DIPLOMARBEIT

Efficient Search for Robust Solutions by Means of Evolutionary Algorithms and Fitness Approximation

von

Ingo Pänke

Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe

Referent: Prof. Dr. Hartmut Schmeck
Betreuer: Dr. Jürgen Branke, Dr. Yaochu Jin (HRI)

Karlsruhe, 31.05.2004

Contents

Acknowledgements **xv**

1 Introduction **1**

2 Related work **5**

3 Foundations **11**

3.1 Evolutionary optimization 11

3.2 Robustness 15

3.2.1 Definitions and Examples 15

3.2.2 Robustness Measures 16

3.3 Multi objective optimization 20

3.3.1 Difficulties 20

3.3.2 Decision makers preferences 21

3.3.3 Pareto dominance 22

3.3.4 NSGA2: An *a posteriori* multi-objective EA 23

3.3.5 Performance metrics 25

3.4 Response surface methodology 27

3.4.1 Definitions 27

3.4.2 Approximate model 27

3.4.3 Design of experiments (DoE) 28
3.4.4 Extensions 28
3.4.5 Illustration 29

4 Evolutionary robustness optimization 31

4.1 Introduction 31
4.2 Collecting response data 32
4.3 An EA for SO robustness optimization 33
4.4 An EA for MO robustness optimization 35
4.5 Constraint handling 36

5 Fitness approximation (RSM) 39

5.1 Assumptions, Definitions 39
5.2 Approximation models 40
5.3 Choice of response data (DoE) 41
5.4 Numerical Difficulties in EA's 42
5.5 Interpolation 43
5.5.1 Standard method 43
5.5.2 Managing singularities 44
5.6 Local regression 47
5.6.1 Background 47
5.6.2 Normal equations method 48
5.6.3 Residual norm method 49
5.6.4 Normal equations vs. Residual norm method 50
5.6.5 Managing singularities 52
5.6.6 Design parameters 53
5.7 Computational complexity 54

5.8 Illustration 56

6 Robustness estimation **59**

6.1 Integral approximation (sampling techniques) 59

6.2 Distribution of approximation models 63

6.2.1 Individual based model distribution 63

6.2.2 Population based model distribution 65

6.2.3 Overview 67

6.3 Extreme outliers 67

6.3.1 Estimation bias 67

6.3.2 Boxplot 68

6.3.3 Outliers detection routine 69

6.3.4 Replacing outliers 70

6.3.5 Additional remarks 71

7 Preliminary 1-d simulation studies **73**

7.1 Introduction and preceding remarks 73

7.2 Natural neighbor interpolation 74

7.3 Test functions 76

7.4 Experiment 1 77

7.4.1 Experimental setup 77

7.4.2 Brief remarks on estimator properties 78

7.4.3 Results 78

7.5 Experiment 2 81

7.5.1 Experimental Setup 81

7.5.2 Results 81

7.6 Experiment 3 82

7.6.1 Experimental setup 83
7.6.2 Results 83
7.7 Discussion 84

8 SO Simulation studies **87**
8.1 Theoretical analysis of estimator properties 87
8.1.1 Estimation error properties in an EA 87
8.1.2 Determinants of the estimation error's standard deviation 89
8.2 Experimental setup 91
8.2.1 Requirements 91
8.2.2 Test problem 92
8.2.3 Analyzed methods 94
8.2.4 Performance criteria 94
8.2.5 Benchmark methods 95
8.2.6 Parameter setting 97
8.3 Results 99
8.3.1 Overview 99
8.3.2 Hypotheses 101
8.3.3 Approximation models 102
8.3.4 Sampling and model distribution 105
8.3.5 Selected regression parameters 112
8.3.6 General observations 116
8.3.7 Comparison with other methods 118

9 MO simulation studies **121**
9.1 Experimental setup 121
9.1.1 Requirements 121

9.1.2 Test problems 122
9.1.3 Analyzed methods 123
9.1.4 Performance criteria 124
9.1.5 Benchmark methods 125
9.1.6 Parameter setting 125
9.2 Results 126
9.2.1 Analysis of Neighbors evaluation method 126
9.2.2 Performance of the approximation methods 128
9.2.3 Selecting the final set of solutions 131

10 Summary and Outlook **135**
10.1 Summary 135
10.2 General remarks 138
10.3 Outlook 139

A Statistical significance tests **141**

B Implementation **145**

List of Figures

3.1 Trade-off f_{raw} - f_{exp} 17
3.2 Expected fitness optima for different noise distributions 17
3.3 Trade-off between f_{exp} and f_{var} 19
3.4 Pareto-dominance 23
3.5 Comparison of different solution sets 26
3.6 Illustration: Response data and approximated response surface 29

5.1 Computational cost: Residual norm vs. Normal equations method 51
5.2 Tricube function 55
5.3 Computational cost: interpolation vs. regression 56
5.4 Examples of fitness approximations 57

6.1 Stratified and Latin hypercube sampling 60
6.2 Single model and multiple models distribution 64
6.3 Nearest model method 66
6.4 Overview of model distribution methods 67
6.5 Example of an outlier 68
6.6 Boxplotting 69

7.1 Natural neighbor interpolation 75
7.2 Test problems for the 1-dim simulation studies 76

7.3 Performance of different estimation methods in experiment 1 80
7.4 Test function f_{2dim1} for different δ 81
7.5 Performance of different estimation methods in experiment 2 82
7.6 Performance of different estimation methods in experiment 3 84

8.1 Test function f_1 (1-dim) 92
8.2 Test function f_1 (2-dim) 93
8.3 Overview of the results in SO (LHC) 100
8.4 Overview of the results in SO (Stratified) 101
8.5 Varying parameters for outliers detection 104
8.6 Model location: Individual location vs. Center of mass 107
8.7 Estimation error standard deviation for different ensemble sizes 110
8.8 PMD-ENS-k performance for different ensemble sizes 111
8.9 Regression performance for different numbers of input data 113
8.10 Regression performance for different bandwidth 114
8.11 Regression performance when assigning equal weights 116
8.12 Performance loss in higher dimensions 117
8.13 Performance of Neighbors evaluation method 118
8.14 Tsutsui method compared to PMD-ENS-k and IMD-MM 120

9.1 Test problem for the MO simulation studies 123
9.2 f_{exp} and f_{var} of 2-dimensional test function f_2 124
9.3 Test problem 1: Neighbors evaluation method for different *numpoints_min* 127
9.4 Test problem 1: Performance comparison in dimension 1 128
9.5 Test problem 1: Performance comparison in dimension 5 129
9.6 Test problem 1: Performance comparison in dimension 2 130
9.7 Test problem 2: Performance comparison in dimension 1 131

9.8 Final set of solutions: Final generation vs. all generations 132

List of Tables

7.1 Tested methods in 1-dim simulation studies 74

8.1 Definition of terms 89
8.2 Tested methods in SO simulation studies 94
8.3 Standard parameter setting for the SO simulation studies 98
8.4 Enumeration of the estimation methods 99
8.5 Standard deviation of the estimation error (PMD-MM) 103
8.6 Standard deviation of the estimation error (IMD-SM) 105
8.7 Standard deviation of the estimation error: PMD-ENS-10 vs. IMD-MM . . 108
8.8 Redefinition of terms 108
8.9 Standard deviation of the estimation error (Interpolation in PMD-ENS-10) 112

9.1 Parameter setting for MO simulation studies 126

A.1 Quantiles of the t-distribution 141
A.2 Student-T significance tests dimension 2 142
A.3 Student-T significance tests dimension 5 142
A.4 Student-T significance tests dimension 10 143
A.5 Quantiles of the f-distribution 143
A.6 Fisher significance tests 144

Acknowledgements

This thesis is a cooperation project between the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe and the Honda Research Institute Europe (HRI-EU) in Offenbach a.M., Germany.

I would like to thank my supervisors Dr. Juergen Branke at the AIFB and Dr. Yaochu Jin at HRI-EU for their continuous support. Thanks goes to the ELTEC research group of HRI-EU which supported this thesis with both technical infrastructure and their scientific experience. Special thanks goes to Tatsuya Okabe from HRI-EU who helped me with all question regarding MOOEALib, a C++ library for multi-objective evolutionary optimization and Christian Schmidt from the AIFB with whom I had fruitful discussions on statistics issues.

Chapter 1

Introduction

Many real-world optimization problems are subject to a wide range of uncertainties. In scheduling a new job arrives unexpectedly, traveling time cannot be reliably forecasted by a route planner, or manufacturing tolerances avoid that a solution can be implemented according to design specifications. Formally, a deterministic optimization problem

$$minf(x, e) \ ,$$

where x represents decision variables and e environmental parameters, is transformed to stochastic optimization problem with $x \mapsto x + \delta$ and/or $e \mapsto e + \delta$, where δ is a stochastic disturbance. In this case, the goal of the optimization is no longer to find a solution with high performance, instead one is interested in solutions that perform well over a range of disturbances. Such solutions are called *robust*. *Disturbance* is also often termed *noise*.

Different measures of robustness have been introduced in the last years. The most common approach is to make (at least implicitly) assumptions on the probability distribution of the expected environments noise, measure the *expected fitness* of an individual, and use this as single optimization criterion. Therefore, most of the evolutionary robustness optimization methods fall into the category of *single objective optimization* algorithms. We refer to this approach as *SO robustness optimization*. However, from decision theory

we know that simply using the *expected* "outcome of a decision" (performance of a solution) as optimization criterion, implicitly assumes *neutral risk preferences* of the decision maker. Risk preferences can be taken into account by including the fitness variance as second objective to the optimization problem. Thus, *multi objective optimization* provides a means for robustness optimization. We refer to this approach as *MO robustness optimization*. In order to distinguish the possible different objectives we define

- f (often denoted f_{raw}) - deterministic fitness function or *raw fitness*
- f_{exp} - *expected fitness* with respect to a probability distribution of the noise
- f_{var} - *fitness variance* with respect to a probability distribution of the noise .

If the fitness function is available in closed form and if the product of fitness function and density functions of the probability distribution is integrable, f_{exp} and f_{var} can be calculated analytically and directly used as optimization criterion. Unfortunately, this scenario is not realistic for most optimization problems. The next idea is to estimate f_{exp} and f_{var} empirically. This can for example be done by drawing sample points in the neighborhood of an estimation point and evaluating these samples with the fitness function. If the locations of the sample points are chosen with respect to the probability distribution of the noise, the expected noise is simulated. However, for most real-world problems, calls to the fitness function incur high computational cost and one should carefully decide which possible solutions are to be evaluated with the fitness function. Therefore, one assumption of our approach to robustness optimization is that fitness function evaluations are very expensive.

The goal of this work is, thus, to develop new methods for *efficient* search for robust solutions, i.e. methods that require only a small number of fitness function evaluations. In particular, we investigate, how information about the fitness surface which is collected throughout the run of an evolutionary algorithm can be exploited. This work is inspired

by ideas from *Response surface methodology* because this framework provides effective statistical methods for fitness surface approximation. Based on fitness function approximations, our new methods estimate f_{exp} and f_{var}, and use these estimations for guiding the evolutionary search for robust solutions. In particular, this work analyzes *interpolation* and *local regression* as means for fitness function approximation in combination with different robustness estimation techniques.

This work is organized as follows: Chapter 2 provides an overview of related work and existing approaches to robustness optimization. In Chapter 3 we provide the foundations of this work. Here we also introduce terminology that is used throughout this work. Although most of Chapter 3 might not provide new insights to the experienced reader, we recommend Section 3.2 where we discuss *robustness*. Chapter 4 presents the evolutionary algorithms which we developed for robustness optimization. In Chapter 5, we describe the fitness approximation methods and their specific features which are necessary in the framework of an evolutionary algorithm. Based on fitness approximations, the estimation techniques which are presented in Chapter 6 estimate the robustness indicators f_{exp} and f_{var}. Chapter 7 reports about preliminary simulation studies which we carried out for the 1-dimensional case. The motivation for these experiments was to gather a deeper understanding for the problem at hand without spending too much time on implementation. Furthermore, the 1-dimensional case allowed us to analyze a wider range of approximation methods. Single objective optimization with f_{exp} as optimization criterion is still a common approach in robustness optimization. We therefore carried out extensive simulation studies on *SO robustness optimization* which are presented in Chapter 8. The simulation studies for *MO robustness optimization* are outlined in Chapter 9. We summarize this work and give an outlook on future research in Chapter 10.

Chapter 2

Related work

As described in the introduction, there exist two measures of robustness, the expected fitness f_{exp} and the variance of the fitness f_{var}. Only the combination of these objectives, makes it possible to fully account for the risk preferences of the decision maker, e.g. the engineer. However, in most of the literature, robustness optimization is treated as single objective optimization problem, with f_{exp} as objective. We refer to the single objective optimization case as SO, and to the multi objective optimization case as MO.

Tsutsui and Gosh present in [1] the idea, to disturb the phenotypic features while evaluating the functional value of individuals in the framework of a genetic algorithm. They refer to this as GAs/RS^3 (Genetic Algorithms with a robust solution searching scheme). Note that the disturbance is only added for the evaluation process and does not represent a mutation. They showed that this method works on simple test functions: The genetic algorithm managed to converge to the global f_{exp} optimum. In [1] Tsutsui et al. also proof, that for infinitly large population size an evolutionary algorithm with single disturbed evaluations (like in GAs/RS^3) performs as if working directly on f_{exp}. In [1] and [2] GAs/RS^3 is also applied to a two-dimensional test problem. However, there are no simulation studies for higher dimensional problems available. The great advantage

of GAs/RS^3 is that it can be added to a standard genetic algorithm with almost no additional cost.

In [3], a robust solution[1] is found by searching for multiple local optima with a standard evolutionary algorithm and evaluating these high performance regions more closely in terms of robustness *a posteriori*. The advantage of this method is the comparably low additional cost as only a small set of solutions is evaluated in terms of robustness. The major drawback of this method is that we cannot assume that an optimal solution in terms of robustness lies in a high performance region. To overcome this drawback, an evolutionary algorithm must be *guided* in terms of robustness, thus individuals must be evaluated according to f_{exp} throughout the run. Several approaches have been made to guide the search in this sense.

The most common approach is to estimate f_{exp} for each individual by repeated sampling with additive noise in the neighborhood. This method was used in several applications. In [4] it is succesfully applied to job shop scheduling to find robust schedules. The authors point out the difficulty of defining a neighborhood in the discrete search space. In [5], a genetic algorithm with a similar robustness searching scheme is used for wing-box optimization and compared to a number of different optimization techniques, e.g. simulated annealing, local search, alopex method. The authors find the genetic algorithm to perform relatively well. Thomson [6] succesfully adds a similar robustness feature to a genetic algorithm, which evolves a control system of an autonomous robot.

Several modifications of this method have been analyzed in order to improve the search for a given constant number of fitness evaluations. Branke [7] shows that the performance is increased by first evaluating all individuals only once, and then reevaluate the best individuals. The performance can further be improved by using more sophisticated sampling techniques instead of simply drawing disturbances randomly. In [8] several sampling meth-

[1] robustness in terms of the single objective f_{exp}

ods are compared and it is empirically shown that Latin hypercube sampling improves the solution quality significantly. A detailed description of Latin hypercube sampling can be found in [9].

In [7], Branke presents a f_{exp} estimation method which requires no additional fitness evaluations, but exploits historical information, collected throughout the run of the EA. Here f_{exp} is estimated by building the weighted average of fitness values of individuals, which are located within a certain neighborhood. Weights are assigned with respect to the distance between the individuals. Ideally, the weight function $w(y)$ reflects the probability that x is turned into y by a disturbance. On one test function the performance was significantly improved by storing a large number of individuals with corresponding fitness values, instead of e.g. just use the data available from the current generation. In this paper, it is also shown that this method performs better than a GAs/RS^3-like approach[2].

The field of fitness estimation is related to this work in two aspects:
First, the RSM approach of this work uses *fitness function approximations* at a large number of sample points in order to empirically estimate an individual's expected fitness and fitness variance. A survey of fitness approximation methods can be found in [10]. In [11], one method, namely neural network approximation, is employed by Jin, Sendhoff and Olhofer in the framework of an evolution strategy. In [12], neural network and kriging approximation are compared in the framework of evolutionary optimization. The authors find that neither the kriging model nor the neural network improve the opimization compared to an EA without approximation models.

Secondly, when response surface methodology is used, the computation of f_{exp} and f_{var} is itself an estimation. Therefore, papers which deal with the behavior of evolutionary algorithms, when using estimations instead of real fitness evaluations are related to this work. The idea of a "controlled evolution" as analyzed in [11] might be transferable to

[2]it needs to be mentioned that in [7] the GAs/RS^3-like approach in Branke's work was run with different EA settings than in Tsutsui's work

robustness optimization, where f_{exp} is the function to be estimated. Two methods of controlling evolution are proposed: In the "controlled individuals" approach, part of the individuals in each population are evaluated with the real fitness, the remaining part is evaluated with estimations. In "controlled generations", generations are either entirely evaluated with the real fitness or the fitness of all individuals of a generation is estimated. The "controlled evolution" approach turned out to be very effective and was applied to evolutionary optimization of turbine blades [13].

One of the only few approaches which deal with robustness optimization as multi objective optimization problem is presented in [14]. Here, the trade-off between "optimality" (f_{raw}) and "robustness" (f_{var}) is evident. The estimation of f_{exp} is done similarly to Branke's weighted average method [7], but Jin and Sendhoff use equal weights for all neighboring individuals. The variance is estimated by computing the empirical fitness variance of all neighboring individuals, again with equal weights. Furthermore, the sample set is restricted to individuals of the current population. Note that for SO robustness optimization, Branke has shown for one test function, that extending the history data base, improves the performance significantly [7]. An earlier work by Chen at al. [15] considers both optimizing mean performance and minimizing performance variations as goals. Similar to our work, they use response surface models. However, Chen et al. do not use evolutionary optimization, but employ a method from decision theory instead. In MO evolutionary optimization there exists an enormous amount of literature which is related to this work. For our purposes, the most relevant one, is the work by Deb et al. [16] which introduces NSGA2, as this is used in the MO part of our work.

Similarly, in the field of spatial statistics, geostatistics and approximation theory, there exists a large amount of literature which is related to our work. One example is the work of Giunta and Watson [17] which compares quadratic regression with kriging interpolation in terms of model accuracy. They found that the quadratic regression models

are more accurate on their test instances. Somewhat surprisingly this was true even for test problems which are highly non-quadratic. A similar comparison was made for a real world problem, aerospike nozzle optimization, by Simpson et al. [18]. They found, kriging models to perform as well as quadratic regression in this application. Unfortunately, most of the work in the field of spatial statistics and geostatistics is restricted to 2-d or 3-d problems. Worth mentioning is the work by Boissonnat and Cazals [19], which use natural neighbor interpolation for smooth surface reconstruction. The proposed method is generally applicable in any dimension. Unfortunately, the authors only present an analysis of the computational overhead for the 3-d case.

Chapter 3

Foundations

This chapter provides a collection of brief introductions to concepts which are the foundations of this work. First, in Section 3.1, we introduce the main ideas of *evolutionary optimization* and define basic terminology. Section 3.2 proceeds with a definition and illustration of *robustness*. An understanding of the *robustness* concept as employed in this work, prerequisites a basic understanding of *multi-objective optimization* (MOO). Therefore a brief introduction to MOO is given in Section 3.3. In Section 3.4 we briefly introduce the basic ideas *response surface methodology*.

3.1 Evolutionary optimization

Evolutionary algorithms (EA's) are inspired by principles of biological evolution. EA's imitate the interaction between *selection* ("survival of the fittest"), *reproduction* and *mutation*. Already in 1795, Charles Darwin was the first who built up an evolution theory in this sense [20]. According to Darwin, living organisms fight a struggle for existence. In this context organisms are termed *individuals*. Weak individuals die at an early age. Thus, weak individuals do not produce *offsprings*. In contrast, strong individuals reach an age at which they are able to reproduce. According to Darwin this strength is mainly

attributed to the genetic material (*genotype*). Strong individuals recombine their genetic material and the resulting offsprings are likely to be even stronger. Darwin also concluded that the composition of the genetic code does not solely depend on the *parents* genetic code, but that some alterations of the genetic code appear randomly. This aspect is known as *mutation*. The process of improving strength from generation to generation is known as *evolution*.

An evolutionary algorithm simulates this evolution process which aims at producing individuals with good *"fitness"*. *"Individuals"* in EA's represent *solutions* to a specific optimization problem. An individual with good fitness is interpreted as the *representation* of a solution with high performance with respect to the given problem. *Solutions* are therefore "real world solutions", whereas the term *individuals* is used in the domain of the artificial simulation - the evolutionary algorithm. Solutions and individuals are related by an *encoding* or, respectively, *decoding* function: An individual is translated to a "real world solution" by applying the decoding function. Thus, each individual is a *representation* of a real world solution. In analogy to biology, a representation of a solution is termed *genotype*.

Algorithm 1 contains the basic operations of an EA:

- *initialization*: A pool of individuals (population) is created. This can be done randomly or heuristically by using problem specific knowledge.
- *evaluation*: Each individual is assigned a fitness value, which represents a measure for the performance with respect to the given optimization problem.
- *selection*: At this step it is decided, which individuals of the current population $P(t)$ produce *offsprings*. This can for example be done randomly or by taking into account the fitness values.
- *recombination*: The actual "reproduction activity". The genetic information of M

BEGIN BASIC_EA

$t \leftarrow 0$

initialization: initialize population $P(0)$

evaluation: evaluate $P(0)$

REPEAT

selection: mating pool $M(t) \leftarrow select(P(t))$

recombination: $M'(t) \leftarrow recombine(M(t))$

mutation: $M''(t) \leftarrow mutate(M'(t))$

evaluation: evaluate $M''(t)$

update-population: $P(t+1) \leftarrow u(P(t) \cup M''(t))$

$t \leftarrow t+1$

UNTIL(*termination condition*)

END BASIC_EA

Algorithm 1: **Basic evolutionary algorithm**

parents is recombined to generate L offsprings (usually $M = 2$). The resulting offspring population is denoted $M'(t)$. This operation is also known as *crossover*.

- *mutation*: In order to introduce new genetic material and ensure a certain degree of diversity, the genotype of an individual is altered by adding some randomly distributed noise to it. $M''(t)$ therefore represents the mutated offspring population.

- *update-population*: At this step, it is decided which individuals survive to the next generation. The *update method* is commonly termed *reproduction scheme*. Here, Darwins principle *survival of the fittest* is at work. Individuals with better fitness are preferably chosen to survive. Note the difference from biological evolution. In the artificial simulation, there exist reproduction schemes, in which the pool of possible survivors is build from both groups parents and offsprings, thus, it may happen that some individuals survive for a large number of generations.

t represents the generation count. In biological evolution there does not exist a *termination condition*. Computer simulations, however need to stop at some time and return a result. Typical termination conditions are, a fixed number of generations or a low degree of diversity within the population, e.g. the fitness of the best individual does not differ significantly from the average fitness of the entire population. A comprehensive introduction to evolutionary optimization is provided in [21].

Traditionally, it is being distinguished between genetic algorithms (GA's) and evolution strategies (ES's). However, Algorithm 1 merges GA and ES. GA's were first used by Holland and Goldberg and use a binary representation of an individual. This category of evolutionary algorithms mainly focus on crossover, whereas mutation plays only a minor role. In contrast, evolution strategies which were introduced by Rechenberg and Schwefel usually work on real-valued representations. Here, the operator which mainly guides the search is mutation. Often, *strategy parameters* are added. Strategy parameters are addi-

tional alleles of the individual's genotype. Their purpose is to guide the mutation, e.g. recognize if a certain setting of mutation parameters works well. This "recognition" is achieved by letting the strategy parameters itself be subject to evolution, e.g. mutation and crossover. Today, many evolutionary algorithms, cannot clearly be categorized into the GA or ES. As this categorization is not important for our purposes, and to avoid confusion, we use the term *evolutionary algorithm (EA)* from now on. Therefore we define an EA as a heuristic optimization algorithm which uses Darwin's principle of *survival of the fittest* to guide the search for good solutions.

3.2 Robustness

3.2.1 Definitions and Examples

In optimization *robustness* of a solution is defined as the property of being insensitive to

1. *changes in the environment* or
2. *noise in the decision variables* (see [22]).

An example for *noise in the decision variables* is the appearance of manufacturing tolerances. In many industrial manufacturing processes, a product can not be produced exactly according to the design specifications, but slight differences between specification and implemented solution are expected. In this example, a solution which is embodied by a product specification is robust, if it's implementation performs well despite slight deviations to the specification. An example for *changes in the environment* can be observed in scheduling. Here, we find many sources of disturbances: A machine breaks down, a necessary input product is not available due to logistic problems[1], or an important job arrives unexpectedly.

[1] often appears in highly synchronized supply chains, with *just in time production*

The fitness function $f(x, e)$ of an individual x depends on environmental parameters e, although these are not subject to optimization. To formalize the two cases: $x \mapsto x + \delta$ and $e \mapsto e + \delta$, where δ is a randomly distributed noise factor.

3.2.2 Robustness Measures

A widely used robustness measure is the *expected fitness* f_{exp} over a range of possible disturbances. This requires that a probability distribution of the disturbance is given. If the distribution of δ is continuous, the expected fitness is calculated

$$f_{exp}(x) = E(f(x+\delta)) = \int_{-\infty}^{\infty} p(\delta) \cdot f(x+\delta)\, d\delta \tag{3.1}$$

Equation 3.1 must be interpreted as (theoretical) expression, because in most cases f_{exp} cannot be derived analytically, either because f is not available in closed form, or if f is available in closed form it cannot be integrated. Furthermore p might not be integrable, e.g. p is a normal distribution. Thus, f_{exp} needs to be estimated. An example which illustrates the difference between raw fitness optimization and robustness optimization is depicted in Figure 3.1. The corresponding fitness function is defined in Equation 3.2. Here, the objective is to minimize f_{1exp}.

$$f_1(x) = \begin{cases} -1 & 2.0 \leq x \leq 4.0 \\ -2 & 6.9 \leq x \leq 7.1 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

For this test problem, f_{exp} can be calculated analytically, because we assume δ to be uniformly distributed on $[-1; +1]$. The raw fitness optimum $opt(f)$ is located in the x-interval $[6.9; 7.1]$, whereas the expected fitness optimum is at $x = 3$.

Note, that f_{exp} depends on the probability distribution of δ. If, e.g., δ is uniformly distributed on $[-0.1; +0.1]$, we see that $opt(f)$ equals $opt(f_{exp})$. If, however, δ is uniformly

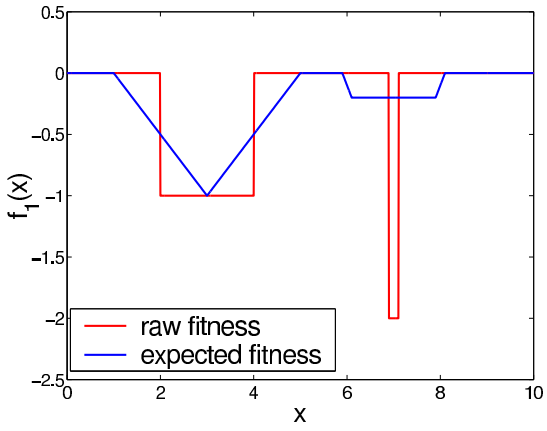
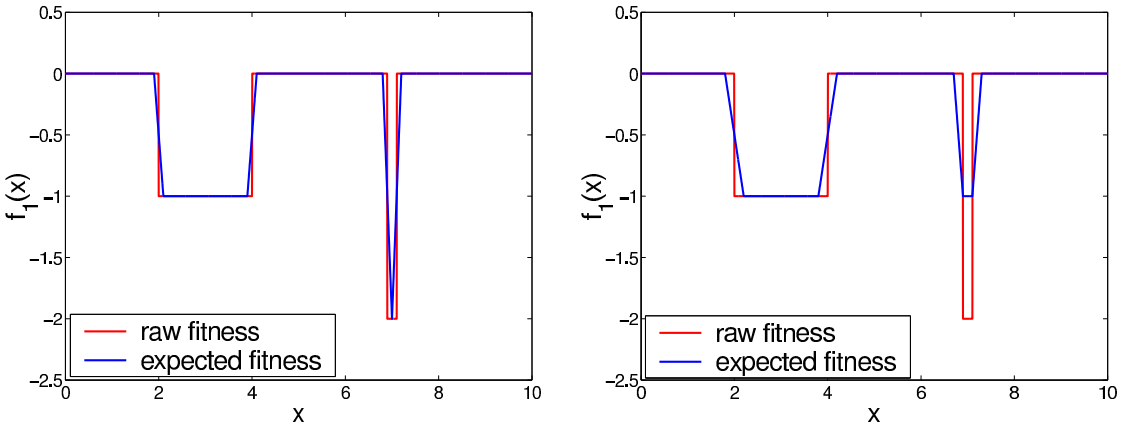


Figure 3.1: Trade-off between raw fitness optimum and expected fitness optimum



(a) δ uniformly distributed on $[-0.1; +0.1]$ (b) δ uniformly distributed on $[-0.2; +0.2]$

Figure 3.2: Expected fitness optima $opt(f_{exp})$ with different intervals of uniform noise distributions

distributed on $[-0.2; +0.2]$, f_{exp} has global optima on the left plateau as well as on the right peak (see Figure 3.2).

We see, that a decreasing variance of the noise distribution lets the shape of f_{exp} approach the shape of f. In contrast, the larger variance of the noise distribution the stronger the fitness landscape is smoothened.

Using f_{exp} as a single optimization criterion is the most prominent robustness measure.

However, from the point of view of decision theory, this approach has an elementary weakness, because it implicitly assumes neutral risk preferences of the decision maker. For a risk neutral decision maker the expected benefit is the only decision criterion, in particular, he does not care about the *expected deviation*, which is usually measured as *variance*. If the distribution of δ is continuous, the variance of the fitness is calculated

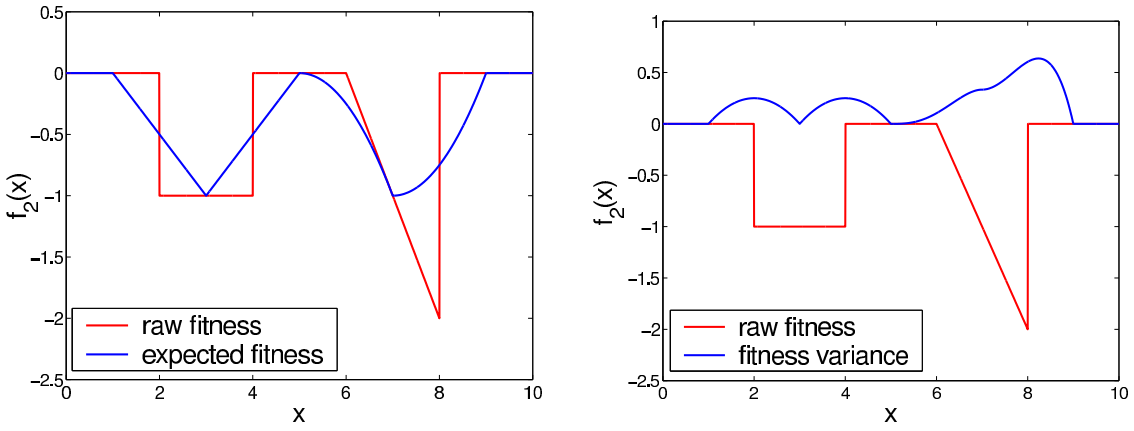
$$f_{var}(x) = var(f(x+\delta)) = \int_{-\infty}^{\infty} p(\delta) \cdot (f(x+\delta) - f_{exp}(x))^2 \, d\delta \tag{3.3}$$

In most real world problems f_{var} cannot be calculated analytically.

