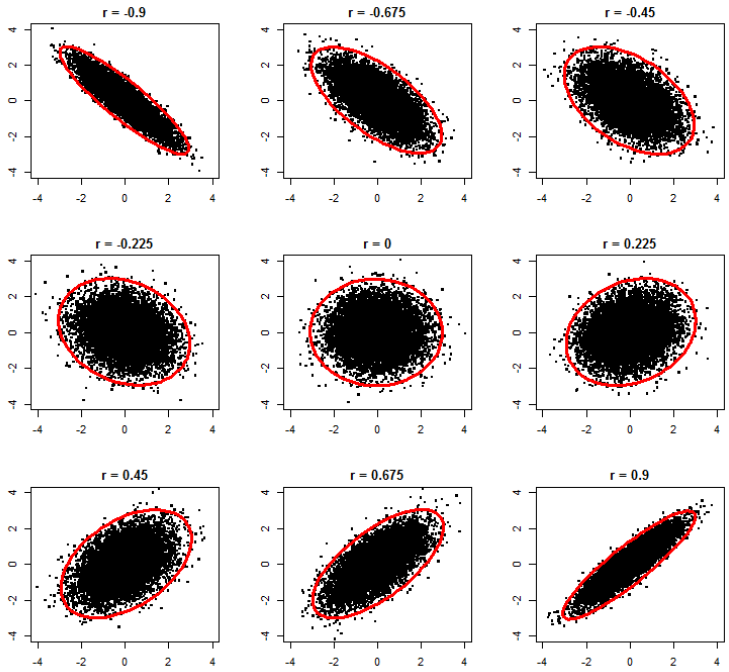


Figure A.4: Sampling a bi-variate normal distribution with various correlation coefficients r. Source: www.santiagobarreda.com/rstuff/rmvtnorm/rmvtnorm.html

Appendix B: Pieces of control theory

This section is to provide basics of control theory that are essential when trying to understand the problem at hand. It is included for the purpose of self-containedness, so that a reader unfamiliar with the control theory is conveniently delivered a brief introduction. It not intended to be neither rigorous nor exhaustive. Most information (and all definitions) come from [16], minor parts from [1]. The excellent textbook [16] is especially recommended for more detailed information.

B.1 The fundamental control problem

Definition. *The fundamental control problem*

The central problem in control is to find a technically feasible way to act on a given process so that the process behaves, as closely as possible, as desired. Furthermore, this approximate behavior should be achieved in the face of uncertainty of the process and in the presence of uncontrollable external disturbances acting on the process [16].

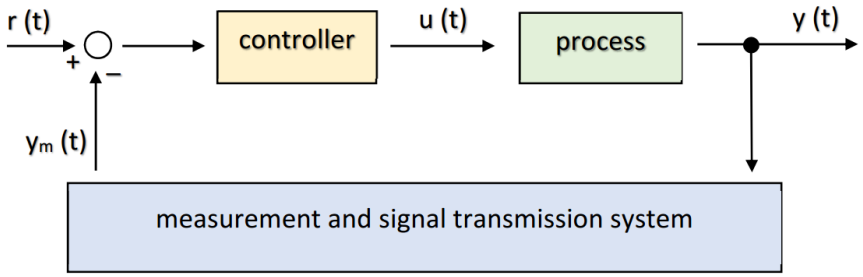


Figure B.1: Basic feedback control loop.

A simple example of a controlled process (system, plant) is keeping a steady temperature in a room by adjusting the heating.

In our case, we do not deal with a real process but with its model. We want to control "one-dimensional" finite-element models of engines. Our goal is to keep one or more quantities within the model at the desired levels. Those can be, for example, the temperature of exhaust gas, mean effective pressure (the average pressure exerted by the working fluid in an engine cylinder throughout the cycle) or volumetric efficiency (the volume of air -fuel mixture drawn into the cylinder at atmospheric pressure during the intake strokes compared to the volume of the cylinder).

B.2 Description of a system

When given a control problem, the first task is to create a model. That is, describe the process by a set of equations (usually ODEs or PDEs), define the input and output (in figure B.1, those are functions $u(t)$ and $y(t)$, respectively).

B.2.1 Transfer function

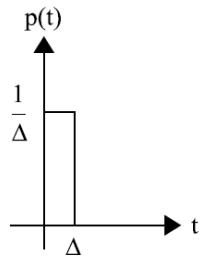
Essential information about a system is contained in the so-called transfer function. It characterizes the system by describing its input versus output properties.

