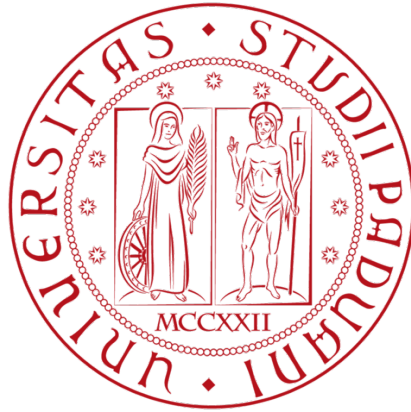


UNIVERSITÀ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria Civile, Edile Ambientale
Department of Civil, Environmental and Architectural Engineering
Corso di Laurea Magistrale in Ingegneria Matematica



Performance assessment of Surrogate model integrated with sensitivity analysis in multi-objective optimization

Relatore:

Chiar.mo Prof. Ernesto Benini

Laureando:

Federico Genovese

Matricola:

1129918

Anno Accademico 2016–2017

*Ai miei genitori,
Morena e Flavio,
e ad Alessia.*

Abstract

This Thesis develops a new multi-objective heuristic algorithm. The optimum searching task is performed by a standard genetic algorithm, the *Non-dominated Sorting Algorithm II (NSGA-II)*. Furthermore, it is assisted by the Response Surface Methodology surrogate model and by two sensitivity analysis methods: the *Variance-based*, also known as Sobol' analysis, and the *Elementary Effects*. Once built the entire method, it is compared on several multi-objective problems with some other algorithms, also these based on evolutionary searching process and on surrogate models. Finally fitting qualities of Response Surface method is tested on an experimental dataset, to measure its predicting features on external data. The final results show that this new meta-heuristic performs well compared to other algorithms and it also seems to be cheap, talking about the computational costs.

This work deals in the first chapters with theoretical aspects, then it introduces the developed model and finally it reports the results of tests on multi-objective problems and of the fitting over the external dataset.

Chapter 2 introduces the mathematical description of a multi-objective optimization problem. Then it describes the concepts of Pareto optimality, which involves the dominance between solutions and the set of best elements. Successively it deals with the main problems arising working in this framework: the curse of dimensionality and the *No free lunch* theorem. Last, the chapter introduces the metrics to evaluate the performances of the optimization algorithms.

Chapter 3 describes the main features of the evolutionary algorithms and their typical frame. Then the chapter goes more in detail through the structures of the most spread algorithms. Here the features and the building blocks of each routine are explained to highlight the positive aspects and the drawbacks of each model. In the end of the chapter some non-evolutionary algorithms are introduced. However, these are widely used or much near to the evolutionary ones.

In chapter 4 the typical frames of surrogate model are described and few of them are reported in detail. These are the Artificial Neural Network, the Kriging filter and the Response Surface Method, which it is adopted in the developed model. Finally, there is a brief section which treats the classification of meta-heuristic hybrid optimization methods.

Chapter 5 regards the sensitivity analysis in all its features. First, it reports the main sampling methods with their pros and cons. Successively the most used sensitivity analysis methods are described, dealing also with the mathematical concepts.

In chapter 6 the building of the developed methods it is explained in details. First of all, it introduces the general results of sensitivity analysis for the test problems at hand. Then it treats in each feature the Response Surface Methodology and another point worth highlighting is the new parameter differing from the literature suggestion.

Finally, it is described also the genetic algorithm used to perform the optimization searching process.

Chapter 7 deals with the results of the optimization task, both for two- and three-objectives test functions. However, before it reports the results, there are introduced the algorithms which allow the comparison. Then just two problems are reported, one from two-objectives test and one from the three-objectives. The complete definition of test functions can be found in appendix A, while the complete results in appendix B.

Finally, chapter 8 analyses the behaviour of the Response Surface on the external dataset. First, it describes the nature of the dataset and its application. Successively, the chapter introduces the methods with which will be evaluated the goodness of predicted data from the Response Surface. Therefore, few results are reported and commented for each method. Again, the complete results can be found in appendix, in chapter C.

Acknowledgements

This work comes up from the knowledges acquired during my entire academic studies, but also due to the curiosity and the appeal that this topic inspired me.

A huge thank goes to all those who make this goal possible, sustaining me during all these years: my parents and my family, all my friends and the class mates, with which I shared disappointments and satisfaction and all the people who have been next to me.

Particular thanks go to Prof. Benini for his valuable suggestions and his kindness. I would also like to extend my thanks to Dott. Venturelli, who shares useful his knowledge, several information and data.

Finally, I wish to thanks all those people who help me with their suggestion to build up, review and correct this work.

Contents

Abstract	v
Acknowledgements	vii
1 Introduction	1
1.1 Objective of the Thesis	1
1.2 Summary of the Thesis	1
2 Multi-objective optimization	3
2.1 Mathematical description	4
2.1.1 Pareto optimality	5
2.1.2 Curse of dimensionality	7
2.1.3 No free-lunch theorem	8
2.2 Algorithm evaluation metrics	9
3 Evolutionary Algorithms	13
3.1 Evolutionary Algorithms	13
3.1.1 Main features	13
3.2 Genetics algorithm	15
3.3 Particle swarm optimization	20
3.4 Differential evolution	22
3.5 Shuffled frog leaping	25
3.6 Ant colony optimization	27
3.7 Other meta-heuristic methods	28
4 Surrogate Models	33
4.1 Main features	33
4.2 Kriging	36
4.3 Artificial Neural Networks	41
4.4 Response Surface Methodology	44
4.4.1 Local reference frame and sampling	47
4.4.2 Surface fitting	48
4.5 Meta-heuristic hybrid optimization	49
5 Sensitivity Analysis	55
5.1 Purpose of the sensitivity analysis	55
5.2 Sampling technique	56
5.2.1 Random sampling	56

5.2.2	Factorial sampling	57
5.2.3	Latin hypercube sampling	58
5.2.4	Multivariate stratified sampling	59
5.2.5	Quasi-random sampling	60
5.3	Sensitivity analysis methods	64
5.3.1	One at a time	64
5.3.2	Elementary Effects	65
5.3.3	Variance-based method	67
5.3.4	Derivative-based method	70
6	Model Implementation and Application	73
6.1	Sensitivity analysis application	73
6.1.1	Sobol' sensitivity analysis	74
6.1.2	Morris sensitivity analysis	77
6.2	Response Surface modelling	81
6.3	Genetic algorithm application	85
7	Result and comparison on Test Function	87
7.1	Sensitivity analysis results	87
7.2	Response Surface configuration	88
7.3	Final results	90
7.3.1	Analysis of two-objective test functions	91
7.3.2	Analysis of three-objective test functions	94
8	Analysis of the Response Surface on an experimental dataset	97
8.1	Fitting and testing sets	98
8.2	One point excluded test	100
8.2.1	Global response surface result	101
8.2.2	Local response surface result	101
8.3	Brief conclusions on response surface methods	102
9	Conclusions	105
A	Test Functions	107
A.1	Two-objectives test functions	107
A.2	Three-objective test functions	109
B	Plots and Data of Test Functions	113
B.1	Results of two-objective problems	113
B.2	Results of three-objective problems	120
C	Box-plots of errors on the experimental dataset	129
C.1	Fitting and testing sets box-plots	129
C.2	One point excluded global tests box-plots	132
C.3	One point excluded local tests box-plots	135
	Bibliography	139

List of Figures

2.1	Pareto Front	6
2.2	MOOP evaluation metrics	9
3.1	Crossover techniques	18
4.1	Parameters θ and p	38
4.2	ANN classification	42
4.3	Sigmoid and Hyperbolic tangent functions	43
4.4	Response surface example	46
4.5	Flow diagram of a Meta-heuristic hybrid optimization	51
4.6	Meta-heuristic calssification	53
5.1	Factorial design	57
5.2	Multivariate stratified sampling	60
5.3	Discrepancy	61
5.4	Sample size of quasi-random sequences	61
6.1	Histogram of Sobol' sensitivity analysis	75
6.2	Variation on sample dimension for Sobol' analysis	75
6.3	Histogram of second order Sobol' sensitivity analysis	76
6.4	Histogram of Elementary Effects analysis	78
6.5	Histogram of Elementary Effects analysis by groups	79
6.6	Variation on sample dimension for Morris analysis	80
6.7	Variation of secondary factors fixed parameter on 2d problem	83
6.8	Variation of secondary factors fixed parameter on 3d problem	84
7.1	ZDT1 Pareto fronts	92
7.2	ZDT1 box-plot of Dmetric and HVnorm	92
7.3	DTLZ2 Pareto fronts	94
7.4	DTLZ2 box-plot of Dmetric and HVnorm	95
8.1	Airfoil profile	97
8.2	Morphing part of airfoil profile	98
8.3	Error on fit&test method response surface of 5 variables	99
8.4	Error on fit&test method response surface of 6 variables	99
8.5	Mean error Vs number of variables on fit&test method	100
8.6	Global response surface of 6 variables	101
8.7	Mean error Vs number of variables in global surfaces	102
8.8	Local response surface of 6 variables	102
8.9	Mean error Vs number of variables in local surfaces	103

B.1	ZDT2 Pareto fronts	113
B.2	ZDT2 box-plot of Dmetric and HVnorm	114
B.3	ZDT3 Pareto fronts	115
B.4	ZDT3 box-plot of Dmetric and HVnorm	115
B.5	ZDT4 Pareto fronts	116
B.6	ZDT4 box-plot of Dmetric and HVnorm	117
B.7	ZDT6 Pareto fronts	118
B.8	ZDT6 box-plot of Dmetric and HVnorm	118
B.9	DTLZ1 Pareto fronts	120
B.10	DTLZ1 zoom on the Pareto front	120
B.11	DTLZ1 box-plot of Dmetric and HVnorm	121
B.12	DTLZ3 Pareto fronts	122
B.13	DTLZ3 zoom on the Pareto front	122
B.14	DTLZ3 box-plot of Dmetric and HVnorm	123
B.15	DTLZ5 Pareto fronts	124
B.16	DTLZ5 zoom on the Pareto front	124
B.17	DTLZ5 box-plot of Dmetric and HVnorm	125
B.18	DTLZ6 Pareto fronts	126
B.19	DTLZ6 box-plot of Dmetric and HVnorm	126
B.20	DTLZ7 Pareto fronts	127
B.21	DTLZ7 box-plot of Dmetric and HVnorm	128
C.1	Error on fit&test method response surface of 1 variable	129
C.2	Error on fit&test method response surface of 2 variables	130
C.3	Error on fit&test method response surface of 3 variables	130
C.4	Error on fit&test method response surface of 4 variables	131
C.5	Global response surface of 1 variable	132
C.6	Global response surface of 2 variables	132
C.7	Global response surface of 3 variables	133
C.8	Global response surface of 4 variables	133
C.9	Global response surface of 5 variables	134
C.10	Local response surface of 1 variable	135
C.11	Local response surface of 2 variables	135
C.12	Local response surface of 3 variables	136
C.13	Local response surface of 4 variables	136
C.14	Local response surface of 5 variables	137

List of Tables

4.1	Axis sample	48
4.2	X matrix construction	49
5.1	Fractional design	58
5.2	Doubled Latin Hypercube sampling	59
5.3	Halton sequence	63
5.4	Sobol sequence	64
6.1	Sobol' analysis results	74
6.2	Sobol' second-order analysis result	76
6.3	Morris sensitivity analysis	77
6.4	Morris sensitivity analysis by group	78
6.5	Number of generation	86
7.1	Sensitivity analysis of ZDT3 and DTLZ2	88
7.2	Surface errors variability on two-objective problems	89
7.3	Surface errors variability on three-objective problems	89
7.4	ZDT1 data of Dmetric and HVnorm	93
7.5	DTLZ2 data of Dmetric and HVnorm	95
B.1	ZDT2 data of Dmetric and HVnorm	114
B.2	ZDT3 data of Dmetric and HVnorm	116
B.3	ZDT4 data of Dmetric and HVnorm	117
B.4	ZDT6 data of Dmetric and HVnorm	119
B.5	DTLZ1 data of Dmetric and HVnorm	121
B.6	DTLZ3 data of Dmetric and HVnorm	123
B.7	DTLZ5 data of Dmetric and HVnorm	125
B.8	DTLZ6 data of Dmetric and HVnorm	127
B.9	DTLZ7 data of Dmetric and HVnorm	128

Chapter 1

Introduction

1.1 Objective of the Thesis

This work aims to develop a new meta-heuristic evolutionary optimization tool building together a surrogate model with sensitivity analysis. While the surrogate is a tool quite diffused in the optimization frameworks, sensitivity analysis is often accounted as a mere statistical one. The coupling of these two implementations can lead to effective information and results.

Besides, they could provide some sort of simplification to the optimization at hand. Building this optimization tool, a further aspect to deal with is the computational cost. Nowadays many high-fidelity models reach astonishing results, e.g. they manage to reproduce with computer-based experiments almost exactly the features of real-based ones, while they also retrieve larger quantity of data than the real experiments. On the other hand, these high-fidelity models require huge amount of computational resources and time, which limit often their field of application. Here the model to be developed will be a cheap one. It should be able to decrease the computational requirements with respect to many other algorithms, while returning a controlled level of accuracy.

Moreover, optimization methods work on an entire process, developing it in all its features. Anyway, external data could be submitted to the model to perform the optimization. Often, in this case external data do not present the desired sample required by the model itself. In fact, most of the times the sampling phase of an optimization problem is realized through the Design of Experiments [10, 48]. The latter provides the most useful information possible to the model, so that it will show better performance. Here the aim is to build a model able to fit also external data.

Thus, the main focus of this thesis is to set up a cheap meta-heuristic model. To realize such algorithm, both a theoretical analysis of each aspect connected to the meta-heuristic model and a detailed implementation of the part involved are necessary. In particular, many parameters can be set in the coding of the response surface model. Despite it is often accounted for as a method not that much suitable, it can prove to be a useful and effective tool, as will be shown in the following.

1.2 Summary of the Thesis

To achieve the above objective, the initial feature of this work is a theoretical analysis of all the involved parts, which are a multi-objective optimization, an evolutionary

search algorithms, surrogate models and sensitivity analysis. The development of each argument through the text follows the sequence in which they had been encountered and studied from the literature.

Once done this, the effective model construction is described. The first step consists in the coupling of sensitivity analysis with the surrogate model. Between those that have been read and reported in the bibliography, few articles use and take advantage of this statistical test to provide further information upon which they base the surrogate. A profitable tool which can give large amount of information without requiring too much computational efforts can be obtained. The sensitivity analysis not only describes to the user which are the main variables influencing the function. It also provides much useful general data about the behaviour of the problem.

The successive phase to build the meta-heuristic deals with the research of a surrogate model able to retrieve good solutions but at the same time it results cheap, computationally speaking. To this scope, some algorithms have been studied and the most suitable was identified in the response surface. This last shows a large simplification to the problems, so it allows to perform the choice of the evolutionary algorithm once again looking to the computational complexity. Finally, the resulting model is evaluated on the test problems to measure its features and its abilities to perform the optimization task for which it has been built. Although test problems cannot measure completely the qualities of the model, they are necessary to realize further tests. In particular, it is important to evaluate the method's ability to fit experimental sample of data coming from a general set up and not based on the required sample set.

At the end of the thesis the results for all these tests are reported, evaluating the overall performance which the model shows. It is also sided to other evolutionary algorithms to compare their results on the considered problems.

Chapter 2

Multi-objective optimization

In engineering Optimization is the process which through an algorithm aims at identifying the combination of variables realizing the best solutions in the problem case at hand. Multi-objective optimization encloses all the problems handling two or more objectives which need to be contemporaneously satisfied. In fact, it can be found variables configurations obtaining good performance only in a single-level or within few-objectives-level. Thus, multi-objective optimization deals not only with the best solutions searching phase, but also with their classification and sorting. The entire process needs to analyse the behaviour of all the variables in their domain to build a profitable and complete optimum search.

Since optimization task needs to interface with complex and various real applications, it faces several problems. The first problem at all is the definition of the objective functions themselves: the mathematical formulation of the case at hand is the introductory step to start with. Without it, it is impossible to begin an analytic search of optima. Mathematical formulation requires first the definitions of variables and their domain. Following this, it is fundamental to identify the proper laws describing the process and the true objectives to optimize. Constrains of each variable also play a main role in the framework. This mathematical translation often is not that easy to set up, because experimental qualitative manner must be reported in a quantitative setting and therefore they become difficult to handle. This sets several errors in the model, which need to be highlighted and possibly solved by running repeated tests before applying the model into practise.

Once the problem is analytically defined, the optimum search sets in. In a multi-objective frame, seldom it is found a sole optimum configuration, dominating all the other framework. Objectives are often mutually competitive and if a solution achieves better values in an objective, it also loses ground on the other sides. Computationally speaking, multi-objective problems require larger quantity of resources (e.g. time and memory) as larger is the number of variables or objectives, the precision required or the sparsity of optima solutions in the domain.

The algorithm chosen to perform the optimum search in this work will be presented later in chapter 3. It is important to highlight since now that several search routines exist, each one with its own peculiarity. Moreover, they often are sided by surrogate models which aim to ease the computational cost. In this optimization framework one can find also many other analytical searching tools, known as Gradient searching methods, as the Conjugate Gradient, the Steepest Descent, and many others. Though, they are seldom used in real-world engineering, where objective functions are too general,

presenting many local optima. Such methods often get stuck in these local optima, preventing the optimization task to come to a proper end.

Finally, dealing with multi-objective problems it is also necessary to define metrics to classify and to order the proposed solutions, but also the differences between algorithms. In fact, once realized which one is the right algorithm, one needs also to know which are its performance with respect to other proposed routines.

This chapter will introduce and describe in detail the mathematical aspects of multi-objective optimization. It will also treat some of the problems hinted here.

2.1 Mathematical description

Single objective optimization aims to find values for the n decision variables $\mathbf{x} = (x_1, x_2, \dots, x_n)$ in the domain of decision Ω , to minimize the objective function:

$$y = \min_{\mathbf{x} \in \Omega} f(\mathbf{x}) \quad (2.1)$$

Often in this problem the goal is not only to discover the optimum value attained by the objective (possibly global optimum and not local). The goal is also to find out the set of decision variables which achieve the optimum value. This because the optimization process is related to a real-life problem and the variables describe a real configuration of the case study. Problems usually are defined as minimization by convention, but nothing changes when dealing with maximisation, since the problems are equal: $\max f(\mathbf{x}) = \min(-f(\mathbf{x}))$. However, searching for the minimum will help to treat multiple objective functions.

The decision variables affect the optimization task with their number, their domain of admissible values and the possible mutual dependencies. Although often decision variables are considered mutually independent and realize a huge relaxation on problems, in almost any application they are continuous in the domain and the objective functions are multi-dimensional. Summing up these features, the single objective optimization can reach a high level of complexity by its own. Therefore, efforts are required to formulate exactly the problem and retrieve the optimum solutions.

Beside the objective functions, commonly the problem is affected by constraint expressions. These can be linear or non-linear equations, as well as inequalities, and they could also involve mixed-integer problem (MIP). In mixed-integer problems some decision variables are integers within their domain. This kind of optimization requires specific solvers [48]. For example, a generic problem like this can be written as:

$$\text{Solve: } y = \min_{\mathbf{x} \in \Omega} f(\mathbf{x})$$

$$\begin{aligned} \text{Subject to: } & g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, k \\ & x_j \in \mathbb{Z}, \quad j \in 1, \dots, n \end{aligned}$$

Constraints involving integer variables make the optimization much more difficult, since it becomes a kind of discrete problem. Anyway, the presence of constraints, their development and evaluation increase the complexity, affecting the optimization process.

Dealing with multi-objective optimization, only the number of objective functions changes to m (with $m \geq 2$). The new problem can be written by defining a new function $\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x}))$, which maps the vector \mathbf{x} of the decision variables of dimension N to the vector \mathbf{y} of the objective functions space with dimension m :

$$\mathbf{y} = \min_{\mathbf{x} \in \Omega} \mathbf{F}(\mathbf{x}) \quad (2.2)$$

Such optimization requires to evaluate simultaneously all the objectives. This implies increasing complexity and computational cost. Moreover, these objectives represent several features of a problem, often bringing opposite contributes to each other. So, as will be analysed in details in the following sections, getting a better value on some objectives often means worsening some others. As the number of objective functions increases, the problem becomes more complex and typically 10 objectives are set as threshold from an easy problem to a harder one. One of the major challenge of multi-objective optimization is to deal with computationally expensive objective functions. Such problem shows up often in simulation based models (e.g. FEM, CFD), which require huge resources to solve even quite simple problems.

2.1.1 Pareto optimality

Mathematical optimization focus on determining the global best solution for a given problem. Often this is not possible in practice, for several reasons: first of all, optimization works with a model of the real problem. Therefore, the solution obtained will be affected by errors due to the fact that equations can't perfectly describe reality. Second, dealing with complex function in many variables, the function value attained at its optimum and the optimum point itself is unknown. So, it is not even possible to understand if it is close to points already evaluated. Finally, it can be proven with test functions that reach the global optimum it is very time consuming when the number of variables is large (typically fairly difficult optimization problems has at least 30 decision variables).

Searching for the best solution of a function, it is possible to identify also two other optimum regions: the local one and the robust one. The former represents usually a problem because the searching process need to overcome this region and proceed to the global optimum. The robust region displays a wide area where, changing the variables, function attains more or less the same value. This could allow to develop more carefully the analytical model, inserting new variables and searching in this area for an optimum in the new formulation.

When some functions achieve multiple good solutions, they get called multi-modal. In these cases, multiple optima provide to the designer a choice, but this multiplicity always depends on the model developed. On the other hand, dealing with a multi-objective problem, when the optimization finds out several good solutions, each of them taking the best value on a different function, these are called Pareto solutions.

The following definition are the fundamental concepts in multi-objective optimization, as stated in [61]:

Pareto Dominance Given two vectors $\mathbf{u} = (u_1, u_2, \dots, u_m)$ and $\mathbf{v} = (v_1, v_2, \dots, v_m)$, \mathbf{u} is said to dominate \mathbf{v} and denoted as $\mathbf{u} \prec \mathbf{v}$ if and only if $\forall i \in \{1, \dots, m\}, u_i \leq v_i \wedge \exists j \in \{1, \dots, m\} : u_j < v_j$. Moreover, \mathbf{u} is said to *cover* \mathbf{v} , denoted as $\mathbf{u} \preceq \mathbf{v}$, if and only if $\mathbf{u} \prec \mathbf{v}$ or $\mathbf{u} = \mathbf{v}$.

Pareto Optimality A solution $\mathbf{x} \in \mathbf{N}$ is said to be Pareto optimal with respect to the whole set \mathbf{N} if and only if there is no other solution $\mathbf{x}' \in \mathbf{N}$ for which $\mathbf{F}(\mathbf{x}') \prec \mathbf{F}(\mathbf{x})$.

Pareto Optimal Set For a given multi-objective evaluation function $\mathbf{F} : \mathbf{N} \rightarrow \mathbf{M}$ the *Pareto Optimal Set* is defined as the subset of \mathbf{N} of all the Pareto optimal vectors in the decision variable set:

$$\text{Pareto Optimal Set} \doteq \{\mathbf{x} \in \mathbf{N} : \nexists \mathbf{x}' \in \mathbf{N} : \mathbf{F}(\mathbf{x}') \prec \mathbf{F}(\mathbf{x})\}$$

Pareto Front For a given multi-objective evaluation function $\mathbf{F} : \mathbf{N} \rightarrow \mathbf{M}$ and a *Pareto Optimal Set*, the *Pareto Front* is defined as the set of vectors mapped from the *Pareto Optimal Set* to \mathbf{M} by the objective function \mathbf{F} :

$$\text{Pareto Front} \doteq \{\mathbf{y} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x})) : \mathbf{x} \in \text{Pareto Optimal Set}\}$$

Many real-world problems involve conflicting objectives which need to be solved together. This framework requires the use of the above concepts to rank and evaluate which are the best solutions. A generic example is reported in figure 2.1, that presents a problem concerning minimization of costs and time delays in a transmission system. As obvious, increasing the velocity of data broadcasting requires better infrastructures, though also higher costs and vice versa, lower investments involve worse performance. The points A, \dots, F on the graph represent the possible solutions individuated, but only the points A, \dots, E are reasonable solution. This happens because F would realize higher delays with the same cost of A , while G higher cost with the same delay of D . Hence solutions F and G are the so called *dominated* solutions, while the others are the *non-dominated* solutions. Each *non-dominated* solution realizes a couple cost-delay which is optimal. Finally the Pareto Front encloses all this trade-off solution, which will be further investigated to find the one that fit better the real application.