To illustrate the weakness of the single criterion approach, take a look at the following fitness function (Equation 3.4, Figure 3.3):

$$f_2(x) = \begin{cases} -1 & 2 \leq x \leq 4 \\ 6 - x & 6 \leq x \leq 8 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The graph on the left side of Figure 3.3, depicts the expected fitness of function f_2. Considering this as single robustness criterion, a decision maker is indifferent between the solutions $x^{(1)} = 3$ and $x^{(2)} = 7$. However, considering the variance as second robustness criterion, as depicted on the right side of Figure 3.3, a decision maker who is risk averse clearly favors solution $x^{(1)}$ because this has a lower expected fitness deviation. In this example, the expected fitness deviation of $x^{(1)}$ is 0. In contrast, a risk-taking decision maker chooses $x^{(2)}$, as he emphasizes the chance of receiving a higher benefit than the expectation. As mentioned earlier, a risk neutral decision maker is indifferent between $x^{(1)}$ and $x^{(2)}$. In most optimization problems, we can assume risk aversity. Examples include design optimization, portfolio optimization, scheduling. In this cases, the objective is to minimize the variance. Nevertheless, it is generally possible that a decision maker is



(a) f_{raw} and f_{exp} (b) f_{raw} and f_{var}

Figure 3.3: f_2 with δ uniformly distributed on $[-1; +1]$)

risk-taking. From the perspective of decision theory, this is the reason why people play lottery. In optimization, however, we have not found a similar example.

We conclude that, if a decision maker has non-neutral risk preferences, robustness optimization requires two objectives, optimizing f_{exp} as well as f_{var}. How we deal with multiple objectives in optimization, will be introduced in Section 3.3.

Differing from the commonly used notation in robustness optimization, we do not use the term *effective fitness* for the expected fitness. This term only makes sense, in single objective robustness optimization. With non-neutral risk preferences, a robustness measure must include (at least) 2 objectives. Although a large part of this work deals with single objective robustness optimization, we use the terms f_{exp} and f_{var} throughout this work in order to avoid confusion.

3.3 Multi objective optimization

In Section 3.2 we conclude that robustness optimization requires two objectives if the decision maker has non-neutral risk preferences. This section describes the difficulties when optimizing more than one objective, presents approaches to that and outlines an example of a multi objective evolutionary algorithm (MOEA) (Sections 3.3.1 - 3.3.4). Furthermore a brief introduction on how to interpret empirical results from MOEA runs is outlined in Section 3.3.5.

3.3.1 Difficulties

Most optimization methods are designed to optimize a single objective f_1. Thus, they try search for a solution x^*, such that $f_1(x^*)$ is minimal or maximal. A straight-forward approach to use SOO methods for MOO problems, is to merge multiple objectives to a single by building a weighted sum of the objectives. In this case the optimization problem is transformed to

$$opt \;\; f_{sum}(x) \;=\; \sum_{1 \leq i \leq n} w_i \, f_i(x), \tag{3.5}$$

where x is a solution, n is the number of objectives, f_i is objective function i and w_i the corresponding weight. The weight vector w reflects the preferences of the decision maker regarding the relative importance of the n objectives. Once a good w is found, this method works well. The major drawback of the weighted sum approach is that in reality the preferences of the decision maker are either not known in advance, or they cannot be quantified. For example, the decision maker has different weight vectors in different domains of the solution space. The fact that a weight vector is not static, is not a problem in general, because the set of different weight vectors could be made available to the optimization algorithm. Still, the task of defining proper weight vectors for any

possible scenario, might be a to complex and time consuming task for a decision maker. However, a good solution to a problem, *must* take into account user preferences somehow.

3.3.2 Decision makers preferences

In order to account for the preferences of the decision maker, we distinguish between three categories of optimization methods:

1. *a priori*
2. *interactive*
3. *a posteriori*

Methods that aggregate multiple objectives to a single one before running the optimization procedure fall into the group of *a priori* methods. One example is given by the *weighted sum* method in subsection 3.3.1. For robustness optimization another aggregation method is mentioned in [15], where f_{exp} and f_{var} are combined to the signal to noise ration $\frac{f_{exp}}{f_{var}}$. For a risk averse decision maker ($min\ f_{var}$), this requires f_{exp} to be formulated, such that it needs to be maximized. However, this method implicitly assumes a weighting, too.

In *interactive* methods, the decision maker improves the search for high performance solutions by providing information on his preferences *during* the run of the optimization algorithm. For example, the algorithm stops at a certain point and asks the decision maker to sort a certain set of solutions with respect to his preferences. Based on this information the algorithm modifies it's parameters and continues searching. Interactive methods reduce the drawbacks of a priori methods. However, the search might be misguided *due to* interaction.

A posteriori methods do *not* make assumptions on the preferences of the decision maker. Consequently, they do not produce a single optimal solution. Instead, the goal of

a posteriori methods is to find a *set* of solutions which can be presented to the decision maker. A desired property of such a solution set is a certain degree of diversity which reflects a trade-off between the objectives.

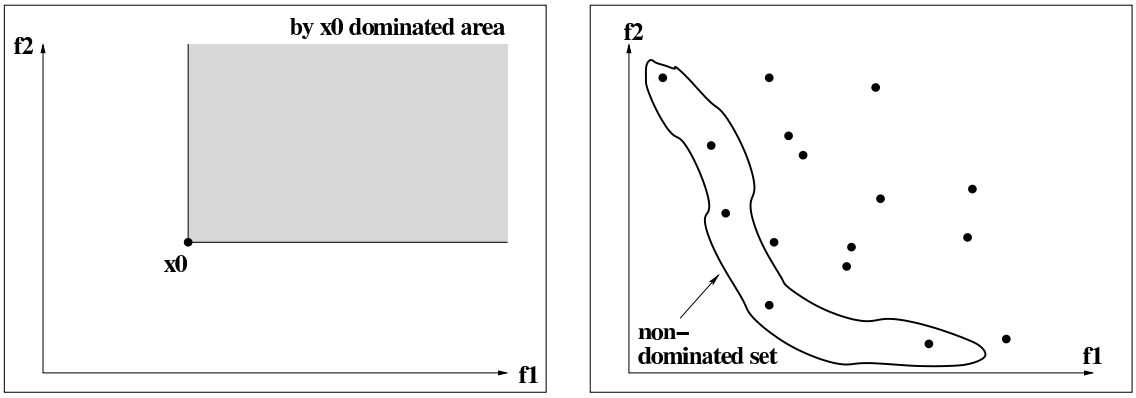
Recalling the basic operations of an EA, we see that evolution requires an update of the population: Individuals with good fitness survive to the next generation. If there are multiple objectives (multiple fitness values) and if we do not make assumption on the preferences to the objectives, a ranking of the individuals might not be possible. The question arises: *How is the population update done in MOEA's?*. Before further discussing this question, we *briefly* introduce the concept of *pareto dominance*. A comprehensive introduction to this concept and multi-objective evolutionary optimization in general can be found in [23].

3.3.3 Pareto dominance

Assume an optimization problem consists of n objectives (criteria) f_i, $i = 1 \ldots n$, which are all formulated such that each f_i is to be minimized. Let x,y be solutions to the optimization problem:

Definition 3.3.1 (Pareto dominance)
x is pareto-dominated by y if y is at least as good as x in all criteria and if y is strictly better than x in at least one criterion:
$x \prec y$ *(x is pareto-dominated by y),* *if* $\forall\, i : f_i(y) \leq f_i(x)$ *and* $\exists\, i : f_i(y) < f_i(x)$

If x is pareto-dominated by y, y is preferred by a decision maker regardless of his preferences. Consider a set of solutions P. Although with the concept of pareto dominance, we might not be able to find a ranking of *all* solutions, we are now able to determine a subset $P' \subseteq P$ of which all elements are not pareto-dominated by any element of P. P' is the non-dominated set:



(a) (b)

Figure 3.4: Pareto-dominance: a) pareto-dominated area, b) non-dominated set

Definition 3.3.2 (Non-dominated set)

The non-dominated set is defined as the largest set P' for which the following statement is true:

$max\,|P'| :\ P' \subseteq P :\ \ \forall x \in P' :\ \ (x \not\prec y)\ \ \forall y \in P$

For illustration, see Figure 3.4. In this example the optimization problem has two objectives *minimize* f_1, *minimize* f_2. In (a), the area which is dominated by a point x_0, is depicted. Members of the non-dominated set, as depicted in (b), are these points which have a location that is not covered by the dominated regions of all other points.

The concept of pareto-dominance can now be used to set up an *a posteriori* multi-objective evolutionary algorithm.

3.3.4 NSGA2: An *a posteriori* multi-objective EA

This subsection *briefly* introduces *Non-dominated Sorting Genetic Algorithm 2* (NSGA2) which was developed by Deb et al. in 2000. For details, we recommend [16]. NSGA2 is an improved version of NSGA which was developed earlier. In principle, the structure of the

basic evolutionary algorithm (Algorithm 1) does not change. However, some operations are modified in order to adapt the algorithm to the requirements of MOO. Obviously, each individual must be assigned multiple fitness values by the *evaluation* procedure. The essential property of NSGA2 is the *reproduction scheme* which is the underlying concept of the *update-population* operation. In NSGA2, this operation can be divided into two operations:

1. *non-dominated-sorting*

2. *crowding-distance-sorting*

In *non-dominated-sorting*, we divide a population P into a set of *pareto fronts* P_i: In a first iteration, we determine the non-dominated set of P which is assigned the name P_1. Then, we remove P_1 from P, apply the same procedure for $(P \setminus P_1)$, and get the next pareto front P_2. We repeat this operation until P is empty. At this point, P is divided into a set of pareto fronts, each with a rank i.

The next step is to apply a *crowding-distance-sorting* procedure which sorts the individuals within the paretofronts. In NSGA2, the *crowding distance* is the sorting criterion. Recall that in MOO the goal is to find a *set of solutions*. A performance criterion for such a set, is the degree of *diversity* within the set. Note that this diversity refers to the *fitness space*[2] which does not imply that the individuals have significantly different genotypes. Diversity is obviously a reasonable criterion, because this a posteriori provides a larger pool of solutions to the decision maker. Thus, it is desirable to embed this criterion into the *reproduction scheme*. The diversity criterion in NSGA2 is the crowding distance. The crowding-distance function assigns large values to individuals which are located at the edge of a set of solutions in the fitness space, and/or which are located in sparsely populated regions of the fitness space. For details, we again refer to [16]. In the

[2]Figure 3.4 is an example of a 2-dimensional fitness space

update population step, the two sorting methods are combined to build the next parent generation from the pool of offsprings (see Algorithm 2).

BEGIN NSGA2_UPDATE_POPULATION

$P' = P(t) \cup M''(t)$

$P(t+1) := \emptyset$, $i = 1$

$F =$ *non-dominated-sorting* (P')

WHILE $(|P(t+1)| + |F_i| < N)$

$P(t+1) = P(t+1) \cup F_i$

$i = i + 1$

END WHILE

assign-crowding-distance F_i

$sort(F_i, \prec_{crowding_dist})$

$P(t+1) = P(t+1) \cup F_i[\ 1\ :\ (N - |P(t+1)|\ )\ ]$

END NSGA2_UPDATE_POPULATION

Algorithm 2: **NSGA2 UPDATE POPULATION**

Note that in Algorithm 2 the same notations are used as in Algorithm 1. F stores the set of paretofronts F_i, N is the size of the parent population. In the last line, the parent population $P(t+1)$ is filled up with those individuals of paretofront F_i which have the largest crowding-distance value.

3.3.5 Performance metrics

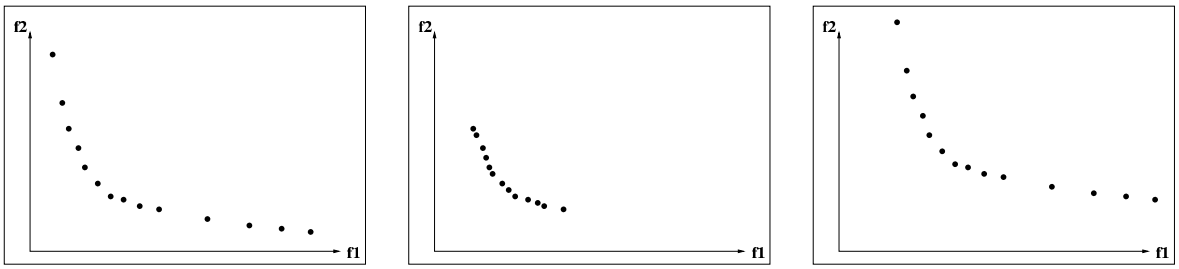
The goal of this subsection is to enable the reader to interpret the results which will be presented in Chapter 9. Particularly, we present guide lines on how to compare two sets of solutions. As mentioned earlier, diversity is one criterion for the quality of a set which

was e.g. returned by NSGA2. Many metrics for MOO have been developed in recent years. A comprehensive survey can be found in [24]. For our purposes, it is sufficient to emphasize two *qualitative* performance properties of a solution set:

1. *diversity within the set*

2. *location of the set (in the fitness space)*

For illustration, have a look at Figure 3.5. In this example, the solution set in (a) obviously



(a) (b) (c)

Figure 3.5: Comparison of different solution sets

has a higher diversity compared to solution set in (b). Although only qualitatively, we conclude that (a) is preferred to (b) due to a higher *diversity*. Comparing (a) and (c), we find that many solutions of (c) are dominated by solutions of (a). In this case, we conclude that solution (a) is preferable to solution (c) due to a better *location* of the solutions. Comparing (b) and (c), we do not get such a clear result. In this case, quantitative performance metrics are required. We will, however, see that this case does not appear in the results of this work. Therefore we believe that this brief illustration is sufficient.

3.4 Response surface methodology

3.4.1 Definitions

Comparing different sources, we found that the term *response surface methodology (RSM)* is not very well defined. A definition which covers most of the aspects is Def. 3.4.1.

Definition 3.4.1 (Response Surface Methodology)
RSM is a collection of mathematical and statistical techniques for empirical model building

A *response* is an *output variable* y of a system. This system also has a vector of input variables x. Each possible $x^{(i)}$ is mapped to a response $y^{(i)}$ by the *response function* f. The set of possible combinations $(x^{(i)}, y^{(i)})$ represent a *response surface*.

The initial intention of RSM was to model experimental responses. In other words, the goal of RSM is to find a functional relationship between input x and output y at acceptable cost. Initially, these experiments were mainly done in the field of physics or chemical processes. In recent years, RSM has been transfered to computer experiments, for example in simulation systems. The two basic RSM concepts are *approximate model* and *design of experiments (DoE)*.

3.4.2 Approximate model

We assume the *structure* of the relationship between the input and the output variables of a system to be unknown. Therefore, we first need to make an assumption on this structure. In most cases, low order polynomials, e.g. linear or quadratic, are used. In principle, arbitrary functions can be chosen. Taking into account problem specific knowledge for the choice of the model improves the approximation. In this work, we use linear and quadratic polynomials as approximate models.

We assume that some examples of input-output-data combinations are given or can be experimentally discovered. The coefficient vector c of the approximate model $\widehat{f}(c)$

are then chosen, such that $\widehat{f}$ explains the given input-output-data best. The resulting approximation function $\widehat{f}(c^*)$ is defined as *best fit*. Most commonly, the *minimum square distance* criterion is used for this purpose: If n input-output-data points $(x^{(i)}, y^{(i)}), i = 1 \ldots n$ are given, c is chosen by minimizing $e(c)$ in Equation 3.6.

$$e(c) = \sum_{1 \leq i \leq n} w(x^{(i)}) \left(\widehat{f}(c, x^{(i)}) - f(x^{(i)})\right)^2 \tag{3.6}$$

$w(x^{(i)})$ assigns a weight $x^{(i)}$. Although this method is widely used, generally, any fitting criterion can be used to determine the *best fit*.

3.4.3 Design of experiments (DoE)

A further important aspect of RSM is the design of experiments. The objective of DoE is to find locations, where the response is evaluated, such that the approximate model can be estimated sufficiently well at acceptable cost. The choice of the design of an experiment is usually very problem specific. As we will see later in more detail, in our application DoE is mainly left to the evolutionary algorithm[3]. The most prominent DoE methodologies are *full factorial design*, *central composite design*, *D-optimal design* and *Taguchi's methods*. These methodologies are not used in our approach, therefore we refer to [25] for details.

3.4.4 Extensions

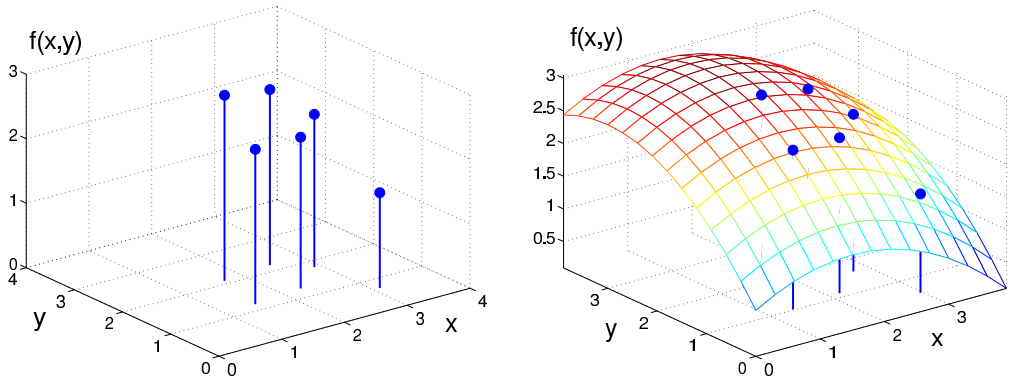
As mentioned earlier, the term RSM is not clearly defined in literature. In particular, it is not solely used, when procedures as explained in Sections 3.4.2 and 3.4.3 are applied. Instead, the term RSM is used whenever the surface of a response is to be estimated. This does not necessarily implicate that the response is noisy, nor does it restrict the approximation methods to regression analysis. In view of our approach to fitness approximation,

[3]this issue is difficult to understand with the information provided so far, however, at this point it is not essential to understand this detail

it needs to be mentioned that interpolation represents an approximation technique which is covered by the term RSM. Furthermore, DoE does not necessarily have to be planned by a human being, but can be left to a computer program, e.g. an evolutionary algorithm.

3.4.5 Illustration

For illustration, see Figure 3.6. In (a), a number of available 2-dimensional response data points are depicted. This can either be chosen by a *traditional* RSM procedure, e.g. a human, or is returned by an EA. In this illustration a quadratic polynomials is chosen as approximation model. The resulting approximation of the response surface is depicted in (b).



(a) (b)

Figure 3.6: Response data (a) and approximated response surface (b)

Chapter 4

Evolutionary robustness optimization

4.1 Introduction

In Chapter 3, we introduced a basic SO evolutionary algorithm (Section 3.1) and an example of a MO evolutionary algorithm (Section 3.3). The *evaluate* operation which assigns a fitness value to an individual, guides the search. If we want robustness to guide the search, *evaluate* must assign the expected fitness f_{exp} (SO), respectively (f_{exp}, f_{var}) (MO) to an individual. As discussed previously, f_{exp} and f_{var} cannot be calculated exactly, due to theoretical limitations and/or high cost of raw fitness evaluations. Thus, f_{exp} and f_{var} are estimated. Different estimation methods which are based on fitness function approximations (Chapter 5) are described in Chapter 6. Thus, the *evaluate* function in the robustness optimization EA returns estimations of either f_{exp} or (f_{exp}, f_{var}). A fitness approximation (Chapter 5) is based on a set of known fitness function values which are known as response data in RSM.

Section 4.2 describes which strategy we applied in this work to collect response data.

Section 4.3 describes the EA operators for the SO approach, Section 4.4 for the MO approach. Finally, in Section 4.5 we briefly discuss the challenges of constraint handling in robustness optimization.

4.2 Collecting response data

Response data collection strategies can be categorized into

1. *online collecting*: response data are solely collected at runtime of the evolutionary algorithm

2. *offline collecting*: response data are solely collected before running the evolutionary algorithm

3. combination of *online* and *offline collecting*: response data are collected before and throughout the run of the evolutionary algorithm[1]

The advantage of online collecting is that response data are likely to be evaluated at locations where they are needed. The major drawback of *pure* online collecting is that in the initial stages of the evolutionary algorithm only a small set of response data is available. This might cause large estimation errors for f_{exp} and f_{var} which could misguide the search. Offline collecting overcomes the difficulties in the initial stages of the EA but might waste computational power by evaluating the response at locations where it is actually not necessary. The combination of both represents a means to benefit from the respective advantages. However, such a method requires careful parameter settings which causes difficulties in the scientific analysis.

In this work we employed a pure online collecting strategy. Throughout the run of the EA, we evaluate the response (fitness function) at the points where individuals are

[1] online/offline collecting is often termed online/offline learning

located. If for example, our EA runs with population size 100 for 50 generations, in the final generation, the response data base has grown to a size of 5000. In analogy to biology, the response data base contains the information of all individuals of the history. We subsequently refer to this as *History*. For approximations solely *History* information is used.

4.3 An EA for SO robustness optimization

Taking into account *data collection*, the basic EA (Algorithm 1), needs to be modified. See Algorithm 3:

Algorithm 1 is extended by the *History_update* operation which adds a population with respective f_{raw} values to the *History*. Furthermore, the *evaluation* operation is replaced by *raw_fit_evaluation* and *robustness_estimation*: *robustness_estimation* calls the estimation procedures for f_{exp}. Our standard EA setting has the following properties (for details on the genetic operators we refer to [26]):

- *mutation*: We employed a standard evolution strategy, i.e. mutation of the objective variables $\vec{x}$ is carried out by adding a $N(0, \sigma_i^2)$ distributed random number to each component of x_i. The"step-sizes" σ_i which are known as *strategy parameters* are also subject to mutations (log-normal distributed).

- *recombination*: For the objective variables we use *discrete* recombination[2], that is for each allele one of the (two) parents is randomly selected for inheritance. For the strategy parameters we used *generalized intermediate* recombination, that is for each allele an uniformly distributed random number which lies between the parents allele's values is chosen for inheritance.

[2]also known as *uniform crossover*

BEGIN ROBUSTNESS_EA

$t \leftarrow 0$

initialization: initialize population $P(0)$

raw_fit_evaluation: evaluate $P(0)$ according to f_{raw}

History_update: add $P(0)$ to the *History*

robustness_evaluation: estimate f_{exp} for all individuals of $P(0)$

REPEAT

selection: mating pool $M(t) \leftarrow select(P(t))$

recombination: $M'(t) \leftarrow recombine(M(t))$

mutation: $M''(t) \leftarrow mutate(M'(t))$

raw_fit_evaluation: evaluate $M''(t)$ according to f_{raw}

History_update: add $M''(t)$ to the *History*

robustness_evaluation: estimate f_{exp} for all individuals of $M''(t)$

update-population: $P(t+1) \leftarrow u(P(t) \cup M''(t))$

$t \leftarrow t+1$

UNTIL(*termination condition*)

END ROBUSTNESS_EA

Algorithm 3: **Evolutionary algorithm for robustness optimization**

- *selection*: From the pool of parents, individuals are chosen randomly for recombination

- *update-population*: We employ a (μ, λ)-*reproduction scheme*, with $\mu = 15$ and $\lambda = 100$, i.e. the best 15 offsprings out of 100 build the next parent generation.

- *termination-condition*: We set a fixed number of 50 generations.

4.4 An EA for MO robustness optimization

As MO robustness EA we use NSGA2 with robustness extensions. As the extensions are exactly as in SO and NSGA2 has the same structure as Algorithm 3 we do not present a pseudo code version of the MO robustness EA. One essential difference should be mentioned. NSGA2 uses binary coded individuals[3]. Therefore the genetic operators are different. Our standard setting for the MO robustness EA has the following properties

- *mutation*: Flip each bit with a certain probability (*flip probability*)

- *selection*: Two individuals of the parent generation at a time are grouped as parents to produce offsprings.

- *reproduction*: A virtual coin toss which has the *crossover probability* as parameter decides whether the genotype of the parents is simply copied to the offsprings or if a *1-point-crossover* of the parents chromosomes is performed.

- *update-population*: The population update of NSGA2 was described in in Algorithm 2.

- *termination-condition*: We set a fixed number of 50 generations.

[3] in particular, we use gray code in our algorithm

4.5 Constraint handling

In most optimization problems it is clearly defined if a solution is feasible, i.e. it does not violate any constraint. Many different techniques to deal with infeasibility have been developed, the most common are

- feasibility-preserving operators
- special representations
- delete infeasible solutions
- penalty functions
- repair infeasible solutions
- constraint violation as additional objective (MO)

Noise in the decision variable, however, causes difficulties. Even if an individual is feasible, this does not guarantee that the actual implementation is feasible, too. An individual which is located close to the infeasibility boundary has a certain probability that it's implementation becomes infeasible. This probability depends on the distribution of the noise. The difficulties in the presence of noise can be summarized by a simply question *How do we assess a solution x that has probability of $p_{inf}(x)$ to become infeasible?* A straight-forward approach is to define a threshold, e.g. $p_{inf,max} = 0.05$. If an individual has a higher probability to become infeasible it is deleted or repaired. However, it is difficult to find a good setting of p_{max}.

Ray [27] proposes a constraint-handling scheme for robustness optimization. This method is based on the pareto concept and considers an individual's *self-feasibility* and *neighborhood-feasibility*. However, this method requires to evaluate the fitness function for k neighbors of each individual.

Further investigation of constraint-handling techniques is beyond the scope of this work. We therefore decide to avoid most of the difficulties related to infeasibility. In our algorithm we use a feasibility-preserving operator known as *bounce-off-the-boundary* to ensure that no individual is infeasible. With this operator all individual are located in the feasible region. If, however, samples are drawn from an infeasible region while estimating robustness of an individual, the sample is assigned a constant penalty. If an individual has high probability of becoming infeasible, a large faction of samples are drawn from the infeasible region, thus, the penalty will be large.

We constructed the test problems such that robustness optima are not located close to the border. Detailed information on the experimental set up can be found in Chapter 8.

Chapter 5

Fitness approximation (RSM)

In Section 3.4, we briefly introduced RSM as a means for fitness approximation. This chapter describes how RSM is actually employed in this work. In particular, we use *Nearest neighbor interpolation* and *Local regression* for fitness surface reconstruction. In Section 5.1 we first outline the underlying assumptions of this chapter. In Section 5.2 we describe which approximation models are used and discuss some properties. The choice of input to the approximation models is discussed in Section 5.3. The main reason to describe our RSM approach in a separate chapter is that we are often faced with numerical difficulties when it comes to approximation in the context of evolutionary algorithms. Reasons for these difficulties are outlined in Section 5.4. How we manage these difficulties for Nearest neighbor interpolation and Local regression is described in detail in sections 5.5 and 5.6. Remarks and comparisons regarding *computational complexity* are given in Section 5.7.

5.1 Assumptions, Definitions

Throughout this chapter, we assume that a set of response data (the *History*) is available in order to apply an approximation procedure. This issue will be discussed in more detail

in Section 5.3. We further assume that the goal of an approximation procedure is to find a setting of coefficients of an approximation function, such that it is most accurate at a given location. We refer to this location as *(model) fitting point* from now on.

5.2 Approximation models

Generally, an arbitrary parametric function can be chosen as approximation model. In this work however, we restrict our attention to polynomials for both *Nearest neighbor interpolation* and *Local regression*. These are

1. linear polynomial (*linear*)

2. quadratic polynomial with independent variables (*quadratic_indp*)

3. quadratic polynomial with cross products (*quadratic*)

$$f(x) = \begin{cases} \beta_0 + \sum_{1 \le i \le n} \beta_i \, x_i & \text{polynomial type} = \textit{linear} \\ \beta_0 + \sum_{1 \le i \le n} \beta_i \, x_i + \sum_{n+1 \le i \le 2n} \beta_{n+i} \, {x_i}^2 & \text{polynomial type} = \textit{quadratic_indp} \\ \beta_0 + \sum_{1 \le i \le n} \beta_i \, x_i + \sum_{1 \le i \le j \le n} \beta_{n-1+i+j} \, x_i \, x_j & \text{polynomial type} = \textit{quadratic} \end{cases} \tag{5.1}$$

Here, n is the number of dimensions, β denote the polynomial coefficients. The number of coefficients of a polynomial depends on the dimensionality and the type of the polynomial (Equation 5.2).

$$\textit{numcoeff}(n) = \begin{cases} n+1 & \text{polynomial type} = \textit{linear} \\ 2n+1 & \text{polynomial type} = \textit{quadratic_indp} \\ (n+1)(n+2) & \text{polynomial type} = \textit{quadratic} \end{cases} \tag{5.2}$$

5.3 Choice of response data (DoE)

In *classical* RSM, data points are chosen for evaluation, such that the resulting set of response data points has the best coverage of the region which is to be approximated. We assume a response function evaluation to be very expensive. Therefore we do not use any additional fitness evaluations but use response date which are stored in *History* (see Section 4.2).

Which History data are used as input to an approximation model depends on the desired *model fitting point* $x^{(0)}$. For both, *Nearest neighbor interpolation* and *Local regression*, the nearest data points of the *History* with respect to the Euclidean distance $\|h^{(i)} - x^{(0)}\|_2$ are chosen. Here, $h^{(i)}$ represents the vector of objective parameters of a History entry i. Thus, for each approximation, $\|h^{(i)} - x^{(0)}\|_2$ is computed for $i = 1 \ldots History_size$ and $h^{(i)}$'s are sorted with respect to this criterion[1].

How many data points k are used to build the model is different for interpolation and regression. In interpolation, the goal is to find a polynomial which intersects the given data points. Thus, the required number of data points k_{intpol} equals the number of coefficients $numcoeff(n)$ of the polynomial. In regression, the number of data points $k_{regress}$ which can be added to the model is infinite. Still, the minimum required number of data points equals $numcoeff$.

$$\begin{aligned} numcoeff(n) &= k_{intpol} \\ numcoeff(n) &\leq k_{regress} < \infty \end{aligned} \tag{5.3}$$

In regression, we add a feature to reduce computational cost: If $k_{regress}$ is larger than a

[1] this procedure is computationally very expensive, especially if the History size is large and the dimensionality is high. In [28] an *approximate* nearest neighbor searching algorithm is developed which speeds up this process. However, we use a brute force algorithm for computing the distances and *quicksort* as sorting algorithm

certain fraction of the History size, e.g.

$$k_{regress} > \frac{1}{10} \cdot History_size$$

we reduce $k_{regress}$ to $(\frac{1}{10} \cdot History_size)$. If, for example, the current History size is 5000 but the History data density around $x^{(0)}$ is such that $k_{regress} = 2000$, we only add every fourth input data point to the model. Thus, $k_{regress}$ is reduced to 500. We tested this feature with the above defined parameter setting and found that the performance did not degrade.