Definition. *Transfer function*
The transfer function of a continuous time system is the Laplace transform of its response to an impulse (Dirac's delta) with zero initial condition [16].

However, as an impulse is implicitly an idealization (and thus unrealizable), it is more practical and common to study a system's dynamic behavior using the step response

$$Y(s) = \frac{1}{s}\, G(s),$$

where $G(s)$ is the transfer function.



An impulse: $\Delta \to 0$

B.2.2 Stability

Definition. *Stability of a system*
A system is stable if any bounded input produces a bounded output for all bounded initial conditions [16].

Without further details, lets state that the stability of the system is (obviously) closely related to the properties of the transfer function and thus to the corresponding stability theory for differential equations.

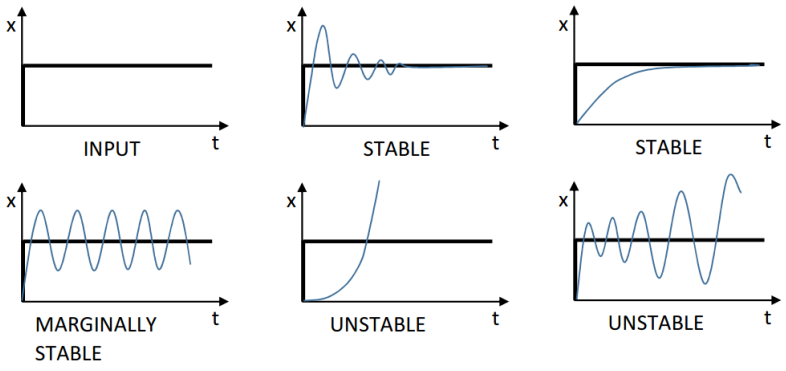


Figure B.3: Examples of stable and unstable step response.

Since we are taking the system as a black box, we do not need to concern ourselves with the theory describing stability. However, what is extremely important, is how a step response of a stable and unstable system looks like, see figure B.3.

B.3 Prototype solution to the control problem

Let's assume that:

- We know how an action at the input affects the system's output.
- We have the desired behavior for the system output.

Then if we inverted the relationship between input and output, we could determine what input action is necessary to get the desired output.

Let u be the system's input (the control signal) and y its output, d be the disturbance and r be the setpoint (the desired value of y). Let f be a function that (exactly) describes the system. Let both f and its inverse f^{-1} be well defined (i.e. they both describe a stable system). Then we have

$$y = f(u) + d$$

Let $y = r$. Then

$$u = f^{-1}(r - d).$$

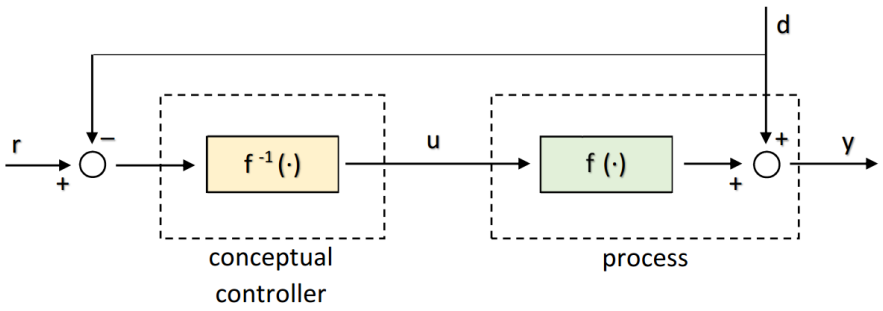


Figure B.4: Prototype solution to the control problem.

This is a conceptual solution to the control problem. The idea is simple and naïve and the requirements are extremely demanding (usually unsatisfiable). Yet it plays a fundamental role in practical applications. Most control strategies try to capture the intent of this inversion idea (approximate the inverse), while also considering real-world limitations that occur along the way.

Quoting [16]:

> "In principle, all controllers implicitly generate an inverse of the process, in so far that this is feasible. Controllers differ with respect to the mechanism used to generate the required approximate inverse."

B.4 PID controllers

A proportional-integral-derivative (PID) controller is a simple but powerful tool of control. It is robust: a small change of its parameters does not usually have a profound effect on the controlled system. Even very rough tuning usually produces an acceptable setting.

B.4.1 Definition

The input for the controller is the error function $e(t)$, i.e. difference between the desired value (the setpoint) and the actual value of a quantity. Its output is called the control signal, denoted $C(t)$.