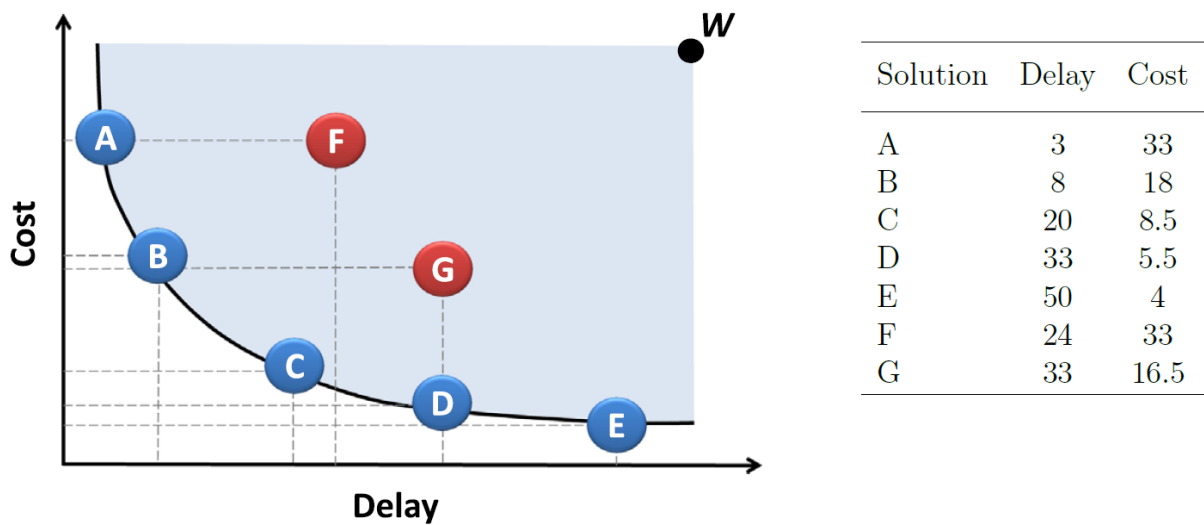


Figure 2.1: A problem with 2 objective functions: elements on the Pareto Front represent the non-dominated solution, in fact their values are a trade-off between the two objectives [10].

2.1.2 Curse of dimensionality

During the modelling phase of a problem it is intuitive that, as the number of variables gets higher, the higher will be also the cost of evaluating objective functions. The measure of the contribute of each single variable to the final prediction can become hard. Having lot of variables concurring to a single or few objectives would lead to difficulties parting the contribute of variables to each objective. Moreover, the striking target is to obtain the optimum value, hence an accurate prediction of the objective. The contribute of a single variable inside the domain needs to be understood both locally and globally, sampling it in n different location. But dealing with a k -dimensional problem, the sampling requires n^k observation to achieve the same sample density as in the single variable problem.

Another problem that comes up is related to the decision variables domain: increasing their bounds or inserting one more variable raise the volume so fast that the available data become sparse. Sparsity is an issue in multi-objective, since the already obtained results lose statistical significance when the domain becomes poorly tested. To get back significant results, density of the sample needs to be enlarged, but its dimension grows exponentially with the number of decision variables and with their bound. In high dimensional problems often the sample appear to be sparse and dissimilar, so group search methods could be required.

Let's consider the example in [18], which study the cost of a car tyre design having complex computational requirements due to its geometrical variables, the range of simulation, its manufacturing process and others aspects. Let assume also for the sake of simplicity that analysis and design process takes one hour of computation per decision variable. Let suppose one requires to study only the diameter of the tyre versus its cost with ten hours at disposal. Hence the simulation can be run ten times varying the diameter inside the entire domain. This would yield ten results which can build reasonable prediction of how the cost varies with the dimension, even if this relation could be highly non-linear. Assume now that, given the model, one wants to study at the same time many different variables to gain refined results and tune properly the model, e.g. introducing tread width, groove spacing, flexing area thickness, shoulder thickness, bead seat diameter and liner thickness side-wall height. Now the model must deal with eight parameters, each having a different domain; always considering to need one hour per computation, to fill the entire design space with ten sample per each variable, this would lead to 10^8 runs, hence hours, which are almost 11416 years of computation.

Trying to evaluate the objective function for all the possible combination of decision variable and building a *full factorial experiment* with k sampling for each design, it can become very expensive and often unrealisable. On the other hand, it is quite evident that the number of variables has a massive impact to the experiment burdening. It is always advisable to perform a deep study of the objective function to highlight which are the most influencing variables and those which do not bring considerable changing in the objective function. This last often can be kept fixed during the analysis and reintroduced at the end if a further accurate study is strictly required. Moreover, studying the objective function one could come out with further constraints to the problem. They would prevent the analysis of variables configurations which are indeed unachievable and would burden the computation in vain.

2.1.3 No free-lunch theorem

As will be described later in chapter 3, there exist many different algorithm which solves the multi-objective optimization problems. Each of them has its main features but, as stated by the No Free Lunch theorem, there does not exist a method which always performs better than all the others.

No Free Lunch Theorem 1. *Given a finite set \mathbf{V} and a finite set \mathbf{S} of real numbers, assume that $f : \mathbf{V} \rightarrow \mathbf{S}$ is chosen at random according to uniform distribution on the set $\mathbf{S}^{\mathbf{V}}$ of all possible functions from \mathbf{V} to \mathbf{S} . For the problem of optimizing f over the set \mathbf{V} , then no algorithm performs better than blind search [65].*

Proof. The proof [23] uses the probabilistic method. We will show that for any learner, there is some learning task (i.e. *hard* concept) that it will not learn well. Formally, take \mathcal{D} to be the uniform distribution over $(x, f(x))$. Our proof strategy will be to show the following inequality

$$\mathcal{Q} \doteq \mathbb{E}_{f: \mathcal{X} \rightarrow 0,1} [\mathbb{E}_{\mathcal{S} \sim \mathcal{D}^m} [\text{err}(A(\mathcal{S}))]] \geq \frac{1}{4}$$

as an intermediate step, and then use Markov's Inequality to conclude.

We proceed by invoking Fubini's theorem (to swap the order of expectations) and then conditioning on the event that $x \in \mathcal{S}$.

$$\mathcal{Q} = \mathbb{E}_{\mathcal{S}} [\mathbb{E}_f [\mathbb{E}_{x \in \mathcal{X}} [A(\mathcal{S})(x) \neq f(x)]]] = \frac{\mathbb{E}_{\mathcal{S}, x} [\mathbb{E}_f [A(\mathcal{S})(x) \neq f(x) | x \in \mathcal{S}]] \mathbb{P}(x \in \mathcal{S}) + \mathbb{E}_{\mathcal{S}, x} [\mathbb{E}_f [A(\mathcal{S})(x) \neq f(x) | x \notin \mathcal{S}]] \mathbb{P}(x \notin \mathcal{S})}{2}$$

The first term is, in the worst case, at least 0. Also note that $\mathbb{P}(x \notin \mathcal{S}) \geq \frac{1}{2}$. Finally, observe that $\mathbb{P}(A(\mathcal{S})(x) \neq f(x)) = \frac{1}{2} \quad \forall x \notin \mathcal{S}$ since we are given that the *true* concept is chosen uniformly at random. Hence, we get that:

$$\mathcal{Q} \geq 0 + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

which is the intermediate step we wanted to show. We conclude the proof by a simple application of the reverse Markov Inequality:

$$\mathbb{P}(\mathcal{Q} \geq \frac{1}{10}) \geq \frac{\frac{1}{4} - \frac{1}{10}}{1 - \frac{1}{10}} \geq \frac{1}{10}$$

□

The theorem in essence state that when all function f are equally likely, then the chosen algorithm does not influence the probability of observing an arbitrary sequence of values during the optimization task. Going back to the searching algorithms, what happens is that on a particular kind of problems an algorithm may outperform all the others. Although on other problems different algorithms would achieve better results. Therefore, any two algorithms are equivalent when their performance is averaged over all the possible problems. However, one can argue that many old search procedures are overtaken by nowadays implementation at almost all the problems. It happens because a better knowledge of the optimization tasks and computational implementation has been achieved. These facts highlight knowledge as a key factor, which enables to understand the problem at hand and globally gain better optima and improved performance.

2.2 Algorithm evaluation metrics

To evaluate the performance of a multi-objective optimization it is necessary to develop a metric. It is useful to properly compare the results obtained by different algorithms, taking advantage of the problem features, as could be the *Pareto Optimal Set* or the *Pareto Front*. The metric should also need to consider both convergence and diversity (or sparsity) of solutions inside the domain.

In fig. 2.2 different metrics are compared: picture 2.2(a) introduces the problem, which requires the minimization of both the argument along the two axes. The black dots represent the best-known approximation set, while the grey dots are the current set of found solutions. Both of them are *Pareto Front*, since one can easily notice that all the solutions of the same family identify a non-dominated point of each argument.

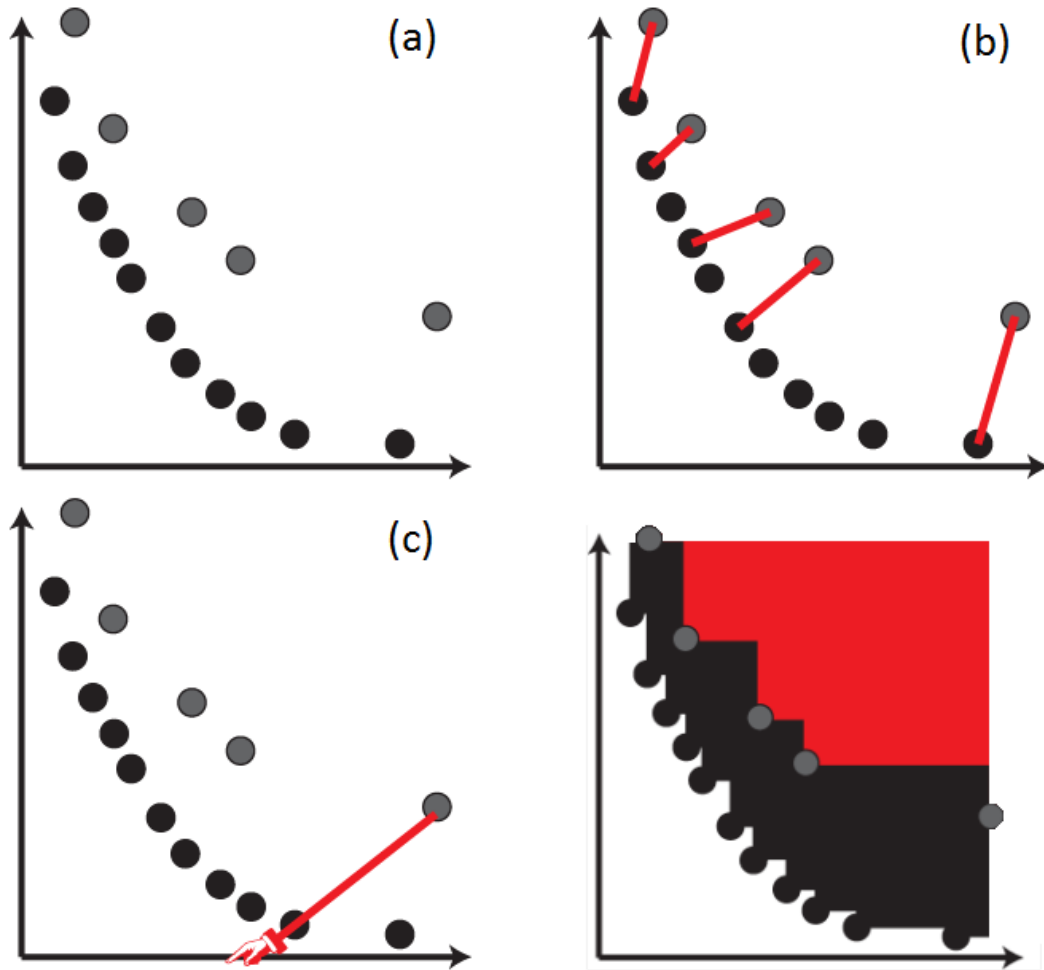


Figure 2.2: Different kind of metric: (a) introduces the problem, presenting the best-known set (black) and the current found solutions (grey), in a minimization problem of both the arguments; (b) Generational Distance metric; (c) Epsilon Indicator metric; (d) Hypervolume metric [46].

When evaluating the quality of a search and the progresses in multi-objective optimization, a typical reference point is in the proximity to the best Pareto Front found so far. In such way it can properly evaluate the full extent of trade-off solutions. Here the main used metrics are briefly introduced and described. Anyway, many different types of metrics exist and each focus often only on a particular chosen aspect.

Generational Distance

It is the easiest metric to realize and measures the Euclidean distance between each point of the current Pareto Front (\mathcal{PF}') and the nearest point of the best reference set (\mathcal{PF}). The mathematical definition of this procedure is:

$$D_G(\mathcal{PF}, \mathcal{PF}') = \frac{\sqrt{\sum_{\mathbf{x} \in \mathcal{PF}'} d_{\mathbf{x}}^2}}{P} \quad \text{where} \quad d_{\mathbf{x}} = \min_{\mathbf{y} \in \mathcal{PF}} \sqrt{\sum_{i=1}^M [f_i(\mathbf{x}) - f_i(\mathbf{y})]^2} \quad (2.3)$$

In eq. 2.3 $d_{\mathbf{x}}$ is the minimum Euclidean distance between each objective value in \mathcal{PF}' and \mathcal{PF} . The denominator P is the total number of values in the current Pareto Front, while $f_i(\mathbf{x})$ and $f_i(\mathbf{y})$ are the objective values of respectively the current Pareto Front and the best one. This metric leads to an averaging process which reduces the impact of single optimal point proximity to the best Pareto set. Furthermore, it does not take in consideration the diversity of solution along the current \mathcal{PF}' itself. This method is often known also as *D-metric*.

Epsilon Indicator

This metric weight differently the elements in the Pareto Front, considering the worst-case value. The distance of the current solution set is obtained as the required translation of the entire set to dominate each nearest neighbour in the optimal Pareto Front. Hence this depends in particular from the worst solution in the current approximation set:

$$D_{\epsilon}(\mathcal{PF}, \mathcal{PF}') = \max_{\mathbf{y} \in \mathcal{PF}'} \min_{\mathbf{x} \in \mathcal{PF}} \max_{1 \leq i \leq M} (f_i(\mathbf{x}) - f_i(\mathbf{y})) \quad (2.4)$$

The ϵ - *indicator* is very sensitive to gaps in current Pareto Front, highlighting the *consistency* of the actual set with the reference one. instead, the additive form of this metric is more influenced by diversity and gaps inside the \mathcal{PF}' . However it always focuses on the worst-case distance. Hence this metric is able to measure quite well either the method convergence and its diversity inside the Pareto Front.

Hyper-volume

Hyper-volume is a more complete metric since equally measures convergence and diversity. As in fig. 2.2(d), it evaluates the volume of objective space dominated by the current Pareto Front. The hyper-volume metric is calculated as the ratio between volumes of the best Front \mathcal{PF} and approximation Front \mathcal{PF}' :

$$D_H(\mathcal{PF}, \mathcal{PF}') = \frac{\int_V \alpha_{\mathcal{PF}}(\mathbf{x}) dx}{\int_V \alpha_{\mathcal{PF}'}(\mathbf{x}) dx} \quad \text{where} \quad \alpha_{\mathcal{PF}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \exists \mathbf{z} \in \mathcal{PF} : \mathbf{z} \preceq \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

The hyper-volume calculation is performed taking a fixed reference point more or less distant from the Pareto Front, such that all the results overtake it. The attainment function $\alpha_{\mathbf{x}}$ makes it possible to measure the volume where the best Pareto Front dominates the current one. Diversity is measured through this method due to the fact that near solutions on the current Pareto front introduce a gain in the total volume. Finally, the name of the method is due to the measurement calculation needed when

the objective functions are more than three. While in 2-dimensional problems hyper-volume deals with areas and in the 3-dimensional ones it realizes volumes, in presence of further objectives the dimensional space grows and the hyper-volume concept must be used.

Maximum Spread

This last method generates a simple measure of the extension of the Pareto Front, hence it deals only with its diversity and it does not consider the convergence. For 2-dimensional problems the metric evaluates the Euclidean distance between the two extreme solutions of the current Front \mathcal{PF}' , while in upper dimension cases different distance measures can be used.

Chapter 3

Evolutionary Algorithms

In recent years, multi-objective optimization is facing problems getting harder and harder, which are related to complex real world scenarios. Dealing with them comes out the so-called hybrid optimization approach. It combines together different heuristic searching algorithms of various nature with methods from mathematical and statistical programming, to treat complex objective functions. This combination aims to take advantages of features of each individual component and often the brand-new algorithm realized performs better than the single parts alone. Anyway, as stated in section 2.1.3, usually each algorithm performs well only in few problems. So it is possible to identify many different kinds of meta-heuristic hybrid formulations which best fit each particular situation.

First of all it is necessary to describe which are these searching algorithms and mathematical models used in the optimization. In this chapter will be described the searching methods of evolutionary algorithms, briefly introducing the most used ones. Then also surrogate models in chapter 4 will be investigated, dealing with their objectives, their searching process and again there will be introduced some models use nowadays.

3.1 Evolutionary Algorithms

Evolutionary algorithms are a mechanism inspired by the biological evolution. It searches for the optimum of the objective function exploiting features of evolution itself, as reproduction, recombination and mutation.

This kind of algorithm performs quite well on all types of optimization, because its nature is not related to any particular feature of the problem. This is the main reason why this tool is powerful in current times, where any real-life task is submitted to a *sort of*¹ optimization process.

3.1.1 Main features

As said previously, the fundamental aspect of this kind of algorithms is their strict relation with the biological evolution theory. First step of any evolutionary algorithm is to build a population whose individuals will be the candidate solutions for the optimization process. Each individual represents a set of decision variables and it gets

¹ A *sort of* since often the optimization process gets developed without complete notion on the real problem at hand and looking for first rough information

evaluated by the fitness function. Once the entire population undergoes the evaluation process, the reproduction and recombination take place. Two or more individuals of the starting population give birth to one or several new solutions, called *children*. The parents are chosen randomly or with some deterministic picking selection between the entire population. Afterwards, some children of the offspring are subject to a mutation. It allows to enhance the domain variables exploration and exploitation, since otherwise the evolution would be constrained within the combinations of the starting population. Then the entire offspring gets evaluated and is merged to the old group of individuals. Since the population size is often kept fixed or at least constrained, the following phase selects the individuals which will survive as a sort of genetic selection. This process then is repeated likewise, until a threshold number of generation is reached or the evolution does not lead to a threshold improvement of the objective function (theoretically this case coincides with the algorithm convergence).

The procedure described is the general development of an evolutionary algorithm. It is possible to recognize some particular features which directly connect to the optimization problem at hand:

Individuals: each individual of the population identifies a set of decision variables, which can be encoded in different ways. The values can be simply described by real-coding, but could also be binary-coded [25]. The former choice is often better because real coded numbers can provide machine precision, while binary coding limit the representation capacity. Moreover, real coding uses much less storage memory since a single number depicts the entire variable. Meanwhile a binary string is required in the latter formulation. Finally, the binary string decoding needs to be performed before evaluating the objective function, as real-coded already yield the desired value. Only in presence on integer variables problems individuals can be more logically represented by binary coding. So, in general real-coded should be chosen and individuals would be described by a string of numbers, each representing a single decision variable.

Population: the population of an evolutionary algorithm is its fundamental base. This because the individual itself cannot evolve alone, but it needs to interact with the other and spread its genetic heritage. Different kinds of algorithms treat population in various ways. Some of them work always on the same starting population, changing step by step their genetic variables. Some others merge old population and offspring and they successively reject part of the individuals. Most of the times a fixed point is the population dimension, which cannot grow nor decrease.

Fitness function: it is the objective function analogue. Also in this case the presence of multiple optimization reflects itself to multiple fitness values for each individual. Dealing with their sorting, which is usually necessary to the reproduction and selection of best fits, it is a complex task and it can be solved through mathematical operation of point allocation.

Reproduction, recombination and mutation: these are the classical operations which chromosomes are subject to. These processes in the algorithm will be ruled by fixed parameters, managing for example the rate of mutation or recombination.

Offspring: depending on the algorithm, new generation could be generated either by new children or by evolution of individuals themselves. In the former case a contention to fill the new generation would raise up, involving both children and the old population. In the latter, any improvement in each individual fitness function would lead to the evolution acceptance, while a worsening would not be directly accepted.

Algorithm 1: Pseudo code of general evolutionary algorithm

```

Step 1: Generate the initial population of individuals randomly (starting
generation);
Step 2: Evaluate the fitness function of each individual;
while termination is not reached do
    1. Select individuals (parents) for reproduction;
    2. Generate new individuals (offspring) through crossover and mutation
operations;
    3. Evaluate the offspring fitness;
    4. Merge old population and offspring;
    5. Delete from the new population least-fit individuals;
end

```

Despite the typical model of evolutionary algorithms may seems ease, many different implementation has been realized over the years. Each of them is capable of dealing at best with a specific kind of problem. So the most spread methods will be described in detail hereinafter.

3.2 Genetics algorithm

Genetic algorithm is the closest method to the general evolutionary algorithm. Also, it is the most used searching tool of this kind. In fact, it is quite easy to build, it fits well on many types of problems and, most important, it is open to a lot of development and implementation. Its main features are all and only those of a generic evolutionary algorithm.

First of all the population needs to be initialized. This is usually performed by a simple random choice inside the domain or inside the search space, when they are defined differently. Another option is to gather most of the initial individuals in areas where likely should be found optimal solutions. This could speed up the convergence but at the same time it would compromise the exploration and exploitation of the entire design space.

Once the population is generated, it takes place the selection stage to choose the individuals which will breed the new generation. Such process requires the following steps:

- Each individual gets evaluated by the fitness function. In presence of a multi-objective problem, the fitness value would be a vector, hence it will be necessary to perform a normalization by the sum of all resulting fitness values.

- The population is sorted by descending fitness values.
- The cumulative normalized fitness is calculated for each individual as the sum of all the previous individual fitness values (those with a higher fitness) and its own value. This process leads to a sort of cumulative distribution function of the population fitness value.
- Finally a random number R uniformly distributed between 0 and 1 is chosen and all the individuals with cumulative fitness lower than R are selected.

This procedure is quite resource demanding if the population is large, which is common nowadays.

Different selecting algorithms are almost always used. Among others the main two are fitness proportionate selection (better known as *roulette wheel selection*) and tournament selection.

Roulette wheel selection

In *roulette wheel selection* the total fitness function gets divided into N parts (with N the population dimension), each proportionate to the relative individual fitness value. So, the probability of being selected for a single individual is equal to:

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j}$$

This can be imagined as a roulette wheel where each sector is scaled by the relative fitness value, instead of divide the wheel equally. A development of this algorithm is the *Stochastic Universal sampling*, which builds the entire sample set using only a single random value. It fills the set starting from this fitness value point and advancing in the cumulative fitness value by a fixed step (usually equal to the total fitness, divided by the number of desired sample). Such implementation exhibit no bias and, what is more important, it works well when the population has few individuals with large fitness value in comparison with the other ones. This further implementation allows to choose the next candidate from the rest of population, avoiding that these few high fitting individuals saturate the candidate space.

Tournament selection

Tournament selection, as the name itself suggest, develops several tournaments among a set of individuals randomly chosen in the population, by using their fitness value as discriminating parameter. Once the set of players is sorted in each tournament, the selection sets in. A probability value p gets fixed, then the best individual is chosen with probability p . The second best with probability $p * (1 - p)$, the third one with probability $p * (1 - p)^2$ and so on. Usually it is $p = 1$, so simply the best individual is selected at each tournament phase and the procedure gets repeated. The tournament set size governs how individuals get chosen: a large set will often present at least a high fitness individual which will be selected, while a smaller set allows the selection of lower fitness individuals. Typically tournament procedure works out between only 2 or 3 elements, to speed up the process and avoid burdening computation. Tournament selection among others benefits can work on parallel architecture and it is independent

of the genetic algorithm scaling in some systems.

A key factor of both this type of selection is that candidate solutions with low fitness are less likely to be eliminated. On the other hand high fitness can be avoided, while on many others algorithms the worst solution get immediately discarded and the selection occurs only between the best ones. Moreover both methods are quite easy to implement, they require low computational efforts. Therefore, they perform a better selection than several other algorithms. Finally they also perform well talking about the sampling stochasticity noise, since these methods are less dependent on the random picking procedure. When desired, they allow to perform more than once the selection of an individual.

Such techniques may seem to contemplate elitism, since they select likely individuals with a high fitness values. Although, it will be described later on a method which considers in a particular frame the best individuals. However also the less fit solutions are necessary in the selection and evolution process, since they manage to perform exploitation and exploration of the search space.

The next step consists in performing the recombination (also known as crossover) to breed the new generation. This process involves the previous selection from which typically two or more individuals, called *parents*, at a time get extracted and combined to generate one or more new generation's individuals, called *children*. Some researchers suggest that combinations which use more than two individuals could increase the offspring quality. The resultant offspring generation usually shows a better mean fitness value. However, once again the genetic diversity and exploration of the search space is kept alive by the presence of low fit individuals.

Different crossover techniques between two parents exist, but only few of them are quite always used. These reported achieve low cost computational effort and rather satisfactory results, meanwhile they are easy to implement:

Single-point given the strings of each parent decision variables and chosen a random point inside both these strings, the children are generated swapping the sub-string data on either crossover point of the parents (Fig. 3.1a).