5.4 Numerical Difficulties in EA's

Theoretically, interpolation in multiple dimensions is straightforward and requires basic linear algebra operations. Whenever it comes to implementation, we are faced with numerical problems which arise from the fact that real numbers are not truly continuous on computers as they are stored in a certain bit format. If we choose data points which are *linearly dependent* or, from the perspective of numerical analysis, *close to linearly dependent*, the algebraic operations have a high probability to fail. As numerical linear algebra is not the main focus of this work, we do not go into further details here and refer to a standard book in numerical analysis, e.g. [29].

In the context of EA's, these problems become even more serious: During the run of the EA, several individuals are evaluated. All visited data points are stored in the History which provides input data to approximation procedures. In later stages of the run, the EA converges, thus many similar or equal data points are stored in the History. In this situation, we build models at fitting points which are located in regions with high History data density. If we choose the k nearest data points for approximation, the probability to choose data points which are similar or equal increases with the number of iterations of the EA. Similar data points have a high probability to be *linearly dependent* or *close to*

linearly dependent. Due to this (desired) behavior of the EA, interpolation and regression become numerically difficult.

It is worth mentioning that in real world applications equal data points should never be stored in the History. In fact they should never be evaluated. On the other hand, truly equal points in the sense that the floating numbers have equal bit representation, are expected to be very rare. Of course one could improve the algorithms by adding some *"equal threshold"*. However, this issue is beyond the scope of this work.

5.5 Interpolation

5.5.1 Standard method

Finding the polynomial coefficients, such that the resulting polynomial intersects the k given data points requires to solve a linear equation system. In this equation system, each row (equation) represents a data point. In matrix form, the equation system can be written as

$$A\beta = b \tag{5.4}$$

β represents the (solution) vector of coefficients, A is commonly termed *design matrix*, b is the vector of fitness values (response values). Rows of the design matrix are specified by applying the desired polynomial to the model input data, i.e. if for instance a data point $x = (x_1, x_2)$ is to be added to *quadratic* model, the following row will be added to the matrix:

$$[1 \;\; x_1 \;\; x_2 \;\; x_1^2 \;\; (x_1 x_2) \;\; x_2^2] \quad .$$

We get the coefficient vector β^* by multiplying the inverse of A with b.

$$\beta^* = A^{-1}b \tag{5.5}$$

If A is regular, A^{-1} exists and is unique, if A is singular, A^{-1} does not exist or is not unique. A common way to compute $A^{-1}b$ is to decompose matrix A into a product of two matrices A_1 and A_2, with A_1 and A_2 having certain properties which make it easy to compute $A^{-1}b$. LU decomposition is a standard method for this purpose. Based on Gaussian elimination, this method decomposes A into the product LU where L is unit lower triangular and U is upper triangular. The product LU is called the LU factorization. Using the LU factorization, Equation 5.5 is easy to solve by forward and backward substitution. The advantage of this method is the low computational cost of approximately $n^3/3$ multiplications and additions. See [30] for details.

5.5.2 Managing singularities

Simply applying LU decomposition and relying on the result does not work. Assume, we pick some data points from the History which are linearly dependent. Although the resulting matrix A is singular, LU decomposition returns a factorization of A which we use for solving the equation system (Equation 5.5). Of course, it is possible to check, whether β^* solves the equation system. However, even if we find that β^* perfectly solves Equation 5.5 which means the resulting polynomial intersects all input data points, this is not necessarily the desired result because it is not unique[2]. Theoretically, there exists an unlimited number of different solutions β^* to the underdetermined system. The β^* returned might not be appropriate for approximation of the fitness function.

A proper interpolation method requires to produce unique solutions. Hence, it needs to detect singularities if present. But, simply recognizing that A is singular is also not

[2]Citing Gentle, "it is neither necessary nor sufficient that a matrix be nonsingular for it to have an LU factorization" ([30], page 92)

sufficient. Our method must be able to identify which rows of A are linearly dependent, in order to replace these by other History data. The method of choice is *QR decomposition with column pivoting*. We refer to this as *QRPT* subsequently. *QR decomposition* (*without* column pivoting) factorizes a matrix A into the product QR, where Q is an orthogonal matrix and R is upper triangular. The QR factorization is obtained by either Householder transformations, Givens transformations or the (modified) Gram-Schmidt procedure (see [30] for details). In our implementation, we use Householder transformations. *With* column pivoting, $A = QR$ is extended to $PA = QR$ where P is a permutation matrix which keeps track of which columns of the original matrix A were exchanged. Without going into further details (see [31], algorithm 5.4.1), we present the procedure for detecting linearly dependent rows of matrix A:

As *QRPT* detects linearly dependent *columns*, we first need to transpose matrix A. Applying *QRPT* to A^T returns Q, R and P. The interesting values are now on the diagonal of R, i.e. r_{ii}. If a r_{ii} is *zero* or *close to zero*[3], we know that the corresponding column of A^T is *linearly dependent* or *close to linearly dependent*. We get the index of the corresponding row in A by applying the permutation P to the index i of r_{ii}. If we detect no linearly dependent row (A is regular), we can now use LU decomposition to solve Equation 5.5, because a regular matrix has a unique LU decomposition. If we identify linearly dependent rows we replace these by the next data points of the sorted History.

In order to quantify what is *close to zero* as indicator for a row to be *close to linearly dependent*, we define a threshold to which we refer as *singular_threshold*. If a certain r_{ii} is lower than *singular_threshold*, we assume the corresponding row in A to be linearly dependent. Two types of bad setting of *singular_threshold* may appear:

1. *singular_threshold* is too low

2. *singular_threshold* is too high

[3] must be defined by a threshold

The first pitfall is more serious: We do not detect any linearly dependent row, although the matrix is singular. In this case, we wrongly assume the matrix to be regular and compute the LU decomposition. As shown above, this will probably return an undesired result. Even checking the result subsequently might not help. We can deal with this by initializing *singular_threshold* to a moderately low value, e.g. 10^{-12} which has shown to work well.

However, with this strategy, the second pitfall might appear: We detect at least one linear dependent row, although the matrix is actually regular. Relying on this, we would continue replacing rows, perhaps without success. Although we do not get a false result, we waste a lot of computation time. In the worst case, the algorithm continues replacing data points until the History database is empty. In order to avoid this, we developed an adaption mechanism for the *singular_threshold*: When $QRPT$ detects singularities although we have repaired a certain number of times, we assume that this is due to a too high *singular_threshold*, and decrease it (by *decrease_factor*).

There exists one additional case in which the *singular_threshold* is adapted: When we do not detect linearly dependent rows, we try to solve Equation system 5.5 with LU decomposition. We test the result by applying it to Equation 5.5 and check if $A\beta^* = b$ with respect to some threshold (*equation_solved_threshold*). If we find that the equation system can not be solved appropriately, we assume that A was singular. As we did not detect singularities we conclude that this was due to a too high *singular_threshold*. We increase *singular_threshold* (by *increase_factor*) iteratively until we detect at least one linearly dependent row which can then be replaced.

Theoretically, it is possible that the interpolation algorithm unsuccessfully proceeds replacing input data points until the History is empty. If this is the case, we try the same procedure with a lower level polynom: The polynomial types are sorted and enumerated

with respect to *numcoeff*. If for example *quadratic* interpolation fails, *quadratic_indp* is tried. If *quadratic_indp* fails, too, *linear* is tried. *Linear* interpolation is the simplest polynom. If interpolation fails even for this type, we perform *unweighted constant regression*, i.e. we choose a certain number of nearest data points and calculate the mean of the corresponding fitness values. The first coefficient (β_0) of the *linear* polynomial is set to the mean fitness value, all other coefficients ($\beta_i, i \in 1 \ldots n$) are set to *zero* (constant function)[4].

5.6 Local regression

5.6.1 Background

Global or *unweighted* regression approximates a function by minimizing the vertical square Euclidean distance between input data points and the function which is to be approximated. This is known as *least square fitting*. *Local* regression tries to approximate a function locally in the neighborhood of a *fitting point* $x^{(0)}$. To achieve a good *local fit* we weight the distances which are to be minimized. Distances which correspond to data points which are close to $x^{(0)}$ are assigned a larger weight. Here we assume that the information provided by these data points is more important. The functional dependence between distances and weights is a design parameter of Local regression which we denote as *weight function*. The number of model input data points $k_{regress}$ is a design parameter, too. Technically, interpolation is a special case of regression with $k_{regress} = k_{intpol}$. However, in most cases $k_{regress} > k_{intpol}$, thus an equation system

$$A\beta = b \tag{5.6}$$

needs to be solved which is overdetermined. Adding an error term e on the right hand

[4]this case did not appear in reality. The motivation for the implementation is to ensure the robustness of the algorithm.

side, we get

$$A\,\beta = b + e \tag{5.7}$$

A least square solution to Equation 5.7 is such that the Euclidean norm of e is minimal. We therefore need to find β^* for which the Euclidean norm of $(b - A\beta)$ is minimized. Differentiating for β and setting the result to *zero* gives us

$$(b - A\,\beta)^T\,(b - A\,\beta) = 0 \tag{5.8}$$

There exist two methods to solve for β, via *Normal equations* or via *Residual norm*.

5.6.2 Normal equations method

Transforming Equation 5.8, we see that the least square solution β^* satisfies

$$A^T\,A\,\beta^* = A^T\,b \tag{5.9}$$

Equation (system) 5.9 is called the *Normal equations*. If we assign weights to distances, the Normal equations are defined as

$$A^T\,W\,A\,\beta^* = A^T\,b \tag{5.10}$$

where W is a diagonal matrix with weights on it's diagonal. Note that $A \in R^{n\times m}$, $W \in R^{n\times m}$ and $b \in R^m$, where $m = numcoeff$ and $n = k_{regress}$. A^TWA as well as A^Tb can be computed by matrix multiplication. Thus, we need to solve a linear equation system exactly as in interpolation. Here, A^TA respectively A^TWA is termed *design matrix*. Analogous to interpolation, the solution to the linear equation system can be found by LU decomposition. However, in this case there exists a method which works more efficiently by taking advantage of the symmetry of A^TA, respectively A^TWA. This method is known as *Cholesky decomposition*. Cholesky decomposition factorizes a quadratic, positive definite matrix A into the product of two upper triangular matrices T,

$$A = T^T\,T\ . \tag{5.11}$$

First, Cholesky decomposition is computationally very efficient. It requires only $1/6n^3$ operations on a $(n \times n)$-matrix and is therefore twice as fast as LU decomposition. Secondly, it is extremely numerically stable. In fact, it is the preferred method, if one wants to test, if a matrix is positive definite. We refer to [32] and [31] for details.

5.6.3 Residual norm method

If we want to compute an unweighted regression, we can use a different technique which works directly on A rather than on the design matrix $A^T A$. Consider again Equation 5.8. The overdetermined matrix A has a QR decomposition such that $A = QR$. For $n > m$, R is of the form

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

where R_1 is upper triangular. Now, Equation 5.8 can be written as

$$\begin{aligned}
(b - A\beta)^T (b - A\beta) &= (b - QR\beta)^T (b - QR\beta) \\
&= (Q^T b - R\beta)^T (Q^T b - R\beta) \\
&= \left(\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \beta \right)^T \left(\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \beta \right) \\
&= \left(\begin{pmatrix} c_1 - R_1\beta \\ c_2 \end{pmatrix} \right)^T \left(\begin{pmatrix} c_1 - R_1\beta \\ c_2 \end{pmatrix} \right) \\
&= (c_1 - R_1\beta)^T (c_1 - R_1\beta) + {c_2}^T c_2
\end{aligned} \tag{5.12}$$

The transformation between the first and second row is allowed, because for orthogonal Q, the euclidean norm has the property $\| Q\beta \|_2 = \| \beta \|_2$. c_1 is a vector of length m, c_2 is a vector of length $n - m$, such that

$$Q^T b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \tag{5.13}$$

. The left hand side of Equation 5.12 is called the *Residual norm*. The residual norm is minimal if $(c_1 - R_1\beta)^T(c_1 - R_1\beta)$ is minimal because quadratic forms are non-negative. The quadratic term is minimal if $(c_1 - R_1\beta) = 0$. Solving for β we get,

$$\beta^* = {R_1}^{-1}c_1. \tag{5.14}$$

This is easy to solve, because R_1 is triangular. Further details can be found in [30].

5.6.4 Normal equations vs. Residual norm method

Comparing the two methods, the numerical analysis is strongly in favor of the *Residual norm* method. Here the concept of *matrix condition* is important. Citing Gentle [30], "data are said to be *ill-conditioned* for a particular computation if the data were likely to cause problems in the computations, such as severe loss of precision. More generally, the term ill-conditioned is applied to a problem in which small changes to the input result in large changes in the output"[5]. For specific problems such as solving equation systems, the *condition* of a matrix can be quantified by a *condition number*. Condition numbers are defined to be positive,, and a large condition number is associated with strong ill-conditioning. We recommend [30] for a comprehensive introduction. It can be shown that the *condition number* of a matrix A^TA is the square of the *condition number* of matrix A. Therefore it seems favorably to work directly on A.

For a comparison of the computational cost, let us assume that no singularities appear and that for both methods the most efficient algorithms are employed. Again, we define $m = numcoeff, n = k_{regress}$. Equation 5.15 summarizes the computational cost for the Normal equations method (*NE*): Matrix multiplication is the most expensive operation. By taking advantage of the symmetry we reduce the cost to approximately 1/2. Computation of the equation system's right hand side is straight forward. As decomposition

[5] pages 75 ff.

method we use *Cholesky* decomposition here. Finally, the equation system is solved by forward and backward substitution.

$$cost(NE) = \underbrace{1/2\,n\,m^2}_{\text{matrix multiplication}} + \underbrace{m\,n}_{\text{right hand side}} + \underbrace{1/6\,m^3}_{\text{Cholesky decomp.}} + \underbrace{m^2}_{\text{forw./backw. substitution}} \tag{5.15}$$

Equation 5.16 summarizes the computational cost for the Residual norm method (*RN*): QR decomposition is the most expensive operation and requires $2/3\,n^3$ multiplications and additions. After Q and R have been computed, c_1 is computed (compare Equation 5.13). Equation 5.14 is solved by backward substitution.

$$cost(RN) = \underbrace{2/3\,n^3}_{QR\text{ decomposition}} + \underbrace{m\,n}_{\text{computing } c_1} + \underbrace{1/2\,m^2}_{\text{backw. substitution}} \tag{5.16}$$

Comparing equations 5.15 and 5.16, we see that for sufficiently large n, the computational cost of the Residual norm method are higher than the cost of the Normal equations method. Recall, that, if $m = n$, regression is equivalent to interpolation. The curves in Figure 5.1 depict $\frac{cost(RN)}{cost(NE)}$ for different n's. Here we set $n = k \cdot m$. Already for relatively

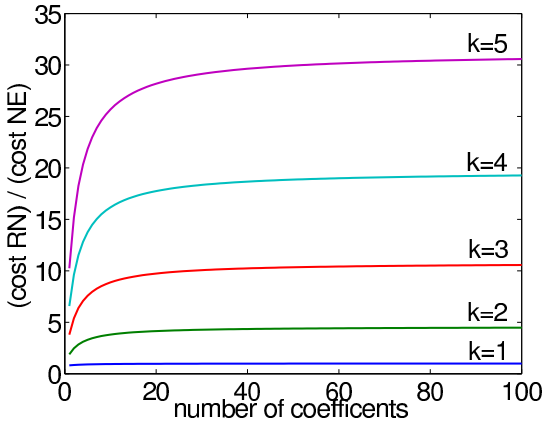


Figure 5.1: computational cost: Residual norm as factor of Normal equations method

small k, *RN* incurs significantly higher computational cost than *NE*, e.g. for $k = 2$ and 10-dimensional quadratic regression ($m = 66$), *RN* requires 4.4313 as many multiplications and additions as *NE*. The *major* disadvantage of the *Residual norm* method is, however,

that it can only be applied to *unweighted* regression. We fail to do the transformation from equation 5.12 with a weight matrix W, because

$$(b - QR\beta)^T W (b - QR\beta) \neq (Q^T b - R\beta)^T W (Q^T b - R\beta)$$

For orthogonal Q, the property $\| Q\beta \|_2 = \| \beta \|_2$ only holds for "pure" L_2 norm. Considering these aspects, we decide to use the Normal equations method with Cholesky decomposition[6].

5.6.5 Managing singularities

Successful application of the regression procedure requires that the input data matrix A has full rank, or in other words: m out of the n input data points are linearly independent, thus the problem is similar to the case of singularities in interpolation (Section 5.5.2). Here we deal with singularities by identifying linear dependent matrix rows and replacing them. However, in regression the number of matrix rows in A is not restricted. If A is singular (has less than full column rank), we iteratively add data points until A is regular. Thus, it is sufficient to detect that A is singular. Applying Cholesky decomposition on $A^T W A$ represents a means to detect singularities in A, because the following statement is true

$$A \text{ regular} \quad \Rightarrow \; A^T W A \quad \text{positive definite}$$

As mentioned earlier, Cholesky decomposition is the preferred method if a matrix is positive definite. If Cholesky decomposition detects $A^T W A$ to be not positive definite, we conclude that A was not regular. We can add a data point to A and repeat the procedure until Cholesky finds $A^T W A$ to be positive definite. This method works correct. However, adding data points one by one and applying the decomposition methods many times, is very time consuming. We developed a simple heuristics which estimates how many

[6]both decomposition methods, Cholesky and QR are implemented and can be chosen by the user

data points are to be added to the model. In particular, we modified the Cholesky routine. Recall the notation from Section 5.6.2. Additionally we define $B := A^T W A$ with corresponding entries b_{ii}. Cholesky decomposition computes the diagonal entries t_{ii} of matrix T as follows:

$$T_{ii} = \left(b_{ii} - \sum_{1 \leq k \leq i-1} T_{ik}^2 \right)^{1/2} \tag{5.17}$$

If the square root can not be taken because the inner term is negative, this indicates that B is not positive definit. In our modification, we simply count how often this case appears, and take this number as heuristic estimation for the required number of additional input data points. We tested this heuristic empirically and found that in most cases the heuristic performes as if an exact rank determining method, e.g. $QRPT$, was used.

Additionally, we modify the standard Cholesky procedure by adding a *positive_definite_threshold*. Instead of checking whether T_{ii} is non-negative, we check whether $T_{ii} <$ *positive_definite_threshold*. This threshold is dynamic and adapts exactly as *singular_threshold* in Section 5.5.2.

In the theoretical case when all History data are added to the model and the regression routine is still not able to solve the system, we follow the same strategy as in interpolation and reduce the number of coefficients by using a lower order polynomial.

5.6.6 Design parameters

Although part of the information of this subsection was already provided in earlier section, it is useful to present it in the context of regression design parameters. This subsection therefore aims at summarizing the regression procedure. The following design parameters should be mentioned

- *fitting criterion*: As mentioned in Section 5.6.1, we use least square estimate, which is a standard fitting criterion. Generally, any criterion can be used.

- *bandwidth*: The bandwidth defines the range (measured as Euclidean distance) in the search space from which data points are added to the regression model. In our application we set the bandwidth such that it covers the range of the noise. Possibly, the number of data points is too small (smaller than *numcoeff*). In this case, the bandwidth is extended, such that a *minimum number* of data points k_{min} lie within the bandwidth. k_{min} is specified by the user. In our application, we set

$$k_{min} = 2 \cdot numcoeff$$

  in order to have a clear differentiation to the interpolation method.

- *weight function*: As weight function we use the *tricube* function which is defined

$$W(x^{(i)}, x^{(0)}) = \left(1 - \left(\frac{\| x^{(i)} - x^{(0)} \|_2}{bandwidth}\right)^3\right)^3 \tag{5.18}$$

  $x^{(0)}$ denotes the fitting point of the model. Figure 5.2 shows the tricube function for $bandwidth = 1.0$. In addition, we define a *minimum weight*, in order to avoid numerical problems[7].

5.7 Computational complexity

This section briefly analyzes the computational complexity of the presented approximation methods. For this analysis, we did not measure actual runtime but use the theoretically known complexity. In order to make the different methods comparable, we assume that no singularities appear. Nevertheless, the cost for checking for singularities are included. We further assume that a matrix of input data A and a vector of corresponding fitness values b is given.

[7] when we detect singularities, we deal with that by adding the necessary number of new History data points to the regression model. If the data points are weighted with tricube function these values would either be very low or zero, as their distance to approximation point exceeds the initial bandwidth.

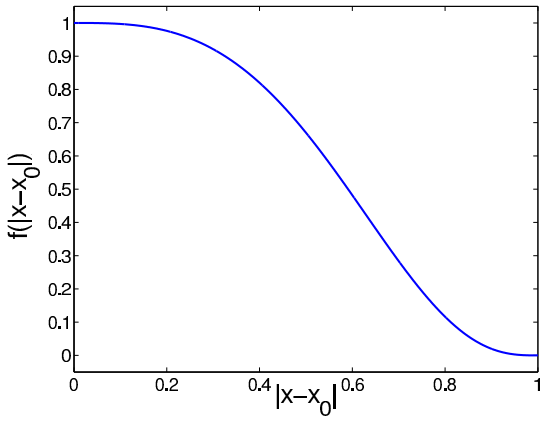


Figure 5.2: tricube function with *bandwidth* = 1.0

Interpolation requires to compute a matrix transpose, perform $QRPT$, check for singularities, perform a LU decomposition and solve the equation system by forward and backward substitution. We assume the cost of computing the matrix transpose to be 0, because in principle, the algorithms could be modified such that they work directly on A rather than on A^T. Further, the cost for checking for singularities after $QRPT$ has been computed are assumed to be negligible. In total, we get the cost of the interpolation procedure. See Equation 5.19 where m is the number of polynomial coefficients.

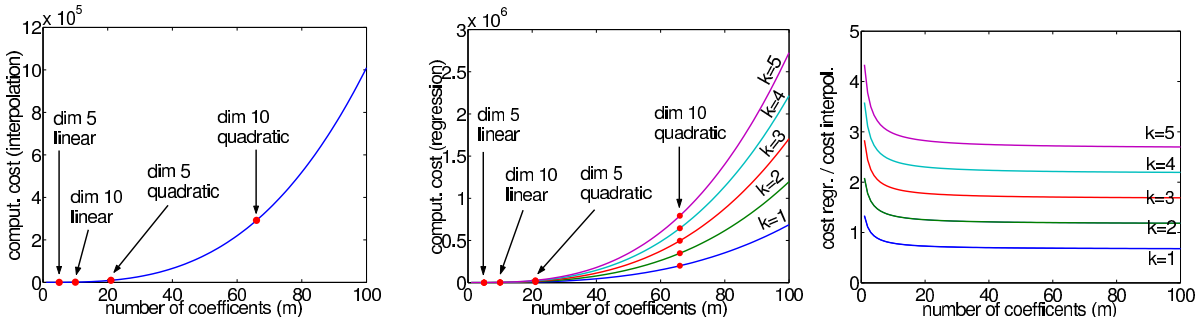
$$\begin{aligned} cost(interpolation) &= \underbrace{(2/3)\,m^3}_{QRPT} + \underbrace{(1/3)\,m^3}_{LU \text{ decomposition}} + \underbrace{m^2}_{\text{forw. + backw. substitution}} \\ &= m^3 + m^2 \end{aligned} \tag{5.19}$$

The computational cost for regression, when using Cholesky decomposition are already shown in Equation 5.15. As in Figure 5.1 of Section 5.6.4, we set the number of input data points $n = k \cdot m$. We get the cost function for regression:

$$cost(regression) = ((1/2)\,k + (1/6))\,m^3 + (\,k + 1\,)\,m^2 \tag{5.20}$$

Figure 5.3 depicts the cost functions for interpolation (a) and regression (b). In (c) the cost of regression as a factor of the cost for interpolation are depicted. For large k, regression

demands significantly more computational power than interpolation. For arbitrary k the factor is asymptotically $((1/2)k + (1/6))$.

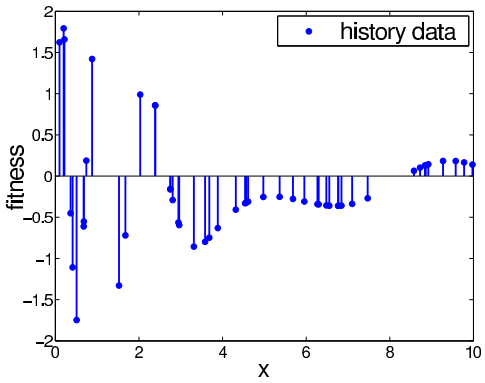


(a) (b) (c)

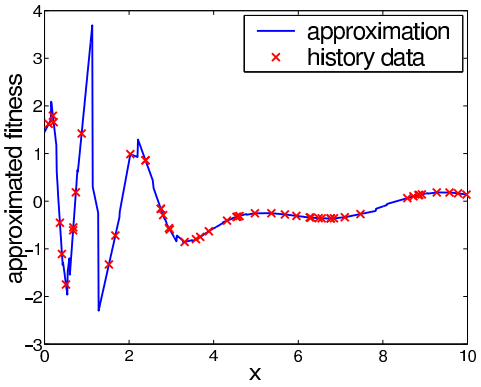
Figure 5.3: Number of required multiplications and additions:
(a) interpolation, (b) regression, (c) comparison: $\frac{\text{cost regression}}{\text{cost interpolation}}$

5.8 Illustration

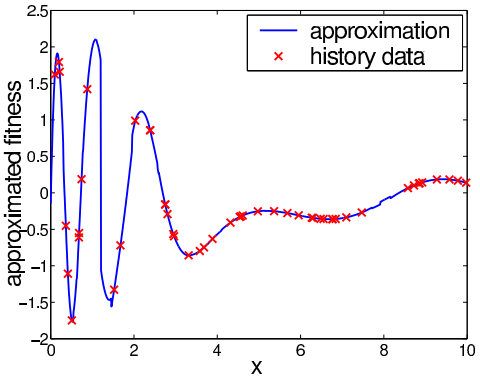
The different techniques produce different fitness surface approximations. A qualitative comparison can be derived from the example in Figure 5.4. The set of available History data is depicted in Figure 5.4(a). Repeated application of the approximation methods produce the fitness surface approximations which are depicted in Figure 5.4 (b-e). In interpolation, all History data points are intersected by the approximation function. The regression approximation is slightly smoother, specifically in the area where the true response surface is very rugged. As smoothening underestimates the variance of a function, we expect interpolation, to work better as f_{var}-estimator. On the other hand, interpolation is more likely to produce extreme outliers and thus is expected to produce a small number of extreme wrong estimations. What effect is dominating will be shown in the simulation studies. Furthermore, we see that both Nearest neighbor interpolation and Local regression produce discontinuous functions. The quadratic models seem to produce



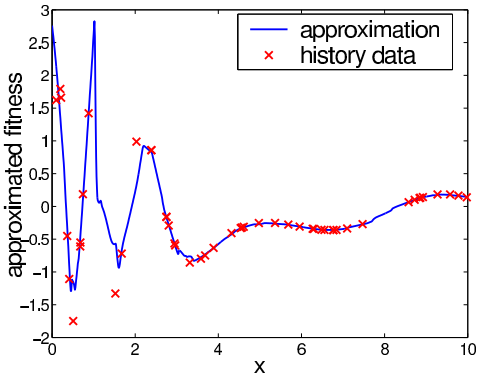
(a) set of history data



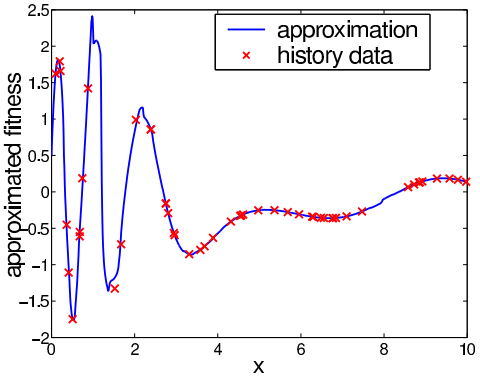
(b) linear interpolation



(c) quadratic interpolation



(d) linear regression



(e) quadratic regression

Figure 5.4: 1-dimensional examples of fitness surface approximations

smoother surfaces than the linear models.

Chapter 6

Robustness estimation

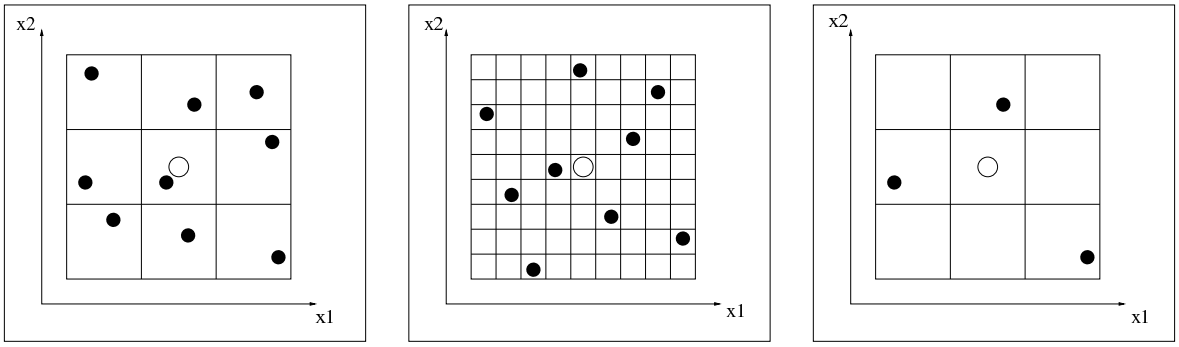
This chapter outlines how the function approximation methods from Chapter 5 are actually used to estimate f_{exp} and f_{var}. Section 6.1 describes how we simulate noise distributions and approximate integrals. Section 6.2 introduces two different approaches to approximation model distribution. In order to improve estimation quality further, we developed a mechanism which detects extremely wrong estimations. See Section 6.3.

6.1 Integral approximation (sampling techniques)

The goal of robustness estimation is to estimate the two integrals in Equation 6.1 most accurately based on approximation methods from Chapter 5.

$$\begin{aligned} f_{exp}(x) &= \int_{-\infty}^{\infty} p(\delta) \cdot f(x+\delta)\, d\delta \\ f_{var}(x) &= \int_{-\infty}^{\infty} p(\delta) \cdot (f(x+\delta) - f_{exp}(x))^2\, d\delta \end{aligned} \tag{6.1}$$

As mentioned earlier, these integrals cannot be calculated analytically, because f is not available in closed form in most applications. Even if f is available in closed form, it is probably not integrable. Instead, we use approximate functions $\widehat{f}$ which are polynomials and thus integrable. Still, even after replacing f by $\widehat{f}$ the integrals are not analytically



(a) (b) (c)

Figure 6.1: Sampling techniques: (a) Stratified sampling with 3 quantiles per dimensions, (b) Latin hypercube sampling with 9 quantiles per dimension, (c) Latin hypercube sampling with 3 quantiles per dimension

solvable for a number of reasons. $p(\delta)$, the density function of the noise distribution might not be integrable. One can argue, that for most density functions sufficiently accurate polynomial approximations are available. However, the second factor $\widehat{f}$ is in some scenarios partially defined and is composed of a large number of polynomials. We therefore estimate the integral of Equation 6.1 by derandomized sampling in the neighborhood taking into account probability distribution $p(\delta)$.