There are several ways how to describe the inner workings of a PID controller. For the purposes of optimization, let's keep to the traditional, though perhaps old-fashioned, notation. The general equation can be written in the form

$$C(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \tag{C}$$

where K_p, K_i and K_d are the proportional, integral and derivative gains, respectively.

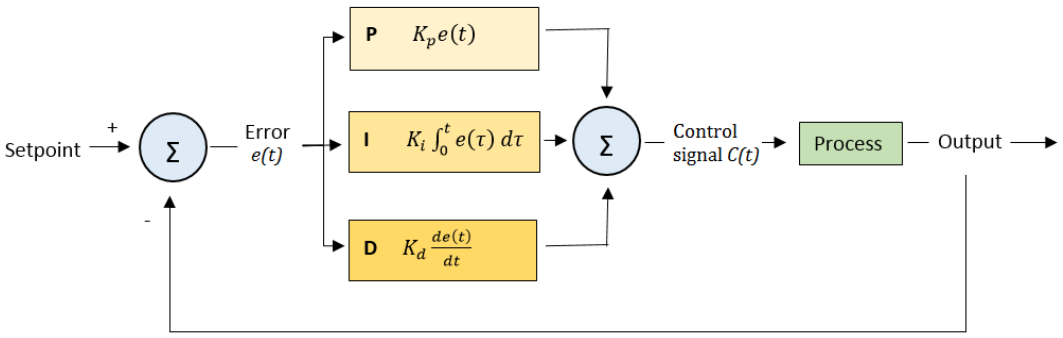


Figure B.5: PID controller scheme. The process (an engine in our application) generates output (e.g. exhaust gas temperature). Its value is measured by sensors and compared with the desired setpoint value (e.g. 760°K). Their difference, error function $e(t)$, is the input for a PID controller. There, the error function is processed as is given by equation (C), to obtain the control signal $C(t)$. The control signal then influences the process' behavior so its output values match the setpoint better. With this new output, the whole procedure is repeated.

However, there are other, equivalent ways how to describe a PID controller. In engineering, it is much more common to use the following notation:

$$C(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right).$$

Now $K = K_p$ is again the proportional gain, $T_i = \frac{K_p}{K_i}$ is called the integral time and $T_d = K_p K_d$ is the derivative time.

B.4.2 PID elements

The proportional element is the most important part of the controller. It provides output proportional to the instantaneous value of the error. It alone could provide basic control to a stable plant, but its performance is limited. In proportional control, there is always a steady state error. That is the offset from the setpoint when the response has converged. The error decreases with increasing gain but the oscillation increases.

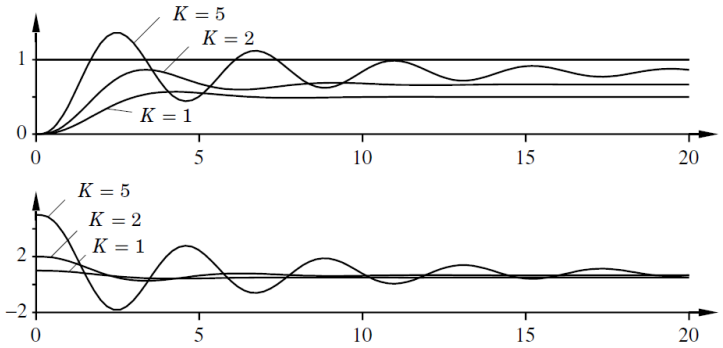


Figure B.6: Simulation of a closed-loop system with proportional control. The process transfer function is $G(s) = \frac{1}{(s+1)^3}$ and $K_p = K$. [1]

The integral action is proportional to the accumulated error, so it reduces the steady state errors. It is the slow control mode. Its main setback is the effect of wind-up. With decreasing integral time T_i (that is increasing K_i), the oscillations increase.