Double-point in this case two random crossover points are selected on both the parents, dividing the strings in three part each. So, one child is generated merging central string of the first parent and side strings of second one and the other child vice versa (Fig. 3.1b)

Uniform this type of crossover uses a fixed mixing ratio between two parents. A random number of crossover points is generated over the parent's strings. Hence each sub-string is handed down to the children, always swapping between the parents. In this fashion each child will have more or less half of its genetic heritage from each parent and the decision variables get treated almost as single gene (Fig. 3.1c). The swapping between parents sub-string can also be controlled by a probability which decides when to realize the swap (e.g. in simple uniform crossover as in relative figure the swapping probability is set to 1, since each sub-string is allocated to the other child).

The succeeding step consists in introducing the mutation. It is used to maintain the genetic diversity from one generation to another and provide exploration and ex-

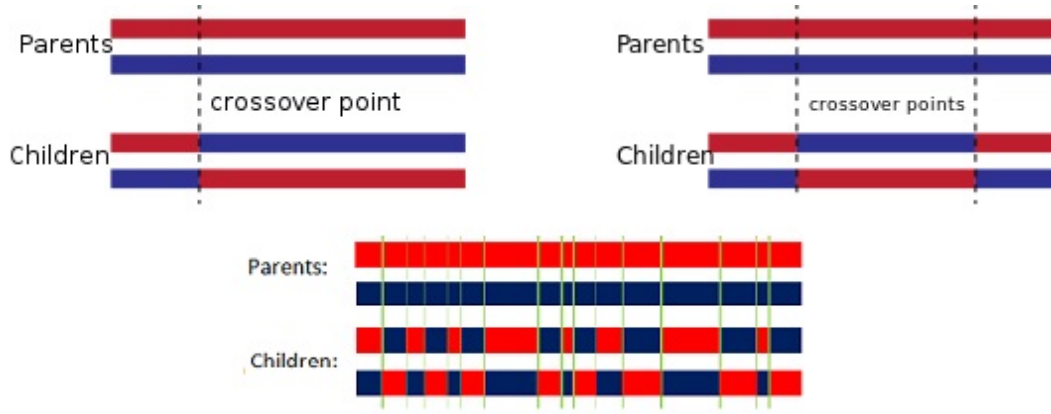


Figure 3.1: **a.** Single-point crossover technique (upper left); **b.** Single-point crossover (upper right); **c.** Uniform crossover (bottom) [63].

ploitation. Without these features the searching algorithm would most of the times get stuck in a local optimum and individuals would become all similar to each other. The chromosome modification of an individual may completely change its fitness value, hence mutation occurrence needs to perform also the evaluation of the new element.

This genetic operator alters some children and often this is done fixing a probability of mutation. Such process needs to happen only few times per generation (on the order of 1 time per 100 individuals or less) or it could lead the algorithm to become a primitive random search.

Mutation occurrence works differently depending on the population definition: if the chromosome is binary or real coded. In the former case each decision variable is represented by a binary string of fixed length. The change could perform a single bit flip in randomly chosen genomes, replacing in these bit string a 0 with 1, or vice versa. Otherwise the mutation can provide an entire sub-string modification. Randomly it is selected a genome portion and its bits are switched with probability $\frac{1}{l}$, where l is the sub-string length (if the selection provide a single bit string, a simple flip bit mutation is performed).

Dealing with real coded chromosomes the easiest mutations are the uniform and Gaussian one. First the genes which undergo modification need to be selected randomly. Then, given the upper and lower bound of each decision variable in the case of uniform distribution, new random values in between the bounds are generated and substituted to the previous. In the Gaussian case for each decision variable gets evaluated the mean from the parent entries. Given the mean value a Gaussian distributed number get extracted and added to the original value. If the final value falls outside the domain boundaries, a new random value is chosen and substituted to the old one.

More sophisticated approaches have been developed and particular among the others is *shrink mutation*, developed by Da Ronco and Benini in [7]. This implementation uses the Gaussian mutation but it is constrained by the *shrink* parameter:

$$Shrink_{i,i_{gnr}} \doteq Shrink_{i-1,i_{gnr}} \left(1 - \frac{i_{gnr}}{n_{gnr}}\right)$$

It represents the current mutation range allowed by the i -th decision variable, while i_{gnr} and n_{gnr} represent the current generation index and the total number of generations. The mutation is built to enhance deeply exploration of the design space during the first part of the optimization, while it performs better exploitation of the already

founded solutions during the last optimization stages.

Once reproduction and mutation over offspring are performed, the old generation and all the children are merged. This stage can be developed substantially in two different ways. The former consists simply in completely substitute the entire old generation with the new one. This implies that the reproduction phase generates a number of offspring at least equal to the old population's dimension. *At least* since the crossover could produce clone individuals, which usually should be delete to avoid useless evaluations of the fitness function and mainly to maintain and preserve the population diversity. The latter method to merge old and new generation arises when the designed offspring number is lower than the population dimension. In this scenario, after mutation stage ends, all the children are evaluated by the fitness function and added to the old population. Hence, the individuals get sorted by fitness value and from the entire set n of them get extracted to build the new generation (where n is the original population dimension). An alternative is to keep from the merged population set the best n individuals, but in this case care must be taken to preserve genetic diversity.

Once built the new generation, all the operations starting from the selection to perform crossovers are repeated until termination criterion is not reached. This happens when the maximum number of generation is achieved or when the best solution improvement is below a fixed tolerance. In such case, the algorithm should has theoretically converged to the optimum.

Elitism

This feature allows the algorithm to build a list containing some particular individuals of the population. These are the candidates with the best objective values and they are carried into the next generation unchanged. These individuals lead to huge impact on the performance of the genetic algorithm, reducing the time required to the convergence and hence the computational cost. So, in the following generation these elements enter again in the tournament selection to develop new generation.

However, in each generation the elements of the elite group get always refreshed and the entire starting population is sorted by the fitness value to select the best individuals. The number of elements in this elite list can deeply affect the algorithm itself, depending on the population's dimension. In fact, choosing a large elite set it would prevent the algorithm to explore the entire space, only performing the exploitation around elite solutions and the algorithm likely would achieve a local optimal solution. On the other hand a small elite set would affect only slightly the genetic operator and this few elements would not enhance the convergence.

Hence elitism can be a powerful tool but it needs to be properly tuned by testing its contribution and analysing the problem at hand.

NSGA

The *Non-dominated Sorting Genetic Algorithm* is one of the first evolutionary algorithm which spread all over the developing community. It is based also on the elitism strategy just described in 3.2. It performs quite well on a huge variety of problems, it is based on simple but effective ideas and it has been updated several times up to

generating the *NSGA-II*². Besides elitism, *NSGA-II* takes advantage from population crowding in the selection phase, to build an offspring which realizes the diversity of solution through the Pareto front.

Algorithm 2: Pseudo code of NSGA-II

Input: Obtain the optimal value of fitness function
Output: Children
 $Population \leftarrow InitializePopulation(Population_{size}, Problem_{size});$
 $Evaluate(Population);$
 $FastSort(Population);$
 $Selected \leftarrow SelectParentByRank(Population, Population_{size});$
 $Children \leftarrow CrossoverAndMutation(Selected, P_{crossover}, P_{mutation});$
while termination criterion is not reached **do**
 $Evaluate(Children);$
 $Union \leftarrow Merge(Population, Children);$
 $Fronts \leftarrow FastSort(Union);$
 $Parent \leftarrow \emptyset;$
 $Front_L \leftarrow \emptyset;$
 for $Front_i \in Fronts$ **do**
 $CrowdingDistanceAssignment(Front_i);$
 if $size(Parents) + size(Front_i) > Population_{size}$ **then**
 $Front_L \leftarrow Front_i;$
 $Break();$
 else
 $Parents \leftarrow Merge(Parents, Front_i)$
 end
 if $size(Parents) < Population_{size}$ **then**
 $Front_L \leftarrow SortByRankAndDistance(Front_L);$
 for $i = 1, \dots, Population_{size} - SizeFront_L$ **do**
 $Parents \leftarrow P_i;$
 end
 end
 $Selected \leftarrow SelectParentsByRankAndDistance(Parents, Population_{size});$
 $Population \leftarrow Children;$
 $Children \leftarrow CrossoverAndMutation(Selected, P_{crossover}, P_{mutation});$
end
Return($Children$)

3.3 Particle swarm optimization

Particle swarm optimization (PSO) is inspired from the behaviour exhibited in swarm of social insects, big bird flock and fish school. This algorithm turns out to be very efficient in a large variety of engineering design problem. As the other evolutionary algorithm,

²Further development and personalization are realized and also the *NSGA-III* has been released. However usually the community report and compare the results with the second version of the algorithm.

it is population based, but instead of developing several generations, at every iteration it modifies the existent candidate solutions, moving them over the entire search space.

Dealing with an n -dimensional PSO, the fundamental constitutive bricks are particles, which represent the candidate solutions. Each of them is characterized by its position $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,n})$ (dealing with the i -th particle), its velocity $\mathbf{v}_i = (v_{i,1}, v_{i,2}, \dots, v_{i,n})$ and its best value $\mathbf{p}_i = (p_{i,1}, p_{i,2}, \dots, p_{i,n})$. To complete the set of information coming with this particle, one finds its local best neighbour particle and the global best particle, respectively $\mathbf{p}_{nb} = (p_{nb,1}, p_{nb,2}, \dots, p_{nb,n})$ and $\mathbf{p}_{gb} = (p_{gb,1}, p_{gb,2}, \dots, p_{gb,n})$. The main feature of these particles is their position, which is the vector containing the n current decision variables required to evaluate the fitness function. In some code implementation few of this information is discarded: usually either local or global best value is employed, while some other algorithm completely neglect the velocity.

During the optimization process these quantities are updated using several fundamental parameters. They can be largely modified to better fit the current problem in analysis: the maximum velocity of the particles V_{max} , the total number of iteration³, the accelerating constants c_1 and c_2 and the inertia weight ω ; a brief description of the last two elements must be given.

The inertia weight is a parameter required to handle properly the particle velocities. When ω is large the algorithm performs a global search and the particle velocity increase steadily up the maximum, but it makes difficult to change the direction of motion, realizing a divergent population. On the other hand, if ω is small particle velocity would decrease slowly until it reaches zero. Any little bias from global or local best would produce a rapid direction shift. So the proper choice for the inertia weight would lead to the introduction of a changing parameter in relation to surrounding condition. A first implementation reported from [21] develops an easy equation to properly modify ω :

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{iter_{max}} iter \quad (3.1)$$

Here in 3.1 terms ω_{max} and ω_{min} are fixed parameters that should be appropriately tuned for any single problem (suggested values are 0.9 and 0.4 respectively). A more advanced implementation, related also with the fitness value, is reported in [30] and defines ω as:

$$\omega = \begin{cases} \omega_{min} - \frac{(\omega_{max} - \omega_{min})(f - f_{min})}{f_{avg} - f_{min}} & \text{if } f \leq f_{avg} \\ \omega_{max} & \text{if } f > f_{avg} \end{cases} \quad (3.2)$$

In 3.2, f identifies current objective function value, while f_{avg} and f_{min} represent respectively average and minimum objective value of all particles. Anyway one of the handling above is necessary to tune properly the inertia weight and to improve both particle intelligence during the searching process and convergence rate.

Instead, the accelerating constants reflect how the considered particle motion is affected by its personal best, the local best or the global best value. Since usually only two out of these three quantities get used, only two parameters are needed. Under these updating rules the particle will move toward the best positions found so far. Also accelerating constants can provide great improvements to the searching abilities

³Number of iteration in PSO is the equivalent of the number of generation in the genetic algorithm.

and convergence, but again they need to be suitably tuned to obtain the best performance (also they have suggested values which are $c_1 = c_2 = 2.05$). Often from these parameters also two other factors get defined and these are: $C = c_1 + c_2$ (constrained to be $C > 4$) and the *constriction factor* $\phi = \frac{2}{|2 - C - \sqrt{C^2 - 4C}|}$.

The remaining part of the method to describe is the updating of particle position and speed. These are performed by an iterative approach, in which all the parameters defined above play a key role to the correct solve of the problem. First, the algorithm needs to update velocity with equation:

$$\mathbf{v}_{i,j}(t+1) = \phi\{\omega\mathbf{v}_{i,j}(t) + c_1r_1[\mathbf{p}_{i,j} - \mathbf{x}_{i,j}(t)] + c_2r_2[\mathbf{p}_{gb,j} - \mathbf{x}_{i,j}(t)]\} \quad (3.3)$$

Note that particle speed has as many entries as the problem's dimension (indexed with the j subscript). So each of them needs to be calculated and it is possible to write 3.3 vectorial form without misunderstanding. The quantities r_1 and r_2 are uniform random numbers between 0 and 1. As already said, global best and local best can be exchanged in the formula, while almost always particle personal best is present. Moreover, since each element has its own fitness value, using equation 3.2 to evaluate inertia weight, it is necessary to recalculate ω for each particle and every iteration.

Finally it is possible to update position:

$$\mathbf{x}_{i,j}(t+1) = \mathbf{x}_{i,j}(t) + \mathbf{v}_{i,j}(t+1) \quad (3.4)$$

Successively to the motion of all the particles, fitness evaluations are performed, also to verify the possible update of personal, local and global best values.

Such algorithm has as only criterion of termination the maximum iteration limit. In fact, convergence cannot be evaluated because particles evolution develops paths which do not necessary aim to better fitness at each iteration or at most of them.

A final remark should be made dealing with the use of local best value. Definition of the neighbourhood of each particle can be extremely variable in different situations. Performing the proper choice of this set can lead to a much more robust and faster algorithm. As neighbourhood set of elements with which the considered particle communicates, it is possible to consider for examples a geometric set, counting its k nearest particles, or all the elements located in its same portion of the domain (e.g. settling a grid), or instead a group set, that initializes a certain number of families and inside them collocate randomly the particles.

3.4 Differential evolution

Differential evolution is an evolutionary algorithm proposed by Storn and Prize in [59] widely used to solve global optimization problems. As all the others, it includes initialization of population, crossover, mutation and selection operations. Although, here they succeed one another differently with respect to classical algorithms as the genetic ones. Its main advantage is the very simple structure, the little computational costs for operation between individuals and the ease of use. Instead, as the others evolutionary based codes, the major drawback is the requirement of large number of function evaluations at each generation. This could be worsened by the presence of a complex objective function, leading to a time-consuming and computationally expensive problem [19].

Algorithm 3: Pseudo code of Particle Swarm Optimization

Result: Obtain the optimal value of fitness function

Input: $c_1, c_2, C, \phi, iter_{max}$

Set $\mathbf{p}_{gb} = \mathbf{None}$;

for each particle $i = 1, \dots, N$ **do**

 Initialize particle position: $\mathbf{x}_i \sim U(\mathbf{b}_{lo}, \mathbf{b}_{up})^4$;

 Evaluate particle fitness: $fit_i \leftarrow f(\mathbf{x}_i)$;

 Initialize particle best position to its initial position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$;

if $f(\mathbf{p}_i) < f(\mathbf{p}_{gb})$ **then**

 | $\mathbf{p}_{gb} \leftarrow \mathbf{p}_i$

end

 Initialize particle velocity: $\mathbf{v}_i \sim U(-|\mathbf{b}_{up} - \mathbf{b}_{lo}|, |\mathbf{b}_{up} - \mathbf{b}_{lo}|)$;

end

$iter = 0$;

while termination criterion is not reached **do**

for each particle $i = 1, \dots, N$ **do**

for each dimension $j = 1, \dots, n$ **do**

 Extract random numbers: $r_1, r_2 \sim U(0, 1)$;

 Evaluate inertial weight: $\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{iter_{max}} iter$;

 Update j -th component of i -th particle velocity:

$\mathbf{v}_{i,j}(t+1) \leftarrow \phi\{\omega \mathbf{v}_{i,j}(t) + c_1 r_1 [\mathbf{p}_{i,j} - \mathbf{x}_{i,j}(t)] + c_2 r_2 [\mathbf{p}_{gb,j} - \mathbf{x}_{i,j}(t)]\}$;

 Update particle position: $\mathbf{x}_{i,j}(t+1) \leftarrow \mathbf{x}_{i,j}(t) + \mathbf{v}_{i,j}(t+1)$;

 Update particle fit: $fit_i \leftarrow f(\mathbf{x}_i)$ **if** $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then**

 | Update particle best position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$;

if $f(\mathbf{p}_i) < f(\mathbf{p}_{gb})$ **then**

 | Update global best position: $\mathbf{p}_{gb} \leftarrow \mathbf{p}_i$;

end

end

end

end

$iter \leftarrow iter + 1$;

end

Dealing with the basic algorithm, as always, the first step consists in building the starting population by randomly sampling N_p individuals from the sample space, where N_p is the population dimension. The following operation creates the mutant vector for each selected individual of the current generation $\mathbf{x}_{i,G}, i = 1, \dots, N_p$. Each of them has n components as the number of decision variables. The mutation operation can be performed in several ways. It could involve two or three individuals different from the selected one, individuals randomly chosen in the population or those with the best

⁴ \mathbf{b}_{lo} and \mathbf{b}_{up} values are the lower and upper boundaries of the search space variables, expressed as vectors.

function value:

$$\begin{array}{ll}
\text{DE Rand 1} & \mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \\
\text{DE Best 1} & \mathbf{v}_{i,G} = \mathbf{x}_{best,G} + F(\mathbf{x}_{r_1,G} - \mathbf{x}_{r_2,G}) \\
\text{DE Rand to Best 1} & \mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) + F(\mathbf{x}_{best,G} - \mathbf{x}_{r_1,G}) \\
\text{DE Curr to Best 1} & \mathbf{v}_{i,G} = \mathbf{x}_{i,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) + F(\mathbf{x}_{best,G} - \mathbf{x}_{i,G}) \\
\text{DE Rand 2} & \mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G} + \mathbf{x}_{r_4,G} - \mathbf{x}_{r_5,G}) \\
\text{DE Rand 2} & \mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G} + \mathbf{x}_{r_4,G} - \mathbf{x}_{r_5,G})
\end{array} \quad (3.5)$$

In these formulas from [39] several new elements can be seen: r_{1-5} are integers which identify the individuals randomly chosen in the population. These differ from the i -th individual which is the subject of mutation. The subscript G indicates the generation, so the operation deals always with members of the same generation. Finally here the most important parameter is F , the (mutation) scaling factor, ranging in $[0,2]$. It amplifies the difference between all the selected individuals.

The mutant vector $\mathbf{v}_{i,G}$ such defined is then added to the original selected vector $\mathbf{x}_{i,G}$. These two generate a trial vector by the following operations:

$$\mathbf{u}_{j,i,G} = \begin{cases} \mathbf{v}_{j,i,G} & \text{if } rand_j(0,1) \leq Cr \text{ or } j = j_{rand} \\ \mathbf{x}_{j,i,G} & \text{otherwise} \end{cases} \quad (3.6)$$

Here $j = 1, \dots, n$ and j_{rand} are a random integers and Cr is the crossover control parameter, again a random number $Cr \in (0,1)$. Due to the use of j_{rand} the trial vector $\mathbf{u}_{i,G}$ always differs from the original one $\mathbf{x}_{i,G}$, since could happens that $rand_j > Cr \forall j$. On the other hand Cr controls how much the generated individuals will be altered with respect to the old one. A small parameter changes few chromosomes, while a big one would almost build a brand-new individual.

Before completing the crossover phase it is necessary to check the mutation introduced, because the values of $\mathbf{u}_{i,G}$ inherited from $\mathbf{v}_{i,G}$ could be outside the boundaries. In those cases the values are reset as follows:

$$\mathbf{u}_{j,i,G} = \begin{cases} \min\{U_j, 2L_j - \mathbf{u}_{j,i,G}\} & \text{if } \mathbf{u}_{j,i,G} < L_j \\ \max\{L_j, 2U_j - \mathbf{u}_{j,i,G}\} & \text{if } \mathbf{u}_{j,i,G} > U_j \end{cases} \quad (3.7)$$

Finally, a simple selection operation is performed, bringing to the next generation the individuals between $\mathbf{x}_{i,G}$ and $\mathbf{u}_{i,G}$ which has the best fitness value:

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G} & \text{if } f(\mathbf{u}_{i,G}) < f(\mathbf{x}_{i,G}) \\ \mathbf{x}_{i,G} & \text{otherwise} \end{cases} \quad (3.8)$$

Once the new population is built, all the operations of mutation, crossover and selection are repeated until convergence or termination criterion is reached.

In the basic version of this algorithm, the parameters which handle the entire optimization are only three. The first encountered is the population size: its typical value ranges between 5 and 10 times the dimension problem. However, it depends a lot on the number of decision variables, which could easily exceed a hundred. Dealing with the scaling factor, its usual range is between 0.4 and 1, but changing this value could be enhanced exploration or exploitation of the design space. Finally there is the crossover factor Cr which needs to be optimized in the different situations. Little values of this parameter are suitable for low coupled functions (e.g. in additive relation $Cr \in (0,0.2)$), while larger values for high coupled ones ($Cr \in (0.9,1)$).

Finally, the suggested mutation strategies are *DE Curr to Best 1* and *DE Curr to Rand 1* from 3.5.

Algorithm 4: Pseudo code of differential evolution algorithm

Result: Obtain the optimal value of fitness function

Input: N_p , F , Cr

Initialize randomly the population $P_G = \{\mathbf{x}_{1,G}, \dots, \mathbf{x}_{N_p,G}\}$;

while *termination criterion is not satisfied* **do**

for *each* i *in* P_G **do**

$\mathbf{v}_{i,G} = \text{Mutate}(\mathbf{x}_{i,G})$ by eq. 3.5;

$\mathbf{u}_{i,G} = \text{Crossover}(\mathbf{x}_{i,G}, \mathbf{v}_{i,G})$ by eq. 3.6;

$P_{g+1} = \text{Selection}(\mathbf{x}_{i,G}, \mathbf{u}_{i,G})$ by eq. 3.8;

end

$G = G + 1$

end

3.5 Shuffled frog leaping

Shuffled frog leaping is an algorithm inspired by the behaviour of frogs searching for the location which has maximum food. Thought and developed first by Eusuff and Lansey [15], it has been applied to several engineering problems, exhibiting a fast convergence speed. It is a genetic-memetic algorithm, which means that it combines population based approach⁵, with individual learning and local improvement (as seen in particle swarm optimization, 3.3). Thus, it takes advantage of a dual-phase evolution: in the first phase it uses the individual characteristics to find a better solution. In the successive phase it cooperates with the population to achieve an even better objective value. This building allows the algorithms to get several benefits from both the structures used.

Shuffled frog leaping deals with a population of possible solutions which are divided into sub-groups called memplexes. Inside the memplex, each individual performs a local search and it communicates with all the other candidate solutions inside the group to gain better values. A special case is the best fit solution, which can communicate with all the other individuals and not only with those in its group. After a certain number of evolution steps inside each memplex, the groups get mixed and new memplexes are generated randomly. In this way each frog performs the local search and it interfaces cyclically with new set of sub-populations. This process goes on until the termination criteria gets reach, which could be a stagnation of fitness best value (its improvement is less than a specified tolerance) or a threshold value of shuffling process is exceeded⁶.

Analysing in detail the algorithm structure, once the population is generated and candidate solutions are evaluated and sorted, individuals are subdivided into memplex. It is possible to locate in each group the best and the worst individuals, respectively \mathbf{x}_w and \mathbf{x}_b , and the global best one, \mathbf{x}_g . Hence, the element having worst fitness value inside the memplex is submitted to the improvement process. First the frog leaping step size gets calculated:

$$D_i = \text{rand}_{(0,1)}(\mathbf{x}_b - \mathbf{x}_w) \quad (3.9)$$

The i subscript always indicate the i -th population individual, which has also the worst fitness in the memplex. Recall also that each i -th element is a vector which

⁵typical of classical genetic algorithm, 3.2

⁶Here the shuffle process can be seen as the number of generations.

Algorithm 5: Pseudo code of Shuffle Frog Leaping

Result: Obtain the optimal value of fitness function**Input:** N_p , $N_{memeplex}$, $iter_{max}$, $G = 1$ Initialize randomly the population $P_G = \{\mathbf{x}_{1,G}, \dots, \mathbf{x}_{N_p,G}\}$;**while** *termination criterion is not reached* **do**

Evaluate and sort the population by fitness function;

Randomly collocate individuals in the memeplexes;

 Define \mathbf{x}_g ; **for** *each memeplex* **do** Define \mathbf{x}_b ; **for** i *in range* $iter_{max}$ **do** Define \mathbf{x}_w ; Calculate the step size: $D_i = rand_{(0,1)}(\mathbf{x}_b - \mathbf{x}_w)$; Calculate the new solution: $\mathbf{x}_{newWorst} = \mathbf{x}_w + D_i$; **if** $f(\mathbf{x}_{newWorst}) < f(\mathbf{x}_w)$ **then** $\mathbf{x}_i = \mathbf{x}_{newWorst}$ **else** Calculate the step size with global best: $D_i = rand_{(0,1)}(\mathbf{x}_g - \mathbf{x}_w)$; Calculate the new solution: $\mathbf{x}_{newWorst2} = \mathbf{x}_w + D_i$; **if** $f(\mathbf{x}_{newWorst2}) < f(\mathbf{x}_w)$ **then** $\mathbf{x}_i = \mathbf{x}_{newWorst2}$ **else** $\mathbf{x}_i = rand(\mathbf{b}_{lo}, \mathbf{b}_{up}^7)$ **if** $f(\mathbf{x}_i) < f(\mathbf{x}_b)$ **then** $\mathbf{x}_b \rightarrow \mathbf{x}_i$ **end** **if** $f(\mathbf{x}_i) < f(\mathbf{x}_g)$ **then** $\mathbf{x}_g \rightarrow \mathbf{x}_i$ **end** **end** **end**

Replace old population with the evolved individuals;

 $G = G + 1$ **end**

components are the n decision variable: $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in})$. After this step, the frog will be located in a position between the best and the worst points:

$$\mathbf{x}_{newW} = \mathbf{x}_w + D_i \quad (3.10)$$

If the fitness value of \mathbf{x}_{newW} improves with respect to the old one, then the new point is accepted. Otherwise the calculation of 3.9 and 3.10 are repeated, using this time global best individual \mathbf{x}_g instead of the local one \mathbf{x}_b . If no improvements are obtained again, then a new random individual replaces the worst one. Once the new element is fixed and evaluated, then the process goes on using the new worst individual in the memplex. After a fixed number of iteration inside a single memplex, this procedure develops on the next group, involving one by one all the memplexes. Hence the new individuals generated take the place of the old population and get sorted by their fitness values. Finally the elements get again divided into new memplexes and the process starts all over again, until convergence or the maximum number of shuffle is reached.