Assume estimation point $\vec{x}$ is 2-dimensional and the random disturbance δ (noise) is uniformly distributed on $[\delta_{min}; \delta_{max}]^2$. We divide the space $[\delta_{min}; \delta_{max}]^2$ into regions of equal probability with respect to the probability function of δ. For uniform distributions, this means equal sizes. In the 2-dim case, we get a grid with squares of equal size. For arbitrary density functions, this grid can be drawn by calculating the respective quantiles of the distribution. We then use this grid to draw representative samples. In particular, we use two different sampling techniques known as *Stratified sampling* and *Latin hypercube sampling*.

In Stratified sampling, we draw one sample from each quantile randomly. A 2-dimensional example is depicted in Figure 6.1(a). Here, the simulation of a uniform

noise distribution around an estimation point which is marked with an unfilled circle is shown. The filled circles depict locations where samples are drawn. The number of quantiles per dimension is 3. One can see that the quantiles grid is well covered by sample points. However, with constant number of quantiles per dimensions, the number of samples increases exponentially with the dimensionality of the problem. Even with small number of quantiles, this method is not acceptable in higher dimensions. We therefore use Latin hypercube (LHC) sampling in higher dimensions. For illustration, see Figure 6.1(b). LHC sampling is the method of choice if we want to sample a high-dimensional space relatively sparsely and is carried out as follows: Again, we divide the space into regions of equal probability (in 2 dimensions, this is a cell grid). Then, we randomly choose one quantile and draw a sample from this. Now, we eliminate all cells that agree with this point in any dimension. In the 2-dimensional case, that is crossing out the respective row and column. Then, we randomly choose one of the remaining cells and repeat this procedure until all cells are eliminated. As a result, we get a set of k samples where k is the number of quantiles per dimension. Note, that this description *illustrates* the idea of LHC. We use an *implementation* with computational complexity of $O(n^2)$, where n is the number of quantiles.

We distinguish between two methods of drawing samples from a given quantile.

1. *randomized sample drawing*: the location of the sample point within the quantile is randomly chosen (uniform distribution)

2. *derandomized sample drawing*: the location of the sample point is the center of the quantile

The sizes of the sample set for the two sampling techniques can be summarized to

$$num_samples(num_quants, dim) = \begin{cases} (num_quants)^{dim} & : \text{Stratified sampling} \\ num_quants & : \text{Latin hypercube sampling} \end{cases} \tag{6.2}$$

where d is the number of dimensions, $num_samples$ is the number of samples and num_quants is the number of quantiles. We see that the number of samples in LHC sampling is constant in the number of dimensions and solely dependent on the number of quantiles. Therefore, LHC sampling represents a means to overcome the difficulties of Stratified sampling which arise from the high computational complexity. The size of the sample set is therefore easily scalable in Latin hypercube sampling. This can either be done by modifying the number of quantiles (example in Figure 6.1 (b),(c)) or by running the sampling procedure multiple times. We refer to the latter parameter as *number of sampling loops*.

The samples are evaluated with an approximation function. The resulting set of fitness approximations is then used to calculate the estimations of f_{exp} and f_{var} empirically. Thus, the integrals (Equation 6.1) are estimated:

$$\begin{aligned} \widehat{f}_{exp} &= \frac{1}{n_s} \sum_{1 \le i \le n_s} \widehat{f}(x^{(i)}) \\ \widehat{f}_{var} &= \frac{1}{n_s} \sum_{1 \le i \le n_s} \left(\widehat{f}(x^{(i)})\right)^2 - \left(\widehat{f}_{exp}\right)^2 \end{aligned} \tag{6.3}$$

Here, n_s is the number of samples which was previously denoted $num_samples$. A sample point is denoted $x^{(i)}$.

An additional sampling parameter needs to be mentioned. Consider a noise distribution which is defined on $[-\infty; +\infty]$, e.g. normal distribution. In this case the first and the last quantile are infinitely large. Of course, in computer representation quantiles have finite size. Still, the quantiles are expected to be very large. Drawing a sample randomly from this quantile, e.g. from the center of this quantile, strongly biases the estimation, especially if the number of quantiles is small. We therefore cut the distribution at the

quantiles α_{cutoff} and $\alpha_{(1-cutoff)}$ where *cutoff* represents an additional sampling parameter. In the multi-dimensional simulation studies of this work we set $cutoff = 0.05$ for the normally distributed noise. Throughout all simulation studies of this work we use a normally distributed noise $N(0, \sigma_{noise})$. As the noise is solely dependend on the parameter σ_{noise}, we subsequently use σ_{noise} as noise indicator.

6.2 Distribution of approximation models

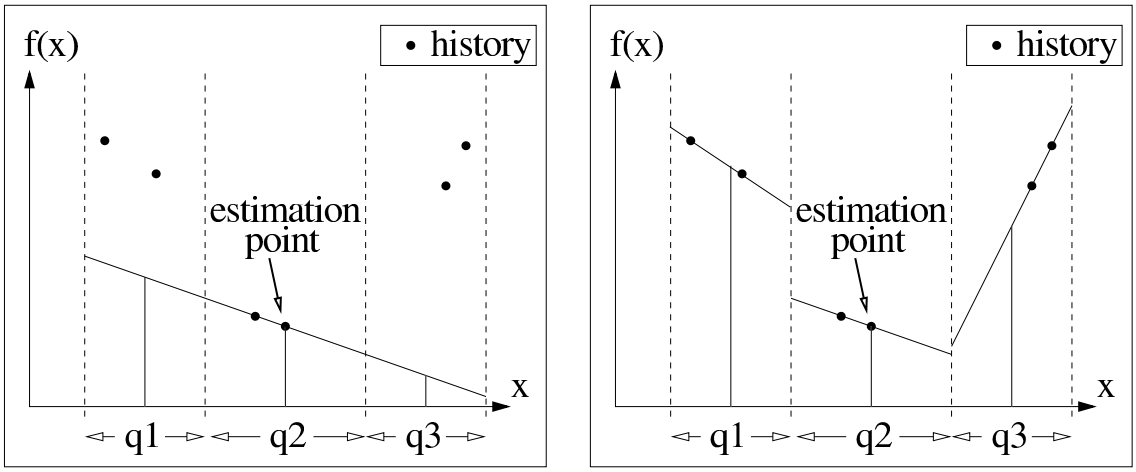
As mentioned earlier, for evaluation of a sample point x we use an approximation of the fitness function. Although the approximation is based on the models as described in Chapter 5, we tested different methods of using approximation models for actually approximating the fitness function at x. In particular, the methods can be categorized into *Individual based model distribution* (IMD) and *Population based model distribution* (PMD).

6.2.1 Individual based model distribution

The basic idea of the individual based model distribution (IMD) approach is to consider all individuals of a population one by one and do the estimation of f_{exp} and f_{var} for each individual independent from the estimations of neighboring individuals. In particular, we distinguish between the two methods *Single model* (IMD-SM) and *Multiple models* (IMD-MM).

IMD-SM works as follows: For each estimation $(\widehat{f}_{exp}, \widehat{f}_{var})$ a single model is built. All samples (compare filled circles in Figure 6.1) of an estimation are evaluated with the approximation function of the single model. Note, that the single model covers the entire noise range[1].

[1] this means all quantiles of the noise distribution



(a) (b)

Figure 6.2: One-dimensional example of model distribution strategies with 3 quantiles using a linear interpolation model: (a) Single model, (b) Multiple models

In IMD-MM, for each sample an own approximation model is built which is only evaluated once at the respective sample point. An illustration for the one-dimensional case is depicted in Figure 6.2. The filled squares represent the History information which is used for approximation. IMD-MM is expected to better approximate the fitness function over the noise range. Of course the IMD-MM is significantly more expensive than IMD-SM. Neglecting the cost of sampling and only taking into account the cost of model computation, the computational cost for IMD-MM is *num_samples* times higher (see Equation 6.2). The drawback of IMD-SM is the often insufficient estimation quality, whereas the major drawback of IMD-MM is the high computational cost. It is worth mentioning that IMD-MM is not expected to perform significantly better if the density of History data is low. Although many models would be built in this case, these models are likely be similar because they are based on the same or similar input data. A drawback of both IMD-SM and IMD-MM arises from the *independent* assignments of models to individuals. If, for example, two individuals are located close to each other, and thus having similar History data in their neighborhood, the models that are to be built for both individuals are likely

to be equal. Computational power is wasted, thus, the IMD methods are inefficient.

6.2.2 Population based model distribution

To overcome the drawback of the IMD methods, "insufficient estimation quality in IMD-SM", "high computational cost in IMD-MM", we investigate another approach. A better method requires to exploit information more efficiently. A source of inefficiency in IMD methods was found in the single use of the models, i.e. a model is expensively built, evaluated for one estimation, and then thrown away. A more efficient method uses an approximation model for multiple estimations. In particular, locations of models are assigned with respect to the entire population (the set of estimation points). This method is named *Population based model distribution* (PMD). The goal of PMD is to find a distribution of approximation model locations which sufficiently covers the population. We split up PMD methods into two steps. In step 1, a model distribution is chosen and the models are built. In step 2, for each individual f_{exp} and f_{var} is estimated by using the available models.

Distributing models in PMD

In this work we use a simple model distribution in step 1: As in IMD-SM, one model is built per individual, having it's fitting point $x^{(0)}$ at the location of the individual. The number of models equals the population size. In contrast to the Single model approach, however, each models is made available to all estimations.

Use of models for estimations in PMD

Generally, a more accurate approximation at sample point x is expected if the nearest available model chosen. In particular, when sampling in the neighborhood of an individual (LHC or Stratified sampling) that model is chosen for evaluation which has the shortest

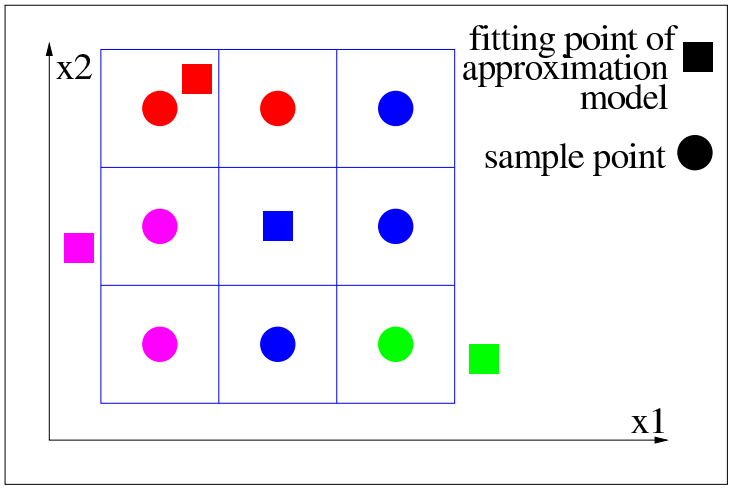


Figure 6.3: Nearest model: Using model with shortest Euclidean distance. The squares represent the fitting points of the available models, these are the locations of the individuals of the current generation. The circles represent sample points for the blue individual. The color of the circles shows which model is used for evaluation.

Euclidean distance between it's fitting point and the sample point. We refer to this technique as *Nearest model*. For illustration, see Figure 6.3.

An extension of the *Nearest model* approach is to build ensembles of models (PMD-ENS). Instead of only using the nearest model for evaluation, an ensemble of the k nearest models is used for evaluation. An ensemble of approximation models ($\widehat{f}_{ens}$) is defined as the weighted sum of approximation models (see Equation 6.4):

$$\begin{aligned} \widehat{f}_{ens}(x^{(0)}) &= \frac{1}{\sum_{1\leq i\leq k} w_i} \sum_{1\leq i\leq k} w_i \, \widehat{f}_i(x^{(0)}) \\ w(\widehat{f}_i, x^{(0)})) &= \frac{1}{\|fittingpoint(\widehat{f}_i) - x^{(0)}\|_2} \end{aligned} \tag{6.4}$$

Here, $x^{(0)}$ denotes the sample point, k is the number of models in the ensemble (*ensemble size*), w is a weighting function which assigns weights to models. The weighting function of Equation 6.4 which is used throughout the simulation studies is a monotonously decreasing function in the Euclidean distance between sample point $\vec{x}$ and the fitting point of $\widehat{f}_i$.

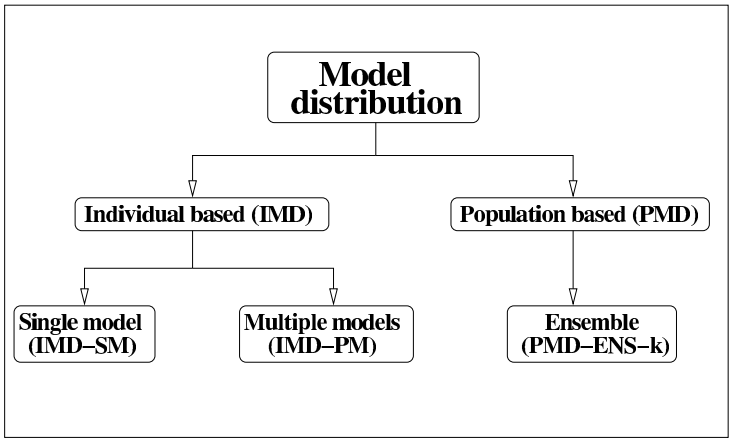


Figure 6.4: Overview of model distribution methods

Technically, the nearest model method is a special case of the ensemble method with ensemble size 1. We therefore refer to both approaches as PMD-ENS-k where k is the ensemble size.

6.2.3 Overview

Figure 6.4 summarizes the three different model distribution methods. The Single model method (IMD-SM) and Multiple models method (IMD-MM) fall into the category of individual based model distribution, the Ensemble method (PMD-ENS) falls into the category of population based model distribution.

6.3 Extreme outliers

6.3.1 Estimation bias

In preliminary experiments, we found that estimations of f_{exp} and f_{var} are sometimes strongly biased by a small number of seriously over- or under- approximated fitness values in the sampling procedure. We refer to these fitness values as *outliers*. Outliers appear

specifically in interpolation, for illustration see Figure 6.5.

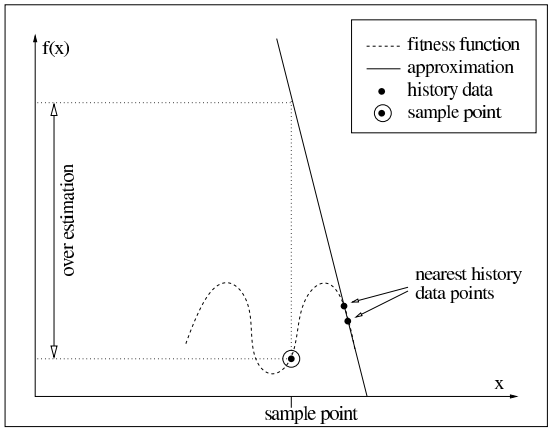


Figure 6.5: Outlier caused by an unfavorable History data distribution and large fitness function gradient

Although the existence of outliers is not due to numerical difficulties but has conceptual reasons, we detect *extreme* outliers and replace them by more reliable estimations. Defining *extreme* requires to set thresholds. Setting a threshold for outliers detection and replacement itself introduces a new bias. Therefore the goal of such a method must be to reduce the bias caused by extreme outliers and at the same time keep the bias which is introduced by this routine at a low level. We use the well known method *boxplotting* with some modifications.

6.3.2 Boxplot

Details on boxplotting can be found in any standard statistics text book, therefore we introduce boxplotting *in brief*. For illustration have a look at Figure 6.6. In boxplotting, outliers of a set X of samples $x^{(i)}$ are detected as follows: After sorting the numbers, a lower quantile (q_{left}) and an upper quantile (q_{right}) are determined. Often q_{left} is the (0.25)-quantile and q_{right} is the (0.75)-quantile. The *interquartile range (IQR)* is defined as $(q_{right} - q_{left})$. An $x^{(i)}$ is defined as outlier, if it it does not lie within the *acceptance*

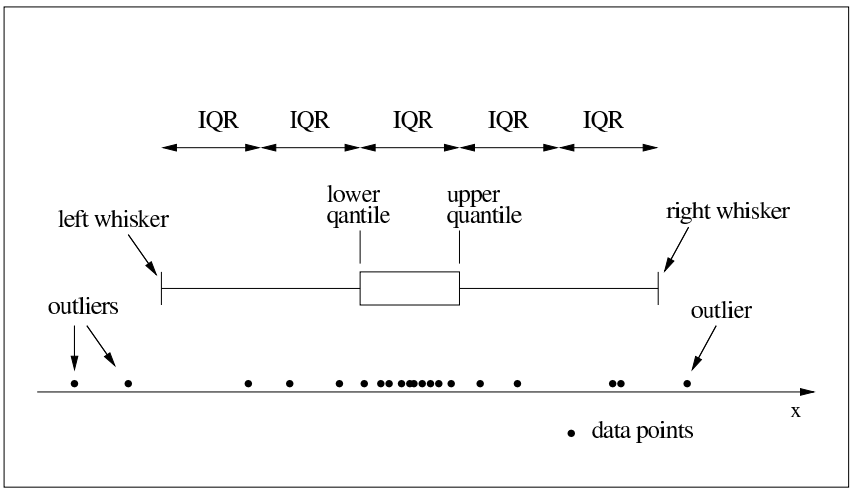


Figure 6.6: Boxplot

interval:

$$x_i \notin [\ q_{left} - k \cdot IQR\ ;\ q_{right} + k \cdot IQR\ ] \tag{6.5}$$

Here k is a factor which needs to be set by the user. We define factor k as *whisker factor*. In Figure 6.6 *whisker factor* is 2.

6.3.3 Outliers detection routine

In classical boxplot applications, data which are detected as outliers are from the same data source as the data which are actually used to compute the *acceptance interval*. In our application, however, this is different: As input data to the boxplot (for computing the acceptance interval), we use all History data. Recall that the History data are known points of the fitness surface. We then use the resulting acceptance interval to decide if the approximated fitness value of a sample point is treated as outlier. To summarize, boxplot input data are History data, potential outliers are approximations. However, this modification requires a careful setting of the boxplot parameters *whisker factor*, *lower quantile* (q_{left}), *upper quantile* (q_{right}). It is required that the parameters have a setting such that no History data would be treated as outliers, because we already know that such

fitness values are feasible and *no* outliers. This could for example be ensured by using the 0.0-quantile as q_{left} and the 1.0-quantile as q_{right}. In any scenario, *whisker factor* must be larger 0.0. In the case of *whiskerfactor* ≤ 0.0 any fitness improvement (even if the approximation has no approximation error) would be treated as outlier. Let us assume *whisker factor* is set to 3, q_{left} is the 0.0-quantile and q_{right} is the 1.0-quantile: In the initial stages of the EA, a *small number* of individuals with *very high* fitness values[2] are likely to appear, after some generations "interesting" regions (with low fitness values) are discovered. If the fitness function has a large range of function values, this results in a large IQR. With *whisker factor* $= 3$, even serious under estimations would not be treated as outliers now. Thus, IQR must be reduced: A standard setting of IQR in many applications is to use the 0.25-quantile as q_{left} and the 0.75-quantile as q_{right}. In the context of an EA a problem arises from such a setting: When the algorithm converges, many similar data are added to the History. IQR shrinks to a small size which causes the acceptance interval to shrink, too. Although the EA has converged, in the sampling procedure some approximations might wrongly be treated as outliers.

A "effective" parameter setting must manage the trade-off between the drawback caused by convergence and the drawback caused by the appearance of high fitness values in the initial stages of the EA. We found that the following works good in this sense: We set *whisker factor* $= 5.0$, q_{left} is the 0.01-quantile, q_{right} is the 0.99-quantile.

6.3.4 Replacing outliers

Once an outlier has been detected, one needs to take further actions in order to proceed with the estimation procedure. Generally, two methods are possible:

1. discard the outlier from the sample set

2. replace the outlier by a more reliable approximation

[2]assuming a minimization problem

Method 1 is cheap and straight-forward. Method 2 requires to compute a more reliable approximation and is therefore more expensive. Although computational cost is one of the major concerns of this work, we decide to use method 2 but use an approximation with low additional cost. If an outlier is detected, we replace the fitness value by the *average fitness* of the current population which (at least in later stages of the EA) is a sufficiently reliable approximation. As the same replacement approximation is chosen for all outliers of the current population we expect the proposed method to work similar to Method 1.

6.3.5 Additional remarks

The target of an outliers detection and replacement mechanism is to reduce the effect of serious outliers. In fact, when switching off this mechanism, the evolutionary algorithm does not manage to converge in many cases, when using interpolation. However, an inappropriate setting of the parameters might introduce a new bias. As one goal of this work is to compare different approximation techniques (including their conceptual difficulties) we chose a parameter setting such that only extreme outliers are detected. In fact, the setting of boxplot parameters has a significant effect on the EA performance when using interpolation. In the simulation studies for SO robustness (see Chapter 8) we found that a setting such that a *larger share* of approximations is identified as outliers has a positive effect on the EA performance. However, this effect is problem specific. We therefore decided to use above described *conservative* parameter setting.

Chapter 7

Preliminary 1-d simulation studies

7.1 Introduction and preceding remarks

This chapter reports about a number of preliminary experiments which were carried out for the special case of 1-dimensional problems. The goal of these experiments was to gather a deeper understanding for the problem at hand without spending too much time on implementation. Another reason for choosing the 1-dimensional case as starting point for further steps was that this allowes us to test a larger range of approximation techniques: In addition to Nearest neighbor interpolation and Local regression, we analyzed *Natural neighbor interpolation*. This technique is introduced in Section 7.2. However, when setting up the 1-dimensional simulation studies, we had not implemented *Local quadratic regression*. Table 7.1 summarizes the methods which were analyzed in 3 experiments. Note that in the 1-dimensional case, we do not need to distinguish between *Stratified* and *Latin hypercube* sampling. For a description of the approximation methods (except Natural neighbor interpolation), we refer to Chapter 5.

The test functions of the simulation studies are presented in Section 7.3. Experiment 1 (Section 7.4) analyzes the quality of different approximation models as estimator for f_{exp}

Table 7.1: Tested methods in 1-dim simulation studies

	Single model			*Multiple models*		
	Nearest neighbor interpolation	*Natural neighbor interpolation*	*Local regression*	*Nearest neighbor interpolation*	*Natural neighbor interpolation*	*Local regression*
linear	yes	yes	yes	yes	yes	yes
quadratic	yes	yes	no	yes	yes	no

and f_{var} for *different noise levels* in a *static environment*. Experiment 2 (Section 7.5) analyzed the estimator quality for f_{exp}, f_{var}, too. Here, the measures are taken over a range of fitness landscapes with different variances. Finally, Experiment 3 (Section 7.6), tests the performance of the methods in an EA environment.

Following remarks are worth mentioning: After gaining experience in multiple dimension simulation studies, we found that some results of the 1-dimensional simulation studies are not transferable to the multi-dimensional case. In particular, we found that the experimental setup is not representative for higher dimensions. However, some findings are worth mentioning. In this report on the 1-d-simulation studies, we therefore only briefly report on some spotlights of the experiments. A critical discussion on the findings is presented in Section 7.7.

7.2 Natural neighbor interpolation

The difference between *Nearest* neighbor and *Natural* neighbor interpolation is the choice of input data points to the model. In the case of linear 1-dimensional *Natural* neighbor interpolation not necessarily the nearest data points but the left and right neighbor with regard to model fitting point. The resulting interpolation is a "real" interpolation. In contrast, if two data points from the same side are chosen the approximation point is

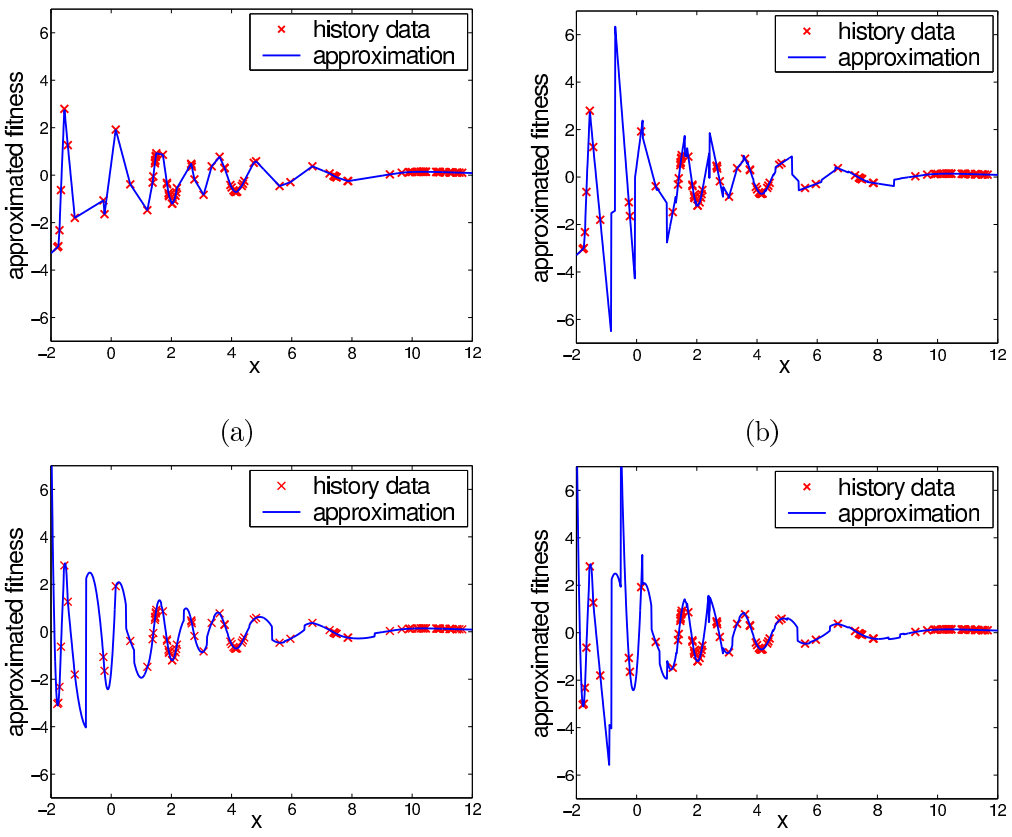


Figure 7.1: Interpolation (a) linear Natural neighbors, (b) linear Nearest neighbors, (c) quadratic Natural neighbors, (d) quadratic Nearest neighbors

extrapolated. The essential difference is that *Natural* neighbor produces a continuous response surface approximation. For illustration, see Figures 7.1(a)(b).

The multi-dimensional equivalent to Natural neighbor interpolation is known as *Delaunay triangulation*. In multiple dimensions the resulting surface approximation is continuous, too. Another property of this triangulation method is that the resulting triangles are relatively equal, that is, the variance of the edge lengths is low. For further information we refer to [33] and [34]. Unfortunately the time to compute a Delaunay triangulation in dimension n is of the order $m^{\lceil \frac{n}{2} \rceil}$, where m is the number of sample points. In the n-dimensional space, Delaunay triangulation constructs a set of n-dimensional hyper

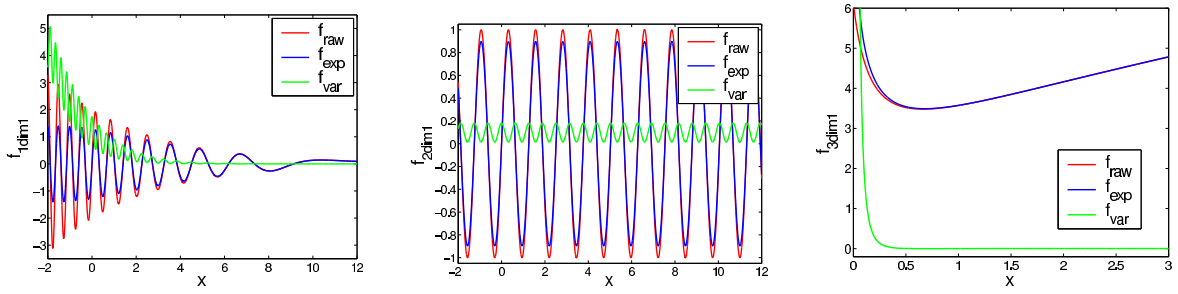
planes. In this form Natural neighbor interpolation can only be used for *linear* interpolation. However, we extended the Natural neighbor interpolation technique for quadratic polynomials: Again, the only difference to quadratic Nearest neighbor interpolation is the choice of input data to the model. If the dimensionality is n, the first $n+1$ data points are chosen equivalently to linear Natural neighbor interpolation. As additional $\frac{n(n+1)}{2}$ data points the nearest remaining data points are chosen. With this extension, the resulting surface approximation is not continuous anymore. Still, the Natural neighbor interpolation is smoother than the Nearest neighbor interpolation. For illustration see Figures 7.1(c)(d).

7.3 Test functions

In Experiments 1-3 we used the following test functions:

$$\begin{aligned} f_{1dim1}(x) &= 2.0\,\sin(\,10\,\exp(-0.08x)\,x\,)\,\exp(-0.25x) \\ f_{2dim1}(x) &= \sin(\,\delta x\,) \\ f_{3dim1}(x) &= \frac{1}{(x+0.2)} + 2.5\,\sqrt{x+0.2} \end{aligned} \tag{7.1}$$

Figure 7.2 depicts f_{raw}, f_{exp} and f_{var} for the three test functions with $\sigma_{noise} = 0.1$. For illustration of f_{2dim1} we set $\delta = 5$.



(a) f_{1dim1} (b) f_{2dim1} (c) f_{3dim1}

Figure 7.2: Test problems for the 1-dim simulation studies with $\sigma_{noise} = 0.1$

7.4 Experiment 1

In Experiment 1, we analyzed the quality of the ten proposed methods (see Table 7.1) as estimator for f_{exp} and f_{var} in a static environment. Particularly, we measure the estimation accuracy for different noise levels σ_{noise}. Here σ_{noise} is the standard deviation of the $N(\mu_{noise}, \sigma_{noise})$ normally distributed noise.

7.4.1 Experimental setup

We randomly generate a set of $n = 100$ History data points $x_i \in [-2; 12]$ based on test function f_{1dim1}. We then compute the true $f_{exp}(x_i)$ and $f_{var}(x_i)$ for all $i = 1 \dots n$ based on the known test function[1]. Here, we use Stratified sampling with 50 quantiles to approximate the integral (for details see Section 6.1). We then compute the approximations $\widehat{f}_{exp}(x_i)$,$\widehat{f}_{var}(x_i)$ for all $i = 1 \dots n$ based on the History data. Again we use 50 quantiles for sampling. For each i we compute the estimation errors $e_{f_{exp}}$ and $e_{f_{var}}$. For convenience we refer to these as e_{exp} and e_{var} (see Equation 7.2):

$$\begin{aligned} e_{exp}(x_i) &= \widehat{f}_{exp}(x_i) - f_{exp}(x_i) \\ e_{var}(x_i) &= \widehat{f}_{var}(x_i) - f_{var}(x_i) \end{aligned} \tag{7.2}$$

By running the experiment 50 times we get empirical distributions e_{exp}, e_{var} for each approximation method we get 10 distributions e_{exp} and e_{var} of 5000 data points for each setting. These distributions are the basis of our analysis. We evaluate mean μ and standard deviation σ for all distributions and get four accuracy measures $\mu_{e_{exp}}$, $\sigma_{e_{exp}}$, $\mu_{e_{var}}$, $\sigma_{e_{var}}$.

σ_{noise} is varied over the domain $[0.0; 1.0]$. Throughout all 1-dimensional experiments we cut the normal distribution to a finite domain by setting the parameter $cutoff = 0.02$ (see Section 6.1).