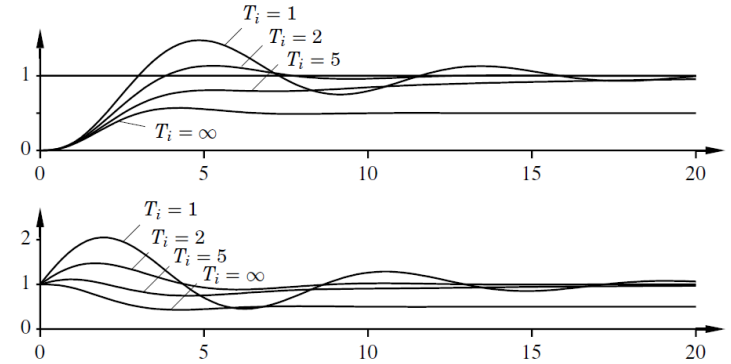


Figure B.7: Simulation of a closed-loop system with proportional and integral control. The process transfer function is $G(s) = \frac{1}{(s+1)^3}$ and $K_i = \frac{K_p}{T_i}$. The proportional gain is set to $K = K_p = 1$. [1]

The derivative element represents the rate of change of the control error. It is thus the fast mode of control. As it follows the error trend and improves the

transient response through high frequency compensation, it can be interpreted as providing prediction by linear extrapolation. Therefore, it is referred to as the predictive mode. However, it is sensitive to noise, as a high-frequency control errors may result in large derivative control signal response. For that reason, the derivative element is often omitted and only PI control is used. Damping increases with increasing derivative time, but decreases again when the derivative time is too large.

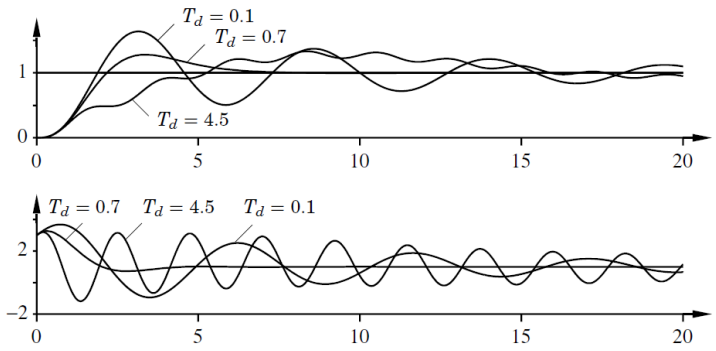


Figure B.8: Simulation of a closed-loop system with proportional, integral and derivative control. The process transfer function is $G(s) = \frac{1}{(s+1}{}^3$ and $K_d = KT_d$. The proportional gain is $K = K_p = 3$ and the integral gain $K_i = \frac{3}{2}$. That corresponds to setting the integral time $T_i = 2$. [1]

B.4.3 Coupled vs. decoupled PID controllers

Let's have n PID controllers connected to the same system. When setting of one influences another one, the controllers are coupled. When they do not interact with each other, i.e. they control independent parts of the plant, they are decoupled.

In case of decoupled controllers, each controlled variable can be linked to a single controller. That means that the controllers' matrix is diagonal:

$$C = \begin{pmatrix} C_{11} & & 0 \\ & \ddots & \\ 0 & & C_{nn} \end{pmatrix},$$

where $C_{ii}(t), i = 1, \ldots, n$, is the control signal function of the i-th controller as described above. When coupling plays a role, off-diagonal elements appear.

Scholarly articles regarding tuning of PID controllers usually consider them to be decoupled. Moreover, they usually assume that we know the system's transfer function [8]. Both these assumptions make the problem much easier.

Appendix C: List of tested algorithms in BBOB 2009

This part is copied directly from [21], where also further references for the individual algorithms may be found.

Used acronyms:

GA/EA: Genetic Algorithm / Evolutionary Algorithm
EDA: Estimation of Distribution Algorithm
CMA: Covariance Matrix Adaptation
ES: Evolution Strategy
PSO: Particle Swarm Optimization

Tested algorithms:

1. ALPS-GA: Age-Layered Population Structure running a standard GA in 12 layers
2. AMaLGaM IDEA: Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Algorithm with no-improvement stretch, anticipated mean shift and interlaced restarts with one large or several small populations
3. iAMaLGaM IDEA: with incremental model building
4. BayEDAcG: An EDA using Bayesian inference to learn the parameters of the continuous Gaussian distribution
5. BFGS: Quasi-Newton method with MATLAB's, more to be found in *fminunc* [50]
6. Cauchy EDA: EDA with isotropic Cauchy sampling distribution
7. BIPOP-CMA-ES: CMA-ES restarted with budgets for small and large population size, more to be found in [26]
8. $(1 + 1)$-CMA-ES
9. DASA: Differential Ant-Stigmergy Algorithm using pheromones on the differential graph that represents parameter difference
10. DEPSO: PSO with Differential Evolution variations
11. DIRECT: Axis-parallel search space partitioning procedure
12. EDA-PSO: hybrid of EDA and PSO with adapted probability of applying one or the other
13. G3-PCX: Generalized Generation Gap with Parent Centric Crossover (local search intensive variant)