3.6 Ant colony optimization

Ant Colony optimization (ACO) is an evolutionary algorithm which deals mainly with combinatorial problems. It uses approximate approaches, based on the behaviour of real ant colonies. The first example of this model is the Ant System [14, 41], which aimed to solve the Travelling Salesman Problem and obtained poor results with respect to other state-of-the-art algorithms. Nevertheless, further developments realize the so-called ACO algorithm, which is used in several applications as sequential ordering, scheduling, assembly line balancing and so on [20, 26] with very good results.

Ant Colony optimization simulates the behaviour of ants searching for food. The real searching process is based on different types of pheromones released by each ant itself. These indicate the presence of food in that path or on the other hand its absence. As the ants pass by, attracted by the food, this signal becomes lower and lower until it gets completely consumed. However, each ant has also the possibility to explore new path in search of other source of food. What is fixed almost always is the path length, hence the number of steps that each ant does during the search, walking away from the colony. Once reached this maximum number, the ant goes back to the colony exactly on the same path, releasing the pheromones along the way to indicate the result of its search. The computational algorithm behaves almost in the same way, making use of pheromones and fixed number of steps. These allow to exploit carefully the searching space around the colony location. Though, to introduce the exploration on the entire domain it is necessary to start the algorithm from different sites. Another choice could tune the path length, but it has been proven that too much steps inhibit the exploitation capacity.

$$s = \begin{cases} \underset{u \in J_k(r)}{\operatorname{argmax}} \{ [\tau(r, u)]^\alpha [\eta(r, u)]^\beta \} & \text{if } q \leq q_0 \\ P_k(r, s) & \text{otherwise} \end{cases} \quad (3.11)$$

⁷ \mathbf{b}_{lo} and \mathbf{b}_{up} values are the lower and upper boundaries of the search space variables, expressed as vectors.

$$P_k(r, s) = \begin{cases} \frac{[\tau(r, s)]^\alpha [\eta(r, s)]^\beta}{\sum_{s \in J_k(s)} [\tau(r, s)]^\alpha [\eta(r, s)]^\beta} & s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \quad (3.12)$$

Equation 3.11 builds the path of each ant and the choice between the two possibilities depends on the value of q_0 and q . Both of them are real numbers between 0 and 1, but while q_0 is a fixed value working as a threshold, q is chosen randomly and it defines which part of eq. 3.11 will be used. If $q \leq q_0$ the ant progresses to the node with largest quantity of pheromones in its neighbourhood, otherwise the following step is based on the eq. 3.12. In this case the ant's move depends on the quantity of pheromones between the two nodes $\tau(r, s)$, on the inverse of their distance $\mu(r, s)$ and on the presence of unexplored adjacent nodes which belongs to $J_k(r)$. Finally, the values α and β are constants which control the dependence on the pheromone presence, hence the way it does affect the taken path. The choice of the constant values α , β and q_0 needs to be properly tuned, running the algorithm for several tests and investigating its behaviour.

$$\tau(r, s) = (1 - \rho) \tau(r, s) + \rho \Delta\tau(r, s) \quad \text{Local pheromone update} \quad (3.13)$$

$$\tau(r, s) = (1 - \sigma) \tau(r, s) + \sigma \Delta\tau(r, s) \quad \text{Global pheromone update} \quad (3.14)$$

$$\Delta\tau(r, s) = \begin{cases} \frac{Q}{L_k} & \text{if the ant move from node } r \text{ to node } s \\ 0 & \text{otherwise} \end{cases} \quad (3.15)$$

The updating process of pheromones prevents the ants from selecting always the best-known path and it leads the exploration to new regions. The parameters ρ in eq. 3.13 and σ in 3.14 control the pheromones evaporation rate respectively along the selected route and on the entire search space, once the paths of all the ants have been chosen. $\Delta\tau(r, s)$ of eq. 3.15 is the reduction of pheromone between nodes (r, s) , controlled by the constant Q and the objective value of the selected route L_k .

Further developments of the algorithm introduce a multi-pheromone procedure, which allows the ants to communicate, defend and cooperate more effectively. These features lead to solve complicated problems without too large computational effort, but it builds a much more complex procedure.

3.7 Other meta-heuristic methods

Simulated annealing

Simulated annealing is another technique to search for the global optimum of a given function. It cannot be classified within the framework of evolution algorithms, since it lacks all their main features, but its mechanisms can easily remind to an evolution process. This algorithm works better with discrete search spaces, since first applications were related to combinatorial optimization in [36], but it has been generalized to process also dealing with continuous domain.

The method's name derives from the analogue process of physical annealing in metallurgy. A crystalline body is subjected to a heating process and a successive very slow cooling down. This allows the crystal lattice to become more regular and the grains to organize better. The aim is to obtain a solid free of defects. Inside the solid structure this happens because each atom has the chance to find its optimum position or to rearrange better if the time of temperature cooling is very long. It means that the solid reach lower and lower free energy configuration the longer is the annealing process.

Dealing now with the numerical code applied on a specific function, its objective is to find the global minimum and this is done always accepting the improving solutions and debating the non-improving ones. While obviously the former will bring better and better function values, the latter are fundamentals to prevent the procedure from getting stuck in a local minimum. Each new solution is in the neighbourhood of current position, hence the following considered point cannot be far and it jumps out the local minimum. Providing the escape from local optima through the *hill climbing* is a peculiar and key feature which always allows the algorithm to aim at the global optima, but this depends on the temperature parameter.

Given the search space and the objective function, the algorithm starts to initialize a solution $\mathbf{x} = (x_1, x_2, \dots, x_n)$, which is a vector whose components are the decision variables values. A neighbour solution \mathbf{x}' is generated randomly changing one or more component or applying a specified rule. Anyway the new solution \mathbf{x}' is always very much similar to the old one. Then the two solutions are compared using the Metropolis acceptance criterion: the new solution is always accepted if it achieves a better fitness value, otherwise it can be accepted with a probability depending on the temperature reached:

$$P\{\text{accept } \mathbf{x}'\} = \begin{cases} \exp[-(f(\mathbf{x}') - f(\mathbf{x}))/T_k] & \text{if } f(\mathbf{x}') > f(\mathbf{x}) \\ 1 & \text{if } f(\mathbf{x}') \leq f(\mathbf{x}) \end{cases} \quad (3.16)$$

In this equation T_k is the value of temperature assumed at iteration k . Despite it never reaches 0 and the hill climbing process is always possible, as the temperature decrease, the probability of accepting a worsening solution becomes lower. Hence at the ending phases it is unlikely to perform a climb and exits the local minimum. On the other hand at the beginning phases temperature is quite high and enables to accept worst solution with fair probability. This gives to the algorithm exploration capacity, while exploitation enhances at the end of the process. A further implementation of the algorithm could allow to test more than one searching step at each temperature level.

While the development of such algorithm is immediate in the discrete case, dealing with the neighbourhood aspects and treating continuous functions can be much more complicate. In the former case an optimum choice to fund near solutions to the current one can be performed simply changing one or more values of the decision variables by just one unit. Instead, in the latter case no unit values can be identified, so the decision variables modifications could be done by introducing a distribution function. The new component value would be randomly extract from a normal distribution centred on the actual value and with a predefined variance. Similarly, it could be adopted any other distribution, as a uniform one, with upper and lower bound which are a fixed quantity away from the actual value.

Talking about the neighbourhood of a general point \mathbf{x} in a discrete search space, it is possible to define a non-negative square matrix \mathbf{P}_k filled with all the transition

Algorithm 6: Pseudo code of simulation annealing

Result: Obtain the optimal value of fitness function
Input: $iter_{max}$, $Restart_{max}$, $T_{annealing}$
while *termination criterion is not reached* or $R < Restart_{max}$ **do**
 Initialize the starting solution: \mathbf{x}_R ;
 Initialize annealing temperature: $T_0 = T_{annealing}$;
 for i in range $iter_{max}$ **do**
 Pick a neighbour randomly or by a rule: $\mathbf{x}'_R \leftarrow neighbour(\mathbf{x}_R)$;
 Calculate $P\{accept \mathbf{x}'_R\}$ using 3.16;
 if $rand_{(0,1)} \leq P\{accept \mathbf{x}'_R\}$ **then**
 $\mathbf{x}_R \leftarrow \mathbf{x}'_R$
 else
 \mathbf{x}_R does not change
 $T_i \leftarrow T_{annealing} \frac{iter_{max}-i}{iter_{max}}$;
 end
 Store final state: \mathbf{x}_R ;
 $R = R + 1$;
end

probabilities. As stated in [20], defined $g_k(\mathbf{x}, \mathbf{x}')$ the probability of generating candidate solution \mathbf{x}' from neighbourhood on \mathbf{x} , then:

$$\sum_{\mathbf{x}' \in nb(\mathbf{x})} g_k(\mathbf{x}, \mathbf{x}') = 1 \quad \forall \mathbf{x} \in \Omega, \quad k = 1, 2, \dots$$

Defined also the difference between two different point as:

$$\Delta_{\mathbf{x}, \mathbf{x}'} \equiv f(\mathbf{x}') - f(\mathbf{x})$$

Then it is possible to write the stochastic matrix \mathbf{P}_k :

$$\mathbf{P}_k(\mathbf{x}, \mathbf{x}') = \begin{cases} g_k(\mathbf{x}, \mathbf{x}') \exp(-\frac{\Delta_{\mathbf{x}, \mathbf{x}'}}{T_k}) & \text{if } \mathbf{x}' \in nb(\mathbf{x}) \quad \mathbf{x}' \neq \mathbf{x} \\ 0 & \text{if } \mathbf{x}' \notin nb(\mathbf{x}) \quad \mathbf{x}' \neq \mathbf{x} \\ 1 - \sum_{\mathbf{x}'' \in nb(\mathbf{x}), \mathbf{x}'' \neq \mathbf{x}} \mathbf{P}_k(\mathbf{x}, \mathbf{x}'') & \text{if } \mathbf{x}' = \mathbf{x} \end{cases} \quad (3.17)$$

Finally in multi-optimization problems, it gets harder to deal with the acceptance criteria for new solutions, since it cannot be found a single value to treat any more. A first choice could be to accept always the non-dominated solutions, storing them in a separate list, while using an aggregate fitness function (weighting the different objectives) to evaluate worsening solutions. Another idea introduces in the formulation multiple annealing temperatures, one for each objective function, building the same framework as working with a single objective [30].

Nowadays the most spread version of this meta-heuristic is the *Adaptive Simulated Annealing*. It builds an algorithm which can properly control and tune the parameters as the temperature schedule and the random step. These features make the method

⁸ $nb(\mathbf{x})$ is the set of the neighbour of \mathbf{x}

more robust and efficient, while also less sensitive to the user defined parameters. Several implementations exist, each one displaying particular features took from real-experience observation or statistical tools [27].

Tabu search

Tabu search is a meta-heuristic algorithm based on a local search algorithm, originally proposed by Glover [22] and used for mathematical optimization. Its main application is in combinatorial problems, which require optimal ordering and selection of options, as resource planning, scheduling, logistics, energy distribution and many others.

The Tabu search takes advantage of local search improvement techniques. It develops an iterative procedure from a feasible starting point inside the decision variables domain. The improvement is characterized by fair modification of the last input, realizing only slightly changes to some decision variable of the input vector. Hence most of the times also the fitness function achieves little gain. The path choice between the current neighbour points could evaluate all its surrounding point and then it should select the best one or it could choose at random a point in the neighbourhood that gives a fitness improvement. The local search terminates when an optimum is found, whether it is local or global. Unless being extremely lucky, one finds local optimum, which is often not a good enough solution.

Algorithm 7: Simple Tabu Search pseudo-code

```

Input: Initialize starting solution  $S_0$ 
 $S_{Best} \leftarrow S_0$ ;
 $TabuList \leftarrow \emptyset$ ;
 $TabuList.push(S_0)$ ;
while do
     $S_{nbb} \leftarrow getNeighbors(Best_{newS})$ ;
     $Best_{newS} \leftarrow S_{nbb}.firstElement$ ;
    for  $S_{newS}$  in  $S_{nbb}$  do
        if  $S_{newS} \notin TabuList$  and  $fitness(S_{newS}) > fitness(Best_{newS})$  then
             $Best_{newS} \leftarrow S_{newS}$ 
        end
    end
    if  $fitness(Best_{newS}) > fitness(S_{Best})$  then
         $S_{Best} \leftarrow Best_{newS}$ 
    end
     $TabuList.push(Best_{newS})$ ;
    if  $TabuList.size > maxTabuSize$  then
         $TabuList.removeFirst()$ 
    end
end
Result: return  $S_{Best}$ 

```

In this framework, Tabu Search allows the path to climb up passing through worse solutions and to exit the local minimum zone. The basic principle of this method is to develop local search and whenever local optima are encountered (no more improving neighbours present), let non-improving moves happen. To prevent the algorithm from visiting twice the same solutions, a list of recent search step is generated, which gets

called *Tabu List* and it gives the name to the method. However, this list does not contain the entire path followed, but only few entries, whose number depends on the problem dimension and the available memory to allocate. The list is then managed by the *last in first out* rule, when the entire memory has been filled up by visited points and a new element needs to be inserted. The oldest one located in the list is eliminated, the entire list scrolls up and the new input is inserted at the last position.

In general, three types of tabu lists exist and these are related to different kind of categories due to their persistence:

Short term It contains the path just walked, forbidding the searching process to visit again those solutions. This list alone could build a simple Tabu search, but usually it cannot satisfy criterion of exploitation and exploration.

Intermediate term This memory drives the search towards promising areas or locations which satisfy a certain criterion. For example, this list could ban solutions which present determined attributes and it evaluates as better those individuals showing some other characteristics, hence performing exploitation of the search space.

Long term This list defines the diversification and exploration rules. It leads the searching process to uncharted locations or it drives search out of local optimum regions.

In this chapter just few meta-heuristic optimization methods were reviewed and about each of them it had been done a brief description. Further more can be find, classified by local or global search as well as single versus population based or hybrid and memetic algorithms. Some of them work better on single objective problems, while others perform better on multi-objective ones. For example, this latter, beside those already described, are *Social Cognitive Optimization*, *Artificial Bee Colony*, *mathematical programming*, *machine learning*, *combinatorial optimization*, *scatter search*, *iterated local search*, *variable neighbourhood search* and many others.

Chapter 4

Surrogate Models

All the optimization methods introduced up to now in chapter 3 can perfectly run on their own, retrieving a solution. In fact, evolutionary algorithms require several runs to properly adjust the parameters. Therefore a large number of fitness function evaluations for each run are needed to realize proper system settings for the optimization process. Unfortunately, simulations' computation is often very expensive and optimization evaluation timing could last up to hours or even days. So the minimization task becomes usually impractical and also prohibitive, even if it could seem simple and direct. The searched results often deal with on-line process or with already deeply simplified models. In these frame, surrogate models set up.

These are the main reasons why optimization requires *surrogate models* (often identified by \hat{f}), which are cheap to evaluate and their response values emulate those of the original function f . This objective function is the starting point not only of the surrogate model which builds itself upon it, but also of the entire optimization task, that starts with the definition of an analytical model. Dealing with the original function, it is always much more complex than the surrogate. In fact, its only objective is to lighten the problem and to evaluate costs, even if it could involve completely different mathematical relations for the experimental data. Although these models are cheap to use, the building phase is still expensive because it needs anyway to perform evaluation of the original function. So surrogates must be used sparingly and their construction requires the definition of a proper input set. This last is itself the base of the model: it needs to be the smallest possible but at the same time it must provide the best prediction possible.

The main phases in building a surrogate model are the definition of a sampling and testing plan. These could be constrained by the problem at hand and be dependent on the degree of complexity and accuracy desired. In this section the building blocks of surrogate models will be treated in detail and a brief description of some widespread model will be given.

4.1 Main features

The first stage to approach the modelling process and realize the surrogate should start with the sensitivity analysis, which will be debated in the next chapter 5. It evaluates the amount of influence to the fitness of each variable taken by its own or interacting with other variables. This is not always simple to carry out since it requires several function evaluations. So this process is proportional to the number of decision variables

and hence it is subject to the curse of dimensionality 2.1.2.

Identification of the most significant input variables could lead to huge problem simplification and mainly to a better knowledge of the fitness function itself. It can help to provide a more accurate and argued choice of surrogate model and evolutionary algorithm. This could realize also huge amount of computational savings. Once the process is done and the critical decision variables are recognized, it is necessary to build the *learning data set*. It is the set of points on which the fitness function f gets evaluated and hence where the surrogate \hat{f} is built. Depending on the number of parameters which are needed to describe the problem, the surrogate will require a minimum number of different points to allow the parameter estimation. Building the learning set with only such points would save lot of computational time but it would often lead to poor prediction capacity of the model. On the other hand, building enormous data set does not enhance the model capacities: in fact, above a given threshold, new learning points would not improve the surrogate performance, realizing the so called *overfitting*. This can grow the amount of evaluation errors and computational efforts, mainly due to the excessive quantity of points considered.

Let's take for example the *needles in haystack* function as in [18]. It takes zero values everywhere but in specific points of the domain, which ideally represent the needles. The perfect surrogate model of this function is rather simple:

$$\hat{f}(\mathbf{x}) = \begin{cases} y^{(1)} & \text{if } \mathbf{x} = \mathbf{x}^{(1)} \\ y^{(2)} & \text{if } \mathbf{x} = \mathbf{x}^{(2)} \\ \dots & \dots\dots \\ y^{(n)} & \text{if } \mathbf{x} = \mathbf{x}^{(n)} \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

The model 4.1 requires only the needles location knowledge to properly define it. Anyway just few examples of engineering functions behave like this.

Talking about the learning set construction, first of all it is always a good idea to scale the domain variable as the unit cube $[0, 1]^k$, where k is the number of variables. Since often the search space is continuous and hence dense, so are the possible learning points. Once it is necessary to define the set, there are not choices that perform always better than others. For example, dealing with very flattened function, if one takes only boundaries points, the set could build a good prediction, but in presence of wide changing inside the domain it would have bad performance. Anyway, by its own nature, the model will not ever perfectly fit the original function everywhere, also because it needs to be simply evaluate. Moreover, the construction should minimize the error of the surrogate with respect to the real function. This can happen when the parameters are less than the chosen points, realizing another minimization task which is the least square problem. In case of the same number of parameters and points, only one configuration of the parameter is possible and it realizes the exact fitting of the set¹, but often it provides poor results on other points.

Differently from experimental design of real processes, a surrogate model does not exhibit noise in the observed responses. Once retrieved the model features, its behaviour and the fitness values resulting from its evaluation will be always the same, even if an input is submitted twice, by the analytical nature of the surrogate.

¹When the numbers of points in the fitting set and of parameter in the surrogate are the same, the model predictions take the exact values of the original test function on the sample set by construction.

The main problems arising from this building are the *lack of fitting* and the *overfitting*. The latter happens when the model is too flexible and it tries to fit all the given point. When there are further points than those needed and the fitting process goes on, the mean error of the entire fitting set increases, but the code does not recognize it. Instead, the former happens when the learning set does not cover uniformly the search space. Adding to this set new points, it could improve its prediction performance. However both of them depend on the surrogate model defined. If one tries to fit a linear function by using a second order model, this model would never reach an adequate error level both increasing or decreasing the learning data set, since is the model itself wrong. Hence the most important part is first of all comprehend and mimic the objective function. So also this framework needs proper choices of different aspects, which can be made only by testing several parameters tuning, because each problem has its own features.

Once identified the critical variables, defined the proper shape of surrogate model and generated the learning data set, the following phase also the model parameters \mathbf{w} need to be identified. Usually this is done using methods as:

Maximum likelihood estimation This method estimates the model parameter given some observations, through looking for the parameters' values which maximize the likelihood of retrieving the observation. For example, assuming that a chosen property of a sample has a normal distribution, *maximum likelihood estimation* selects mean and variance as parameters and it tunes them to build a prediction as closer to the observation as possible.

Bayesian estimation It minimizes the posterior expected value of a loss function or equivalently it maximizes the posterior expectation of a utility function. Also here a parameter tuning is required, but in this case the performance depends on the chosen loss or utility function. Mean square error and median-unbiased estimators are example of loss functions.

Cross validation As the name suggests, this method focuses on the validation technique to estimate how accurately a model performs. Once defined the model parameters to search for, the observed data are divided into complementary subsets, performing the analysis on one subset and validating the analysis on the other subset.

These search for the best configuration of values for \mathbf{w} to realize the most truthful model possible. It could be also introduced a weighting vector to highlight which are the most important data to fit, but this is seldom done.

The most applied and common fitting criterion is the least squares, which minimizes the sum of squared errors: considering each data point as $\mathbf{x}^{(i)}$, the objective writes as:

$$\min_{\mathbf{w}} \sum_{i=1}^n \left[y^{(i)} - \hat{f}(\mathbf{x}^{(i)}, \mathbf{w}) \right]^2 \quad (4.2)$$

This approach however can be simply solved using matrix formulation, since the minimization problem can be analytically translated in mathematical formulas. Consider $\mathbf{y} = (y_1, y_2, \dots, y_n)$ the vector of data output, $\mathbf{w} = (w_1, w_2, \dots, w_k)$ the vector of unknown parameters and $\mathbf{X} = \hat{f}(\mathbf{x}, \mathbf{w})$ the matrix whose rows are related with the input

data and columns represent the respective parameters. Then, eq. 4.2 can be written and developed as:

$$\min_{\mathbf{w}} [\mathbf{y} - \mathbf{X}\mathbf{w}] [\mathbf{y} - \mathbf{X}\mathbf{w}]^T$$

This happens because in matrix notation the sum of the squares can be simply translated into the scalar product. Deriving the previous expression with respect to \mathbf{w} and setting it equal to zero, one performs the optimization task and obtains:

$$\begin{aligned} -\mathbf{X} [\mathbf{y} - \mathbf{X}\mathbf{w}]^T - [\mathbf{y} - \mathbf{X}\mathbf{w}] \mathbf{X}^T &= 0 \\ 2\mathbf{X}^T [\mathbf{y} - \mathbf{X}\mathbf{w}] &= 0 \\ \mathbf{X}^T \mathbf{y} - \mathbf{X}^T \mathbf{X} \mathbf{w} &= 0 \\ \mathbf{X}^T \mathbf{X} \mathbf{w} &= \mathbf{X}^T \mathbf{y} \end{aligned}$$

Therefore, the final solution is:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (4.3)$$

This phase is often called the model training. Dealing with particular surrogate as the *artificial neural network* (4.3), instead of using the least squares method, one by one the data gets fitted by the model and parameters are properly tuned. The fitting process stops when the entire data set has been used. Otherwise after a given threshold, it stops if an input generates a decay in the estimation.

The final stage consists in testing the model to evaluate its error. Typically the starting data set gets divided into 2 part, the training and testing one. The latter is much smaller than the former: often less than 20% of the total set is allocated to the testing phase. The sole purpose of the test is to measure the error once the entire model has been built. The most spread error evaluations are root mean square error (RMSE) and correlation coefficient.

This phase is not always performed, in particular dealing with initial testing model, when different surrogates are compared to measure their computational requirements. The reason why the testing set is taken from the original one and it is not built brand new, is because such construction fixes a maximum number of fitness function evaluation. Therefore it is possible to avoid too many burdening evaluations and it allocates a relative amount of time to this phase.

Let's now take a look to some models used nowadays.

4.2 Kriging

Kriging is a spatial statistical technique developed by Danie Gerhardus Krige. Initially it was applied in geological settings and later it became used widely in engineering after deep reviews, modifications and improvements. Kriging is a Gaussian process² based modelling method, alternative to the conventional response surface or regression. It fits data obtained from large experimental areas. It builds global interpolation rather than local one, successively used for the prediction phase.

²A Gaussian process is statistical model where the observations occur in the domain space distributed as a normal random variable [18].