[1] note that we use the terms "true fitness" and "real fitness" synonymously

7.4.2 Brief remarks on estimator properties

In general, both mean μ_e and standard deviation σ_e of the estimation error are important quality criteria for an estimator. However, the relative importance of both varies from application to application. After we had gained experience in multi-dimensional simulation studies we were able to demonstrate that in the context of evolutionary algorithms the σ_e is the important criterion[2]. Nevertheless, we report on both μ_e and σ_e, because this chapter represents a starting point of the analysis.

7.4.3 Results

As expected, the Multiple models approach performs significantly better. Considering the single objective f_{exp}, we see that the mean of the estimation error $\mu_{e_{exp}}$ is closer to zero for the multiple model methods (Figures 7.3(a)(b)). We also see that the standard deviation of the estimation error is closer to zero for the multiple model methods. Both findings hold specifically for larger σ_{noise}(Figures 7.3(c)(d)). The reasoning is straightforward: If σ_{noise} is small, the number of History data points within the σ_{noise}-range is small. Although the Multiple models method computes an own model at each sample point, this does not improve the estimation because the models are likely to be equal. We will later see that application of the Multiple models method in this form is restricted to low-dimensional problems. Therefore some attention should be spent on the Single model methods. In Figure 7.3(b), we see that in the Single model application, the quadratic interpolation methods perform relatively well for $\sigma_{noise} < 0.2$. However, for larger σ_{noise} the linear interpolation methods are more stable.

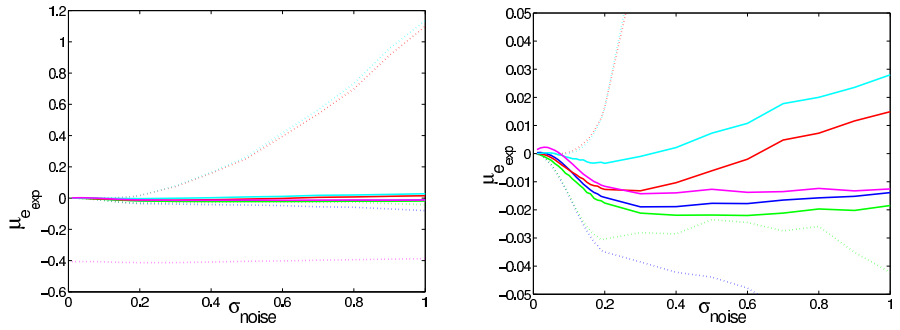
Considering the Multiple models approach, we see that for large σ_{noise} the linear methods outperform the quadratic methods (Figure 7.3(d)).

Surprisingly, linear regression performs best for large σ_{noise} (Figure 7.3(d)). The rea-

[2]this issue will extensively be discussed in Chapter 8

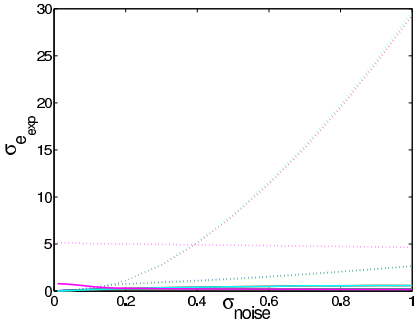
son is easily found. A larger σ_{noise} smoothens f_{exp}. Thus the drawback of regression (smoothening of the landscape) is less severe. A negative aspect of the smoothening effect of regression is the consistent underestimations of the variance. Figure 7.3(e) shows that the extent of underestimation is increasing with increasing σ_{noise}. However, at a certain level, here $\sigma_{noise} = 0.2$, the underestimation remains constant. This can be attributed to the problem specific smoothening of f_{var} for large σ_{noise}.

Comparing Natural and Nearest neighbor interpolation we do not find a significant performance difference for the estimation of f_{exp} (Figure 7.3(a)-(d)). As expected, we find that linear Natural neighbor interpolation tends to underestimate the variance, whereas Nearest neighbor interpolation slightly overestimates the variance (Figure 7.3(e)). Considering $\sigma_{e_{var}}$ (Figure 7.3(f)), we find that Natural neighbor interpolation performs slightly better.

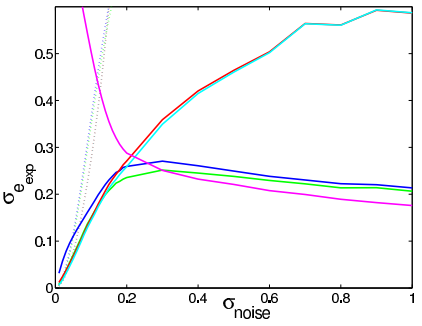


(a) $\mu_{e_{exp}}$

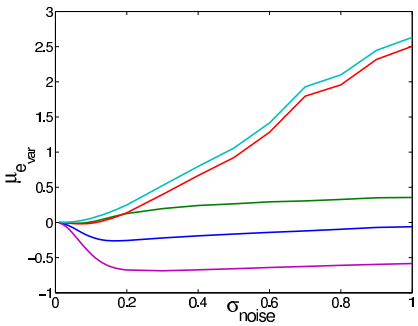
(b) $\mu_{e_{exp}}$



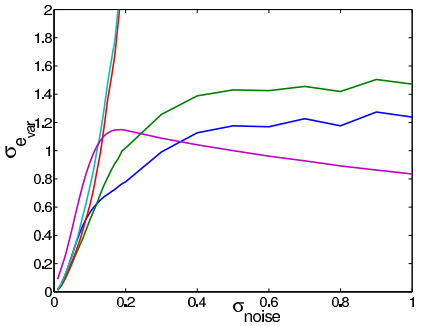
(c) $\sigma_{e_{exp}}$



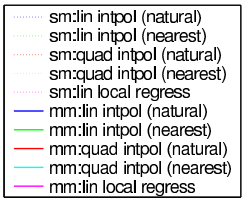
(d) $\sigma_{e_{exp}}$



(e) $\mu_{e_{var}}$



(f) $\sigma_{e_{var}}$



(g) legend

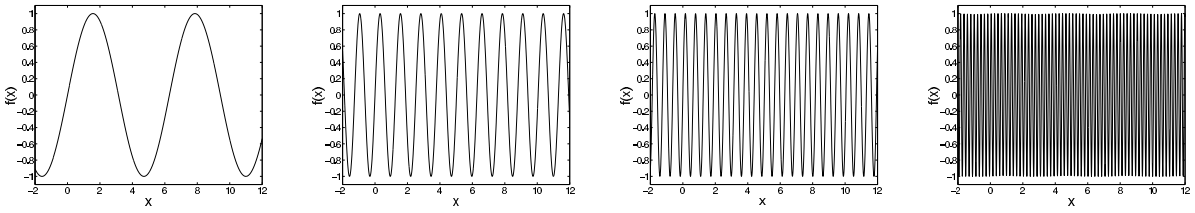
Figure 7.3: Performance of estimation methods. 'sm'- Single model, 'mm'- Multiple models. (b) zooms into (a), (d) zooms into (c)

7.5 Experiment 2

Experiment 2 aims at evaluating the quality of the ten proposed methods as estimator for f_{exp} and f_{var} in fitness landscapes with different variances. Thus question to answer by this experiment is *"How do the estimators perform if the fitness landscape is smooth, and how do they perform if it is very rugged?"*

7.5.1 Experimental Setup

The setup of Experiment 2 is identical to Experiment 1, except the following parameters: We set $\sigma_{noise} = 0.1$ constant for all runs. As test function we use f_{1d2} and run the experiment for $\delta \in [1; 30]$ with stepsize 1. Figure 7.4 illustrates the effects of increasing δ.



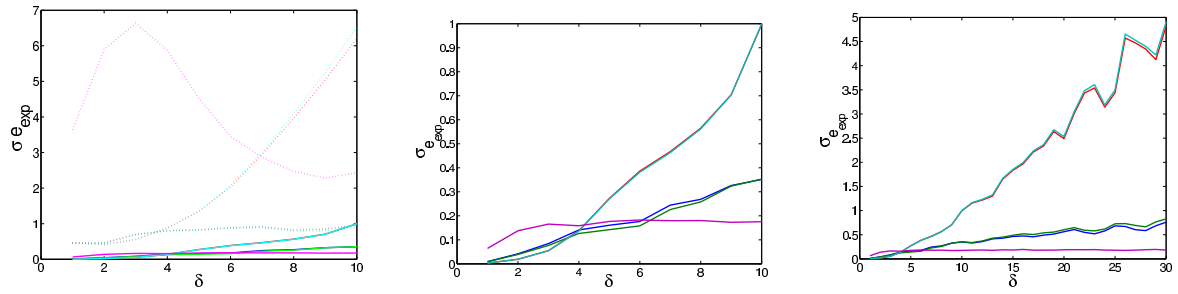
(a) $\delta = 1$ (b) $\delta = 5$ (c) $\delta = 10$ (d) $\delta = 30$

Figure 7.4: $f_{1d2} = \sin(\delta x)$, for different δ

7.5.2 Results

The effect of increasing δ in experiment 2 is similar to the effect of increasing σ_{noise} in experiment 1. We therefore do not repeat a detailed discussion on the results. However, one spotlight should be taken from the results. Figure 7.5 depicts the standard deviation of the estimation error for f_{exp}. Again, Multiple models performs significantly better than Single model (a). For low δ, linear and quadratic interpolation perform similar and both better than linear Local regression. For larger δ, linear interpolation performs better than quadratic interpolation. The reason is that in very rugged landscapes quadratic models

are suspect to extreme outliers. Surprisingly, Local linear regression performs best for large δ, because one would expect that regression works best on smooth surfaces. This is of course true. Evidence is provided by Figure (b) where we see that the performance of Local regression degrades with increasing δ on $\delta \in [1;5]$. However, Local regression performs only relatively better than interpolation, because it does not produce severely wrong estimations.



(a) $\delta = 1$ (b) $\delta = 5$ (c) $\delta = 30$

Figure 7.5: $\sigma_{e_{exp}}$: (a) all methods on $\delta \in [1;10]$, (b) Multiple models on $\delta \in [1;10]$, (c) Multiple models $\delta \in [1;30]$, coloring as in Figure 7.3

7.6 Experiment 3

Experiment 3 aims at evaluating the quality of the ten proposed methods in the framework of an evolutionary algorithm. Experiments 1 and 2 analyzed the quality of f_{exp}- and f_{var} estimators in a static environment. However, it is of essential interest for our work, how the estimations actually guide the evolutionary search. Questions to be addressed are concerned with convergence rates, whether the approximation finds the true optimum or whether the approximation methods guides the search similarly as the true fitness function would do. We treat robustness as single objective optimization problem, i.e. we use an SOEA.

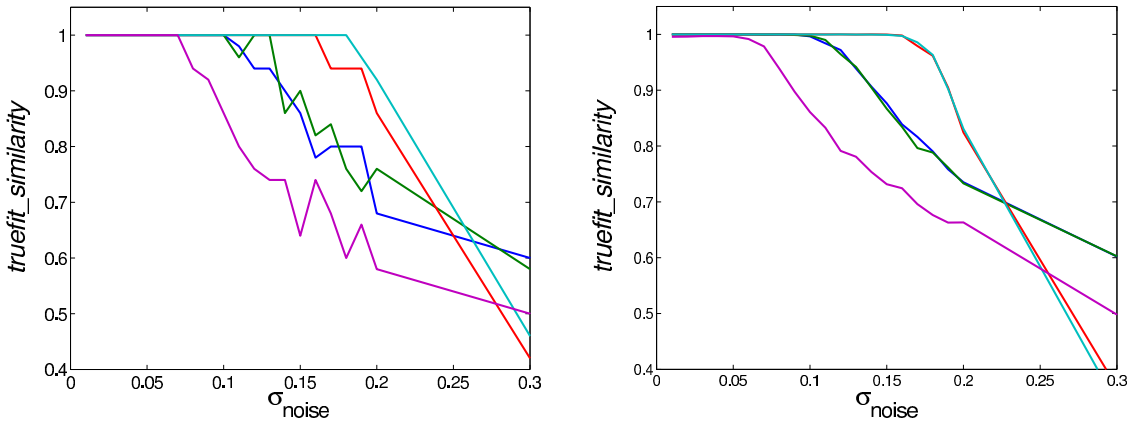
7.6.1 Experimental setup

For Experiment 3, we use a modification of Algorithm 3 in Chapter 4. Instead of the standard evolution strategy we used *covariance matrix adaptation* (*cma*)[3]. All other genetic operators and EA parameters are set as described in Section 4.3. Constraint handling is done as described in Section 4.5. The sampling procedure is configured as in Experiments 1 and 2. As test problems we used functions f_{1dim1} on $x \in [-2; 12]$ and f_{3dim1} on $x \in [0; 3]$ for different $\sigma_{noise} \in [0; 1]$. For each of the 10 approximation methods we run the algorithm 50 times with different random seeds. Additionally we run the algorithm with the same random seed but use the true fitness function instead of the approximations. As benchmark criterion we define *truefit_similarity*: In each generation of each run we measure if the best individual (of the approximation run) is located close the best individual of the true fitness run. As maximal distance we set a threshold of 0.05. *truefit_similarity* is the relative share of individuals which are close to true fit run individuals.

7.6.2 Results

Figure 7.6 shows the *truefit_similarity* performance for both test functions when employing the Multiple models method. We see that the performance of the approximation methods is consistent. For low σ_{noise} all approximation methods perform as if the true fitness function is used for f_{exp} estimation. For larger σ_{noise} the quadratic interpolation methods perform better than the linear methods. Linear Local regression performs worst. Again, we see that there is no significant difference between Nearest neighbor and Natural neighbor interpolation for both linear and quadratic models.

[3]for details we refer to [26]



(a) f_{1dim1} (b) f_{3dim1}

Figure 7.6: Performance of Multiple models in experiment 3: *truefit_similarity* for test function f_{1dim1} and f_{3dim1}, coloring as in Figure 7.3

7.7 Discussion

The results of the three experiments are consistent in the following points: There is no significant performance difference between Nearest neighbor and Natural neighbor interpolation methods. For low σ_{noise}, quadratic interpolation performs better than linear interpolation, on larger σ_{noise}, however, linear interpolation methods perform better. We conclude that with increasing σ_{noise}, the probability of significantly wrong estimations (outliers) increases. Quadratic methods seem to be more suspect to such outliers. This reasoning is supported by the fact that linear Local regression works comparably well on large σ_{noise}. Here we assume regression to be less suspect to significantly wrong estimations. We found supporting evidence for a consistent f_{var} underestimation of linear regression and Natural neighbor interpolation. An expected results was the better performance of the Multiple models method compared to the Single model method.

With the experience gained in multi-dimensional simulation studies, some critical points concerning the experimental set up need to be mentioned "a posteriori". In Exper-

iment 1 we used a History data base of size 100. For the 1-dimensional case this means the Euclidean distance between two neighboring data points is only $\frac{x_{max}-x_{min}}{100} = 0.14$ on average. A 2-dimensional setting with this property already requires $100^2 = 10000$ History data points, in n dimensions 100^n data points are required. Although this comparison is somewhat far-fetched, we see that the transfer of the findings to multi-dimensional is questionable. For the multi-dimensional case, it is specifically interesting to see how the methods perform in regions with low History data density. Another difficulty arises from the computational overhead in multiple dimensions. The number of models to be calculated in the Multiple models method increases exponentially in the number of dimensions when Stratified model distribution is applied (for details we refer to Section 6.1), i.e. even for a small number of quantiles this is not acceptable for dimensions larger than 5.

After all, the simulation studies served two goals: First, it brought difficulties to light which we were better able to treat in the multi-dimensional simulation studies. Secondly, the results enabled us to reduce the number of methods to be tested in multiple dimensions. Specifically, we discarded Natural neighbor interpolation. On the one hand, because it did not improve the performance compared to Nearest neighbor interpolation, on the other hand because the computational cost to perform Natural neighbor interpolation in multiple dimensions is not acceptable. We conclude that linear Local regression is less suspect to significant outliers. The drawback of linear Local regression is that it strongly smoothens the surface. Therefore we will additionally implement quadratic Local regression for the multi-dimensional simulation studies which might overcome this drawback.

Chapter 8

SO Simulation studies

This chapter is divided into three sections. Section 8.1 provides a discussion on estimator properties in the context of an evolutionary algorithm and lays the theoretical foundations for an analysis of the empirical results which are presented in Section 8.3. The experimental setup is outlined in Section 8.2.

8.1 Theoretical analysis of estimator properties

Subsection 8.1.1 analyzes the influence of estimation error properties on the evolutionary search and concludes that the standard deviation of the estimation error is the essential property for guidance of the EA. Subsection 8.1.2, then takes a closer look on what determines the standard deviation of the estimation error in our application.

8.1.1 Estimation error properties in an EA

An estimation $\widehat{f}_{exp}$ of the *expected fitness* f_{exp} introduces an error term $e_{\widehat{f}_{exp}}$, to which we refer as e_{exp}. e_{exp} is defined

$$e_{exp} = \widehat{f}_{exp} - f_{exp} \quad .$$

We assume that for a given approximation method, e_{exp} has a probability distribution. Generally, whenever fitness estimations f_{est} are used instead of the real fitness, the standard deviation $\sigma_{e_{est}}$ of the error distribution e_{est} is the essential parameter which guides the search of an EA. This is illustrated by the following example: Assume an estimation method to have an estimation error distribution with expectation $\mu_{e_{est}}$ and standard deviation $\sigma_{e_{est}} = 0$. In this case, the estimation *constantly* over- or underestimates the real fitness. Thus, we get

$$f_{est} = f + \mu_{e_{est}} \quad .$$

Assume, we employ a (μ, λ)-reproduction scheme. As the *rank* of the fitness values determines, who survives to the next generation, a constant fitness increase for all individuals has no effect on the composition of the next parent generation. The effect of such an estimation method on the course of the run, solely depends on the selection strategy. With uniform or rank based selection, the fitness differences have no influence on the selection of individuals. In this case, the evolutionary algorithm performs as if the real fitness is used instead of estimations. If fitness proportional selection is chosen, a constant overestimation smoothens the fitness differences between individuals. However, we treat the influence of $\mu_{e_{est}}$ as negligible [1].

To summarize: If $\sigma_{e_{est}} = 0$, the algorithm performs as if the real fitness function is used. An estimator is not required to be unbiased, i.e. we do not require

$$E(\widehat{f}(x)) \; = \; f(x) \quad .$$

Instead, it is sufficient if the estimator has the property

$$E(\widehat{f}(x)) \; = \; f(x) \; + \; c \quad ,$$

where c is an arbitrary constant. We conclude that the reduction of $\sigma_{e_{est}} = 0$ improves the search.

[1] this discussion also shows that rank based or uniform selection are the preferred methods for our purposes

With rank-based selection we do not even require c to be a constant. An estimator is expected to perform as if the real fitness is used if the following statement is true:

$$\widehat{f}(x^{(1)}) \ < \ \widehat{f}(x^{(2)}) \Leftrightarrow f(x^{(1)}) \ < \ f(x^{(2)}) \quad .$$

However, the latter property is difficult to measure. We, thus, concentrate on the minimization of the standard deviation.

8.1.2 Determinants of the estimation error's standard deviation

The terminology is defined in Table 8.1. The estimation $\widehat{f}_{exp}(x^{(0)})$ of $f_{exp}(x^{(0)})$ is calculated

Table 8.1: Definition of terms

Term	*Meaning*
$\widehat{f}_{exp}$	estimation function for the expected fitness
f_{exp}	real expected fitness
e_{exp}	estimation error $\widehat{f}_{exp} - f_{exp}$
$\mu_{e_{exp}}$	expectation of the e_{exp} distribution
$\sigma_{e_{exp}}$	standard deviation of the e_{exp} distribution

by approximating the fitness function at n_s sample points $x^{(i)}$, $1 \leq i \leq n_s$ and taking the empirical mean over the $\widehat{f}(x^{(i)})$ (see Chapter 6). $\widehat{f}(x^{(i)})$ itself is an approximation which has an error distribution $e_{\widehat{f}}$. Analogous to the notation used so far, $\mu_{e_{\widehat{f}}}$ and $\sigma_{e_{\widehat{f}}}$ denote mean and standard deviation of the error distribution. For the sake of illustration, we assume $e_{\widehat{f}}$ to be normally distributed[2].

$$e_{\widehat{f}} \sim N(\mu_{e_{\widehat{f}}}, \sigma_{e_{\widehat{f}}})$$

[2]without loss of generality, because for sufficiently large n_s arbitrary distributions can be used here

Unfortunately, the $e_{\hat{f}}$ distribution is not explicitly given but it is implicitly defined by a number of problem properties, including

- the fitness function,
- the approximation method,
- the sampling technique,
- the distribution of History (surface) points .

The error distribution of the mean e_{exp} can be calculated by applying a convolution of n_s error distributions $e_{\hat{f}_i}$. The resulting distribution must then be transformed by multiplying it with $\frac{1}{n_s}$.

$$e_{exp} = \frac{1}{n_s} e_{\hat{f}_1} + \ldots + \frac{1}{n_s} e_{\hat{f}_{n_s}} = \frac{1}{n_s} \sum_{1 \leq i \leq n_s} e_{\hat{f}_i} \tag{8.1}$$

Note that in equation 8.1 the '+'- symbol represents the binary convolution operator. The '$\sum$'- symbol represents the convolution operator for multiple distributions.
For now, we assume the $e_{\hat{f}_i}$ to be statistically independent. With this assumption, $\mu_{e_{exp}}$ and $\sigma_{e_{exp}}$ are derived.

$$\begin{aligned} \mu_{e_{exp}} &= \frac{1}{n_s} n_s \, \mu_{e_{\hat{f}}} = \mu_{e_{\hat{f}}} \\ \sigma_{e_{exp}} &= \sqrt{(\frac{1}{n_s})^2 \, n_s \, {\sigma_{e_{\hat{f}}}}^2} = \frac{\sigma_{e_{\hat{f}}}}{\sqrt{n_s}} \end{aligned} \tag{8.2}$$

A convolution of normal distributions returns a normal distribution[3]. We get the estimation error distribution of the expected fitness:

$$e_{exp} \sim N(\mu_{e_{exp}}, \sigma_{e_{exp}}) \sim N(\mu_{e_{\hat{f}}}, \frac{\sigma_{e_{\hat{f}}}}{\sqrt{n_s}})$$

[3]a convolution of arbitrary equal distributions can be approximated by a normal distribution for sufficiently large n_s

If we want to compare different approximation techniques with respect to their performance in an EA environment, we need to compare $\frac{\sigma_{e_{\hat{f}}}}{\sqrt{n_s}}$ or just $\sigma_{e_{\hat{f}}}$, if the number of samples n_s is constant for all estimations.

However, the assumption of statistically independent $e_{\hat{f}_i}$ might not be realistic. If we apply the Single model method (IMD-SM), all samples are evaluated with the same model. Thus, the estimation errors are expected to be correlated. But also with Multiple models (IMD-MM) independence is not guaranteed. Consider the case when only a small set of History data is available in the neighborhood of $x^{(0)}$. Although at each sample point $x^{(i)}$ a separate model is built, these models are likely to be identical. We expect the error distributions to be correlated. Thus, $\sigma_{e_{exp}}$ of Equation 8.2 is increased by the sum of the covariances between all models and we get

$$\sigma_{e_{exp}} = \frac{1}{n_s} \sqrt{\Big(\sum_{1 \le i \le n_s} \sigma_{e_{\hat{f}i}} + 2 \sum_{1 \le i < j \le n_s} cov(e_{\hat{f}_i}, e_{\hat{f}_j}) \Big)} \quad . \tag{8.3}$$

8.2 Experimental setup

8.2.1 Requirements

In general, results of simulation studies are subject to a wide range of biases which arise from the fact that most problems have many parameters which need to be set by the researcher. In order to reduce *side effects* a minimal number of parameters is desired. Furthermore, test problems should be *neutral*, i.e. a test problem should not favor a certain method.

A test problem for SO robustness optimization is required to provide a clear trade-off between f_{raw} and f_{exp}, i.e. the f_{raw} optimum should be different from the f_{exp} optimum.

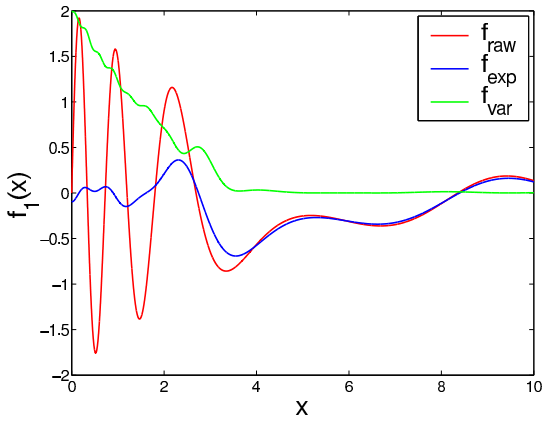


Figure 8.1: 1-dimensional test function f_1 with f_{raw}, f_{exp} and f_{var} with $\sigma_{noise} = 0.5$

8.2.2 Test problem

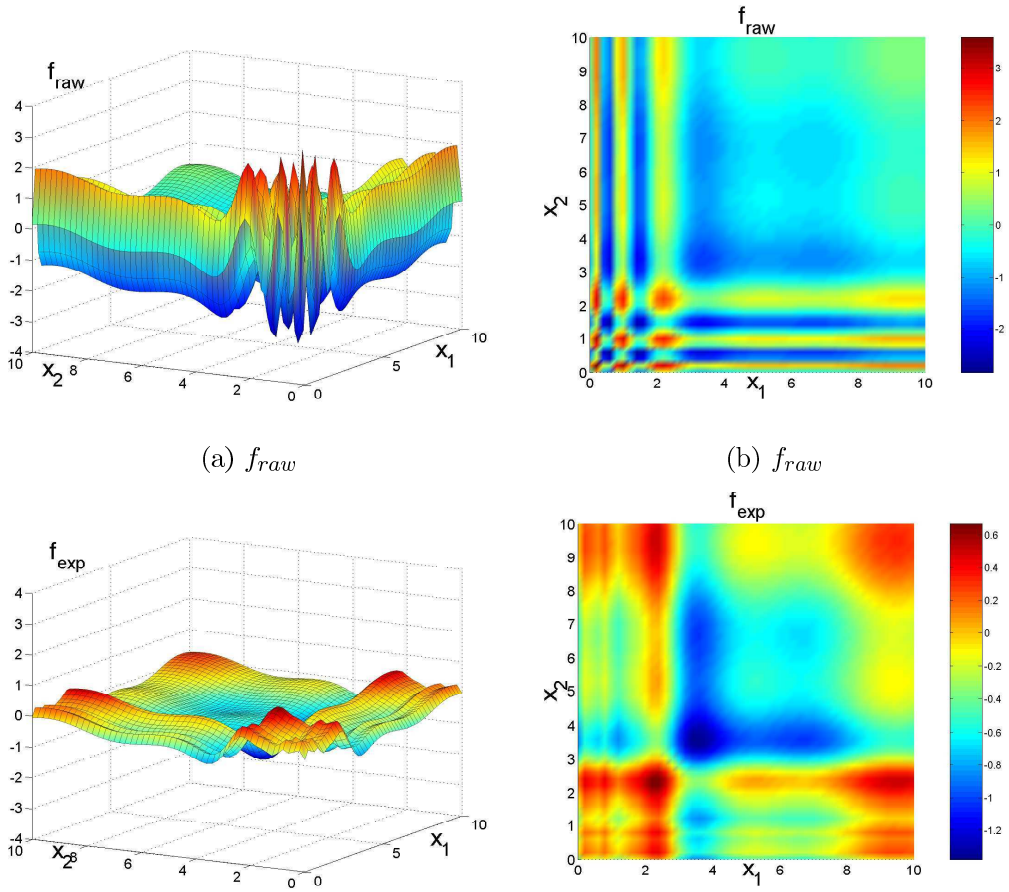
The test function of this chapter is defined

$$f_1(x) \;=\; \sum_{1 \le i \le n} 2.0 \sin(\,10 \exp(-0.2 x_i)\, x_i\,) \exp(-0.25 x_i) \tag{8.4}$$

where $x \in [0; 10]^n$. The optimization for raw fitness would be defined

$$\min f_1(x) \quad s.t. \quad x \in [0; 10]^n \tag{8.5}$$

However, for optimization of f_{exp}, the constraint is difficult to define. Here, we use a penalty as described in Section 4.5. The test problem is chosen such that constraint handling is not expected to have an influence on the search. Throughout all runs the noise that is added to the problem is $N(0, \sigma_{noise})$ distributed with $\sigma_{noise} = 0.5$ for each dimension. For the 1-dimensional case, f_1 is depicted in Figure 8.1. We see a clear trade-off between f_{raw} and f_{exp}, i.e. the f_{raw} optimum is located at $x^{(opt_raw)} = 0.5$ whereas the f_{exp} optimum is located at $x^{(opt_exp)} = 3.5$. f_{var} has no meaning in this chapter but is shown for illustration. An EA that uses f_{raw} as optimization criterion is expected to converge to $x^{(opt_raw)}$ whereas an EA that uses f_{exp} as optimization criterion is expected to converge to $x^{(opt_exp)}$. However, if f_{exp} is poorly estimated the EA might have problems to



(a) f_{raw} (b) f_{raw}

(c) f_{exp} (d) f_{exp}

Figure 8.2: 2-dimensional test function f_1 with $\sigma_{noise} = 0.5$ for each dimension: (a) f_{raw}, (b) f_{raw} projection on (x_1, x_2)-plane, (a) f_{exp}, (b) f_{exp} projection on (x_1, x_2)-plane

converge to $x^{(opt_exp)}$. The optimization is becoming more difficult in higher dimensions. Figure 8.2 compares f_{exp} and f_{raw} of test function f_1 for the 2-dimensional case. We see that the f_{exp} -surface is smoother is smoother than the f_{raw} -surface. In the projection on the (x_1, x_2)-plane (Figure 8.2(b)), we see that f_{raw} has 9 local optima including one global optimum. Three of the local optima have similar (raw) fitness values as the global optimum. f_{exp}, however, has a clear global optimum (Figure 8.2(d)). Of course f_{exp} is not available to the EA.

8.2.3 Analyzed methods

In the SO simulation studies, we analyzed a wide range of methods. Table 8.2 summarizes the analyzed methods and shows in which dimensions each method was tested.

Table 8.2: Tested methods in SO simulation studies

Tested dimension	*Single model*		*Nearest model*		*Ensemble*		*Multiple models*	
	LHC	*Strat*	*LHC*	*Strat*	*LHC*	*Strat*	*LHC*	*Strat*
lin. interpolation	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5
quad. interpolation	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5
lin. regression	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5
quad. regression	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5, 10	2, 5

For the remainder of this chapter we will also use the abbreviations, IMD-SM (*Single model*), PMD-ENS-1 (Ensemble with ensemble size 1 - Nearest model), PMD-ENS-k (Ensemble with ensemble size k), IMD-MM (Multiple models). For the most important parameters, e.g. *number of quantiles*, *ensemble size*, *number of sampling loops*, *outliers detection parameters*, we tested various settings.

8.2.4 Performance criteria

For each approximation method, we run an EA 20 times with different random seeds for each setting. In each run we take two measures in the final generation.

1. We choose the best individual according to the estimation $\widehat{f}_{exp}$ and reevaluate f_{exp} using the real fitness. We refer to this criterion as *best real fitness according to the approximation* (BRA). BRA is averaged over 20 runs.