14. simple GA: binary coded GA
15. GLOBAL: Sampling, clustering and local search using BFGS or Nelder-Mead
16. LSfminbnd: Axis-parallel line search with MATLAB's fminbnd univariate search
17. LSstep: Axis-parallel line search with the univariate STEP Select The Easiest Point, based on interval division
18. MA-LS-Chain: Memetic Algorithm with Local Search Chaining using a steady-state GA and CMA-ES for local search with a fixed local/global search ratio
19. MCS: Multilevel Coordinate Search like DIRECT with additional local searches by triple search
20. NELDER (Han): Nelder-Mead downhill simplex restarted using coordinate-wise projections
21. NELDER (Doe): Nelder-Mead downhill simplex with restarted half-runs
22. NEWUOA: NEW Unconstraint Optimization Algorithm builds a second order model using $2n + 1$ points and with minimal Frobenius norm
23. full NEWUOA: using $(n^2 + 3n + 2)/2$ points
24. $(1 + 1)$-ES: with 1/5th success rule for step-size adaptation
25. POEMS: Prototype Optimization with Evolved Improvement Steps using hypermutations and stochastic local search
26. PSO: standard PSO with swarm size 40 and no restarts
27. PSO Bounds: as PSO and diving the search domain based on a concept from PBIL
28. Monte Carlo: uniform random sampling
29. Rosenbrock: Local search algorithm maintaining an orthogonal basis for the adaptation of search directions
30. IPOP-SEP-CMA-ES: CMA-ES restarted with Increasing POPulation size, first run with SEParable CMA
31. VNS: Variable Neighbourhood Search combining CMA-ES, PBX-α-EA and μCHC

Bibliography

[1] K. J. Åström. Control system design lecture notes for ME 155A. *Department of Mechanical and Environmental Engineering University of California Santa Barbara*, 2002.

[2] A. Auger, S. Finck, N. Hansen, and R. Ros. BBOB 2009: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-0383, INRIA, April 2010.

[3] A. Auger, S. Finck, N. Hansen, and R. Ros. BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-388, INRIA, September 2010.

[4] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In B. McKay et al., editors, *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, volume 2, pages 1769–1776, 2005.

[5] A. Auger and N. Hansen. Theory of Evolution Strategies: a New Perspective. In A. Auger and B. Doerr, editors, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, chapter 10, pages 289–325. World Scientific Publishing, 2011.

[6] Beyer, H.-G. and Schwefel, H.-P. Evolution Strategies – A Comprehensive Introduction. *Natural Computing*, 1(1):3–52, May 2002.

[7] J. Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu.com, 1st edition, 2011.

[8] R. Dittmar, S. Gill, H. Singh, and M. Darby. Robust optimization-based multi-loop {PID} controller tuning: A new tool and its industrial application. *Control Engineering Practice*, 20(4):355 – 370, 2012. Special Section: {IFAC} Symposium on Advanced Control of Chemical Processes - {ADCHEM} 2009.

[9] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[10] J. Dréo, A. Petrowski, P. Siarry, and E. Taillard. *Metaheuristics for hard optimization: methods and case studies*. Springer Science & Business Media, 2006.

[11] Engelbrecht, A. P. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006.

[12] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[13] F. W. Glover and G. A. Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[14] Goldberg, David E. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[15] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley Publishing Company, 1989.

[16] G. C. Goodwin, S. F. Graebe, and M. E. Salgado. *Control System Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[17] D. Graham and R. C. Lathrop. The Synthesis of Optimum Transient Response: Criteria and Standard Forms. *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, 72(5):273–288, Nov 1953.

[18] N. Hansen. cma 1.1.06: Python Package. `https://pypi.python.org/pypi/cma`.

[19] N. Hansen. The CMA Evolution Strategy: A Tutorial. `https://www.lri.fr/~hansen/cmatutorial.pdf`, 2015.

[20] N. Hansen, D.V. Arnold, and A. Auger. Evolution Strategies. In J. Kacprzyk and W. Pedrycz, editors, *Springer Handbook of Computational Intelligence*, chapter 44, pages 871–898. Springer Berlin Heidelberg, 2015.

[21] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posik. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. *Workshop Proceedings of the GECCO Genetic and Evolutionary Computation Conference 2010*, pages 1689–1696, 2010.