There exist two different types of Kriging [40]. The mainly adopted is the Ordinary Kriging, which takes advantage of a constant drift function to develop a simple construction of the model. On the other hand, it limits the fitting capacity to complex data. Other references use the Universal Kriging, which adopts as drift function a polynomial generated by base functions. This is much harder to deal with, since the choice of appropriate basis functions is not easy and it burdens the entire computational process.

Anyway, Kriging has been widely applied in optimization problems, exploiting its interpolation accuracy. It develops an optimization that can be used to approximate complicate fitness functions and to assist evolutionary algorithms. However Kriging can also work on its own, as an optimum searching tool per se, called Sequential Kriging Optimization [40].

The basic idea of Kriging is to predict the function value at a given point by computing a weighted average of its neighbour point values, similarly to regression analysis. Let's consider a stochastic process $y(x)$ of the form:

$$y(\mathbf{x}) = \mu(\mathbf{x}) + \epsilon(\mathbf{x}) \quad (4.4)$$

In 4.4 the \mathbf{x} vector is the k -dimensional point representing the decision variables. $\mu(\mathbf{x})$ is a mean function called *drift function*, which describes the average behaviour of $y(\mathbf{x})$. Finally $\epsilon(\mathbf{x})$ is a white noise stochastic error function, therefore distributed as a Gaussian variable with zero mean and unknown variance.

A typical basis function used to fit the data is of the form:

$$\psi^{(i)} = \exp \left(- \sum_{j=1}^k \theta_j |x_j^{(i)} - x_j|^{p_j} \right) \quad (4.5)$$

The relation with this last formula and the equation 4.4 will be explained in the following.

Analysing formula 4.5, its similarities with a Gaussian distribution are evident. Here, instead of the the variance as denominator at the exponential, Kriging basis has the vector $\boldsymbol{\theta} = \theta_1, \dots, \theta_k^T$, which allows to fit each variable. Furthermore, instead of squaring the numerator, Kriging has another vectorial exponent $\mathbf{p} = p_1, \dots, p_k^T$ which again varies with each single variable. This parameter often takes values $p_j \in [1, 2]$, but nothing forbids to widen the range. This kind of basis let the model fit the data remarkably well, but on the other hand it requires a careful building to its proper functioning.

Let's now see how the Kriging interpolation works. Given a set of sample data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and the relative observed responses, $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$, the objective of Kriging is to find a prediction value at a new point \mathbf{x} . In this framework, the responses \mathbf{Y} are considered as results of a stochastic process and they are denoted as:

$$\mathbf{Y} = \begin{pmatrix} y(\mathbf{x}^1) \\ \vdots \\ y(\mathbf{x}^n) \end{pmatrix}$$

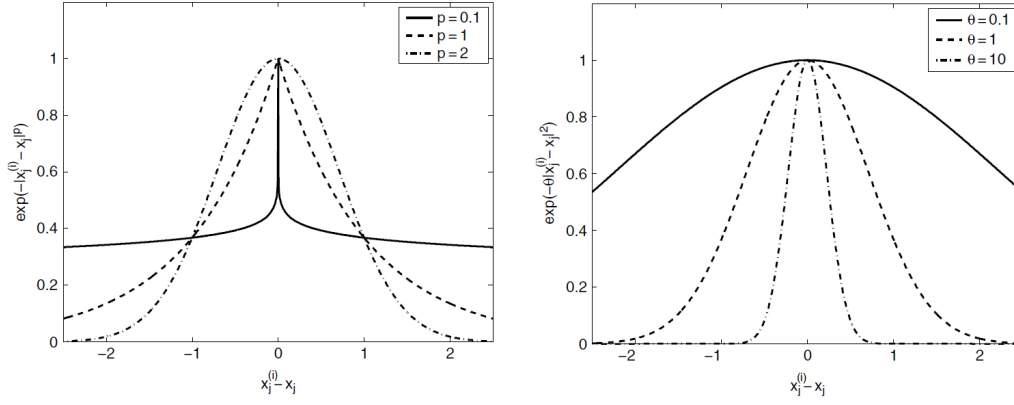


Figure 4.1: Correlation of the basis function by varying the distance between sample points. The figure on the left describes the correlation with varying \mathbf{p} , while the figure on the right with varying $\boldsymbol{\theta}$ [20].

From the output vector \mathbf{Y} can be defined a mean vector $\mathbf{1}\mu$ which is an n -dimensional column vector³. Each response variable can be related to the others by a correlation, which is expressed using the basis function:

$$Cor[y(\mathbf{x}^{(m)}), y(\mathbf{x}^{(n)})] = \exp \left(- \sum_{j=1}^k \theta_j |x_j^{(m)} - x_j^{(n)}|^{p_j} \right) \quad (4.6)$$

Consequently it is possible to build the $n \times n$ correlation matrix of the observed data, denoted as $\boldsymbol{\Psi}$, written explicitly in 4.7.

$$\boldsymbol{\Psi} = \begin{pmatrix} Cor[Y(\mathbf{x}^{(1)}), Y(\mathbf{x}^{(1)})] & \cdots & Cor[Y(\mathbf{x}^{(1)}), Y(\mathbf{x}^{(n)})] \\ \vdots & \ddots & \vdots \\ Cor[Y(\mathbf{x}^{(n)}), Y(\mathbf{x}^{(1)})] & \cdots & Cor[Y(\mathbf{x}^{(n)}), Y(\mathbf{x}^{(n)})] \end{pmatrix} \quad (4.7)$$

From this matrix it is possible to define the covariance matrix⁴:

$$Cov(\mathbf{Y}, \mathbf{Y}) = \sigma^2 \boldsymbol{\Psi} \quad (4.8)$$

All the entries of this last matrix depend on the distance between the sample points $|x_j^{(m)} - x_j^{(n)}|$ and on the parameters p_j and θ_j , which need to be properly evaluated. Figure 4.1 shows how the basis function varies as the points' separation increases or decreases. It highlights the correlation behaviour, in particular, as the p_j parameter gets lower, the correlation decreases much rapidly and it is almost discontinuous as the points distance approaches zero. Instead, the θ parameter affects the width of the correlation curve, thus how far the mutual influence between variables extents. A low value of θ

³ $\mathbf{1}$ is an $n \times 1$ column vector of ones.

⁴The covariance is a measure of the correlation between a set of two or more random variables:

$$Cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y$$

This gives rise to the relation between covariance and correlation:

$$Cor(X, Y) = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} \implies Cov(\mathbf{Y}, \mathbf{Y}) = \sigma_{\mathbf{Y}}^2 Cor(\mathbf{Y}, \mathbf{Y})$$

means a high correlation even for distant points, while a larger value realizes higher correlation for nearer points. Using this parameter, sometimes it is possible to rank and order the importance of variables. Even if this parameter does not say anything about the interaction, in some ways it can provide a brief order of importance.

The most used way to build this model is the *maximum likelihood method*, fitting the $\boldsymbol{\theta}$ and \mathbf{p} parameters to resemble \mathbf{y} . Building the model it is possible to assume that it will interpolate the data which is completely determined, so no error affects the \mathbf{Y} values. The likelihood problem aims to minimize the mean square error and it can be written as:

$$L(\mathbf{Y}^{(1)}, \dots, \mathbf{Y}^{(n)} | \mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp \left[-\frac{\sum (\mathbf{Y}^{(i)} - \mu)^2}{2\sigma^2} \right] \quad (4.9)$$

It can be expressed in terms of the sampling data and the correlation matrix, obtaining:

$$L(\mathbf{Y}^{(1)}, \dots, \mathbf{Y}^{(n)} | \mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2} |\boldsymbol{\Psi}|^{1/2}} \exp \left[-\frac{(\mathbf{Y} - \mathbf{1}\mu)^T \boldsymbol{\Psi}^{-1} (\mathbf{Y} - \mathbf{1}\mu)}{2\sigma^2} \right] \quad (4.10)$$

Finally the maximum likelihood estimate gives the following values, through the differentiation and minimization process:

$$\hat{\mu} = \frac{\mathbf{1}^T \boldsymbol{\Psi}^{-1} \mathbf{Y}}{\mathbf{1}^T \boldsymbol{\Psi}^{-1} \mathbf{1}} \quad (4.11)$$

$$\hat{\sigma}^2 = \frac{(\mathbf{Y} - \mathbf{1}\hat{\mu})^T \boldsymbol{\Psi}^{-1} (\mathbf{Y} - \mathbf{1}\hat{\mu})}{n} \quad (4.12)$$

Further explanations on the calculations above can be found in [18] and they work with matrix differentiation and minimization tasks.

However, the maximum likelihood estimation of eq. 4.10 can be substituted with the *concentrated ln-likelihood function* [18]:

$$\ln(L) \approx -\frac{n}{2} \ln(\hat{\sigma}^2) - \frac{1}{2} \ln |\boldsymbol{\Psi}| \quad (4.13)$$

This formulation leads to retrieve the parameters $\boldsymbol{\theta}$ and \mathbf{p} more simply. It takes advantage from the logarithmic function, since the results variation between $\theta = 0.1$ and $\theta = 1$ or between $\theta = 1$ and $\theta = 10$ is strongly enlarged. Then it becomes easier to search for $\hat{\boldsymbol{\theta}}$ in this scale. It is advisable to scale the search space between $[0, 1]$, in order to obtain values of the parameter θ_j which are more or less of the same degree varying the problem.

Dealing with $\hat{\mathbf{p}}$, in many problems it is suggested to fix this parameter as $\hat{\mathbf{p}} = 2$ and tune only the $\boldsymbol{\theta}$ entry. This because the problem already presents a large number of variables to control. Anyway, taking $\hat{\mathbf{p}} \in [1, 2]$ or without any constrain enhances the resulting fit and therefore the model qualities. However, this implies a higher computational cost.

Once the model has been built by the fitting process, it can be used to provide prediction on the function at hand. The estimation generated is of the form:

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \boldsymbol{\psi}^T \boldsymbol{\Psi}^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu}) \quad (4.14)$$

Algorithm 8: Pseudo-code of Kriging filter from [62]

Input: $offPop$, $offPop_1$, $offPop_2$
Define: $Dmin_1(offPop) \doteq$ minimum genotype Euclidean distance between each individual in $offPop$ and all the remaining individuals;
Define: $Dmin_2(offPop_1, offPop_2) \doteq$ minimum genotype Euclidean distance between each individual in $offPop_1$ and individuals in $offPop_2$;
Set: $strPop$ = population evaluated so far from beginning;
Set: $d_{Toll} = 10^{-3}$;
Set: M = number of objectives;
Set: I = number of offspring required;
Set: $strFit$ = population fitness evaluated so far from beginning;
for $i = 1, \dots, M$ **do**
| **Build** Kriging model on database ($strPop, strFit(:, i)$)
end
Set: $offEHVI$ = individual selected by EHVI⁵criterion;
Set: $dmin_2 = Dmin_2(offPop, [strPop; offEHVI])$;
Find: index vector i for $dmin_2(i) < d_{Toll}$;
Delete $offPop(i, :)$;
Set: $dmin = Dmin_1(offPop)$;
while $dmin < d_{Toll}$ **do**
| Find: index vector i for $dmin(i) < d_{Toll}$;
| Set: $popX = offPop(i, :)$;
| Set: $dmin_2 = Dmin_2(popX, strPop)$;
| Set: $dmin_{2,sort} = \text{sort } dmin_2 \text{ in ascending order; index vector } j \doteq dmin_{2,sort} = dmin_2(j)$;
| **Delete** $offPop(i(j[1 : end - 1]), :)$;
| Set: $dmin = Dmin_1(offPop)$;
end
for $i = 1, \dots, M$ **do**
| Set: $offFit(:, i)$ = prediction Kriging model of $offPop$;
end
Set: $rankPF$ = Pareto ranking of $offFit$;
Set: $rankPF_{sort} = \text{sort } rankPF \text{ in ascending order; index vector } j \doteq rankPF_{sort} = rankPF(j)$;
Set: n_{PF} = number of individuals on the Pareto Front;
if $size(offFit) > I - 1$ and $n_{PF} > I - 1$ **then**
| Set: $dmin_2 = Dmin_2(offPop, strPop)$;
| Set: $dmin_{2,sort} = \text{sort } dmin_2 \text{ in descending order; index vector } k \doteq dmin_{2,sort} = dmin_2(k)$;
| Set: vector index $iSel = j(k(1 : I - 1))$;
else
| Set: vector index $iSel = j(1 : I - 1)$;
Set: $selPop = [offPop(iSel, :); offEHVI]$;

Eq. 4.14 contains the vector ψ , which is the vector of correlation between observed data and new prediction. Since correlation does not depend on the estimation, its

evaluation is simple:

$$\psi = \begin{pmatrix} Cor[Y(\mathbf{x}^{(1)}), Y(\mathbf{x})] \\ \vdots \\ Cor[Y(\mathbf{x}^{(n)}), Y(\mathbf{x})] \end{pmatrix} = \begin{pmatrix} \psi^{(1)} \\ \vdots \\ \psi^{(n)} \end{pmatrix} \quad (4.15)$$

Thus, the estimation depends on all data points used in building the model. So its interpolation takes advantage of all the data available and expanding the relation it is possible to retrieve the contribution of each of those points. In [18] the entire construction is deeply explained in all its parts.

4.3 Artificial Neural Networks

Artificial neural networks (ANN) are processing systems that can be considered as a rough approximation of the biological neural networks. ANN nowadays are applied in the most different subjects. They are used to solve complex problems which requires algorithms able to identifies solutions without a specific rule-based approach. The building blocks of a network are the information processing units or neurons and the connections between layers of neurons. Each neuron realizes the operational structure, while the connection represents the synapses, characterized by scalar weights. These structures organize in three types of layers:

Input layer This layer receives the input decision variables of the fitness function to be approximated. Each input is associated to a neuron, so the number of nodes in this level is the same of the decision variables.

Hidden layer This layer receives the information from the input layer or another hidden layer. There could be more than one hidden layer in a single network. This layer elaborates the information through the connection weights and the transfer function and it sends them to the output layer or another hidden layer. The number of neurons in each hidden layer and the number of hidden layers themselves are chosen arbitrary. However, depending on the problem at hand many suggestion are present in literature.

Output layer It contains the output node, again one for each objective function. It receives the information from the last hidden layer or directly from the input one and it processes them by the transfer function and the connection weights.

As just seen, each ANN can be built arbitrarily and its classification is based on the number of layers. A single-layer network connects directly inputs with outputs, while multi-layer networks own one or more hidden layer and these realize greater processing of the information. Both are represented in fig. 4.2.

⁵EHVI is the *Expected Hyper-Volume Improvement* function, constructed as:

$$EHVI(\mathbf{x}) = \mathbf{E} \left[HV \left(Y \left(\mathbf{x}^{(1)} \right), \dots, Y \left(\mathbf{x}^{(n)} \right), Y(\mathbf{x}) \right) - HV \left(Y \left(\mathbf{x}^{(1)} \right), \dots, Y \left(\mathbf{x}^{(np)} \right) \right) | Y \left(\mathbf{x}^{(i)} \right) = f \left(\mathbf{x}^{(i)} \right) \right]$$

Here Y is a vector-valued objective function and $HV(Y^{(1)}, \dots, Y^{(np)})$ if hyper-volume of the set dominated by the point $Y^{(1)}, \dots, Y^{(np)}$. In any hyper-volume calculation, there has to be a reference point selected [62].

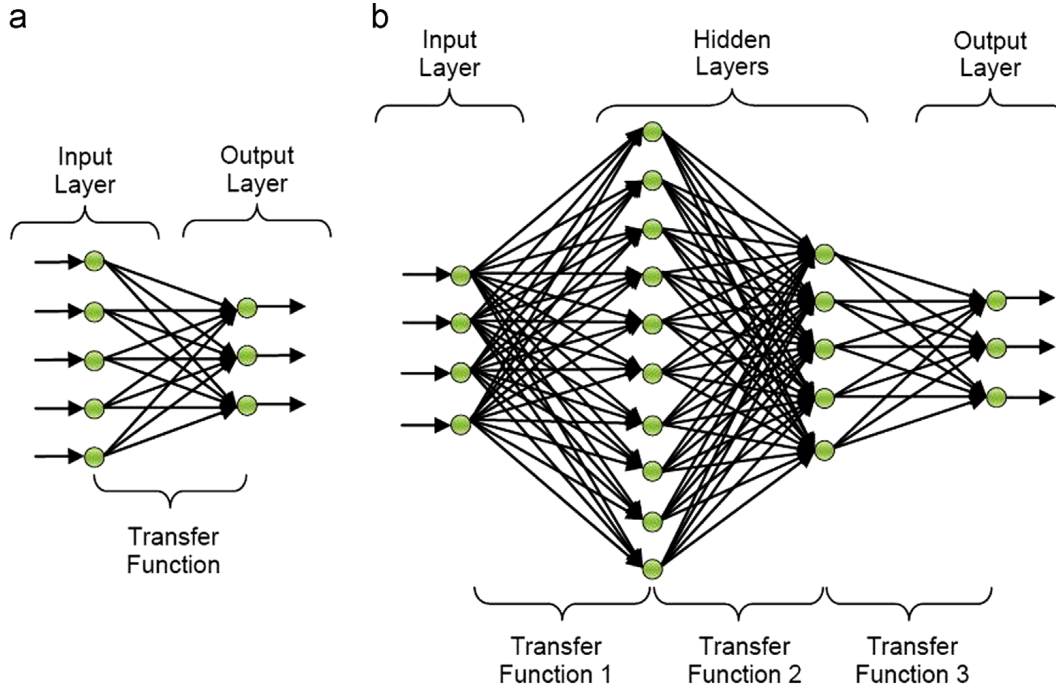


Figure 4.2: Classification of artificial neural network: (a) single-layer network, (b) multi-layer network [17].

In the layer's descriptions a transfer function was introduced. It is the function that process the information coming from the previous layer and it can take any shape. This function is made of two parts: the propagation function and the activation one. The propagation function regroups all the inputs in a single value before their use, so it computes the input $p_j(t)$ to the j -th neuron from the outputs $q_i(t)$ from the previous neurons. Typically, it has the following form:

$$p_j(t) = \sum_i q_i(t)w_{ij} + b_j \quad (4.16)$$

In 4.16, $p_j(t)$ is the information elaborated in the j -th neuron by the activation function. It depends on time since the neural network undergoes a learning process and in each phase the transmission sequence could change. On the other hand, $q_i(t)$ is the input coming from the i -th node of the former level. The value w_{ij} represents the weight between the neurons of the two different levels and it is the objective of the learning process. In fact, during the fitting procedure only these weights and the bias term (b) are subject to the tuning procedure. Finally, the information retrieved in the propagation function gets further processed in the neuron by the activation function, which ensures that the neuron's response is bounded and it allows to turn on or off the neuron. Despite in real neural network the perception is linear, usually the activation function has a non-linear shape to compute non-trivial problems with a small number of neurons. It takes the following shape:

$$z_j = \phi_j(p_j(t)) \quad (4.17)$$

This equation generates the actual neural response and its arguments are the outputs of the propagation function. The same procedure happens in the biological neurons to either large or small stimuli [38].

A first typical non-linear relation describing eq. 4.17 is the *sigmoid* or *logistic function*:

$$z_j = \frac{1}{1 + e^{-ap_k}}, \quad (4.18)$$

Here the parameter a controls the amount of non-linearity and hence the curve slope.

Another used function is the hyperbolic tangent, which yields the response:

$$z_j = \frac{e^{ap_k} - e^{-ap_k}}{e^{ap_k} + e^{-ap_k}} \quad (4.19)$$

Again, the parameter a tunes the curve slope, so the non-linearity. Both the functions, as already said, are bounded, although the respective co-domain is different. The sigmoid maps the input to $[0, 1]$, the hyperbolic tangent maps it to $[-1, 1]$ and are reported in fig. 4.3. In applications this aspect could imply better fit using one function instead of the other. The result of propagation function $p_j(t)$ works also as activation parameter. The set-up of neurons could provide particular fixed responses (or the *shut-down* of the neuron in particular implementation, when a given threshold is not exceeded), which depend on the value taken by the activation function.

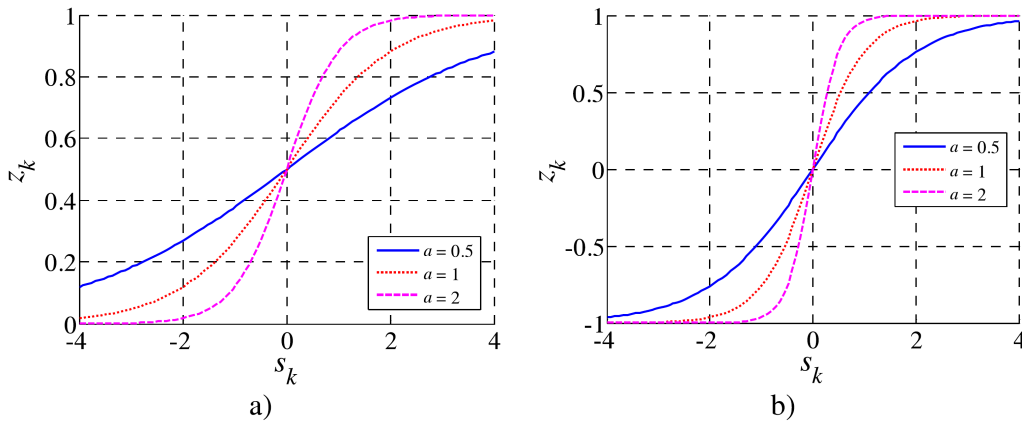


Figure 4.3: Most popular activation functions. a): sigmoid or logistic function. b): hyperbolic tangent function [38].

Such framework realizes an easy-to-develop method and it requires simple mathematical objects. Artificial Neural Network can profitably substitute statistical and other empirical methods that involve polynomials and regression based systems. Moreover it can be further simplified without using the activation function and, if necessary, mapping linearly the propagation output to a fixed bounded interval.

Once built the Neural Network, the following step consists in its training and testing. During the learning process, the ANN adapts itself to the various inputs to fit the outputs. It tunes the weights of synapses and the bias factors until it produces the best response possible. Different learning process exists to suit various applications. The latest ones allow also the network to change its own topology to realize the best model possible [9]. Adaptation could remove and generate neurons and synaptic connection to further simulate biological networks. The training develops a new optimization tasks, which can be solved by several methods, as quasi-Newton, conjugate-gradient, and

back-propagation. This last is nowadays one of the most common learning techniques. Given a configuration of the network and an input vector, the fitness function and the prediction is evaluated. Therefore, loss function is measured, as the distance between correct fitness \mathbf{y} and prediction \mathbf{y}' :

$$E(\mathbf{y}, \mathbf{y}') = \frac{1}{2} \|\mathbf{y} - \mathbf{y}'\|^2 \quad (4.20)$$

The standard choice to evaluate their distance is the *Euclidean distance* as in eq. 4.20. However, several other metrics can be used depending on the data at hand [9, 25]. The error function over n training input data will be simply the sum of the single errors $E = \frac{1}{2n} \sum_x \|\mathbf{y}(\mathbf{x}) - \mathbf{y}'(\mathbf{x})\|^2$. The objective of the training phase is to minimize or reach a threshold value before testing the built network and starting its use as a surrogate. The back-propagation optimization task is a two-steps procedure. Each input propagates through the network and it realizes the prediction. Successively this is compared with the correct value as in 4.20, obtaining the error. Thus, the error propagates backward in the network, giving the name to the method. So, each neuron gets attached with an error value. Last, the synapses weights and the bias factors are recalculated minimizing the error to the following nodes, for example using the least squares method. The aim is to minimize the loss function and to generate the best prediction possible. Typically to refresh the weights values a gradient method is used [38]. It evaluates a delta parameter related to the gradient of the weight and it gets partially removed or add to the current weight to gain a better performance.

Many others learning processes exist, as supervised or unsupervised techniques, knowledge based, competitive learning, each of them coming with its own main features. Also different ideas on the data set can be used, developing long-short-term memory, group methods and other variants.

Finally there is the testing phase which rates the network capacity to produce correct prediction. The training part of the starting data set is used in this step. Again each correct and predicted values are compared, but this time just to evaluate the error ratio. If this value is under a given threshold, for example 5% or 1%, the network can be used to the forecasting task for which it has been built. Otherwise the training phase must be performed again, modifying its frameworks, e.g. changing the learning data set, the network's topology or the learning process of the parameters.

4.4 Response Surface Methodology

Response Surface Methodology is a statistical technique which builds approximate and relative simple models. It takes advantage from design and analysis of experiments [3, 33, 34, 43]. Using output data related to chosen input variables of a certain problem, it is possible to mimic the behave of the original function and to obtain a cheap method to perform prediction. Depending on the number of modelled variables, the response surface can build different searching space, as lines, for 1-dimensional surface, surfaces themselves, in 2-dimensions, volumes and hyper-volumes when the design involves 3 or more parameters. However, as one draws the graph of the surface, $k + 1$ dimensions are necessary. This happens because the k value represents the number of decision variables and the further dimension displays the values taken by the surface itself point by point.