2. For each of the 20 runs, we measure whether the best individual according to $\widehat{f}_{exp}$ is located within the neighborhood of the known global optimum. For the

n-dimensional test problem we define $[3;4]^n$ as neighborhood. By counting how often the best individual is located within the neighborhood, we get the relative share of successful runs. We refer to this criterion as *Global Optimum Neighborhood* (GON).

For evaluation of the best individual with the real f_{exp}, we use Stratified sampling with 5 quantiles per dimension for 2- and 5-dimensional test problems and 3 quantiles per dimension for the 10 dimensional test problem.

Additionally we run the EA with the same setting but use the real fitness function instead of approximations for estimation of f_{exp}. This, on the one hand represents a means to check whether the EA is generally able to converge to the global optimum with the respective parameter setting, on the other hand the results from this run represent an upper performance bound[4].

8.2.5 Benchmark methods

With above described performance criteria we are able to analyze which estimation methods work best when using fitness approximations. We further compare our approach to other robustness optimization methods. In particular, we implemented two methods to which we refer as

1. *Tsutsui* method

2. *Neighbors evaluation* method

Both approaches are briefly described in Chapter 2. Recall that the idea of *Tsutsui*[5] is to add disturbances to the decision variables while an individual is being evaluated.

[4]however, theoretically it is possible that approximations perform better than the real fitness. If for example the EA converges too early, approximations errors implicitly introduce exploration

[5]which was denoted GAs/RS^3 in [1]

As the EA is revisiting promising regions of the search space, it implicitly averages over a set of disturbed solutions. We refer to [1] for details. In our implementation, we run a standard evolutionary algorithm but add a normally distributed disturbance to the genotype of an individual before evaluating it. Note, that noise is only added for evaluation, the individual's genotype is not modified. In fact, this is exactly what Tsutsui et al. describe in their work. However, there are some differences between Tsutui's and our implementation: Besides a different setting of the genetic operators, Tsutsui et al. choose a binary representation, whereas we use real valued representation. In order to avoid the results to be biased by the choice of EA parameters, we compare our approach to Tsutsui's for 6 different EA parameters settings.

The *Neighbors evaluation* method runs an EA and evaluates all individuals with the real fitness f_{raw}. In contrast to our method the Neighbors evaluation approach does not store fitness evaluations in a History. Instead, only the individuals of the current population are available to estimation procedures. The estimator for $f_{exp}(x^{(0)})$ is the average fitness of individuals of the current population within a certain neighborhood of $x^{(0)}$. The neighborhood is chosen with respect to σ_{noise}. In particular, an individual $x^{(k)}$ is a neighbor of $x^{(0)}$ if $\|x^{(k)} - x^{(0)}\|_2 < q_{(1-\alpha)}$, where $q_{(1-\alpha)}$ is the $(1-\alpha)$-quantile of the $N(0, \sigma_{noise})$-distributed noise. In our experiments we set $\alpha = \mathit{cutoff} = 0.05$ (see Section 6.1). Note, that the neighborhood is not congruent to the space covered by the sampling procedure of estimation methods. In the 2-dimensional case the sampling covers a square (see Figure 6.1 for illustration). In contrast, the neighborhood in the Neighbors evaluation method is a circle with the same diameter as the edge length of the square.

Often the number of History data points which lie within the neighborhood borders is very small. We therefore introduce the parameter *numpoints_min*. If less than *numpoints_min* History data points are in the neighborhood of the estimation point we continue adding nearest data points to the estimation model such that the estimation is

based on *numpoints_min* data points. The Neighbors evaluation estimator was also used in [14] for MO robustness optimization. Branke [7] uses a similar estimator but additionally assigns weights to individuals. We allow both benchmark method the same number of fitness evaluations as used for the approximation methods: With population size 100 and 50 generations, this means 5000 evaluations are allowed.

8.2.6 Parameter setting

The parameters for the algorithm of the SO simulation studies are summarized in Table 8.3. Parameters can be categorized in *EA parameters* (Chapter 4), *approximation parameters* (Chapter 5) and *sampling parameters* (Chapter 6). Details on the parameters are described in the respective chapters. This parameter setting was used for all experiments throughout this chapter if not stated otherwise. The small number of parameters which need to be set for the *Tsutsui* and the *neighbors evaluation* approach are outlined in Section 8.3.7.

Table 8.3: Standard parameter setting for the SO simulation studies

EA parameters	
(μ, λ) - *reproduction scheme*	$(15, 100)$
standard evolution strategy (mutation)	$\sigma_{init} \in [0.01; 1.0]$ (randomly)
discrete recombination (objective variables)	no parameters
generalized intermediate recomb. (strategy var.)	no parameters
selection	randomly (uniform distribution)
termination condition	50 generations (fixed)
infeasibility penalty	$2.0 \cdot$ *dimensions*
Approximation parameters	
History size (max)	5000 (stores *all* individuals)
interpolation: *singular_threshold* (init)	10^{-12}
interpolation: *increase (decrease) factor*	10 $(\frac{1}{10})$
regression: *singular_threshold* (init)	10^{-20}
regression: *increase (decrease) factor*	10 $(\frac{1}{10})$
regression: *bandwidth*	σ_{noise}-*range* $[\alpha_{cutoff}; \alpha_{(1-cutoff)}]$
regression: k_{min} (min. number of data points)	$2 \cdot$ *numcoeff*
regression: *weight function*	*tricube function*
equation_solved_threshold	10^{-2}
Sampling parameters	
cutoff	0.05
sample drawing	*derandomized*
outliers (boxplot): *whisker_factor*	5
outliers (boxplot): *quantiles*	$(\alpha_{0.01}, \alpha_{0.99})$

8.3 Results

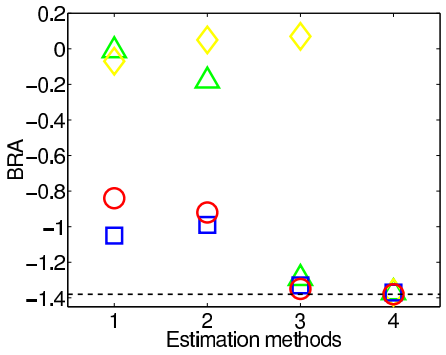
8.3.1 Overview

As a starting point, have a look at Figures 8.3 and 8.4. Here the performance of different approximation models and when using the real fitness is shown for different estimation methods for both Latin hypercube and Stratified sampling. The estimation methods are enumerated as follows (Table 8.4). As ensemble sizes we choose $k = 3$ (dim 2), $k = 10$ (dim 5), $k = 10/30$ (dim 10, linear/quadratic).

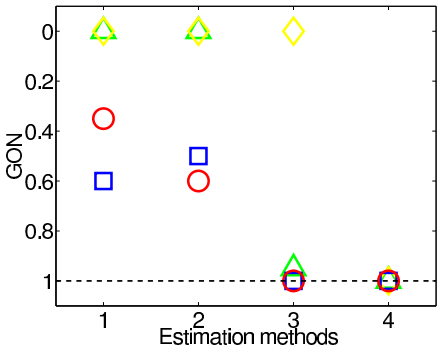
Table 8.4: Enumeration of the estimation methods

No.	description	*num_samples*					
		LHC			*Stratified*		
		dim 2	dim5	dim10	dim 2	dim5	dim10
1	Single model (IMD-SM)	5	50	100	25	243	–
2	Nearest model (PMD-ENS-1)	5	50	100	25	243	–
3	Ensemble (PMD-ENS-k)	5	50	100	25	243	–
4	Multiple models (IMD-MM)	5	50	100	25	243	–

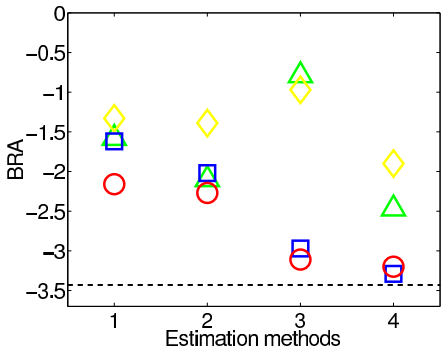
We see that the EA is generally able to converge to the global optimum. In all settings (in all dimensions) the EA converges to the global optimum when using the real fitness instead of approximations: This can be seen from Figure 8.3(b,d,e) where the GON criterion is 1.0 for the real fitness, and similar in Figure 8.4.



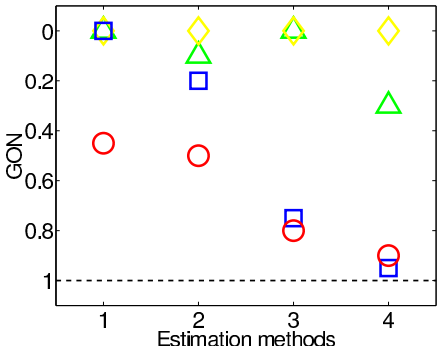
(a) dimension 2, BRA



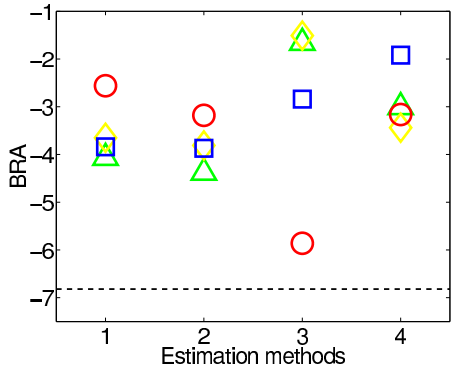
(b) dimension 2, GON



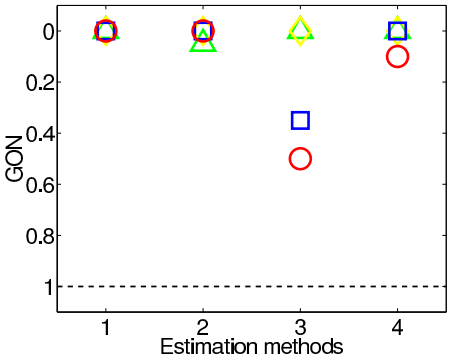
(c) dimension 5, BRA



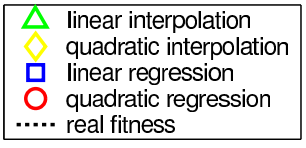
(d) dimension 5, GON



(e) dimension 10, BRA

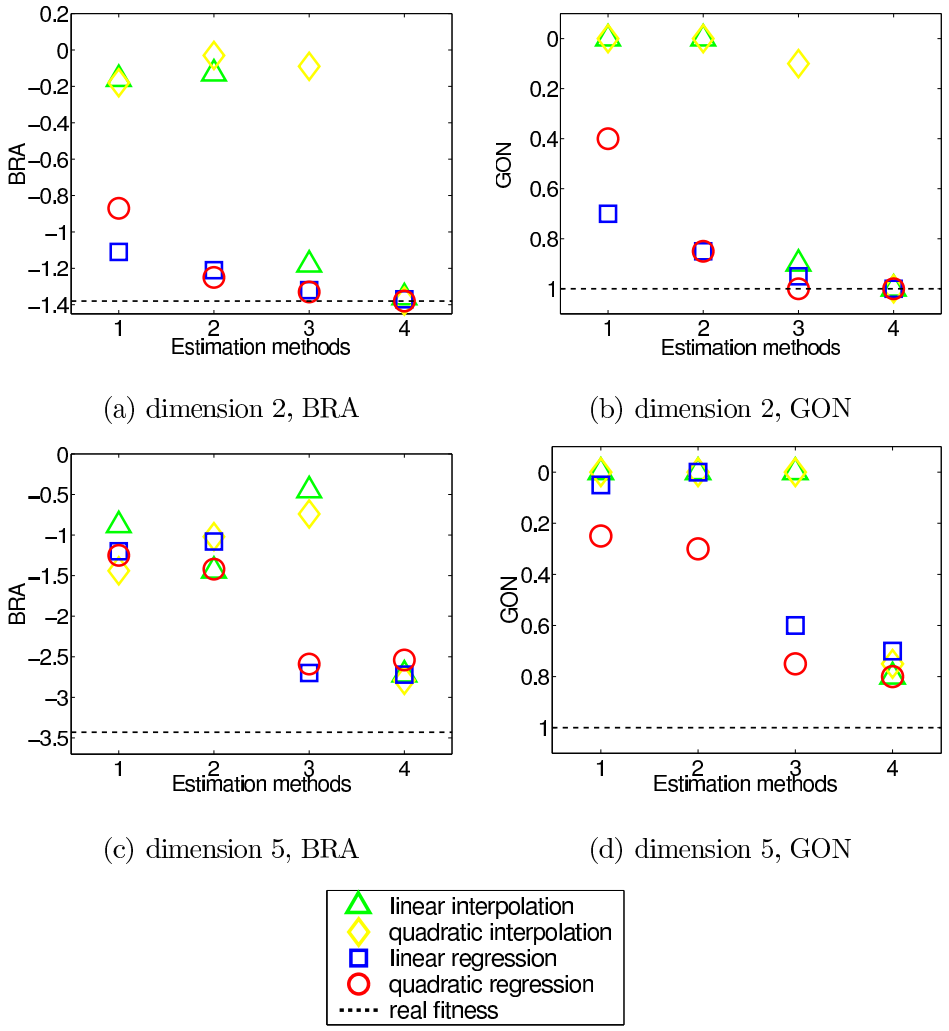


(f) dimension 10, GON



(g) legend

Figure 8.3: Comparison of approximation models for different estimation methods in dimensions 2,5,10 with Latin hypercube sampling. Method 1 - IMD-SM, Method 2 - PMD-ENS-1, Method 3 - PMD-ENS-k, Method 4 - IMD-MM



(a) dimension 2, BRA

(b) dimension 2, GON

(c) dimension 5, BRA

(d) dimension 5, GON

(e) legend

Figure 8.4: Comparison of approximation models for different estimation methods in dimensions 2,5 with Stratified sampling. Method 1 - IMD-SM, Method 2 - PMD-ENS-1, Method 3 - PMD-ENS-k, Method 4 - IMD-MM

8.3.2 Hypotheses

We draw up the following hypotheses:

1. *Local Regression* is more effective than *Nearest neighbor interpolation*

2. A *quadratic* approximation model is more effective than a *linear* model

3. *Multiple models* is more effective than *Single model*

4. *Nearest model* is more effective than *Single model*

5. There exist scenarios in which *Ensemble* is more effective than *Multiple models*

Hypotheses 1 and 2 are discussed in Section 8.3.3. Hypotheses 3-5 are discussed in Section 8.3.4.

8.3.3 Approximation models

Hypothesis 1 - Local Regression is more effective than Nearest neighbor interpolation

The results clearly support Hypothesis 1. Already in dimension 2 the local regression methods perform better than the interpolation models. Only in IMD-MM, interpolation shows comparable performance. In Dimension 5, regression works better than interpolation also when using Multiple models. This at least holds when using Latin hypercube model distribution (Figure 8.3). One-sided T-tests provide significance up to a significance level of 0.99 (see Tests 3-6 in Table A.3). In dimension 10, all methods show poor performance. However, *quadratic* regression works significantly better than all other models.

If the argumentation of Section 8.1 is correct, we expect $\sigma_{e_{exp}}$ to be lower for regression than for interpolation. In order to test this, we set up the following experiment: For the 5-dimensional case, we randomly generate a set of History data. Out of these, we choose 100 data points $x^{(i)}$ for which $\widehat{f}_{exp}(x^{(i)})$ is computed with different estimation methods. The estimation error is computed $\widehat{f}_{exp}(x^{(i)}) - f_{exp}(x^{(i)})$, where $f_{exp}(x^{(i)})$ is based on the real fitness. By repeating this experiment 50 times we get a distribution of 5000 e_{exp}-values

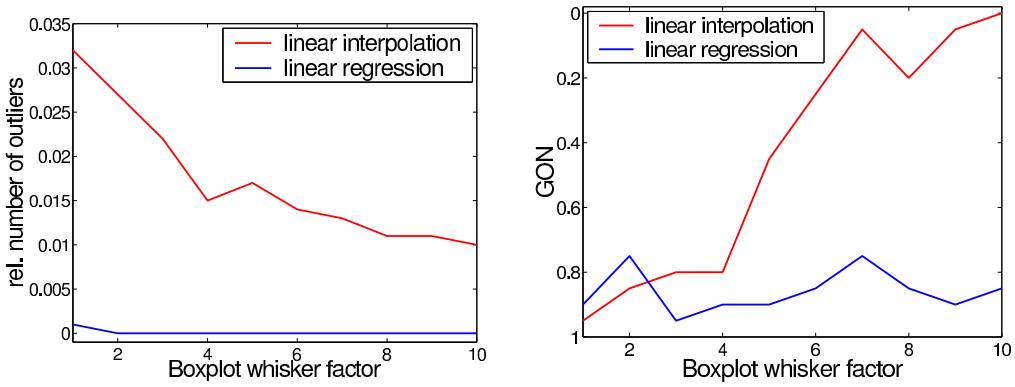
Table 8.5: Standard deviation of the estimation error (Dimension 5, PMD-MM)

History data	*lin. interpol.*	*quad. interpol.*	*lin. regress.*	*quad. regress.*
1000	0.38	0.47	0.22	0.30
10000	0.25	0.28	0.18	0.13

for which we compute mean and standard deviation. As we will see later, we run this experiment to provide evidence for other hypotheses, too. Therefore we refer to this experiment as σ *measure experiment*. Table 8.5 presents the results. Here, we use PMD-MM in 5 dimensions. Statistical Fisher tests show that the $\sigma_{e_{exp}}$ of regression is smaller than the $\sigma_{e_{exp}}$ of interpolation for a significance level of 0.99 (see Fisher Tests 13-16 in Table A.6).

How can the high standard deviation be explained? We found that interpolation is likely to produce extreme outliers (recall Section 6.3). We further show that the bias introduced by outliers significantly degrades the performance. Evidence is provided by the following experiment: We vary the size of the *acceptance interval* in the outliers detection routine (see Section 6.3.2). Setting *whisker factor* to a small number, i.e. setting a small *acceptance interval*, will cause many estimations to be treated as outliers. In Figure 8.5 the relative number of outliers and the GON performance is shown for different *whisker_factor* settings for linear interpolation and linear local regression (PMD-MM, Dimension 5). As expected, the number of outliers decreases with increasing size of *acceptance interval*. Local regression is significantly less suspect to outliers (a). Therefore the performance is stable for different *acceptance intervals*(b). However, the performance of linear interpolation decreases with increasing size of the acceptance interval.

We conclude that Hypothesis 1 can be accepted.



(a) rel. number of outliers (b) GON performance

Figure 8.5: Varying parameters for outliers detection (Boxplot whisker factor)

Hypothesis 2 - A quadratic approximation model is more effective than a linear model

Hypothesis 2 focuses on the order of the approximation polynomials. Here we consider local regression as this has shown to be the preferred method. One would expect that a quadratic model better guides the evolutionary search than a linear model. However, it is important in which scenario the models are employed.

In IMD-MM, the model is only evaluated at the fitting point $\vec{x}_0$. If the History data density is low, a linear model might perform better, because a quadratic model requires $\frac{dim+1}{2}$ as many input data points as a linear model (see Chapter 5.2). With low History data density, data points with large distances to the model fitting point $\vec{x}_0$ need to be added to the model. Thus, the local fit becomes worse. Considering Table 8.5 we find empirical evidence. For low History data density (1000 data points) the linear regression model performs significantly better whereas for a large History data base (10000 data points) the quadratic regression model performs better (see Fisher Tests 17-18 in Table A.6).

In IMD-SM the approximation model needs to approximate the fitness function over a certain range. Therefore we expect a quadratic model to perform better in this case.

Table 8.6: Standard deviation of the estimation error (dimension 5, IMD-SM)

History data	*lin. regress.*	*quad. regress.*
1000	0.24	0.29
10000	0.26	0.23

Figure 8.3 shows for the 5-dimensional IMD-SM case (c,d) that quadratic local regression performs best. Somewhat surprising, this does not hold for the 10-dimensional case. However, as in IMD-SM, this can be attributed to the History data density which is very low in dimension 10. Empirical evidence is provided by the data from Table 8.6. Here, the *sigma measure experiment* from Table 8.5 is run in IMD-SM mode. We see that for low History data density the linear model performs better whereas for large History data base the quadratic model performs better[6].

Significance is shown by Fisher Tests 19-20 in Table A.6. However, we can not accept Hypothesis 2 *in general*.

8.3.4 Sampling and model distribution

Hypothesis 3 - IMD-MM is more effective than IMD-SM

Evidence for Hypothesis 3 is again provided by Figures 8.3 and 8.4: For dimensions 2 and 5 IMD-MM clearly performs better (Methods outperforms method 1). Significance can also be shown statistically (T-Tests 1,9 in Tables A.2,A.3). The reasoning is straightforward and was already outlined in Section 6.2. However, in Dimension 10 IMD-SM and IMD-MM show similar performance. This can again be attributed to the low History data density in Dimension 10. If only a small number of data points are available in the neighborhood of estimation point $x^{(0)}$, building a large number of models around

[6]somewhat surprising the standard deviation is slightly larger for a larger History data base when using linear regression

$x^{(0)}$ does not improve the estimation quality, because the models are likely to be similar. Technically spoken, the *covariance* from Equation 8.3 increases the standard deviation $\sigma_{e_{exp}}$. Evidence for the existence of correlation is provided by the fact that the EA does not improve further by increasing the number of models over a certain level.

However, if a sufficiently large History data base is available, Hypothesis 3 can be accepted.

Hypothesis 4 - PMD-ENS-1 is more effective than IMD-SM

One would expect the performance to be increased if the nearest available approximation model is chosen for evaluation (PMD-ENS-1) instead of applying IMD-SM, where the same model is used for all samples of an estimation point. However, the performance is only slightly improved by PMD-ENS-1 as Figures 8.3 and 8.4 show. T-Tests (2,10 Tables A.2 - A.3) fail to show significantly higher performance of PMD-ENS-1. A possible reason for this surprising result may be the definition of the *model location*. This is defined as the location of the individual for which the model is built. The drawback of this definition is that it does not account for the location of the input data of the models. A method that overcomes this drawback is to use the *center of mass* of model input data as location. Figure 8.6 shows an example for the 1-dimensional case in which the center of mass method works better. With our method the sample point ("sample point of individual 1") chooses model 1 for evaluation, because individual 1 is the nearest individual. With the center of mass method, model 2 is chosen for evaluation of the sample point. We did not implement this feature, but believe that it should be analyzed in future research.

We conclude that Hypothesis 4 cannot be accepted.

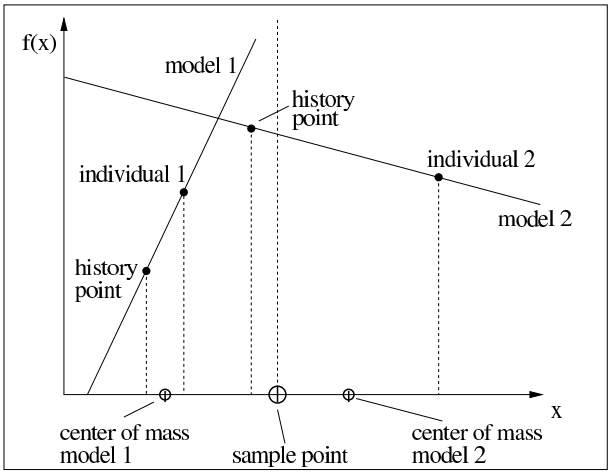


Figure 8.6: Model location: Individiual location vs. Center of mass

Hypothesis 5 - There exist scenarios in which PMD-ENS-k is more effective than IMD-MM

As a starting point for Hypothesis 5, again, consider Figure 8.3. For the regression methods in Dimension 2 and 5, PMD-ENS-k performs only slightly worse than IMD-MM. In dimension 10, PMD-ENS-k shows significantly higher performance than IMD-MM (see T-Tests 11-12 in Table A.4). This result is most surprising. IMD-MM is significantly more expensive because for each sample point an own model is built. PMD-ENS-k requires only one model to be built per individual and some additional computation time for calculation of the ensemble. The key reason for the observed behavior is again the standard deviation of the estimation error. Table 8.7 shows results from the σ *measure experiment* which are consistent with our expectation: As the PMD-ENS-k estimator has lower $\sigma_{e_{exp}}$ it better guides the search. For quadratic regression, we show that the standard deviation of the estimation error is significantly lower (see Fisher-tests 21-22 in Table A.6).

Questions to be addressed to this result are *"How can the low $\sigma_{e_{exp}}$ be explained despite the significantly smaller number of models per generation of PMD-ENS-k?"* and *"What influences $\sigma_{e_{exp}}$ in PMD-ENS-k?"*.

Table 8.7: Standard deviation of the estimation error: PMD-ENS-10 vs. IMD-MM (quadratic regression, 1000 History data)

	PMD-ENS-k	*Multiple models*
Dimension 5	0.21	0.30
Dimension 10	0.39	0.73

In Section 8.1, we showed that $\sigma_{e_{exp}}$ depends on the $\sigma_{e_{\hat{f}i}}$ and on the covariance between error distributions at sample points $x^{(i)}$. Let us for a first analysis of $\sigma_{e_{exp}}$ assume the covariance to be zero. In this case only the $\sigma_{e_{\hat{f}i}}$ matter. To avoid confusion we redefine some variables for this section (see Table 8.8). The distribution of e_{ens} can theoretically be calculated by applying a convolution of the error distribution $e_{\hat{f}_i}$ of all ensemble members. If n_{ens} is sufficiently large, or if the $e_{\hat{f}_i}$ are normally distributed we derive

Table 8.8: Redefinition of terms

Term	*Meaning*
$\widehat{f}_i$	an approximation model, $i = 1 \ldots$ *population_size*
$e_{\hat{f}}$	approximation error $\widehat{f} - f$
$\mu_{e_{\hat{f}}}$	mean of the $e_{\hat{f}}$ - distribution
$\sigma_{e_{\hat{f}}}$	standard deviation of the $e_{\hat{f}}$ - distribution
$\widehat{f}_{ens}$	weighted sum of a number of approximation models $\widehat{f}_i$ (*Ensemble*)
e_{ens}	approximation error $\widehat{f}_{ens} - f$
$\mu_{e_{ens}}$	mean of the e_{ens} distribution
$\sigma_{e_{ens}}$	standard deviation of the e_{ens} distribution
n_{ens}	number of models in an ensemble (*ensemble size*)

$$e_{ens} \sim N(\mu_{e_{ens}}, \sigma_{e_{ens}}) \sim N(\mu_{e_{\hat{f}}}, \frac{\sigma_{e_{\hat{f}}}}{\sqrt{n_{ens}}})$$

Under the assumptions made so far, we conclude that a maximum n_{ens} performs best. This implicitly assumes $\sigma_{e_{\hat{f}}}$ to be equal for all ensemble members. In reality, we expect $\sigma_{e_{\hat{f}}}$ to increase if the distance d between the fitting point of $\widehat{f}$ and the location where $\widehat{f}$ is evaluated increases. Larger n_{ens} requires to add models with larger d.
To summarize, there exist two counteracting effects of increasing the ensemble size n_{ens}:

1. the reduction of $\sigma_{e_{ens}}$ caused by the convolution
2. the increase $\sigma_{e_{ens}}$ caused by the increasing distances between models fitting points and their location of evaluation

Both effects are weakened. In our implementation, we use a weighting of the ensemble members which takes into account distances (see Section 6.2.2). Models with large d are assigned a low weight. Thus, Effect 2 is reduced. Effect 1 is reduced, because some models of the ensemble might be correlated.

For the actual estimation of f_{exp} another negative effect is expected: The estimation of f_{exp} averages over n_s ensembles $(\widehat{f}_{ens})^{(i)}$.The σ_{exp}-reducing convolution effect is strongly weakened because all ensembles are a composition of the same n approximation models, where n is the population size. Thus, the ensembles, respectively their error distribution, are likely to be correlated. Nevertheless, we find empirical evidence for the existence of the counteracting effects. In Figure 8.7 the results from the σ *measure experiment* are depicted for quadratic regression, when varying the ensemble size. For the 5 dimensional case, the History data base contains 1000 data points, in Dimension 10, 10000 History data points are available. We see that there exists an optimal ensemble size, i.e. effect 1 is stronger for smaller ensemble sizes, effect 2 is dominating for larger ensemble sizes.

Further evidence is provided by measuring the EA performance for different ensemble sizes. In Figure 8.8 the performance criteria BRA and GON are depicted for linear and

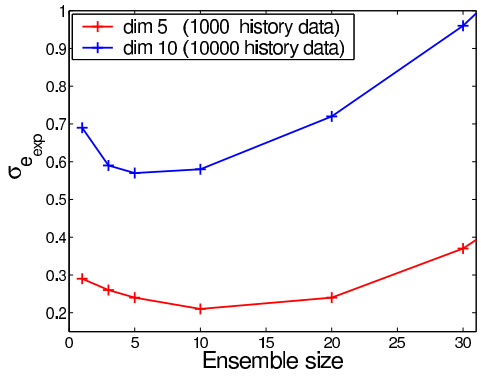
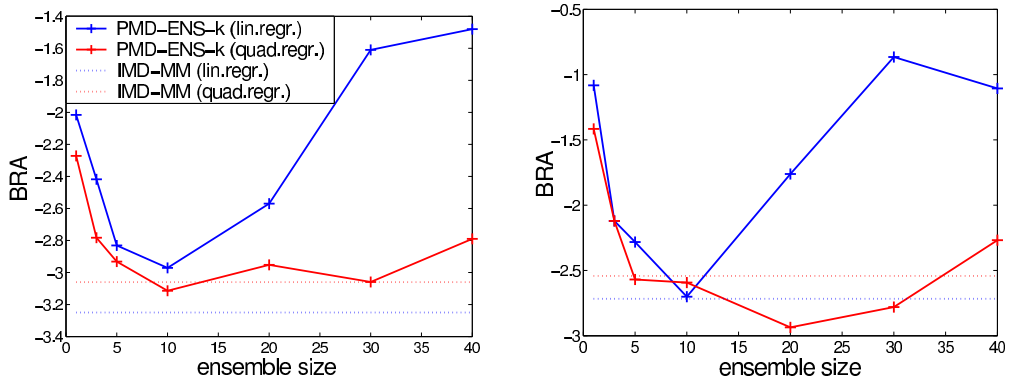


Figure 8.7: Estimation error standard deviation of PMD-ENS-k for different k (quadratic regression in dimensions 5 and 10)

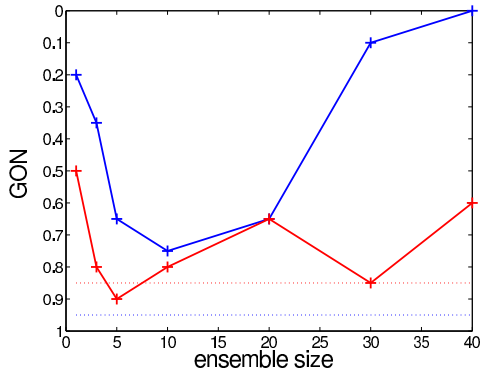
quadratic local regression. Despite some deviations it seems that there exists an optimal ensemble size. Particularly for Stratified sampling, where the sampling bias is expected to be low, we find an optimal ensemble size for both linear and quadratic local regression. For illustration, the performance of IMD-MM is depicted: In some cases PMD-ENS-k performs better than IMD-MM if optimal k is chosen.