[22] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[23] N. Hansen and R. Ros. Benchmarking a Weighted Negative Covariance Matrix Update on the BBOB-2010 Noiseless Testbed. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, pages 1673–1680, 2010.

[24] Nikolaus Hansen. CMA-ES with Two-Point Step-Size Adaptation. Research Report RR-6527, INRIA, 2008.

[25] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

[26] 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*, GECCO '09, pages 2389–2396, 2009.

[27] Hansen, N. and Finck, S. and Ros, R. COCO - COmparing Continuous Optimizers : The Documentation. Research Report RT-0409, INRIA, May 2011.

[28] Hansen, N. and Finck, S. and Ros, R. COmparing Continuous Optimisers: COCO. `http://coco.gforge.inria.fr/`, 2015.

[29] Holland, J. H. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* University of Michigan Press, 1975.

[30] G. A. Jastrebski and D. V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *IEEE Congress on Evolutionary Computation – CEC 2006*, pages 2814–2821, 2006.

[31] Jaynes, E.T. and Bretthorst, G.L. *Probability Theory: The Logic of Science.* Cambridge University Press, 2003.

[32] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`.

[33] Jones, D. R. and Schonlau, M. and Welch, W. J. Efficient Global Optimization of Expensive Black-Box Functions. *J. of Global Optimization*, 13(4):455–492, December 1998.

[34] Kennedy, J. and Eberhart, R. C. and Shi, Y. *Swarm Intelligence.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[35] I. Loshchilov, M. Schoenauer, and M. Sebag. Alternative Restart Strategies for CMA-ES. *CoRR*, abs/1207.0206, 2012.

[36] Loshchilov, I. and Schoenauer, M. and Sebag, M. Comparison-based Optimizers Need Comparison-based Surrogates. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, pages 364–373, Berlin, Heidelberg, 2010. Springer-Verlag.

[37] Loshchilov, I. and Schoenauer, M. and Sebag, M. Black-box optimization benchmarking of IPOP-saACM-ES and BIPOP-saACM-ES on the BBOB-2012 noiseless testbed. In *Workshop Proceedings of the (GECCO) Genetic and Evolutionary Computation Conference*, Philadelphia, Pennsylvania, United States, July 2012.

[38] Loshchilov, I. and Schoenauer, M. and Sebag, M. Self-adaptive Surrogate-assisted Covariance Matrix Adaptation Evolution Strategy. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 321–328, 2012.

[39] Loshchilov, I. and Schoenauer, M. and 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*, GECCO '13, pages 439–446, 2013.

[40] Mohammadi, H. and Le Riche, R. and Touboul, E. *Learning and Intelligent Optimization: 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, chapter Making EGO and CMA-ES Complementary for Global Optimization, pages 287–292. Springer International Publishing, Cham, 2015.

[41] Passino, K. M. *Biomimicry for optimization, control, and automation.* Springer Science & Business Media, 2005.

[42] M. Pelikan. *Hierarchical Bayesian optimization algorithm.* Springer, 2005.

[43] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

[44] M. Pelikan, K. Sastry, and E. Cantú-Paz. *Scalable optimization via probabilistic modeling: From algorithms to applications*, volume 33. Springer, 2007.

[45] Pham, D.T. and Ghanbarzadeh, A. and Koc, E. and Otri, S. and Rahim, S. and Zaidi, M. The Bees Algorithm – A Novel Tool for Complex Optimisation. In *Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006*, page 454. Elsevier, 2011.

[46] Poli, R. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008:3, 2008.

[47] Powell, M. J. D. *Large-Scale Nonlinear Optimization*, chapter The NEWUOA software for unconstrained optimization without derivatives, pages 255–297. Springer US, Boston, MA, 2006.

[48] Price, K. and Storn, R. M. and Lampinen, J. A. *Differential Evolution: A Practical Approach to Global Optimization.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[49] Ricardo Software. WAVE Manual, 2016.

[50] Ros, R. Benchmarking the BFGS Algorithm on the BBOB-2009 Function Testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2409–2414, 2009.

[51] S. M. Ross. *A first course in probability.* Pearson Education, 2010.

[52] R. Storn. On the usage of differential evolution for function optimization. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 519–523, Jun 1996.

[53] R. Storn and K. Price. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. international computer science institute, berkeley. Technical report, CA, 1995, Tech. Rep. TR-95–012, 1995.

[54] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.