Each variable can represent different aspects but in the mathematical framework any kind of problem is written as:

$$\eta = f(\xi_1, \xi_2, \dots, \xi_k) = f(\boldsymbol{\xi}) \quad (4.21)$$

Observing eq. 4.21, η represents the surface, while ξ_1, \dots, ξ_k are the k *natural variables*. These describe the surface and they are expressed in the local frame. Each decision variable (written as x_i) has often its own domain and usually it gets translated through an opportune mapping to the local reference frame, where its domain becomes $[-1, 1]$ (or sometimes $[0, 1]$). Another typical feature of the variables is their independence in the surface model, despite this is often not true in the associated real application. This simplification is necessary, otherwise dealing with the problem would become much more difficult. It would involve further expensive tasks, as it could be correlation analysis [11], Principal Component Analysis [1], Cross-correlation [66] or Canonical Correlation [55].

The actual relationship between response surface and exact results can be written as:

$$y = f(\xi_1, \dots, \xi_k) + \epsilon \quad (4.22)$$

In eq. 4.22, beside the response surface $f(\boldsymbol{\xi})$, the ϵ term represents other source of variability not accounted for in the surface and so the error. Dealing with a pure computational model the error is only due to the approximation realized during the model fitting. Instead, treating a real problem, error would account also for measurement deviations, due to background noise or experimental variation, effects of uncontrolled or unknown variables and so on.

Now, analysing the response surface function, it can be defined as k -th order polynomial in a specific region of the domain space. The general expression of the surface is:

$$\eta = \beta_0 + \sum_{i=1}^k \beta_i x_i + \sum_{i=1}^k \sum_{j \geq i}^k \beta_{ij} x_i x_j + \sum_{i=1}^k \sum_{j \geq i}^k \sum_{l \geq j}^k \beta_{ijl} x_i x_j x_l + \dots \quad (4.23)$$

It is evident in equation 4.23 that the surface involves the k selected decision variables x_i . These can be opportunely substituted with the ξ_i natural variables by performing a reference frame changing. Furthermore, the expression presents some new parameters β , which represent the coefficients that properly model the surface. These are always coupled with the relative decision variable, with exception for β_0 :

β_0 : it is the known term of the surface and it is not related to any model variables. However, dealing with a k dimensional surface which fits a function having n decision variables, not all the factors will be directly described in the model. Therefore their contributes is counted in the value of other parameters, as also this one.

β_i : it represents the first order term coefficient of the relative i -th decision variable. The model up to the first order term involves simple linear relation. So the first order response surface does not present any kind of curvature.

⁶In the second and third summation symbols the subscript j is always greater or equal to the subscript i to avoid the repetition of combination of decision variables already considered.

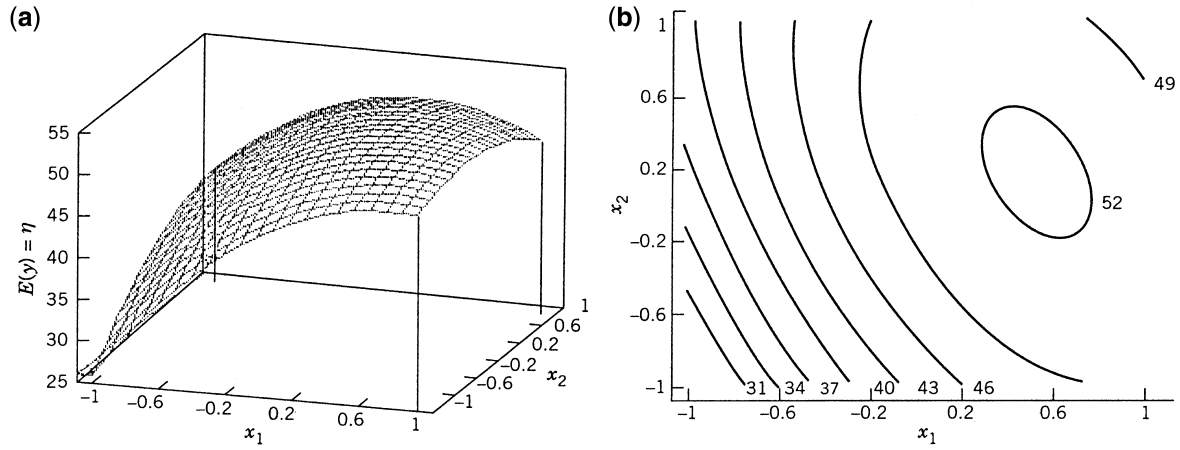


Figure 4.4: (a) Response surface for a second order model. (b) Contour plot of the same surface [43].

β_{ij} : it is the interaction model coefficient, describing the relation between the i -th and j -th variables. Looking at eq. 4.23, this parameter is attached also with the self-interaction terms: it models the behave of any combination of quadratic variables. It introduces in the response surface the curvature. A second order model realizes often a much more complex adaptive surface than the first order one.

β_{ijl} : it describes the third order interaction behaviour between variables i , j and l . It works just as the second order coefficient β_{ij}

Dealing with computational aspects and given a response surface with k variables, the first order model would have at most $k + 1$ coefficients, representing the k different variables and the known term. Introducing the interaction it requires k further coefficient to model the self-interaction contribute and $\binom{k}{2}$ coefficients due to the interaction of different variables between each other. Considering instead the third order model, it would involve $k + k\binom{k}{2} + \binom{k}{3}$ ulterior coefficient and so on, always realizing a greater augmenting of the complexity.

The typical representation of a response surface displays the so called *main effects*. In fact most of the times it includes only those variables which deeply affect the function. The most used graph for response surface is the contour plot. It has as axis the main variables and it connects by lines the points which display the same response value, as in fig. 4.4(b). When the main effects are just one or two, it is possible to plot the surface against the selected variables fig. 4.4(a), obtaining the real surface model.

The use of response surface methodology to build a simple but effective model of the problem can be summarised in few reasons:

1. The surface model is very flexible: second or higher order model can take wide variety of functional forms. So it can approximate quite well many different functions.
2. The parameter estimation β can be done with Least Square method. Hence the required calculations for most of the models does not burden the system, until the surface order and the number of variables grow too much.

3. The use of surfaces to perform local searching of an optimum is easier than the search through complicate functional forms.

4.4.1 Local reference frame and sampling

To build the surface it is first necessary the definition of local reference frame in the chosen region to model. This is a necessary step that comes before the sampling phase. In fact, depending on the chosen surface domain region, points outside it could be selected, if possible, to realize a better model.

The change of reference to natural variables $\boldsymbol{\xi} = (\xi_1, \dots, \xi_k) \in [-1, 1]^k$ it is quite immediate. Let's consider the decision variables $\boldsymbol{x} = (x_1, \dots, x_k)$, while the selected region to model is defined as $[L_i, U_i]^7 \forall i = 1, \dots, k$. There can be defined the mean value and the spread of the design region for each variable:

$$\begin{aligned}\mu_i &= \frac{U_i + L_i}{2} \\ \delta_i &= \frac{U_i - L_i}{2} \quad \forall i = 1, \dots, k\end{aligned}\tag{4.24}$$

Now it is outright the definition in natural reference [13], using the parameters defined in eq. 4.24:

$$\xi_i = \frac{x_i - \mu_i}{\delta_i} \quad \forall i = 1, \dots, k\tag{4.25}$$

Once defined the change of reference, points on the initial design frame can be selected to build up the sample. Various methods exist and often they are mixed up to interlace benefits and to limit the weaknesses. However, before investigating these methods, one needs to select the chosen variables which enter in model and those which are kept fixed. Objective function can be defined by a huge number of variables and using all of them would burden the surrogate. These fixed variables need anyway to take a value, since they give a contribution in the objective function. As stated in [3, 43], the suggested value⁸ for each of them is the midpoint of the respective domain's region. This way of dealing with the *secondary* variables make the surrogate much simpler than many other models. However it gives also little control on the taken choice.

Instead, talking about the chosen variables, their sampling is one of the most important features to realize a well-defined model looking forward to perform useful prediction. Almost all surface models start the sample set taking a full or fractional factorial (or tri-factorial) sampling, which is described in detail in section 5.2.2, the axis sampling and the origin of the surface domain region. The first sample gives important information about the boundary behaviour of objective function. The axis sample takes two opposite values for each chosen variable, while all the other are set to zero. If it is possible, the values can be outside, if it is possible, the surface domain region, or inside this last. Knowledge of the outer behave could help the surface to better fit the objective function. An example of this is reported in tab. 4.1. Finally, the centre point gives a brief knowledge of what happens inside. Some models are developed to build a surface which fits exactly the sampling points, so they take as many sample point as it is the number of parameters of the surface. However the resulting model often shows large lack of fitting for points not present in the sample set.

⁷ L_i and U_i represent the lower and upper bound of the chosen region for the i -th variable.

⁸These texts only report this way of modelling, but successively treating the problems at hand, the model works differently with the fixed variables; this is explained in detail in chapter 6.

Observation	ξ_1	ξ_2	ξ_3	\dots	ξ_k
1	$-\sqrt{2}$	0	0	\dots	0
2	$\sqrt{2}$	0	0	\dots	0
3	0	$-\sqrt{2}$	0	\dots	0
4	0	$\sqrt{2}$	0	\dots	0
\dots	\dots	\dots	\dots	\dots	\dots
$2k-1$	0	0	0	\dots	$-\sqrt{2}$
$2k$	0	0	0	\dots	$\sqrt{2}$

Table 4.1: Axis sample for k variables in natural reference frame. The chosen axis point is outside the selected region.

Further points can be selected using different techniques, as random, pseudo-random, Latin-hypercube and many other sampling, described in the following chapter 5.2. Selection of a larger sampling quite often leads to better result. Although, a too wide sample could bring over-fitting of the parameters and useless computational costs. So this building phase of the model needs always to be treated carefully.

4.4.2 Surface fitting

The final step in Response Surface Methodology consists in performing the evaluation of model parameters β , through a statistical regression. Once defined the sample set, it is possible to evaluate it and to obtain the fitting data. Recall that only the chosen variables x_1, \dots, x_k are changing, while the secondary ones x_{k+1}, \dots, x_n have a fixed value for all the sampling points. So, their contribute to the objective function is constant. Since the coefficients β are scalar quantity, the fitting procedure can be done using statistical tools. The relation between real objective and predicted value can be written as:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{bmatrix} \xi_{1,1} & \xi_{2,1} & \dots & \xi_{r,1} \\ \xi_{1,2} & \xi_{2,2} & \dots & \xi_{r,2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{1,p} & \xi_{2,p} & \dots & \xi_{r,p} \end{bmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_r \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_r \end{pmatrix} \quad (4.26)$$

In eq. 4.26 the matrix \mathbf{X} represents the value of chosen variables. Each row is one of the p sample points, while each column refers to a chosen variable in natural reference frame or a combination of different chosen variables, as table 4.2 reports. In fact, selecting a second or upper order model, also the interaction terms need to be described. To retrieve their β coefficients they need to enter in the \mathbf{X} matrix. Finally, the ε is the error, due to the deviation of the predicted value from the exact objective y .

Since the model aims to the best possible fit of the data, one wishes to minimize the error of the following function:

$$\min \left(\sum_{i=1}^p \varepsilon_i^2 \right) = \min(\varepsilon' \varepsilon) = \min[(y - \mathbf{X}\beta)'(y - \mathbf{X}\beta)] \quad (4.27)$$

Eq. 4.27 is the *Least-Square* estimation procedure. It yields the coefficients β minimizing the square error. Simply developing the algebra and the minimization problem,

Observation	ξ_1	ξ_2	\dots	ξ_k	$\xi_1\xi_1$	$\xi_1\xi_2$	\dots	$\xi_k\xi_{k-1}$	$\xi_k\xi_k$
	ξ_1	ξ_2	\dots	ξ_k	ξ_{k+1}	ξ_{k+2}	\dots	ξ_{r-1}	ξ_r
1	-1	-1	\dots	1	1	-1	\dots	-1	1
2	-0.5	0.6	\dots	-0.9	0.25	-0.3	\dots	-0.7	0.81
3	0.2	-0.4	\dots	0.6	0.04	-0.08	\dots	0.6	0.36
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
p	-1	-0.45	\dots	0.5	1	0.45	\dots	-0.4	0.25

Table 4.2: \mathbf{X} matrix for a second-order surface, with k chosen variables and all the interaction terms.

the above relation gives:

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'y \quad (4.28)$$

The dimension of the sampling set can build a surface that pass exactly through the selected point. Such sample would give the exact solutions over its point. On the other hand, most of the surfaces are low-order and they are described by few number of variables, so the sampling set built would be quite small and it would realize a huge lack of fitting in the not-sampled points.

Several methods of fitting exist and the most recent ones develop an on-line tuning of the parameters, just as happens in Artificial Neural Networks 4.3. A new point is added to the sample if it gives better prediction over the testing set. This process is much more expensive dealing with computational costs. However it is often more accurate in terms of final prediction. Other fitting strategy that can be adopted are similar to the least square but they work on different aspects, as the Bayesian estimation, the maximum likelihood estimation.

The reported algorithm 9 will be explained in detail later in the chapter 6, dealing to the implementation of the model.

4.5 Meta-heuristic hybrid optimization

A hybrid optimization algorithm takes advantage of a meta-heuristic built, e.g. evolutionary approach and surrogate models. The motivation behind its development is to obtain better performance in solving hard optimization problems. The combination of various optimization methods, such as genetic algorithms or Tabu search, with also surrogate model, as branch-and-bound, simplex method or neural network, realizes hybrid models. This often can lead to great advantages and they gain better performance then each search algorithm alone.

Nowadays it is quite recognized the improvement brought by this hybrid model. Although, years ago researchers based their development just focusing on a specific or favourite class of models, considering this superior to the others. Such behaviour led to different and specialized ways of thinking, which built up their own model, bounded and isolated in its properties. Later, the coming of *no free lunch theorem* 2.1.3 broke down this class separation. It proved that on a wide variety of problems all these different algorithms would perform similarly. What came after was a deep study of each problem, to provide the best way to treat it, and each model, to define

Algorithm 9: Pseudo-code of the Response Surface Methodology

Input: *Problem*, *chosenFactor*, *surfOrder*
Define: $k = \text{size}(\text{chosenFactor})$ dimension of the Response Surface;
Define: $k_{\text{Problem}} = \text{dimSize}(\text{Problem})$ number of variables in the Problem;
Define: $N_{\text{obj}} = \text{objNum}(\text{Problem})$ number of objective function;
Define: *Domain* = *bounds(Problem)* bounds of the problem's variables;
Define: *SurfBound* = *defineSurfBound()* selects the bounds of Response Surface;
FactSample = *factorialDesing(k)* build factorial design sample;
FactSample₂ = *TriFactorialDesing(k)* build tri-factorial design sample;
AxisSample = *axisDesign(k)* build axis design sample;
RandomSample = *addRandPoint(k, N_{point})* create a k -dimensional pseudo-random sample of N_{point} points;
sample_{dim} \leftarrow Insert the desired dimension of the sample set, knowing the number of parameter to model;
DesignSample =
mergeDel(k, FactSample, FactSample₂, AxisSample, RandomSample, sample_{dim})
builds the complete sample array merging all the above and selecting randomly *sample_{dim}* of them;
GlobalSample =
convert2bound(DesignSample, SurfBound, Domain, chosenFactor) change the point definitions reference frame to the global one;
InputsSample = *buildInput(k, k_{problem}, Domain, chosenFactor)* constructs the sample to submit to objective function;
 $\mathbf{y} = \text{Problem}(\text{Input}_{\text{sample}})$ return the objective functions output;
 $\mathbf{X} = \text{buildSurfInp}(k, \text{GlobalSample}, \text{SurfOrder})$ define the \mathbf{X} matrix of the inputs to retrieve the surface coefficients;
 $\boldsymbol{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$;
Output: $\boldsymbol{\beta}$

the main features that are needed. The knowledge on the specific problem leads to the development of adaptive and customize optimization algorithms.

Below will be reported a brief classification of the hybrid meta-heuristic algorithms [20], based on four criteria, summarized in Fig. 4.6: the kinds of algorithms that are hybridized together, the level of hybridization, the order of execution of the searching process and the part playing the control strategy.

Hybridized Algorithms: the combination of different algorithms could involve various kind of strategies. First, one might combine part or entire meta-heuristic optimization process, siding an Ant Colony algorithm, to realize a first search, and then it might introduce the Particle Swarm to exploit the found solutions. Secondly, a simulation algorithm could take advantage of a meta-heuristic routine to converge faster to the desired results (e.g. this happens in fluid dynamics or finite elements simulations, which both require huge quantity of time and computational resources⁹). As third class, one could develop combination of

⁹These simulations can be coupled with evolutionary algorithms to reduce the problem complexity during the resolution [17].

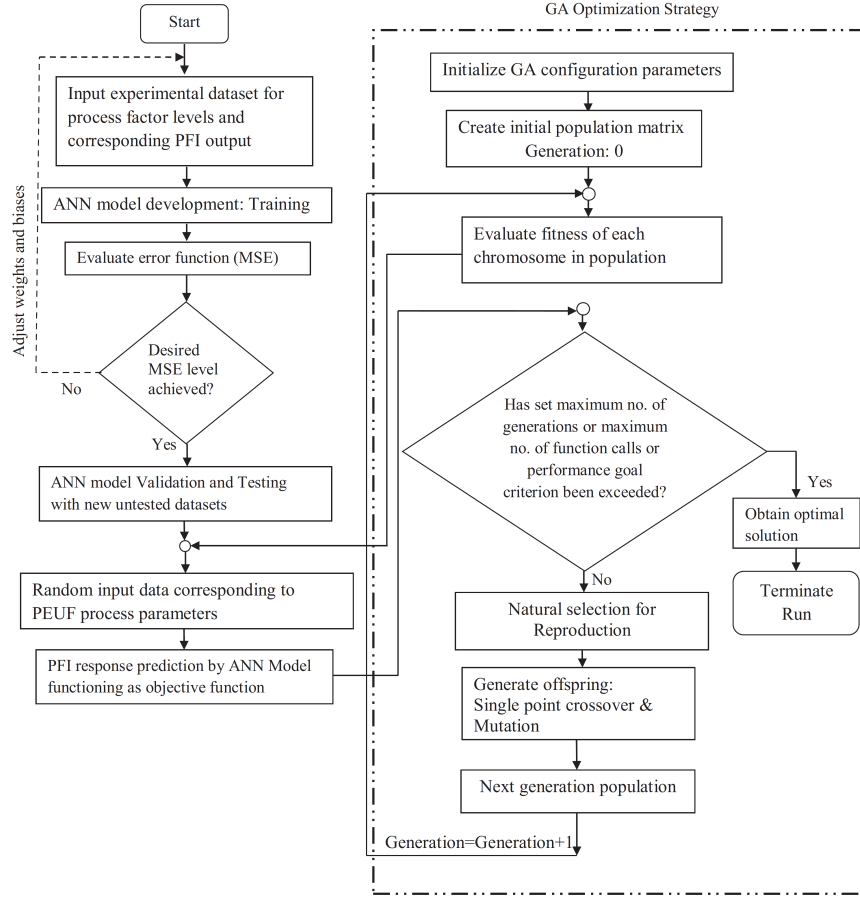


Figure 4.5: Flow diagram of a Meta-heuristic hybrid strategy involving Artificial Neural Network and Genetic Algorithm [9]

meta-heuristics with surrogate models as in fig. 4.5. These could be general techniques coming from other research fields. They could build accurate but simplified model to submit to the optimization task (it is the case of this work, as will be explained later on). As fourth class, some *human guided search* processes exist, which are combined with meta-heuristics. This happens when the solution quality is difficult to evaluate mathematically. Here it sets in the human contribute and intuition, building an interactive system which is often highly effective.

Level of hybridization: this classification differentiates the *level* at which the algorithms collaborate and are coupled. *High-level* hybridization makes use of two or more algorithms just by siding them. So each one develops all its iterations just as it would be working alone. The collaboration is not much highlighted and the involved parts often just share result to proceed in the analysis. On the other hand, *low-level* hybridization combines deeply the structure of the algorithms and the routine of each heuristic gets completely modified. The components of different algorithms are connected and exchanged around a share structure, which let them work all together.

Order of execution: the order with which the algorithms are executed can be *batch*, *intertwined* and *parallel*. In the first case different algorithms are performed in

sequential order and the results of the former are used as input to its follower. Second and third cases develop several sharing and exchanging of information from all the involved parts. *Intertwined* structure often follows a sort of sequentiality. It does not run the algorithms separately but chooses at each phase the most proper one to run. Instead, *Parallel* meta-heuristics realize independent runs of the algorithms. However, these exchange information and results, so each part in the game can take advantage of the others. Each one investigates promising region either performs exploration, due to its design properties.

Control strategy: the way of control a meta-heuristic can be either *integrative* or *collaborative*. The first case includes all those codes in which an algorithm is submitted to another one. The subordinate algorithm works as source of information, decoder, or local searching help, while the outer one realizes the main computational aspects, e.g. the global search. In contrast, *collaborative* approach builds separately the algorithms. These work on their own but they provide information exchanging, as happens in the *Island model* for parallel evolutionary algorithms [6].

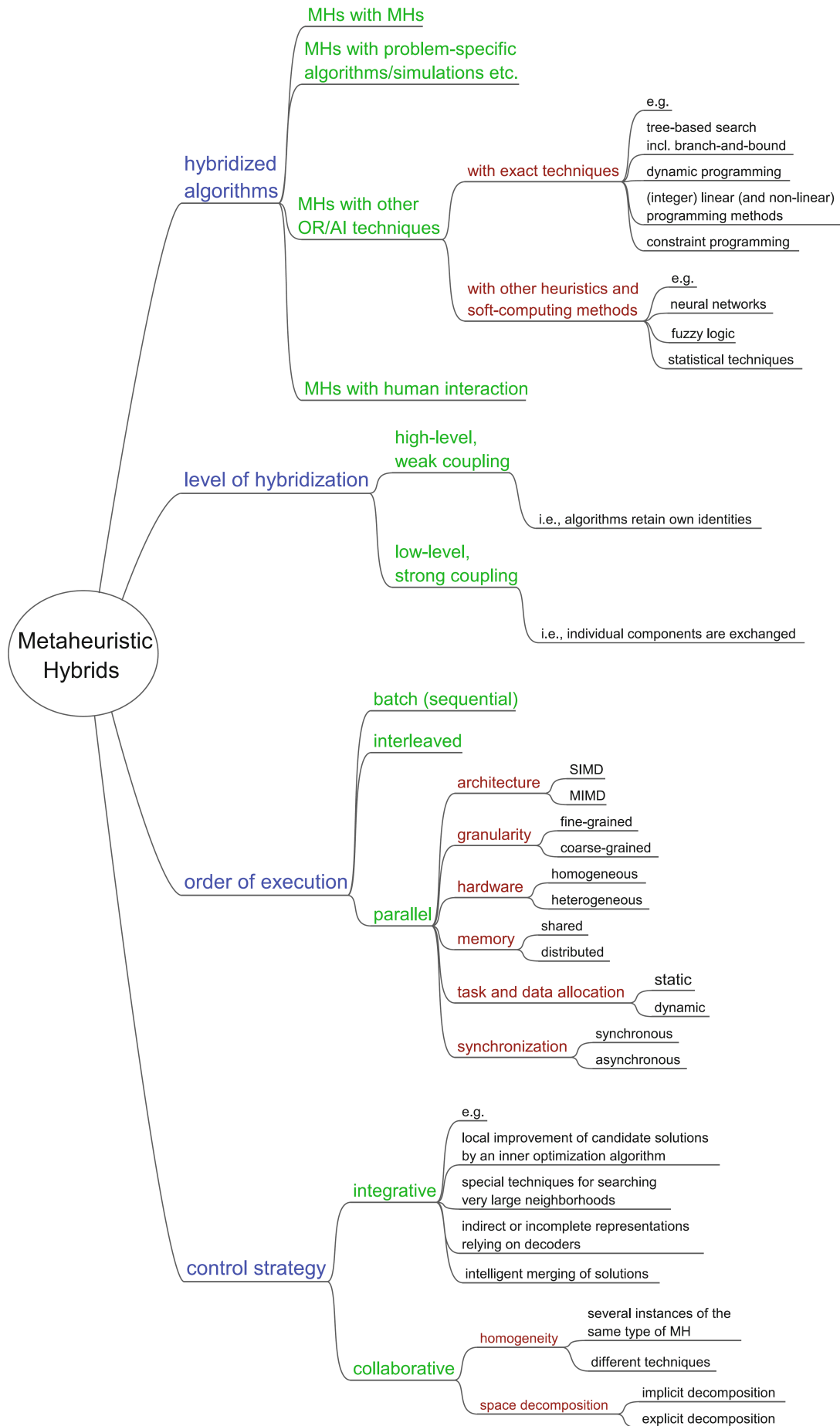


Figure 4.6: Classification of meta-heuristic hybrid [20].

Chapter 5

Sensitivity Analysis

As stated in [51] and [54], a possible definition of sensitivity analysis is:

The study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input.

A first important element in this citation is the idea of *model*, which has been already used in this text. However here, dealing with the sensitivity analysis and a more statistical framework, it gets a little different accent. Often one refers to a model as an analytical and mathematical description of an observed event, which has its own causality. Usually a model is made up of differential equations that rule the entire system. They are typically dictated by the physics of the process in study, but nothing keeps it from being based on completely different expressions, if these can properly describe the events. Then it is completely accepted that many different models can describe accurately the same set of data and evidence. Moreover, even if relating a process with its physical laws could already seem a simplifying idea, often it brings cumbersome mathematics and burdening numerical aspects that can be avoid building a physical-detached model.