We now better understand why PMD-ENS-k works better in some scenarios: The neighborhood of an individual for which f_{exp} needs to be estimated contains a certain number of data points. In both methods IMD-MM and PMD-ENS-k the neighborhood information is most likely used multiple times, either by adding the same data point to a large number of models, or by building a comparably small number of models, and use these models multiple times for evaluation. If, in PMD-ENS-k the neighboring models have an advantageous distribution, the PMD-ENS-k estimation of f_{exp} takes more information into account than IMD-MM. As a result σ_{eexp} is lower in PMD-ENS-k. This was denoted Effect 1. However, the model accuracy is expected to be better in IMD-MM. The decrease of estimation accuracy with larger number of models was denoted Effect 2. Whether IMD-MM or PMD-ENS-k works better depends on which of the effects is dominating. Taking into account computational cost, PMD-ENS-k is the preferred method.

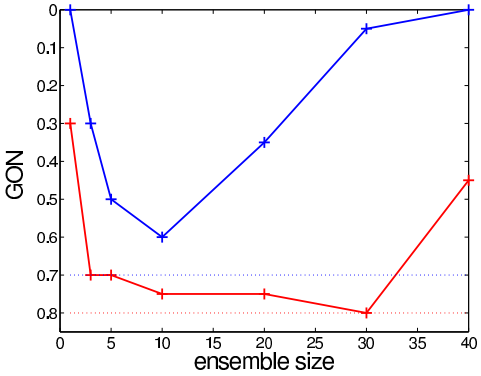


(a) BRA, LHC sampling

(b) BRA, Stratified sampling



(c) GON, LHC sampling



(d) GON, Stratified sampling

Figure 8.8: PMD-ENS-k performance for different ensemble sizes compared to IMD-MM (Dimension 5). All subfigures use the legend in (a).

Some remarks should be made on ensembles of interpolation models. In Section 8.3.3 we showed that interpolation is likely to produce extreme wrong estimations (see for example Figure 8.5) which arise from approximation models with very large gradients (illustrated by Figure 6.5). With large distances between evaluation point and model fitting point the wrong estimation becomes even more severe. Therefore we expect interpolation to be a poor estimator in PMD-ENS-k with $k > 1$. The results from Figures 8.3 and 8.4 provide evidence. Both, linear and quadratic interpolation perform poor in PMD-ENS-k

Table 8.9: Standard deviation of the estimation error (Interpolation in PMD-ENS-10)

		IMD-SM	PMD-ENS-10
dim 5	*lin. interpol*	0.31	8.85
	quad. interpol.	0.66	9.53
dim 10	*lin. interpol*	0.68	31.57
	quad. interpol.	1.11	21.98

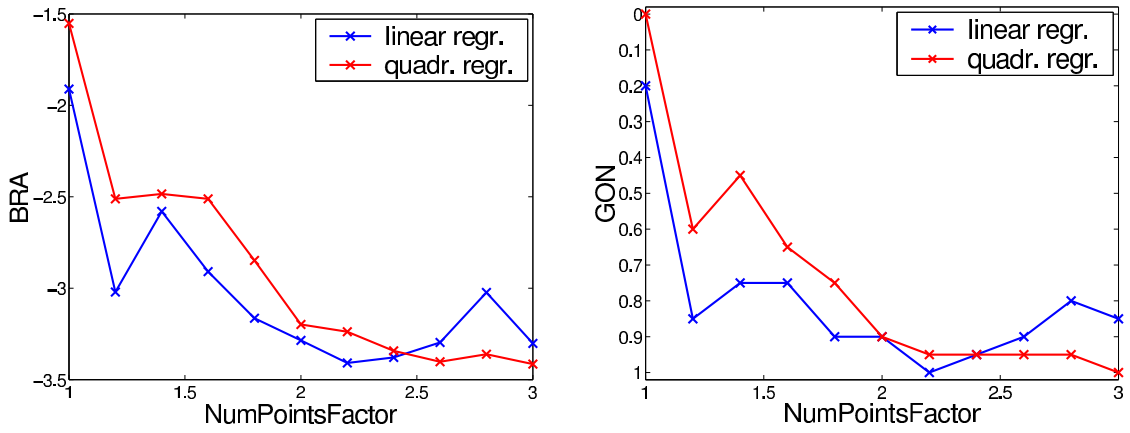
in dimensions higher 2. Quadratic interpolation produces severe outliers already in Dimension 2 and is therefore a poor estimator in combination with PMD-ENS-k. Additional evidence is provided by the σ *measure experiment*. We see in Table 8.9 that interpolation produces large estimation errors in PMD-ENS-10. From this analysis, we conclude that Hypothesis 5 can be accepted.

8.3.5 Selected regression parameters

As local regression turned out to be the method of choice, we take a closer look at a selected set of regression parameters which are expected to have an influence on the performance. See Section 5.6 for a detailed description. In particular, we vary the *minimum number of input data points*, *bandwidth* and run some experiments with a *weight function* which assigns *equal weights* to input data.

Minimum number of input data points

In our standard setting the number of input data points is not fixed. Instead, we choose all data points which are located within the range of the noise distributions (σ_{noise}-range). However, the number of data points within the σ_{noise}-range might be smaller than the required minimum number of data points (equals *numcoeff*). In this case, the number of input data points must be increased to a number larger or equal *numcoeff*. In order to



(a) BRA (b) GON

Figure 8.9: Regression performance for different numbers of input data (IMD-MM, Latin hypercube sampling, dimension 5)

have a clear distinction between interpolation and local regression we set the minimum number of input data k_{min} to a number larger than *numcoeff*, in particular

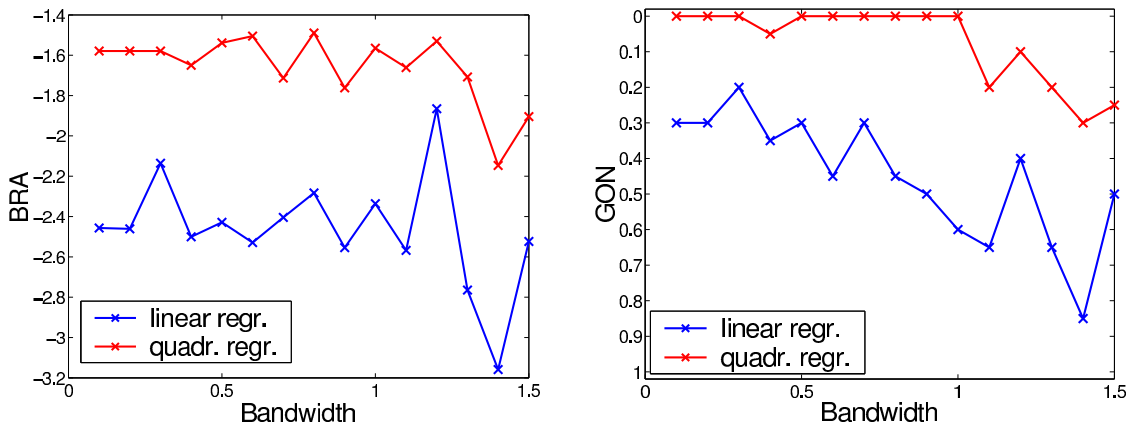
$$k_{min} = NumPointsFactor \cdot numcoeff$$

with $NumPointsFactor > 1.0$. In the standard setting $NumPointsFactor$ was set to 2.0. For the empirical analysis on the influence of this parameter we run IMD-MM in dimension 5 with Latin hypercube sampling (50 samples) and varied $NumPointsFactor$ in $[1.0; 3.0]$ with step size 0.2. Figure 8.9 presents the results. We see that the performance is not constant for different settings of $NumPointsFactor$. We conclude that the History provides less than k_{min} data points in the σ_{noise}-range. Thus, $NumPointsFactor$ has an influence on the performance. For $NumPointsFactor = 1.0$, both linear and quadratic regression perform poor: If less than *numcoeff* data points lie within the σ-range, local regression works exactly as interpolation regardless of the weight function. In fact, for this setting we find local regression to perform similar to interpolation (compare Figure 8.3 (c,d), Method 4). For larger $NumPointsFactor$ the performance is more stable. One could expect to find a clear optimum, because adding more data points to the model is expected to have a

negative influence on the local fit of the model. However, by using the tricube weight function, input data points with large distances to the model fitting point are assigned a low weight. Furthermore, adding more data to the model reduces the standard deviation of the estimation error.

Bandwidth

In this experiment the *bandwidth* does not depend on the availability of input data points, except there are less than *numcoeff* data points within the bandwidth. In this case, the *bandwidth* is extended such that *numcoeff* data points lie within the bandwidth. We varied bandwidth in the interval [0.1; 1.5]. Figure 8.10 presents the results. In fact, we see that for low bandwidth settings the algorithm has constant performance. Here, the bandwidth is extended such that *numcoeff* data points lie within the bandwidth. This holds particularly for quadratic regression because a quadratic polynomial has larger *numcoeff*. For different bandwidth the performance has a high variance and does not allow reliable conclusions on an optimal bandwidth.



(a) BRA (b) GON

Figure 8.10: Regression performance for different bandwidth (IMD-MM, Latin hypercube sampling, dimension 5)

However, compared to the previous experiment (compare Figures 8.9 and 8.10), we find that it might be better to set the bandwidth via k_{min} rather than setting a fixed bandwidth. From a theoretical point of view, this can be explained as follows: The "k_{min}-adjustment" adapts to different History data density around a model's fitting point. For high History data density "k_{min}-adjustment" chooses a small bandwidth. This works well, because the small bandwidth provides a sufficiently large number of input data to achieve a good local fit. If the History data density is low, "k_{min}-adjustment" chooses a larger bandwidth such that a sufficiently large number of input data are added to the model. In contrast, setting a fixed bandwidth does not account for different History data density. This performs poor, because throughout the run of an evolutionary algorithm the availability of History data changes.

Assigning equal weights

In an additional experiment we use a weight function which assigns equal weights to all input data instead of using the tricube weight function. This is equivalent to *unweighted* or *global* regression. We run this setting for Dimension 5 and got surprising results, see Figure 8.3.5

Comparing Tricube weight function and equal weights (compare Figure 8.3 (c,d) with Figure 8.3.5 (a,b)), we find that assigning equal weights works better. For Methods 1-3, that is, when only one model is built per individual, this is plausible. Tricube weight function produces a good *local* fit. If, however, the model is evaluated at different locations, a global model works better. Surprisingly, we find unweighted regression to perform better when using IMD-MM, too. Although the performance difference is less significant than in Methods 1-3 this is an open question. One possible reason is, that unweighted regression produces a smaller standard deviation of the estimation error $\sigma_{e_{exp}}$ because even data points with large distance to the model fitting point significantly contribute to

a reduction of $\sigma_{e_{exp}}$, although at the same time the local fit in terms of expected estimation error becomes worse.

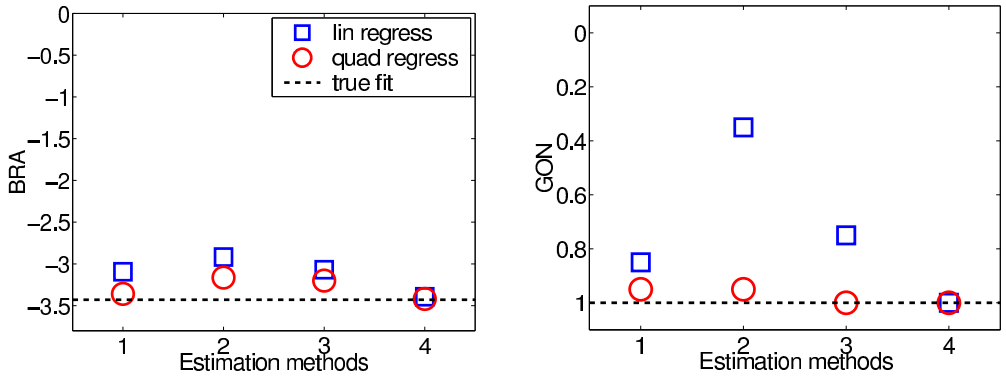
8.3.6 General observations

Influence of sampling technique

Comparing the two different sampling techniques we find that Stratified sampling (Figure 8.4) does not improve the performance compared to Latin hypercube sampling (Figure 8.3). In fact, in Dimension 5 Latin hypercube sampling works even slightly better. This is at first sight surprising because Stratified sampling is expected to have a better coverage of the neighboring search space and the number of samples is approximately 5 times higher than in Latin hypercube sampling[7]. However, increasing the number of samples does not necessarily improve the estimation quality. An estimation error has two sources: First, the estimation error introduced by the approximation models. Secondly, the estimation error which results from sampling[8]. The effect from the second source

[7]Dimension 5: Latin hypercube sampling: 50 samples, Stratified sampling: $3^5 = 243$ samples

[8]the sample set must have finite size



(a) BRA (b) GON

Figure 8.11: Regression performance when assigning equal weights
(Latin hypercube sampling, dimension 5, method numbering as in Table 8.4)

can be reduced, by more expensive sampling methods. The effect from the first source, however, is not reduced by increasing the number of samples. Latin hypercube sampling perhaps works better because it uses more quantiles (see illustration in Figure 6.1). One might argue, that Latin hypercube sampling works specifically well, if the dimensions are independent which is the case in our test function. However in our application, the sampling is done based on the approximation models which do not have independent dimensions.

Performance loss in higher dimensions

Figure 8.12 shows the performance loss in higher dimensions. This is measured by dividing the BAR criterion of the best PMD-ENS-k setting and the best IMD-MM setting by the BAR of the real fitness. As the number of evaluations is held constant over all dimensions this observation is expected. The performance loss can be attributed to theoretical limitations. Interestingly, the performance loss is weaker in PMD-ENS-k. Theoretical evidence is provided in Section 8.3.4 (Hypothesis 5).

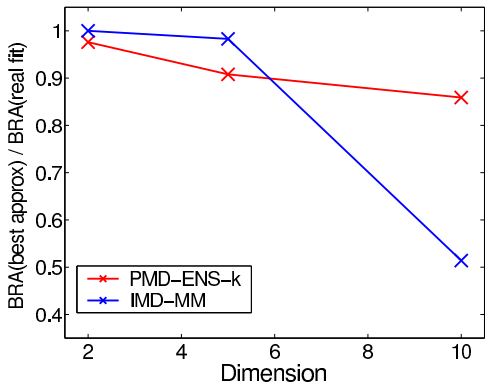
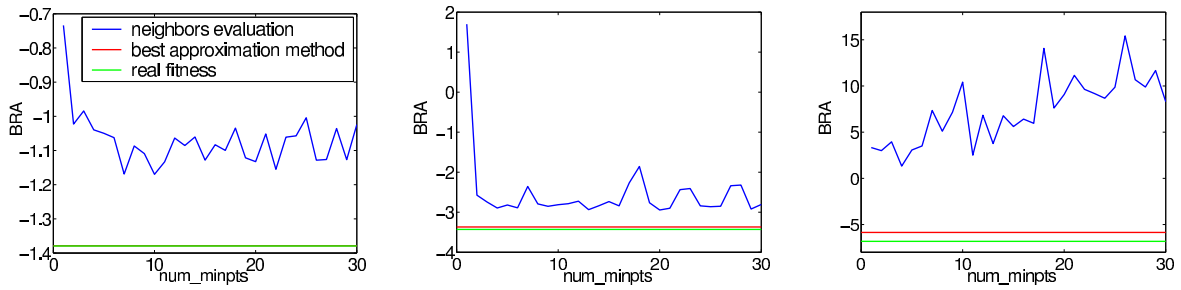


Figure 8.12: Performance loss in higher dimensions $\frac{BAR(bestapproxmation)}{BAR(realfitness)}$

8.3.7 Comparison with other methods

Neighbors evaluation method

The Neighbors evaluation method is described in Section 8.2.5. In order to have a fair comparison we run the Neighbors evaluation for different settings, i.e. we vary the minimum number of data points (*num_minpts*) that are to be added to the estimation model. As described in Section 8.2.5 the neighborhood size is set to the range of σ_{noise} and held constant for all runs. We compare the performance (BRA) for different parameter settings with the performance achieved by the estimation method. Additionally we add the performance achieved when using the real fitness. Figure 8.13 shows the results. We clearly see that the approximation models perform better. In Figure 8.13(a) the green and the red line overlap. We also see that the performance of the Neighbors evaluation significantly degrades in higher dimension: If the number of fitness evaluations (History size) is held constant for arbitrary dimension the number of data points which are in the neighborhood of an estimation point decreases with increasing dimension.



(a) dim2 (b) dim5 (c) dim10

Figure 8.13: Comparing Neighbors evaluation method with different parameter settings to the best approximation method and to the performance when using the real fitness

Tsutsui method

The *Tsutsui* method is described in Section 8.2.5. In preliminary tests with the Tsutsui method we found this method to be highly sensitive to parameter settings of the EA. In order to have a fair comparison, we run the Tsutsui method as well as selected estimation methods with 6 different parameter settings. These parameter settings are modifications of our standard setting which is described in Table 8.3. The 6 settings are different from the standard setting in the following parameters.

Setting 1 uses *normal distribution mutation* with $\sigma = 0.05$ instead of a *standard evolution strategy* (no strategy parameters)

Setting 2 as Setting 1 but uses $(50, 100)$ (μ, λ)-reproduction scheme

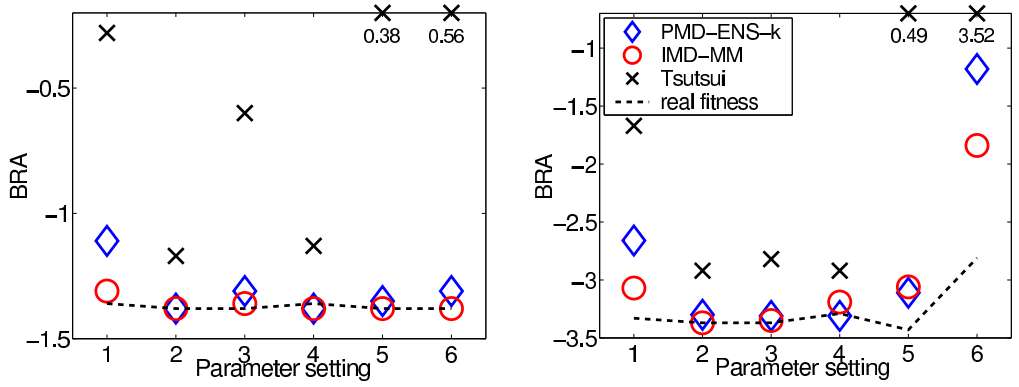
Setting 3 as Setting 1 but uses *generalized intermediate recombination* for the objective variables (no strategy parameters)

Setting 4 as Setting 3 but uses $(50, 100)$ (μ, λ)-reproduction scheme

Setting 5 standard setting (Table 8.3)

Setting 6 as Setting 5 but uses $(50, 100)$ (μ, λ)-reproduction scheme

Figure 8.14 summarizes the results. We see that for dimension 2 Tsutsui performs well for some parameter settings. For example in Setting 4, the Tsutsui method converges near to the global optimum in 85 percent of the runs (Figure 8.14 (b)). However, this finding is problem specific. In fact, all approximation methods converge to the global optimum in 100 percent of the runs. Therefore this can be attributed to the EA parameter setting: In particular, *generalized intermediate recombination* tends to converge to the center of the search space. In our test problem, the global optimum is located close to the center of the search space and therefore favorizes *generalized intermediate recombination*.



(a) Dimension 2, BRA

(b) Dimension 5, BRA

Figure 8.14: Tsutsui method compared to PMD-ENS-k (k=5 in dim 2, k=10 in dim 5) and IMD-MM in combination with quadratic regression - BRA criterion

In both dimensionality's 2 and 5 we find the Tsutsui method to be outperformed by PMD-ENS-k and quadratic regression in all EA parameter settings. In the standard setting of this chapter (Setting 5) Tsutsui achieves a poor BRA. We also see a performance loss for Tsutsui in higher dimensions. This is expected because holding the number of evaluations constant and increasing the dimensionality reduces the *relative* number of evaluations. In particular, the neighborhood size increases exponentially with the number of dimensions. We conclude that the approximation models in combination with a proper estimation method work better than Tsutsui. Obviously, Tsutsui incurs less computational cost.

Chapter 9

MO simulation studies

An introduction to multi objective optimization is given in Section 3.3. There, we also introduce NSGA2, the multi objective evolutionary algorithm (MOEA) which we use in the simulation studies of this chapter. Section 9.1 outlines the experimental setup of the simulation studies. Section 9.2 presents the results.

9.1 Experimental setup

9.1.1 Requirements

In SO robustness optimization the test problem is required to provide a clear trade-off between f_{raw} and f_{exp}. In the MO case we want to analyze if an algorithm method finds a non-dominated set of solutions. In other words, we test whether the algorithm recognizes the trade-off between the objectives. In order to analyze this issue, a test problem must be chosen which provides a trade-off between the objectives. In our case this is a test problem which provides a trade-off between f_{exp} and f_{var}.

9.1.2 Test problems

We run experiments on two test problems. Both of which use the same test function f_2 which is defined

$$f_2(\vec{x}) \;=\; \sum_{1\le i\le n} 2.0\,\sin(\,10\,\exp(-0.08x_i)\,x_i\,)\,\exp(-0.25x_i) \tag{9.1}$$

where $\vec{x} \in [x_{min}; x_{max}]^n$. This test function is taken from [14] where it was used for MO robustness optimization using the *Neighbors evaluation* method. In raw fitness optimization the optimization problem is defined:

$$\min f_2(x) \quad s.t. \quad x \in [x_{min}; x_{max}]^n \tag{9.2}$$

However, for optimization of f_{exp} and f_{var} the constraint is difficult to define. As in the SO simulation studies we use a penalty which is described in Section 4.5. The test problem is chosen such that the constraint handling technique is not expected to have a significant influence on the search. For both test problems we set $\sigma_{noise} = 0.1$ (normally distributed) and $cutoff = 0.05$ for each dimension. We get a σ_{noise}-range of approximately $[-0.165; 0.165]^n$.

If the individuals are uniformly distributed over the search space, the expected number of individuals ($nearneighbors_{exp}$) which are located within the σ_{noise}-range of an individual $x^{(0)}$ can be calculated. We define p_{near} as the probability that an individual (of the current population) is located within the σ_{noise}-range of $x^{(0)}$.

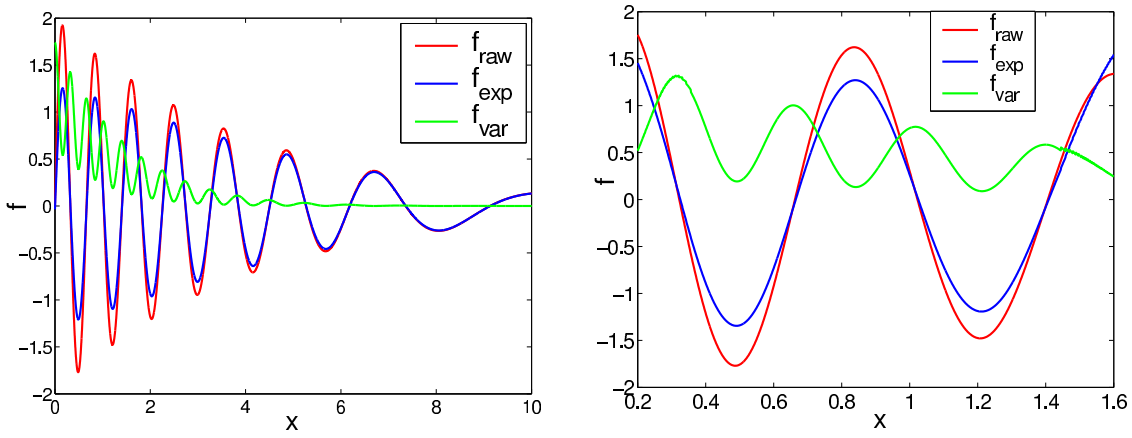
$$p_{near} \;=\; \frac{\sigma_{noise}\text{-range}}{(x_{max} - x_{min})^n} \quad , \tag{9.3}$$

and derive,

$$nearneighbors_{exp} \;=\; \sum_{1\le i\le m} i \cdot \binom{m}{i} \; {p_{near}}^i \,(1 - p_{near})^{(m-i)} \quad , \tag{9.4}$$

where $m = populationsize - 1$.

As Test problem 1 we choose $[x_{min}; x_{max}] = [0.0; 10.0]$. With this setting, $nearneighbors_{exp} = 0.11$ for dimension 2. In order to increase $nearneighbors_{exp}$, we reduce interval $[x_{min}, x_{max}]$ in test problem 2. In particular, we choose $[x_{min}; x_{max}] = [0.2; 1.6]$ and get $nearneighbors_{exp} = 5.5$ for dimension 2.



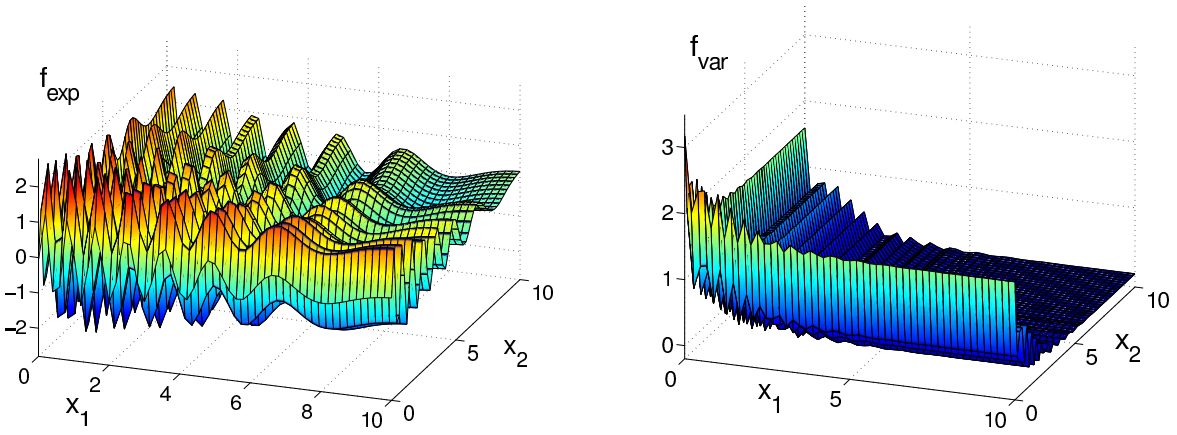
(a) Test problem 1 (b) Test problem 2

Figure 9.1: 1-dimensional Test function f_2 (Test problem 1 and 2)

For the 1-dimensional case, f_2 is depicted in Figure 9.1. We see a trade-off between f_{exp} and f_{var}. For illustration, consider an arbitrary local minimum: The nearest local minimum to the left has *lower* f_{exp} but *larger* f_{var}. The set of non-dominated solutions is given by the local minima. In Figure 9.2, f_{exp} and f_{var} of the 2-dimensional test function f_2 is depicted. We see that in regions where f_{var} is large, f_{exp} is very rugged.

9.1.3 Analyzed methods

We analyze the same methods as in the SO simulation studies (compare Section 8.2.3). Again, we use the term *Nearest model* for the special case of *Ensemble* with ensemble size 1 (PMD-ENS-1).



(a) f_{exp} (b) f_{var}

Figure 9.2: f_{exp} and f_{var} of the 2-dimensional test function for the MO simulation studies

9.1.4 Performance criteria

In Section 3.3 we introduced the *Non-dominated set* as a qualitative performance measure of MOEA's. Based on the data from the non-dominated set, we use an illustration method known as *attainment surface*. The concept of attainment surface is introduced by Fonseca and Fleming in [35]. Citing Fonseca and Fleming "the attainment surface is the boundary in the objective space separating those points which are dominated by or equal to at least one of the data points, from those which no data point dominates or equals"[1]. For an empirical analysis, an algorithm needs to be run multiple times with different random seeds to reduce the effect of randomness. Drawing all resulting attainment surfaces does not allow to compare the performance achieved by a number of different methods. Therefore we need to find a "typical" attainment surface of a number of runs. This allows to compare different methods in one figure. For this purpose, we compute the 50% attainment surface of a number of runs which is interpreted as *median attainment surface*. We refer to [35] for an illustrative introduction. In our simulation studies we

[1] [35] page 586

build the *median attainment surface* based on 11 runs with different random seeds.

In SO robustness optimization we select the final solution based on the estimation $\widehat{f}_{exp}$ and evaluate this with the real fitness. In the MO case we need to determine the final *set* of solutions. An analogous method for MO would be to find the non-dominated set of solutions based on the estimated fitness values of the final generation, and evaluate this set of solutions with the real fitness. With real fitness values the set might not be non-dominated anymore. Of course, one could again get the non-dominated front out of this set of solutions. However, the resulting sets are often very small. An analysis on *"How to select the final set of solutions"* is beyond the scope of this work. Therefore we use a simple method for determining the final set of solutions: All individuals of the final generation of each run are evaluated with the real fitness. Based on these fitness values the non-dominated front is computed.

9.1.5 Benchmark methods

As benchmark method we use *Neighbors evaluation* from Chapter 8 which was also used by Jin and Sendhoff [14] for MO robustness optimization. Additionally, we run the algorithm with the real fitness instead of approximations.

9.1.6 Parameter setting

For approximation and sampling we use an identical parameter setting as for the SO simulation studies (see Table 8.3). The setting of the NSGA2 parameters is given in Table 9.1. Note that instead of the SBX crossover in the original algorithm [16], conventional 1-point crossover and mutation have been adopted.

Table 9.1: Parameter setting for MO simulation studies

NSGA2 parameters	
representation	gray code
number of bits (representation)	30
crossover probability	0.9
number of cross points	1
flip probability (mutation)	0.01
population size	100
termination condition	50 generations (fixed)
infeasibility penalty	$2.0 \cdot$ *dimensions*

9.2 Results

9.2.1 Analysis of Neighbors evaluation method

Before comparing Neighbors evaluation to the approximation methods some properties are worth mentioning. In Figure 9.3 we see the median attainment surface for different settings of *numpoints_min* [2]. First of all, we see a significant difference between *numpoints_min* $= 1$ and *numpoints_min* $= 5$. This shows that only a small number of individuals are located in the neighborhood of an estimation point, otherwise increasing *numpoints_min* would not be effective.