The other key term is *uncertainty*: it is the substance of the scientific method. In fact anything real can be measured or evaluated exactly, without errors. Any built model will never be proven really true, since it cannot fit the relative process without errors. These are just inside the own definition of the model, as the measurement themselves bring loss of information for example. A model can only be accepted within certain threshold, arbitrary chosen. It means that its result or prediction will always differ from the observation at least by an unknown quantity. Evaluate the uncertainty, being able to understand and to predict the process at hand it is a fundamental goal, often even more than developing the model itself.

5.1 Purpose of the sensitivity analysis

Sensitivity analysis is an indispensable tool to evaluate the quality of inference based on mathematical models [50]. It allows a third part to concretely understand and judge the model at hand, to obtain the substance of its behaviour. Dealing with the construction of a model and looking for positive results, the involved parties deal with information and data in a selective or even manipulating way. Therefore, the model realized would be erroneous and biased. Performing a sensitivity analysis, this approach

can be detected and avoided, because often it is not intention of the research to prove untruth results.

Another application of the sensitivity is related to the risk assessment analysis. This tool studies the variability of objective function and the contribute of each variable to the problem. So it allows to analyse how the model is affected by fixing a parameter. It evaluates the reduction of variability attached with each factor and this can be very useful in many issues.

5.2 Sampling technique

A key phase to perform a sensitivity or uncertainty analysis passes through the generation of a sequence of input to study the objective function at hand. The sampling, even if could seems a trivial matter, can lead to large computational savings or to better performance of the analysis. The correct sampling definition has always been one of the main features of statistical tests. Choosing properly specific individuals inside the population affects later the whole estimated characteristics of the population itself.

All problems at hand deal with multi-variables functions, where each parameter describes a certain feature of the test. For the sake of simplicity let's consider as domain space the interval $[0, 1]^k$, such that each variable is bounded between 0 and 1. For a general problem this is not true, but it is always possible to retrieve this frame simply performing an easy change of variable¹. Let's also assume that all the variables are independent and their distribution unknown. Typically the former assumption can be satisfied properly modelling the experimental test and it is an important property in most part of sensitivity analysis. Despite this, it is often possible to relax problem features and suppose independence, this introduces in the model a first error. About latter assumption, usually it is preferable to work with uniform distribution. This is done to avoid assumptions that could reveal false² and to treat easily all the variables. However, passing to a given specific distribution often requires quite immediate calculation, related to probability distribution function.

5.2.1 Random sampling

Random sampling is the most common and known method to build a sequence of simulation points. Despite the *random* word, the numbers generated with the relative computational routine are not truly random. In fact these numbers are all determined by a starting initial value and by the hardware features of the machine realizing the sequence. The initial value can be taken from a so-called seed of pseudo-random number generator (which contains real random numbers), or it may be based on hardware features. This method is commonly used because of its speed and reproducibility and it is implemented in any programming language.

Random sampling exhibits several aspects that could prevent its usage in specific applications, as in sensitivity analysis. It has been proven [51] that the sequences generated by a random sampling routine, can highly influence the results of researches

¹In any case each factor domain needs at least to be finite to be able to perform a computational analysis.

²This is anyway an assumption, but it builds a well-defined and singular roadmap to tackle all the possible test problems.

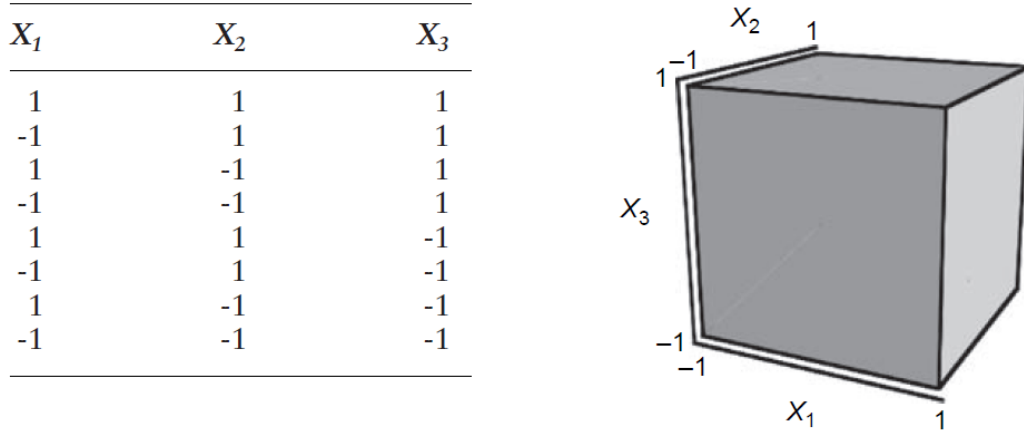


Figure 5.1: Two level factorial design for 3 parameter problem [51].

and studies as well as practical applications. The main erroneous feature individuated are:

- Repetition periods shorter than expected for some seed sates;
- Lack of uniformity for large sequence of generated numbers;
- Correlation of successive values and poor dimensional distribution;

These features lead to random sequences showing clusters and gaps, because the points are not evenly distributed across the domain space [51]. Clusters show several points quite close together, gaps are region without samples. Dealing with a function analysis, when a cluster occurs the function properties coming out from there are emphasized. On the other hand in presence of a gap the features within the gap are not sampled and so the analysis does not account these. Measuring the net effect that this problem generates on the sample, to reduce an estimate uncertainty by a factor of 10, the random sampling must increase the starting sample N by a factor of $10^2 = 100$ [31, 51]. Other models achieve the same results with much less effort: the uncertainty of Sobol' Quasi-random sequences (5.2.5) decreases with the sample size N .

5.2.2 Factorial sampling

This kind of sampling contains all the possible combination of low and high values for each variable. This model does not require to reshape to $[0, 1]^k$, since it uses as characteristic values -1 and 1 . These represent a low and a high value of each variable respectively. Usually and for practical reasons these two are exactly the lower and upper bound of the domain variables. Considering for example a 3-dimensional problem, the eight points individuated are the corners of a cube as in fig. 5.1.

The sampling built is known as full factorial, since all the possible combinations are considered. Dealing with a generic k -dimensional problem, this would require 2^k evaluation points. This sample would represent all the corners of a k -dimensional hypercube, hence it grows fast. The growth can be reduced by developing a fractional factorial sampling. First of all, it requires the choice of the variables which will build the full factorial design. Let assume that in a k -dimensional problem n parameters are chosen, which build a full factorial design. The others $k - n$ are sampled by selecting

two or more of the previous variables and then multiplying their factorial design values in the relative sample. Table 5.1 presents an example of fractional factorial design problem with 7 variables and a 3-dimensional full factorial development.

X_1	X_2	$X_3 =$ X_1X_2	X_4	$X_5 =$ X_1X_4	$X_6 =$ X_2X_4	$X_7 =$ $X_1X_2X_4$
1	1	1	1	1	1	1
-1	1	-1	1	-1	1	-1
1	-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1	1
1	1	1	-1	-1	-1	-1
-1	1	-1	-1	1	-1	1
1	-1	-1	-1	-1	1	1
-1	-1	1	-1	1	1	-1

Table 5.1: Fractional factorial design of a 7-dimensional problem starting from a 3-dimensional full factorial [51].

All the above examples deal with 2-levels factorial design, but also this parameter can be modified, developing the s -levels design. Instead of taking 2 sample points for each variable, s points are picked. Often these are selected realizing $s - 1$ step of fixed size starting from the lower bound of each domain variable (the step size in such frame would be the domain length divided by $s - 1$). Considering for example the standard factorial set $[-1, 1]$, a 3-levels design would select $\{-1, 0, 1\}$, a 4-level design $\{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$ and so on.

5.2.3 Latin hypercube sampling

Latin Hypercube sampling is a method which enhances the s -levels factorial design. Again it divides the search space in different levels, but it takes in each subset more than one point if desired. Here the sample space gets separated by s , generating N sub-sample spaces³. The key objective in Latin Hypercube is to ensure that each variable is individually stratified over the s levels. It means that each level must contain exactly the same number of points. All this building is typically developed by integer number and discrete spaces, but it can be translated to continuous variables as later will be described. Supposing now to deal with a k -dimensional model, divided in s levels and with n picks, each level presents n sampling, divided between the k factors. The final sequence will have $n \times k$ points. The possibility of taking n picks in each level could be also performed by repeating n time the Latin Hypercube sampling with only 1 sample per level. This would build n independent sequence that can generate separate estimates of the objective function.

Now, dealing with the transition from discrete sampling to continuous variables, different choices can be made. First, recall that the domain of each variable is divided into s levels. However it does not imply that the interval dimension of different factors is equal, since one be larger or smaller in length. Anyway, let's consider for simplicity $[0, 1]^k$ the domain space, therefore a function described by continuous variables.

³often the model requires s and N to be equal

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
0	4	5	0	5	4	0	2	0
4	3	1	1	4	3	5	1	4
2	2	5	4	1	0	2	0	2
5	0	0	3	0	5	3	3	5
3	1	3	2	2	2	1	4	3
1	5	2	4	3	1	4	5	1
2	3	3	3	3	4	1	5	5
5	1	0	2	1	3	2	0	4
1	5	4	1	4	2	5	4	2
0	2	1	0	5	1	3	3	1
3	0	2	5	2	0	0	1	0
4	4	5	4	0	5	4	2	3

Table 5.2: Doubled Latin Hypercube sampling with 9 factors ($X_1 - X_9$), 6 levels (0–5) and 12 simulation [51].

Two simple transition to continuous domain can be introduce. The first identifies through discrete sampling the picked interval and the chosen value is a fixed location inside the interval (e.g. its middle point). A second way could perform a pick with $s + 1$ levels and then it selects as value the relative grid line⁴. A further model could consider a uniform distribution: once chosen the discrete interval, a random value is generated with uniform distribution and it locates the relative point.

Latin Hypercube has attractive properties, because it leads to good estimate of mean features of a population without large samples. It rapidly converges to the true value as the number of simulation N increases. Moreover, when the number of samplings N is much larger than the factor k , Latin Hypercube can be very effective for the sensitivity analysis. On the other hand, when these values are more or less the same, the model cannot properly give information on the factor. It happens because there are not enough points to perform the estimates and the correlation between data is excessive and persistent.

An improvement to general Latin Hypercube deals with orthogonal arrays, which has particular properties. These are used to add further design requirement to the sampling sequence. The Orthogonal Latin Hypercube has the additional property that each couple of variables contains all the combination of the s -levels parameters. So each sample space is divided in equally probable subspaces and each of them is filled with equal probability. So it gives very good representation of the input variability.

5.2.4 Multivariate stratified sampling

Multivariate stratified sampling couples the fractional factorial and Latin Hypercube design to obtain advantage from both the models. The first stage consists in applying the fractional factorization which divides each dimension in s sub-levels (often a simple 2-levels factorial is performed as in Fig. 5.2). Then, fixed the total number of points, Latin Hypercube samples the sequence in each part of the grid. In the meanwhile it

⁴The division of the domain variable in s levels locates two values at the boundary and $s - 1$ values inside the variable domain.

takes care that in each column and in each row the constraints are satisfied and the total number of samples per row and column is met. This method could seem quite similar to the Latin Hypercube construction with continuous variables and uniform distribution, but, anyway these two are quite different.

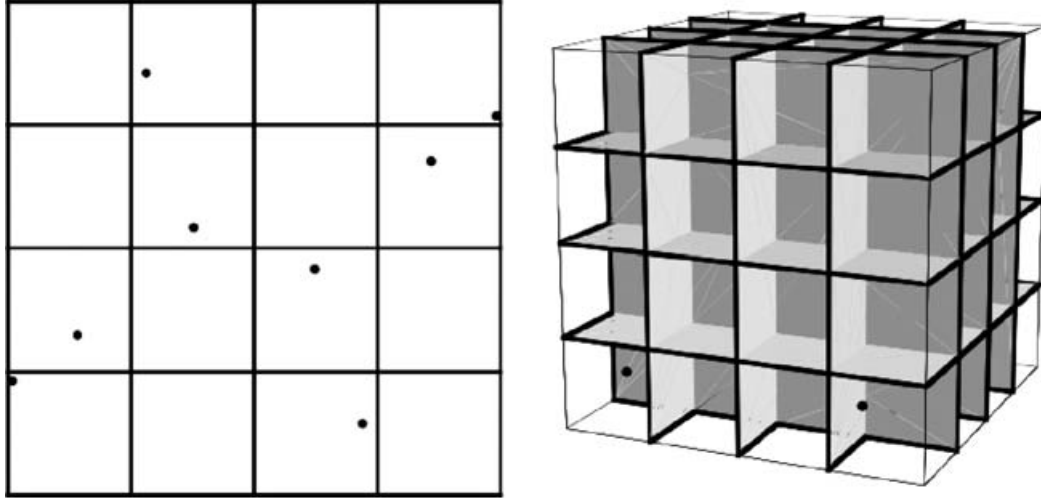


Figure 5.2: Combined fractional factorial and Latin Hypercube design in 2 and 3 dimensions, with a 2-levels FF and a further 2-levels LH [51].

5.2.5 Quasi-random sampling

Quasi-random sampling develops a much more complex computational structure to build a random sequence of points. However, it is far more accurate than the similar pseudo-random sampling. Its first property, that only quasi-random numbers have, is the low *discrepancy*: discrepancy is the measure of lumpiness of a sequence of point in a multidimensional space. Considering an interval $[a, b]$ and a sequence of N values $\{s_1, s_2, \dots, s_N\}$, the discrepancy is defined as:

$$D_N = \sup_{a \leq c \leq d \leq b} \left| \frac{|\{s_1, \dots, s_N\} \cap [c, d]|}{N} - \frac{d - c}{b - a} \right| \quad (5.1)$$

Therefore, the lower D_N is as N tends to infinity, the more equidistributed is the sequence. However, the concept of equidistribution itself is weak. As said for random sequence, even if the distribution is quite uniform in a given interval, large gaps can be present, compared to the rest of the sequence generated, but the results of equidistribution would be fair. Back to the discrepancy, the further concept realized is that each part of the sampling sequence has quite the same gap between all points, using it as a criterion to contract the sequence. In fact it is from these concept that quasi-random sampling is known also as low-discrepancy sequence. The attribute *quasi* means that the sequences such built are neither random nor pseudo-random, but they follow certain rules that lead to very good samplings. They are not completely unpredictable, in fact to simultaneously maintain an even spread, avoid clustering and gaps, the algorithm which generates the sequence of points need to bias the *prediction* and they fill with properly samples the space.

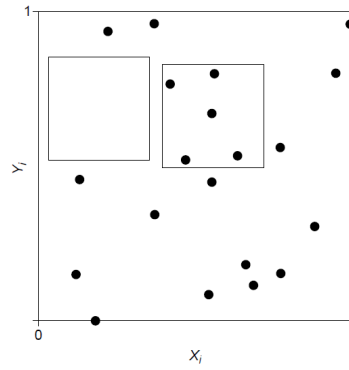


Figure 5.3: Discrepancy is the maximum absolute difference between the fraction of the area a square occupies and the fraction of the points it contains [51].

Considering now k -dimensional cases, these sequences still work well, but they need a very large sequence length N to show their properties and to reach the theoretical optimal rate of discrepancy. As already hinted in section 5.2.1, this sequence provides best convergence characteristics. To estimate the mean features of a multidimensional function $f(X_1, \dots, X_k)$ which is evaluated on points of coordinates $\{X_{i1}, \dots, X_{i,k}\}_{i=1, \dots, N}$, it is required a much little sample dimension N with low-discrepancy sequences than with random sampling to obtain similar accurate results.

In any case, the properties of quasi-random numbers are effective and reach low-discrepancy levels only when a large sample size comes into play. Until the analysis works with *small* sequences, the properties may not be evident. Instead, when the sample overtakes a threshold dimension, its discrepancy decreases rapidly. However, the required size of the sample is strictly connected with the function dimension: as the problem is described by more decision variables, as the required sample gets bigger and bigger to achieve the low discrepancy.

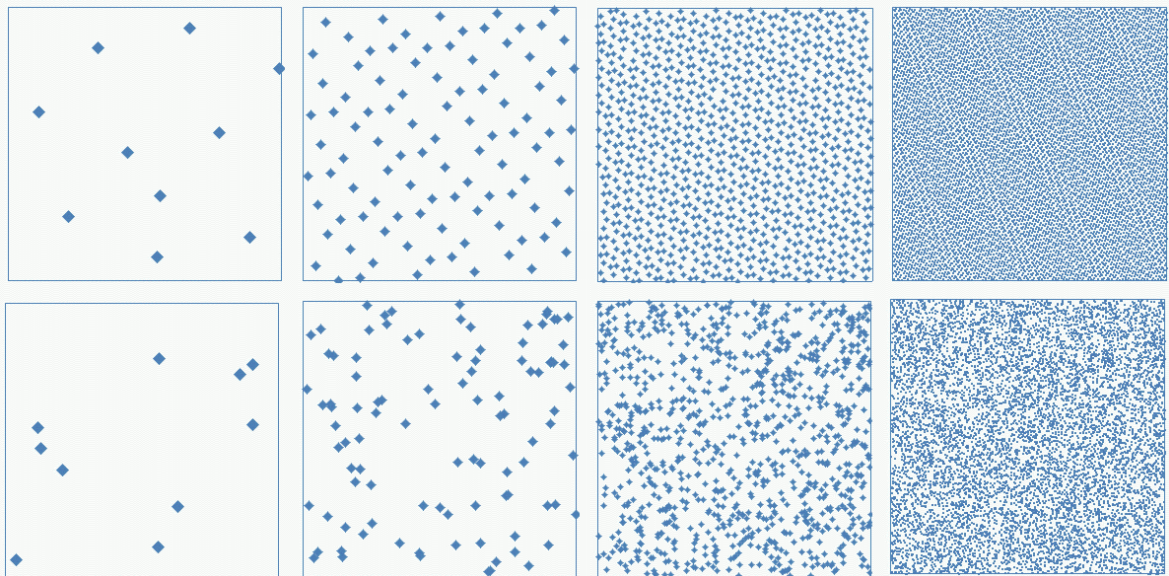


Figure 5.4: Coverage of the unit square for different sample: starting from the left, the samples has 10, 100, 1000 and 10000 points; the top boxes are filled with quasi-random number, while those below with random numbers [64].

A simple test to evaluate the performance and uniformity of a quasi-random sequence with N sample points is the following: it calculates the average value of each variable sequence independently and the cross-product value between two chosen sequences. Because of the properties of quasi-random sequences, these should give:

$$\int_0^1 X_j dX_j = 0.5 \cong \frac{1}{N} \sum_{i=1}^N X_{ij} \quad (5.2)$$

$$\int_0^1 (X_j X_k) dX_j dX_k = 0.25 \cong \frac{1}{N} \sum_{i=1}^N X_{ij} X_{ik} \quad (5.3)$$

Here the subscript j and k indicate the chosen variables. If the sequences cannot approximately take this value, probably they would not perform well in sensitivity analysis. Let's treat now briefly the most spread quasi-random sequences, which are the Halton and the Sobol' sequences.

Halton sequence

The Halton sequence is a well-known low-discrepancy sequence quite simple to generate. The principle under this kind of sample exploits the change of bases of floating numbers between a general base and the classical base-10. The first step consists in generating k different sequences of N consecutive numbers starting from an arbitrary one, with N the size of the sample. The k sequences sample the values for the relative problem variables. So a specific base, arbitrary chosen between prime numbers, is uniquely associated to a single variable. The following phase translates the integer values into decimal numbers, inverting their writing⁵ and placing before them the floating-point separator (e.g. the *base-10* value 11 in *base2* is written as 1011 and after the translation it becomes 0.1101; this is the radical inverse transformation and for further example see table 5.3).

The main implementations develop Halton sequence up to 100 variables, so they require each prime number up to the 100th one, which is 541. Furthermore, as table reports, observing the sequences resulting from base 2, 3 and 5, it is possible to notice that all of them approximatively realize a first coverage of the entire domain with only a sample of 12 numbers. On the other hand the sequence generated from base 541 is still far from covering the domain (it covers little more than 2% of the entire domain and the final value is in fact equal to $\frac{12}{541}$). This fact highlights the necessity of realize sequences of proper size, related not only to the dimension of the problem at hand, but also to the arbitrarily chosen base.

Sobol' sequence

Sobol' sequence was first introduced by the Russian mathematician Ilya M. Sobol, which tried to develop a sequence that converges faster than all the other methods. However, later theorems proved that all the quasi-random sequence has the same convergence rate and they differ only for few aspects. These few particular features are the numbers of required inputs, the complexity of introducing further dimension and the implementation difficulties.

⁵Writing the new number copying the old one from the right to the left.

Index base					Parameter value			
10	2	3	5	... 541	X_1	X_2	X_3	... X_{100}
1	1	1	1	1	$0.1_2 = 0.5$	$0.1_3 = 0.333$	$0.1_5 = 0.2$	$0.1_{541} = 0.002$
2	10	2	2	2	$0.01_2 = 0.25$	$0.2_3 = 0.667$	$0.2_5 = 0.4$	$0.2_{541} = 0.004$
3	11	10	3	3	$0.11_2 = 0.75$	$0.01_3 = 0.111$	$0.3_5 = 0.6$	$0.3_{541} = 0.006$
4	100	11	4	4	$0.001_2 = 0.125$	$0.11_3 = 0.444$	$0.4_5 = 0.8$	$0.4_{541} = 0.007$
5	101	12	10	5	$0.101_2 = 0.625$	$0.21_3 = 0.778$	$0.01_5 = 0.04$	$0.5_{541} = 0.009$
6	110	20	11	6	$0.011_2 = 0.375$	$0.02_3 = 0.222$	$0.11_5 = 0.24$	$0.6_{541} = 0.011$
7	111	21	12	7	$0.111_2 = 0.875$	$0.12_3 = 0.556$	$0.21_5 = 0.44$	$0.7_{541} = 0.013$
8	1000	22	13	8	$0.0001_2 = 0.062$	$0.22_3 = 0.889$	$0.31_5 = 0.64$	$0.8_{541} = 0.015$
9	1001	100	14	9	$0.1001_2 = 0.562$	$0.001_3 = 0.037$	$0.41_5 = 0.84$	$0.9_{541} = 0.017$
10	1010	101	20	A	$0.0101_2 = 0.312$	$0.101_3 = 0.370$	$0.02_5 = 0.08$	$0.A_{541} = 0.018$
11	1011	102	21	B	$0.1101_2 = 0.812$	$0.201_3 = 0.704$	$0.12_5 = 0.28$	$0.B_{541} = 0.020$
12	1100	110	22	C	$0.0011_2 = 0.188$	$0.011_3 = 0.148$	$0.22_5 = 0.48$	$0.C_{541} = 0.022$

Table 5.3: Generating coordinates of points in a Halton sequence using the radical inverse transform

The algorithm that generates Sobol' sequence makes use of a primitive polynomial of fixed degree s in the integer field \mathbb{Z}_2 :

$$x^{s_j} + a_{1,j}x^{s_j-1} + a_{2,j}x^{s_j-2} + \dots + a_{s_j-1,j}x + 1 \quad (5.4)$$

In eq. 5.4 the j parameter indicates the points component in the sequence, while coefficients $a_{1,j}, a_{2,j}, \dots$ are either 0 or 1. Then it is defined a sequence of positive integers $\{m_{1,j}, m_{2,j}, \dots\}$ by recurrence relation:

$$m_{k,j} \doteq 2a_{1,j}m_{k-1,j} \oplus 2^2a_{2,j}m_{k-2,j} \oplus \dots \oplus 2^{s_j-1}a_{s_j-1,j}m_{k-s_j+1,j} \oplus 2^{s_j}m_{k-s_j,j} \oplus m_{k-s_j,j} \quad (5.5)$$

The \oplus operator is the *exclusive or* operator, which works in bit strings. Each $m_{k,j}$ can be arbitrary chosen within the constraints that $1 \leq k \leq s_j$, it is odd and it is less than 2^k . Finally the *direction numbers* $\{v_{1,j}, v_{2,j}, \dots\}$ are required and are defined as:

$$v_{k,j} \doteq \frac{m_{k,j}}{2^k} \quad (5.6)$$

Once retrieved all these parts, it is possible to define x_{ij} , the j -th component of the i -th point of the Sobol' sequence:

$$x_{ij} \doteq i_1v_{1,j} \oplus i_2v_{2,j} \oplus \dots \quad (5.7)$$

Here in eq. 5.7 the index i_k is the k -th digit from the right when i is written in binary coding: $i = (\dots i_3i_2i_1)_2$ [31].

The above steps represent the original implementation of Sobol, but later more efficient codes were released, based on new operator definition. Nowadays the generation of Sobol' sequences relies often on precomputed *direction numbers* arrays and *primitive polynomials*,. Both these are obtained from various search criteria for the sequences and they yield the parameter for several problem dimensions. In fact each direction numbers array refers to its relative dimensional problem.