In Section 9.1.2 we showed that the expected number of individuals which are located within the σ-range of an individual x_0 is very small in Test problem 1 if we assume uniform distribution of individuals. This assumptions holds in the initial generations of an EA. We calculate the expected number of individuals which are located in the neighborhood

[2] the minimum number of neighbors independent of the neighborhood size, see Section 8.2.5

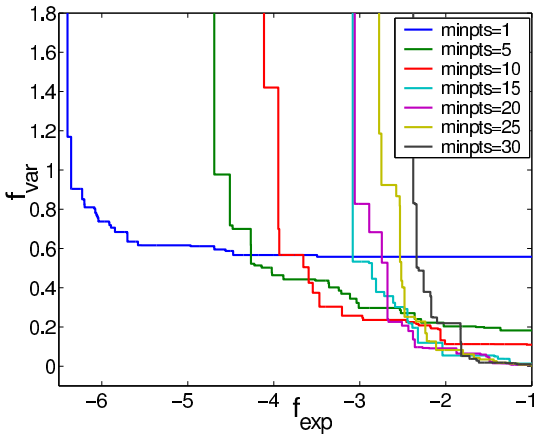


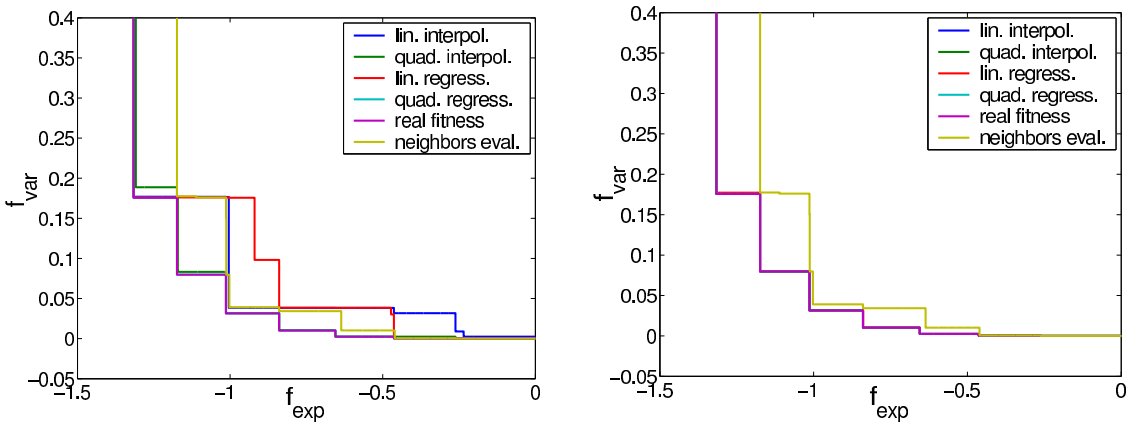
Figure 9.3: Test problem 1, dimension 5: Neighbors evaluation method for different *numpoints_min*

($nearneighbors_{exp}$) similarly as in Section 9.1.2 by using

$$p_{near} = \left(\frac{\text{neighborhood size}}{x_{max} - x_{min}} \right)^n , \tag{9.5}$$

instead of Equation 9.3. Recall that the borders of the neighborhood in the Neighbors evaluation method is represented by a n-dimensional sphere. Thus, the neighborhood size is the volume of a n-dimensional sphere.

By setting $numpoints_min = 1$ the estimation of f_{exp} and f_{var} is often based on a very small number of data points, or on just a single data point. In this case f_{var} would be estimated zero. From this perspective the poor performance with respect to f_{var} can be explained. Surprisingly, with $numpoints_min = 1$ Neighbors evaluation has high performance with respect to f_{exp}. This can be attributed to the test problem properties. In particular, the f_{raw} local optima equal the f_{exp} local optima. By using only a single individual for sampling which is located at a local optimum, f_{exp} is overestimated. However, on this test function this does not matter because at local minima f_{exp} is *consistently* underestimated. In order to deal with this side effect we set $numpoints_min = 5$ for all experiments. In Test problem 2 this effect is reduced due to a higher density of



(a) IMD-SM (b) IMD-MM

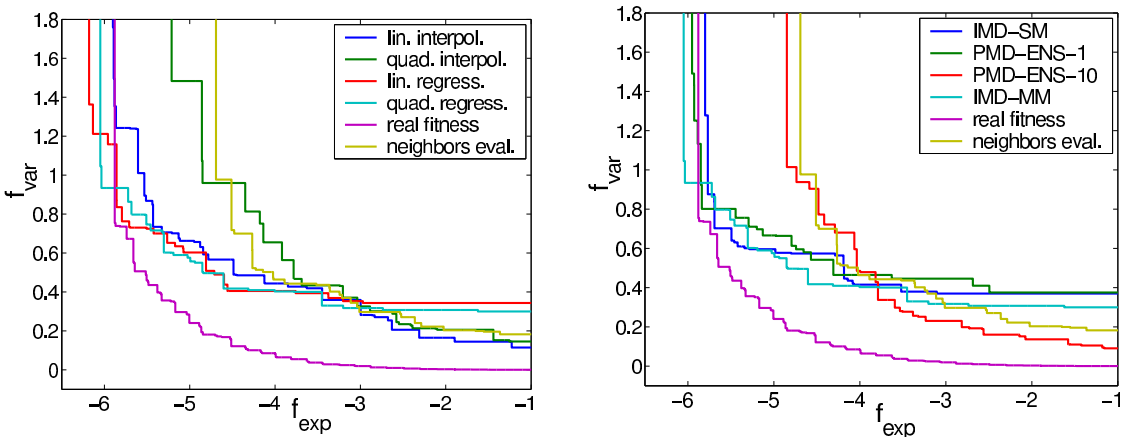
Figure 9.4: Test problem 1, dimension 1: Approximation methods, real fitness, Neighbors evaluation

individuals.

9.2.2 Performance of the approximation methods

Test problem 1

In Figure 9.4 we see the median attainment surfaces for all approximation methods for Single model (IMD-SM) and Multiple models (IMD-MM) compared to the performance achieved when using the real fitness and Neighbors evaluation in the 1-dimensional case. Many curves overlap and can not be distinguished visually. In IMD-MM all approximation methods achieve the same median attainment surface as the real fitness. Only Neighbors evaluation does not achieve the attainment surface of the real fitness. Although the 1-dimensional case seems to be a too simple test scenario, we already see that the Neighbors evaluation performs worse than the approximation methods. We also see that IMD-MM works better than IMD-SM. In particular, when using IMD-SM only quadratic regression draws an equal median attainment surface as the real fitness. This would be visible when removing the real fitness attainment surface from Figure 9.4(a). Figure 9.5 presents



(a) approximation models (IMD-MM) (b) estimation methods (quadr. regression)

Figure 9.5: Test problem 1, dimension 5

results from the simulation studies on the 5-dimensional Test problem 1. The results do not allow to draw conclusions on what is the preferred method in MO robustness optimization. None of the methods performs similar to the real fitness. Particularly, the estimations have difficulties in detecting individuals with low f_{var}. Surprisingly, the regression methods find solutions with larger f_{exp} than the real fitness. This can be explained as follows: As an estimation produces an error, some exploration is implicitly introduced. It is expected that the real fitness finds individuals with large f_{exp}, too, if the parameters of NSGA2 are modified such that it better explores the search space. From this observation we draw the conclusion that the parameters of an evolutionary algorithm should have different settings when using estimations compared to the case when using the real fitness.

In Figure 9.5a we see that all approximation models, except quadratic interpolation detect individuals with larger f_{exp}. The regression methods fail to find solutions with low f_{var}. In (b) we get a similar picture. Those methods which find good individuals with respect to f_{exp} fail to find individuals with low f_{exp}. As we see in Figure 9.2, the surface of Test problem 1 is very rugged already in dimension 2. All approximation models and

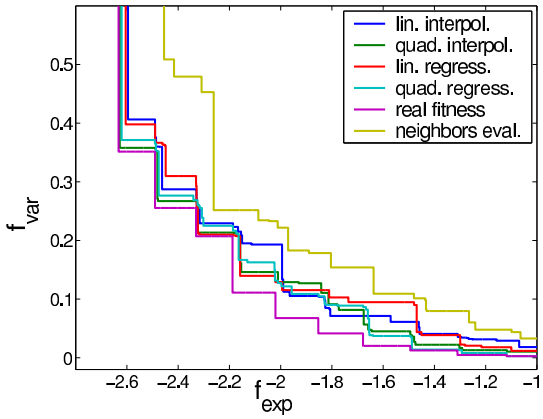


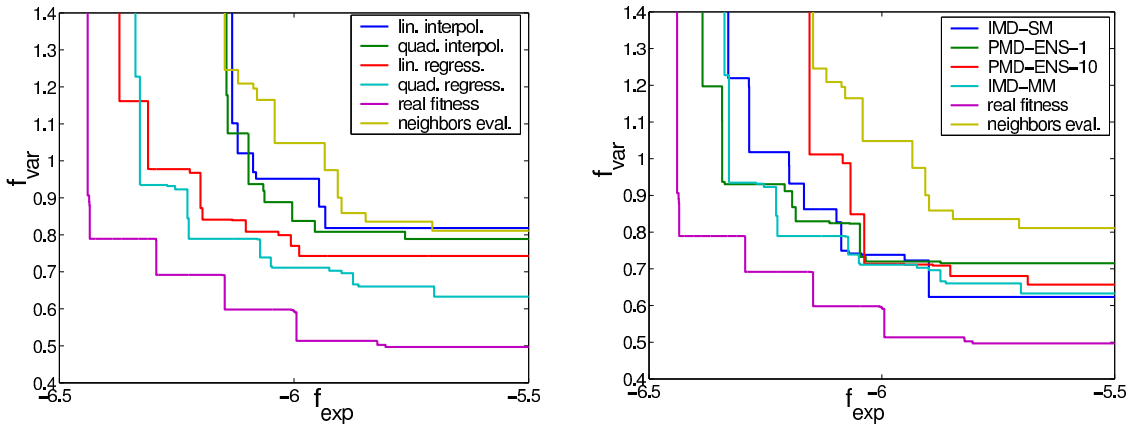
Figure 9.6: Test problem 1, dimension 2, IMD-MM: approximation models, real fitness, Neighbors evaluation

estimation methods have difficulties in higher dimensions on Test problem 1.

In order to derive conclusions for the relative performance of the approximation models and Neighbors evaluation, we analyze another low-dimensional case: In Figure 9.6 the results for dimension 2 when using IMD-MM are depicted. Compared to the 1-dimensional case (Figure 9.4), we find the gap between the median attainment surfaces of the real fitness and the estimations to be widened. We further see that all approximation models perform better than the Neighbors evaluation method. However, the results of Test problem 1 do not allow to draw conclusions for higher dimensions. Test problem 1 does not seem to be useful for comparison of different methods in higher dimensions.

Test problem 2

Compared to Test problem 1, Test problem 2 is easier to solve. Unfortunately, the trade-off between f_{exp} and f_{var} is less clear than for Test problem 1. Although Test problem 2 provides a set of only 2 non-dominated solutions in dimension 1, in higher dimensions the set of non-dominated solutions has sufficiently large size. The results are shown in Figure 9.7. Here we draw median attainment surfaces for the 5-dimensional Test problem 2.



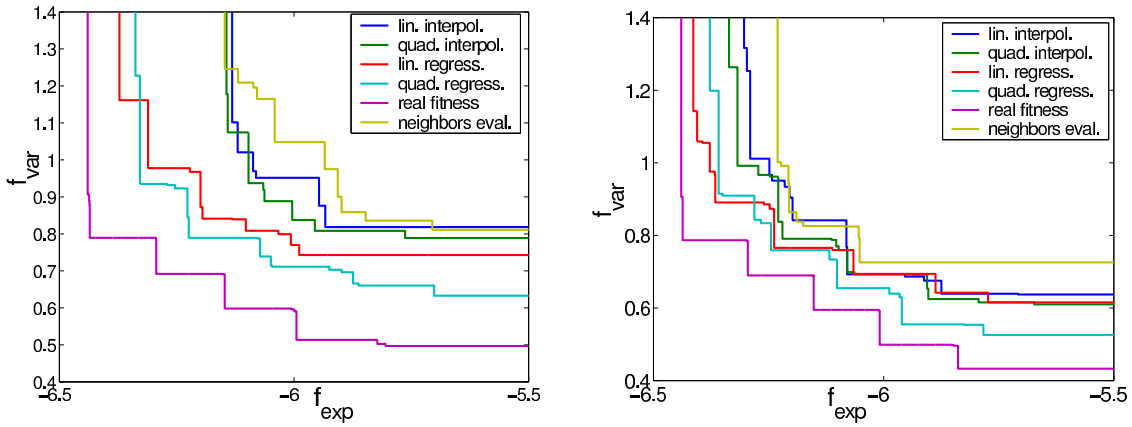
(a) approximation models (IMD-MM) (b) estimation methods (quadr. regression)

Figure 9.7: Test problem 2, dimension 5

In (a) we compare the different approximation models to Neighbors evaluation and the real fitness when using IMD-MM. Without using quantitative metrics one can conclude from the figures that regression is the method of choice. Although worse than regression, the interpolation methods work better than the Neighbors evaluation method. In (b) we compare different estimation methods for quadratic regression. We have difficulties to clearly conclude which estimation method works best. It seems that IMD-MM performs slightly better than IMD-MM and PMD-ENS-1. Somewhat consistently to the findings on Test problem 1, we conclude that PMD-ENS-10 is no good estimation method for MO robustness optimization.

9.2.3 Selecting the final set of solutions

When using estimations in a SOEA, the question *"How to select the final solution?"* is highly important. A discussion on this question can be found in [22]. When using estimations in a MOEA, the question is *"How to select the final set of solutions?"*. This topic is even more interesting under the assumption that fitness evaluations are very expensive. An analysis on these issues is beyond the scope of this work. However, we show with an



(a) real fitness of final generation

(b) real fitness of all generations

Figure 9.8: Final set of solutions (Test problem 2, dimension 5, IMD-MM): Final generation (a) vs. all generations (b)

example how the selection of the final set of solutions influences the results. In Figure 9.8, we show the median attainment surfaces based on two different collections of data for the 5-dimensional Test problem 2. In (a), we store the individuals of the final generation, evaluate them with the real fitness and compute the median attainment surface based on the real fitness evaluations. In (b) we store the individuals of all generations throughout the run, evaluate them with the real fitness and compute the median attainment surface. As the final generation is a subset of all generations the median attainment surfaces in (b) are located south-east from those in (a). Of course the scenario in (b) is not realistic for real world applications, otherwise the real fitness could be used instead of approximations to guide the search. A large discrepancy between fronts in Figures 9.8(a) and 9.8(b) is an indicator for a high sensitivity to the choice of the final set of solutions. One can see that the regression methods are insensitive, whereas interpolation as well as Neighbors evaluation are more sensitive to the choice of the final set of solutions. In interpolation, this can be attributed to a higher probability of significant wrong estimation. However, the performance ranking remains equal, i.e. regression performs better than interpolation.

Interpolation, still performs better than Neighbors evaluation. Further investigation of this issue is reserved for future research.

Chapter 10

Summary and Outlook

10.1 Summary

The goal of this work was to investigate fitness surface approximation techniques (RSM) which can be used to improve evolutionary search for robust solutions.

For this purpose we developed approximation methods, namely *nearest neighbor interpolation* and *local regression* for different polynomials within the framework of an evolutionary algorithm. In preliminary experiments in the 1-dimensional case, we additionally analyzed *natural neighbor interpolation* and concluded that this interpolation technique does not improve the estimation quality compared to the standard nearest neighbor interpolation. The transfer of this conclusion to higher-dimensional cases is questionable. However, even if natural neighbor interpolation improves the estimation quality in higher dimensions, the high computational complexity is not acceptable. As input data to the approximation models we used the solutions that were visited by the evolutionary algorithm. This data collection strategy represents a challenge to the approximation methods, as numerical difficulties are likely. Thus, additional features to the standard approximation algorithms were necessary, in order to make the approximations reliable in the framework

of an evolutionary algorithm. Based on fitness approximations, we developed several estimators for the *expected fitness* f_{exp} and the *fitness variance* f_{var} which both can be used as objectives in robustness optimization. The estimators use the models which are provided by the approximation methods in various ways and can be categorized into *individual based* and *population based model distribution* estimation methods. Depending on the number of models which need to built per generation, the estimators incur different levels of computational cost. In order to reduce the computational cost and to be able to carry out estimations in higher dimensions we used *Latin hypercube sampling*.

In most of the literature, robustness optimization is treated as single objective optimization problem, with f_{exp} as optimization criterion. This implicitly assumes that the decision maker is risk-neutral. If, however, the decision maker is risk-averse which is realistic in most real world applications, f_{var} is an additional objective which is to be optimized. We therefore carried out simulation studies for both, *single objective optimization* (f_{exp}) and *multi objective optimization* (f_{exp} and f_{var}).

One of the most important findings was that the essential property of an estimator is the *standard deviation* σ of the *estimation error*. We found empirical as well as theoretical evidence for a positive correlation between a low σ of the estimation error and a high performance of the evolutionary algorithm. Especially in an EA environment, σ of the estimation error, is influenced by correlations between input data, respectively between input models, of the estimators.

In the SO simulation studies we found the regression methods to clearly outperform interpolation. This is at first sight surprising because the goal of regression is the smoothening of a set of fitness values which are assumed to be stochastic. In robustness optimization, however, the fitness function is not stochastic, but the decision variables. Thus, there exists no noise that could be taken out by regression. The reason, why regression per-

forms better than nearest neighbor interpolation was found by analyzing the respective estimation error properties: Nearest neighbor interpolation is likely to produce severe wrong estimation if the distribution of possible model input data is unfavorable. This was particularly observable in higher dimensions. The quadratic regression model never performed worse than the linear regression model, and seems to work better in higher dimensions. However, the quadratic model incurs significantly higher computational cost. Comparing the different estimation methods, we found the individual based approach *Multiple models* to perform best in most scenarios. Somewhat surprising, the simple population based approach which we denoted *Ensemble* performs well when applied in combination with regression. Although in most cases the individual based *Multiple models* approach performed slightly better, there exist scenarios in which Ensemble achieves better results than Multiple models. We conclude that this is due to the low σ of the estimation error in the Ensemble method which is achieved by weighted averaging over a set of approximation models. We further conclude that the idea of *sharing models* (population based model distribution) has the potential to improve the performance. In the SO simulation studies, we finally compared our new techniques to *Neighbors evaluation* (used similarly in [14] and [7]) and a *Tsutsui*-like method [1]. On our test problem the new techniques clearly outperform Neighbors evaluation and Tsutsui, particularly in higher dimensions.

The conclusions we draw from the experiments in the MO case are not as clear as in the SO case. One difficulty was to find a test function which allows to draw reliable conclusions from comparisons of different estimation methods in higher dimensions. In particular, such a test function must provide a clear trade-off between f_{exp} and f_{var} but the best f_{exp} local optima should be different from the best f_{raw} local optima. However, in the test problems we used, quadratic regression in combination with Multiple models performed best. This is consistent with the results from SO experiments. Furthermore,

we saw that estimations implicitly introduce exploration and conclude that parameters of an evolutionary algorithm (both SO and MO) must be set carefully when using estimations instead of the real fitness. Our new methods outperform the Neighbors evaluation method in MO, too.

10.2 General remarks

Some thoughts regarding the assumptions for research on robustness optimization are worth mentioning. With regard to many real world problems, fitness evaluations are assumed to be very expensive. In this scenario the number of available input data to approximation models is very limited. At the same thime, the noise level is often assumed to be relatively low compared to the size of the search space. When using approximations, it is desirable that the resulting models approximate the surface sufficiently accurate in the noise range of the individuals. If, however, the search space is only sparsely filled with known surface points, the accuracy has strong theoretical limitations. This of course also holds if the noise level is high. However, if this is the case, noise ranges of different local optima might overlap. Thus, data points can be used for estimations at different local optima. Of course estimation accuracy strongly depends on how rugged the fitness landscape is. In order to achieve reliable results on real world problems, we believe that a substantially larger amount of fitness evaluations is necessary.

However, in many high dimensional real world problems, noise is not present in all dimensions, but only a small number of dimensions are noisy. Accounting for this, the problem complexity is reduced. We tested problems with noise in up to 10 dimensions. This might cover a large fraction of real world problems.

10.3 Outlook

In future research the potential of population based model distribution needs to be exploited. Based on an improved (population based) distribution of approximation models, the *Ensemble* method might work even better. *Clustering* is a candidate for further improvements: In densely populated regions the number of approximation models per individual could be reduced, at the same time in sparsely populated regions the number of models per individual could be increased.

If the computational cost of computing approximation models are negligible, the individual based model distribution technique Multiple models can be improved by using elements of the Ensemble idea. In particular, at each approximation point, a set of approximation models is built, each using different input data, if available.

Another way of improving the exploitation of available information, is to use the set of known surface points more directly: If in the neighborhood of an approximation point, a true surface point $x^{(true)}$ is available from previous evaluations, the estimation procedure can be improved by using $x^{(true)}$ instead of an approximation. In a next step, the estimation technique could adapt to the availability of known surface points.

Another interesting issue for future research is related to strategies for collecting fitness surface points. In our work we used a rather straight-forward technique. In a first step our method could be improved by defining a maximum distance between data that are to be evaluated. In a second step the time savings could be used to do additional evaluations in interesting regions.

Finally, the question on *"how to select the final set of solutions?"* in MO robustness optimization is highly interesting for future work. One related aspect is the prediction of estimation errors. This information can then be used to decide whether extra evaluations for the final generation are necessary in order to get a more reliable set of non-dominated solutions.

This work has shown the potential of fitness surface approximation in evolutionary search for robust solutions and has found ideas for improvement. Further development of the presented ideas seems to be very promising.

Appendix A

Statistical significance tests

Tables A.2- A.4 show the significance tests which were done for the EA runs in Chapter 8. The hypothesis test whether the BRA difference of two methods is significant. Throughout all experiments we set the number of runs per parameter setting to 20. Therefore a one-sided t-test is the method of choice. With 20 runs per setting, the number of degrees of freedom is $20 + 20 - 2 = 38$. The important quantiles of the t-distribution are shown in Table A.1.

In Tables A.2- A.4 the "No" column assigns numbers to the tests. The "(X,Y)" column denotes the properties that are common to both estimation methods that are to be compared. The "X" and the "Y" column denote the distinctive properties. "H_0" is the null hypothesis, H_1 the alternative hypothesis. $\bar{X}, \bar{Y}, s_X^2, s_Y^2$ denote empirical mean and empirical variance. T shows the resulting T-score.

Table A.1: Quantiles of the t-distribution (38 DoF)

α	0.05	0.025	0.01
$t_{38;1-\alpha}$	1.6860	2.0244	2.4286

Table A.2: Student-T significance tests dimension 2

No	(X,Y)	X	Y	H_0	H_1	$\bar{X}$	$\bar{Y}$	s_X^2	s_Y^2	T
1	LHC	best Sing	best Mult	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-1.05	-1.38	0.31	0.001	4.85
2	LHC	best Sing	best Near	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-1.05	-0.99	0.31	0.46	-0.48

Table A.3: Student-T significance tests dimension 5

No	(X,Y)	X	Y	H_0	H_1	$\bar{X}$	$\bar{Y}$	s_X^2	s_Y^2	T
3	Strat, Ens(10)	best intp	best regr	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-0.74	-2.70	1.06	0.774	6.68
4	LHC, Ens(10)	best intp	best regr	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-0.97	-3.11	0.23	0.73	12.52
5	Strat, Mult	intp	regr	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-2.71	-3.25	0.80	0.53	2.53
6	LHC, Mult	intp	regr	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-2.47	-3.29	0.88	0.37	3.84
7	LHC, Ens	best lin	best quad	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-2.97	-3.1	0.37	0.23	1.49
8	LHC, Mult	intp	regr	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-3.25	-3.06	0.53	0.88	-0.81
9	LHC	best Sing	best Mult	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-2.16	-3.25	1.30	0.53	3.48
10	LHC	best Sing	best Near	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-2.16	-2.27	1.30	1.07	0.30

Table A.4: Student-T significance tests dimension 10

No	(X,Y)	X	Y	H_0	H_1	$\bar{X}$	$\bar{Y}$	s_X^2	s_Y^2	T
11	LHC, best lin	best Mult	best Ens	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-3.31	-5.14	0.72	1.14	6.07
12	LHC, best quad	best Mult	best Ens	$\mu_X \leq \mu_Y$	$\mu_X > \mu_Y$	-3.55	-5.86	1.00	0.73	8.35

Table A.6 shows the significance tests for the results of the σ *measure experiment* in Chapter 8. The hypothesis test whether the difference in $\sigma_{e_{exp}}$ (the standard deviation of the estimation error) is significant. The method of choice is a one-sided Fisher test. As the estimation is based on 5000 samples for each setting, each random variable has $5000 - 1 = 4999$ degrees of freedom. The important quantiles of the f-distribution are shown in Table A.1.

Table A.6 uses the same notations as Tables A.2 - A.4. Additionally, "dim" denotes the problem dimension and "hist" the number of history data points on which the σ *measure experiment* is based. Instead of the variance s_X^2, s_Y^2, the standard deviation s_X, s_Y is depicted in Table A.6. The last coloumn shows the resulting F-score for each test.

Table A.5: Quantiles of the f-distribution (both variables with 4999 DoF)

α	0.05	0.025	0.01
$F_{4999,4999;1-\alpha}$	1.05	1.06	1.07

Table A.6: Fisher significance tests

No	dim	hist	(X,Y)	X	Y	H_0	H_1	s_X	s_Y	F
13	5	1000	LHC, mult, lin	intpol	regr	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.376	0.217	2.99
14	5	1000	LHC, mult, quad	intpol	regr	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.474	0.300	2.49
15	5	10000	LHC, mult, lin	intpol	regr	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.251	0.181	1.93
16	5	10000	LHC, mult, quad	intpol	regr	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.277	0.126	4.80
17	5	1000	LHC, mult, regr	quad	lin	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.300	0.217	1.908
18	5	10000	LHC, mult, regr	lin	regr	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.181	0.126	2.041
19	5	1000	LHC, single, regr	quad	lin	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.285	0.238	1.438
20	5	10000	LHC, single, regr	lin	quad	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.255	0.229	1.245
21	5	1000	LHC, quad, regr	mult	ens	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.300	0.209	2.060
22	10	1000	LHC, quad, regr	mult	ens	$\sigma_X \leq \sigma_Y$	$\sigma_X > \sigma_Y$	0.728	0.388	3.516

Appendix B

Implementation

The simulation studies are based on a C++ implementation. Whenever possible we used well known programming libraries. In particular, for linear algebra methods we used *GNU Scientific Library* which is an open source project and therefore freely available. For most of the functionalities in the evolutionary optimization part we used *EALib*, an open source project from the Institut für Neuroinformatik at the Ruhr-Universität Bochum. For the multi-objective EA part we used an extension of EALib, *MOOEALib*. MOOEALib was developed at the Honda Research Institute Europe by Tatsuya Okabe and is also freely available. The author wishes to thank the developers of the libraries.

Bibliography

[1] S.Tsutsui and A.Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, pages 201–208, 1997.

[2] S.Tsutsui and L.C.Jain. Properties of robust solution searching in multi-dimensional space with genetic algorithms. In *2nd International Conference on Knowledge-Based Intelligent Electronic Systems*, 1998.

[3] I.C.Parmee, M.Johnson, and S.Burt. Techniques to aid global search in engineering design. In *Proceedings of International Conference on Industrial and Engineering Applications of AI and Expert Systems*, pages 377–385, 1994.

[4] M.Tjornfeld-Jensen. Robust solutions to job shop problems. *IEEE Transactions on Evolutionary Computation*, 2000.

[5] M.McIlhagga, P.Husbands, and R.Ives. A comparison of search techniques on a wing-box optimisation problem. In H.-M.Voigt, editor, *Parallel Problem Solving from Nature 4*, number 1141 in LNCS, pages 614–623, 1996.

[6] A.Thompson. Evolutionary techniques for fault tolerance. In *Proc. UKACC International Conference on Control*, pages 693–698, 1996.

[7] J. Branke. Creating robust solutions by means of an evolutionary algorithm. In A.E.Eiben, T.Baeck, M.Schoenauer, and H.-P.Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V*, pages 119–128. Springer, 1998.

[8] J.Branke. Reducing the sampling bias variance when searching for robust solutions. In L.Spector, editor, *Genetic and Evolutionary Computation Conference (GECCO '01)*, pages 235–242. Morgan Kaufmann, 2001.

[9] J.E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, 1998.

[10] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*, 2003. In press.

[11] Y.Jin, M.Olhofer, and B.Sendhoff. On evolutionary optimization with approximate fitness functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 786–792, 2000.

[12] L.Willmes, T.Baeck, Y.Jin, and B.Sendhoff. Comparing neural networks and kriging in fitness approximation in evolutionary optimization. In *IEEE Congress on Evolutionary Computation*, pages 663–670, 2003.

[13] M.Olhofer, T.Arima, T.Sonoda, and B.Sendhoff. Optimization of a stator blade used in a transonic compressor cascade with evolution strategies. *Adaptive Computation in Design and Manufacture*, pages 45–54, 2000.

[14] Y.Jin and B.Sendhoff. Trade-off between optimality and robustness: An evolutionary multiobjective approach. In C.M. Fonseca et al., editor, *Proceedings of the Second International Conference on Evolutionary Multi-criterion optimization*, pages 237–251, 2003.

[15] W.Chen, J.Allen, K.Tsui, and F.Mistree. A procedure for robust design: minimizing variations caused by noise factors and control factors. *ASME Journal of Mechanical Design*, pages 478–485, 1995.

[16] K. Deb et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, 2000.

[17] A.Giunta and L.Watson. A comparison of approximation modeling techniques: Polynomial versus interpolating models. Technical Report 4758, American Institute of Aeronautics and Astronautics, 1998.

[18] T.Simpson, M.Maurey, J.Korte, and F.Mistree. Comparison of response surface and kriging models for multidisciplinary design optimization. Technical Report 4755, American Institute of Aeronautics and Astronautics, 1998.

[19] J-D. Boissonnat and F.Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *ACM Symposium on Computational Geometry*, pages 185–203, 2002.

[20] C.Darwin. *On the origin of species*. Harvard University Press, 1979. written 1795, online available http://www.literature.org/authors/darwin-charles/the-origin-of-species/.

[21] T.Baeck, D.Fogel, and Z.Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[22] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2002.

[23] K.Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[24] T.Okabe, Y.Jin, and B.Sendhoff. A critical survey of performance indices for multi-objective optimization. In *IEEE Congress on Evolutionary Computation*, pages 878–885, 2003.

[25] R.Meyers and D.Montgomery. *Response surface methodology: process and product optimization using designed experiments*. Wiley series in probability and statistics. Wiley, 2nd edition, 2002.

[26] M.Kreutz, B.Sendhoff, and C.Igel. Ealib: A c++ class library for evolutionary algorithms. Technical report, Institut fuer Neuroinformatik, Ruhr-Universitaet Bochum, 2000. version 1.5.

[27] T.Ray. Constrained robust optimal design using a multiobjective evolutionary algorithm. In D.Fogel, M.El-Sharkawi, X.Yao, G.Greenwood, H.Iba, P.Marrow, and M.Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 419–424. IEEE Press, 2002.

[28] S.Arya, D.M.Mount, N.S.Netanyahu, R.Silverman, and A.Y.Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, pages 891–923, 1998.

[29] J.Stoer and R.Bulirsch. *Introduction to Numerical Analysis*. Springer, 3rd edition, 2002.

[30] J.E. Gentle. *Numerical Linear Algebra for Applications in Statistics*. Springer, 1st edition, 1998.

[31] G.H.Goloub and C.F.Van Loan. *Matrix Computations*. The John Hopkins Press, 3rd edition, 1996.

[32] W.Press, B.Flannery, S.Teukolsky, and W.Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[33] R. Sibson. A brief description of natural neighbor interpolation. In V Barnet, editor, *Interpreting Multivariate Data*, pages 21–36. John Wiley, 1981.

[34] F.Preparata and M.Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[35] C. Fonseca and P. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Parallel Problem Solving from Nature - PPSN IV*, pages 584–593, 1996.

Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabesteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 31.Mai 2004, Ingo Pänke