A final aspect to briefly discuss is the skip of initial entries of the sequence to obtain a better sampling. This recommendation was given by Sobol himself but later studies do not confirm neither deny this advice. Anyway seldom the first points are skipped because the algorithm would require almost the doubling of the sampling dimension⁶.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
0.2500	0.7500	0.2500	0.7500	0.2500	0.7500	0.2500	0.7500	0.7500	0.2500
0.7500	0.2500	0.7500	0.2500	0.7500	0.2500	0.7500	0.2500	0.2500	0.7500
0.1250	0.6250	0.8750	0.8750	0.6250	0.1250	0.3750	0.3750	0.8750	0.6250
0.6250	0.1250	0.3750	0.3750	0.1250	0.6250	0.8750	0.8750	0.3750	0.1250
0.3750	0.3750	0.6250	0.1250	0.8750	0.8750	0.1250	0.6250	0.1250	0.8750
0.8750	0.8750	0.1250	0.6250	0.3750	0.3750	0.6250	0.1250	0.6250	0.3750
0.0625	0.9375	0.6875	0.3125	0.1875	0.0625	0.4375	0.5625	0.8125	0.6875

Table 5.4: First eight points of 10-dimensional Sobol quasi-random sequence

5.3 Sensitivity analysis methods

5.3.1 One at a time

One-at-a-time (OAT) is the most simple and basic sensitivity analysis method. As the name suggests, it changes one factor at a time, while all the other variables are kept fixed. It evaluates the result produced measuring the objective function variation. This first method is generally considered as a local one, because the changing of the factor starts always from the same chosen point. Hence this model can give sensitivity results only in the neighbourhood of the starting point. To obtain a wider analysis of the function many different points need to be picked, but again the results cannot be extended to the entire domain until the density of the observation set reaches great values. For the relative computational cost, this sensitivity technique is seldom used to analyse functions globally, but is rather common to localized and brief studies.

The first step requires the factor changing and the evaluation of the function at hand at the entire point set. Then the differences with the original function values are performed and the sensitivity of the considered variable gets evaluated by partial derivatives or linear regression. Typically the factor step value is fixed for all the variables and all the points, but nothing forbids to change the step size for some factors or around some points.

Another weak point of this method is that it cannot evaluate the simultaneous variation of different variables, so it is not able to measure interaction between factor. On the other hand it is frequently used because of practical reasons, since its results are easy to understand. Moreover, if during the analysis the model crash, it is extremely simple to retrieve the input factor causing the failure due to its sequential behaviour. Finally, the implementing phase is immediate and it is quite cheap in terms of computational costs, until the point set does not become too large.

⁶Sobol advised to skip the largest power of 2, smaller than the number of points to be used. For a sample set of 500 points, it should be generated 256 more points.

5.3.2 Elementary Effects

Elementary Effects method (EE) is one of the most used screening technique in sensitivity analysis. It develops the idea of Morris method, which in turn is based on the OAT method. Elementary Effects determine which input factors have larger effect on the objective functions, measuring their type of contribution (e.g. linear, non-linear, interactive or negligible) by few simple indicators. Originally only two measures were computed. The most important is μ , which describes the overall influence of the selected variable to the objective functions. Then it comes σ , estimating how the variable contributes inside the entire domain and how it interacts with other variables. Later reviews of the model introduced a new parameter μ^* . It estimates the mean contribute of the absolute value of the chosen decision variable. This last parameter allows to highlight factors whose contributes take opposite sign in the domain, realizing a null μ value and a non-zero σ one. However, this information could be already understood from the previous two parameters, but the evaluation of μ^* is done with no extra computational cost. In alternative to the absolute value it could be considered the squared effects, which has been proven to be a less robust measure.

The Elementary Effect procedure develops a randomized OAT experiment, requiring the following entries: a random starting point in the domain space $\mathbf{x} = (x_1, \dots, x_k)$, a number of step p , the grid division of the k -dimensional region of experiment in p -levels and a step size. Moreover all the input factors of the objective function are considered uniformly distributed, so any value inside the domain space is equally probable. In [5], the EE of the i -th input factor associated to the point \mathbf{x} is defined as:

$$d_i(\mathbf{x}) = \left(\frac{y(x_1, \dots, x_{i-1}, x_i + \Delta, x_{i+1}, \dots, x_k) - y(\mathbf{x})}{\Delta} \right) \quad (5.8)$$

In eq. 5.8 the argument of the first function evaluation $y(x_1, \dots)$ is the vector $\mathbf{x} + \mathbf{e}_i \Delta \in \Omega \quad \forall i = 1, \dots, k$, with Ω domain space and \mathbf{e}_i the vector, having all components null but the i -th one. Δ is the step size and it takes value in $\{\frac{1}{p-1}, \dots, 1 - \frac{1}{p-1}\}$. The elementary effect of the i -th input factor is obtained by randomly sampling different point \mathbf{x} in the domain and building its final distribution F_i , such as $d_i \mathbf{x} \sim F_i$. Suggested values for the parameter p and Δ are respectively a general even value for p and $\Delta = \frac{p}{2(p-1)}$. These allow the method to realize a sampling which guarantee more or less an equal-probability for each F_i .

Finally to properly obtain the distribution of each elementary effect, Morris in [42] suggested to build r trajectories each with $k + 1$ points. These provide k elementary effects, for a total cost of the experiment of $r(k + 1)$ evaluation runs. Saltelli later proved that the performance of such sensitivity analysis was quite satisfying, however they are still liable to statistical error of Type II⁷ while quite robust against Type

⁷Typically three types of error are reported (different authors introduce others of them):

Type I *Rejecting the null hypothesis when it is true.* It occurs when a non-influential factor is wrongly identified as so.

Type II *Accepting the null hypothesis when it is false.* This occurs when the analysis fails in highlight a factor which has considerable influence.

Type III *correctly rejecting the null hypothesis for the wrong reason.* Type III errors are difficult to identifies and can be translated as a researcher providing the right answer to the wrong question.

I error. It happens because with a non-monotonic objective function, negative and positive contribution can cancel out each other, thus producing a low μ value. Anyway one could still identify this factor by the associated σ value. In fact, in this case it would be rather large, if the factor is non-negligible but it cancels out itself. The introduction of μ^* leads to a simpler treating of this question.

A further improvement of the method was presented in [5]. The article introduces a sort of modification on the trajectory, since the original one often leads to non-optimal coverage of the input set. The idea consists in generating much more trajectories than those necessary (10–50 times the parameter r) and then chose those with the highest *spread* over the search space. Classification of the best trajectories is performed using a distance metric d_{ml} between two of them:

$$d_{ml} = \begin{cases} \sum_{i=1}^{k+1} \sum_{j=1}^{k+1} \sqrt{\sum_{z=1}^k [\mathbf{x}_i^m(z) - \mathbf{x}_j^l(z)]^2} & \text{for } m \neq l \\ 0 & \text{otherwise} \end{cases} \quad (5.9)$$

Here $\mathbf{x}_i^m(z)$ identifies the z -coordinate of the i -th point of the m -th trajectory. Then d_{ml} evaluates the geometric distance between each couple of point of the trajectories. Obviously one needs to define a 0 in this metric, therefore a reference trajectory which allows the distances evaluation. Finally the choice of the r trajectories is done evaluating the distance covered by each possible combination of an r sample of trajectories over M possibilities. The total covered distance is defined as:

$$D_{comb} = \sqrt{d_1^2 + d_2^2 + \dots + d_r^2}$$

where each d_i is a different path d_{ml} . Literature suggests to prefer always this kind of sampling strategy, since it provides much better performance than the Morris sampling. This happens mainly due to its improved ability in scanning the design space without further model evaluation.

The Elementary Effect has also the possibility to work with groups of variables, thanks to the parameter μ^* . This gives to the model an advantage over many others, since it can deal quite easily with problems with a great number of factors. EE can evaluate the contribution of a group of factors, despite in such frame it loses information on the single variables. The idea consists in realizing a step which involves all the factors in the same group, considering it as a single variable. However in this formulation the step size keep on being fixed, but its direction varies for each factor, e.g. the step could increase or decrease by Δ each factor randomly. To properly measure the contribute and the effect of this different jump, only the parameter μ^* can be used. In fact it has been observed that μ can hardly obtain correct results. The elementary effect for groups is evaluated by:

$$|d_{\mathbf{u}_i}(\mathbf{x})| = \left| \frac{y(\tilde{\mathbf{x}}) - y(\mathbf{x})}{\Delta} \right| \quad (5.10)$$

In eq. 5.10 subscript \mathbf{u} identifies the vector of variables $\mathbf{u}_i = (x_{i1}, x_{i2}, \dots)$, which enumerates all the variables inside the i -th group. The $\tilde{\mathbf{x}}$ represents the vector having \mathbf{u}_i entries moved by a $\pm\Delta$ quantity.

In conclusion, improved Morris method can gather quite good sensitivity analysis results without a too wide number of function evaluations. This is a key point of all the screening techniques. Comparing it to the Variance-based method 5.3.3, it can

obtain good results with a smaller number of evaluation, although it lacks in accuracy with respect to this other cited. On the other hand it can easily perform group analysis which are very useful in first testing. If the grouping is properly designed, it can retrieve results similar to those achieved in simple Morris model.

5.3.3 Variance-based method

This model, as its name suggests, studies how the variance of the output depends due to the uncertainty of the input and how this last can be decomposed accordingly. The method develops the use of variance, since it is the expression of the uncertainty itself and it can properly describe which are the main variables that rule the objective function. Here one puts interest on how the fitness can vary inside each variable domain and what causes this variation. The target of this test is no more the best objective value of the function, as happens in the optimization process, or the mean of fitness [51].

Variance-based is a global sensitivity analysis method which is able to gain information not only on the single variable, but also on its interaction with detailed results. Let's consider an objective function as $Y = f(X_1, X_2, \dots, X_k)$, where Y is a scalar objective and X_i are the k variables. The variance-based first order effect for the generic variable X_i is defined as:

$$S_i = \frac{V_{X_i}(E_{\mathbf{X}_{\sim i}}(Y|X_i))}{V(Y)} \quad (5.11)$$

Here X_i is the i -th factor and $\mathbf{X}_{\sim i}$ identifies the matrix containing all the variables but X_i . The numerator of eq. 5.11 evaluates the variance associated with the factor X_i . The expectation of Y is performed over all the possible values $\mathbf{X}_{\sim i}$, so with X_i as a fixed factor. Therefore, to evaluate the fixed factor variance, the expectation is taken over all the possible value associated to X_i . This builds a set of values generated by varying only the X_i [53]. Due to the identity:

$$V_{X_i}(E_{\mathbf{X}_{\sim i}}(Y|X_i)) + E_{X_i}(V_{\mathbf{X}_{\sim i}}(Y|X_i)) = V(Y), \quad (5.12)$$

the sensitivity index S_i is also normalized with respect to the total objective function variance and it varies in $[0, 1]$. This sensitivity index highlights how the factor influences by its own the objective, so its additive effect on the output. A constrain to the first order sensitivity coefficient is:

$$\sum_{i=1}^k S_i \leq 1 \quad (5.13)$$

Taking for example a pure additive objective function, each factor affects only by its own the function. Since no interaction is present, the entire variance of the objective needs to be described by first order index and their sum will take value 1. On the other hand, considering a function such as:

$$f(\mathbf{x}) = x_1 x_2 + \sum_i x_i, \quad (5.14)$$

it shows a simple product relation between two variables and it introduces an interaction in the system. The interaction cannot be described by the first order indexes alone. Their sum won't be any more equal to 1, because the coefficients S_i lack in describe the interaction and cannot capture its contribute to the variance.

A second fundamental measure which the variance-based method performs is the total effect index S_{Ti} . This quantity describes the total variance introduced in the function due to the i -th variable and all its interactions:

$$S_{Ti} = \frac{E_{\mathbf{X}_{\sim i}}(V_{X_i}(Y|\mathbf{X}_{\sim i}))}{V(Y)} = 1 - \frac{V_{\mathbf{X}_{\sim i}}(E_{X_i}(Y|\mathbf{X}_{\sim i}))}{V(Y)} \quad (5.15)$$

This index measures the total effect of X_i , which is given by all the order of interactions up to the greatest one. Instead of let varies all the other variable and successively the one under study as it happens in the first order index 5.21, now the model makes first vary the chosen variable and takes its expectation and successively let all the other factors vary. So it can evaluate all the variable interactions. However, in this case it does not exist a constrain to the sum of the total effects. In fact, observing a function which presents interactions, as eq. 5.14, the sensitivity gain of total indexes of the interacting variables is added to all the factors which are related. So in the previous function both x_1 and x_2 present the interaction shared. Summing all the total sensitivity index this contribution would appear doubled, even if again the indexes are normalized by the total variance $V(Y)$. However here a further inequality holds true:

$$0 \leq S_i \leq S_{Ti} \leq 1 \quad (5.16)$$

Eq. 5.16 states that the total sensitivity index is always greater than the first order one. This holds true because the total index incorporates all the interaction orders.

A further index can be analytically calculated and it is the second order effect of each variable [52]. This value, similarly to the former two, describes the second order moment of each variable, which is the interaction between two of them. It is described by the equation:

$$V_{ij} = V_{X_i X_j}(E_{\mathbf{X}_{\sim ij}}(Y|X_i, X_j)) - V_{X_i}(E_{\mathbf{X}_{\sim i}}(Y|X_i)) - V_{X_j}(E_{\mathbf{X}_{\sim j}}(Y|X_j)) \quad (5.17)$$

As can be seen, instead of accounting also for the first order sensitivity as happen in the total order index, the second order evaluates only the contribute of coupled factors. Observing once again eq. 5.14 and using only the introduced indexes, it is possible to completely retrieve the entire objective function variance.

Later the numerical and computational aspects will be treated in detail. Therefore analytically the indexes obtained by 5.11 and 5.17 completely assign to each factor the relative amount of variance up to a second order equation. On the other hand the total order index from 5.15 gives a quite good view on any general function. This, sided to first and second order indexes, would achieve a deep knowledge of the function at hand. From a theoretical point of view all the higher order terms can be calculated. Dealing with a general function f of k variables, from functional the decomposition scheme:

$$f = f_0 + \sum_{i=1}^k f_i + \sum_{i=1}^k \sum_{j>i}^k f_{ij} + \dots + f_{12\dots k} \quad (5.18)$$

All the terms in former equation can be obtained by proper decomposition, and this can be translated to the variance of the objective function, giving:

$$V(Y) = \sum_{i=1}^k V_i + \sum_{i=1}^k \sum_{j>i}^k V_{ij} + \dots + V_{12\dots k} \quad (5.19)$$

Diving both sides the eq. 5.19 by the total variance $V(Y)$, it becomes:

$$\sum_{i=1}^k S_i + \sum_{i=1}^k \sum_{j>i}^k S_{ij} + \dots + S_{12\dots k} = 1 \quad (5.20)$$

This last equation proves the constraint previously introduced by eq. 5.13. Each S term represents exactly an i -th sensitivity order index and all together they give the total function variance.

Dealing now with the computation of these indexes, several implementations have been proposed since the first introduction of variance-based model. Anyway, the most common of them can be found in the articles [56], [52], [53], [45] and books [51], [4], [54]. The first step consists in generating two independent sampling matrices \mathbf{A} and \mathbf{B} , with the respective elements a_{ij} and b_{ij} . The j identifies the number of factor, from 1 to k for each sample, which is the domain dimension. Instead i represents the number of total simulation points, N . Once done this sampling, the following step builds the matrices $\mathbf{A}_B^{(j)}$ (or equivalently $\mathbf{B}_A^{(j)}$). Each of them combines both matrix \mathbf{A} and \mathbf{B} . The j -th $\mathbf{A}_B^{(j)}$ matrix has all the columns from \mathbf{A} , but the j -th one which is taken from \mathbf{B} . This allows to calculate the sensitivity index of each factor. Recall that their variance is evaluated first fixing the selected factor, letting all the others vary and then changing only this variable. The next phase requires to evaluate with the objective function the samples generated. Since the problem has k dimensions and the starting sample size is N , the total evaluations required are $N(k+2)^8$. The last phase calculates the total variance of the objective function $V(Y)$ and its mean value f_0 , before computing the sensitivity indexes. To retrieve this value all the sampled points could be used, but since often N is pretty large, just the entries of matrix \mathbf{A} typically are used.

Finally it is possible to compute first and total order sensitivity indexes. These two require the same computational effort and data, so they can be calculated without any further burden of the system:

$$S_j = \frac{\frac{1}{N} \sum_{i=1}^N f(\mathbf{A})_i \left(f(\mathbf{B}_A^{(j)})_i - f(\mathbf{B})_i \right)}{V(Y)} \quad (5.21)$$

$$S_{Tj} = \frac{\frac{1}{2N} \sum_{i=1}^N \left(f(\mathbf{A})_i - f(\mathbf{A}_B^{(j)})_i \right)^2}{V(Y)} \quad (5.22)$$

Equation 5.22 was first proposed by Jansen in [28] and later it was proved to perform better than the others.

Dealing with second order index, it requires more computational efforts. It needs to perform $N(2k+2)$ function evaluations, which are almost doubled with respect to the former case. This happens since treating two indexes simultaneously, both of them need to first be fixed and later vary, but each on its own. Hence in this case both matrices $\mathbf{A}_B^{(j)}$ and $\mathbf{B}_A^{(j)}$ are needed. From the theoretical equation 5.17 to calculate second order index, one needs also the first order indices S_j and S_k . However it comes

⁸The $\mathbf{A}_B^{(j)}$ matrices are k , one for each column changing, and then there are obviously also \mathbf{A} and \mathbf{B} . Each matrix then has N lines, which are the sampling points.

with no further cost or burden since it has been already calculated or it requires a simple evaluation:

$$S_{jk} = \frac{\frac{1}{N} \sum_{i=1}^N \left(f(\mathbf{B}_A^{(j)})_i f(\mathbf{A}_B^{(k)})_i - f(\mathbf{A})_i f(\mathbf{B})_i \right)}{V(Y)} - S_j - S_k \quad (5.23)$$

A last comment is about variance-based method dealing with the sampling strategy to build matrices \mathbf{A} and \mathbf{B} . Literature suggests to choose Quasi Random sequences of N points in a $2k$ -dimensional space and then to build the matrices cutting in half the sample, dividing the starting one in the $(N \times k)$ desired structures. The \mathbf{A} matrix is the left half of the sequence, while \mathbf{B} is the right part. Moreover Sobol' sequence gets usually suggested and performed, since it is the most common one. The matrices \mathbf{A} , $\mathbf{A}_B^{(j)}$ and \mathbf{B} contain higher number of *good points*. These namely cover uniformly the domain space and they are best equidistributed, which are the properties of Quasi Random sampling 5.2.5.

5.3.4 Derivative-based method

This model deals with square-integrable functions $f(x_1, \dots, x_k)$ defined in the unit hypercube with associated the Lebesgue measure $dx = dx_1 \dots dx_k$ [57]. As described for variance and sensitivity indexes, such functions can be decomposed due to the ANOVA decomposition⁹:

$$f(x) = f_0 + \sum_{s=1}^k \sum_{i_1 < i_2 < \dots < i_s} f_{i_1, \dots, i_s}(x_{i_1}, \dots, x_{i_s}) \quad (5.24)$$

Thanks to this equation it is possible to rewrite the function as in equation 5.18. It is possible to define also partial variance constants:

$$D_{i_1, \dots, i_s} = \int_{[0,1]^k} f_{i_1, \dots, i_s}^2(x_{i_1}, \dots, x_{i_s}) dx \quad (5.25)$$

Summing up them all, one obtains the total variance:

$$D = \int_{[0,1]^k} f^2(x) dx - f_0^2 \quad \text{or} \quad D = \sum_{s=1}^k \sum_{i_1 < i_2 < \dots < i_s} D_{i_1, \dots, i_s} \quad (5.26)$$

From this last equation it is possible to calculate the sensitivity indexes, as $S_{i_1, \dots, i_s} = \frac{D_{i_1, \dots, i_s}}{D}$, which are all possible combination of order. In particular to retrieve the first and total order indexes defined in 5.3.3, the required calculation are $S_i = \frac{D_i}{D}$ and $S_{Ti} = \frac{\sum_{(i)} D_{i_1, \dots, i_s}}{D}$.

Dealing with partial derivatives, it can be found a connection between Morris analysis (or Elementary Effect, 5.3.2) and Variance-based method (5.3.3). Proofs of the following theorems can be found in the original paper of Sobol [57]. These theorems set the fundamentals estimation to build the derivative sensitivity measures.

⁹ANOVA stands for *Analysis of Variance*, which analyse the contribute of each single variable and all the possible grouping of the same variable, obtaining the contribute of each factor

Theorem 1. Assume that $c \leq \left| \frac{\partial f}{\partial x_i} \right| \leq C$. Then:

$$\frac{c^2}{12D} \leq S_{Ti} \leq \frac{C^2}{12D} \quad (5.27)$$

The constant factor 12 in 5.27 cannot be improved.

This first theorem and the relative proof yield a good approximation formula for the total order sensitivity index:

$$S_{Ti} \approx \frac{1}{12D} \int_{[0,1]^k} \left(\frac{\partial f}{\partial x_i} \right)^2 dx \quad (5.28)$$

The second theorem states:

Theorem 2. Assume that $\frac{\partial f}{\partial x_i} \in L_2$. Then:

$$S_{Ti} \leq \frac{1}{\pi^2 D} \int_{[0,1]^k} \left(\frac{\partial f}{\partial x_i} \right)^2 dx \quad (5.29)$$

Theorem 2 gives further knowledge on the sensitivity index. However, the relation does not bring any error estimate, so there is not strong reliability in the formula. A recommended test to evaluate the approximation suggests to analyse the second derivative $\frac{\partial^2 f}{\partial x_i^2}$ and it validates the estimate if it is negligible with respect to the first order derivative.

Now it is possible to introduce a new set of parameters, ν_i, \dots, ν_k , which are the sensitive values in derivative-based analysis. Each one takes value:

$$\nu_i = \int_{[0,1]^k} \left(\frac{\partial f}{\partial x_i} \right)^2 dx \quad \text{with } 1 \leq i \leq k \quad (5.30)$$

The calculation of this parameter is quite similar to that of μ^* in Elementary Effect. There, instead of the square, the operator was the absolute value. These two criteria are equivalent from the practical point of view (in fact the development of Morris analysis took in consideration also the square operator) and they are evaluated by the same numerical computation, giving rise to the relation:

$$\begin{aligned} \nu_i &\leq C\mu_i \\ \mu_i &\leq \sqrt{\nu_i} \end{aligned} \quad (5.31)$$

The only difference between these parameters is the further inequality that holds true only for the derivative factors:

$$S_{Ti} \leq \frac{\nu_i}{\pi^2 D} \quad (5.32)$$

Eq. 5.32 provides an estimate of S_{Ti} , even without knowledge on the upper bound C of the partial derivative.

So, such method can evaluate quite well a factor analogue to the total sensitivity index and to μ with less computation than those required in variance-based method. Although Sobol' method leads to further knowledge of the factors, which are the first sensitivity indexes. Moreover derivative-based method requires also less computational cost than elementary effect method. This last results to be also inaccurate sometimes,

in particular for non-monotonic functions, even if this problem is solved with the use of μ^* . On the other hand this former method computes also the variance of each parameter by σ , if it is able to retrieve further aspects of the objective function in its computational time.

Although, also derivative-based method does not always perform well: dealing with strong non-linear terms it often obtains an higher prediction value than the real total sensitivity index, so it falls in a Type-I error.

Chapter 6

Model Implementation and Application

The model that has been developed in this study involves a genetic algorithm 3.2 and a surrogate model of Response Surface methodology 4.4. This choice has been done trying to realize a model which could side pretty satisfying optimization ability with quite cheap computational costs.

The first step of this work dealt with the investigation of the most spread evolutionary algorithms and surrogate models, to analyse their pro and cons. The following phase worked on statistical and mathematical models to catch the main features of the objective functions: this leads in particular to readings about correlation and sensitivity analysis, between which the second one has been selected as it seemed more suitable for the task. Hence, reviewing surrogate models knowing the abilities of sensitivity analysis, Response Surface Method appeared the proper choice, as it works well with relative small dimensional problems and it is very much adaptive to various situations. The final analysis treated the best optimization algorithm to use among those met in literature. However, due to the relative large lightening of the problems at hand thanks to the surrogate model, this choice fell on a quite basic genetic algorithm, as the *NSGA-II*. Moreover, this choice was taken after a brief literature analysis of some articles [8, 16, 35, 44, 47, 49], which describe the resulting performance of collaboration between Response Surface and Evolutionary Algorithms.

The development of such model includes aspects from statistics and numerical computation and it gives the chance to deepen and enlarge the knowledge of these subjects. On the other hand, this kind of models are nowadays widely applied in engineering fields and proper adjustments could lead to interesting applications.

Let's now study step by step the features of the implemented method, analysing at each phase the resulting information and their following uses.

6.1 Sensitivity analysis application

The sensitivity analysis methods chosen are the Sobol' 5.3.3 and Morris analysis 5.3.2. The first choice is due to the fact that Variance Based Method can capture with good accuracy first and total order variances and it gives ready-to-use results. On the other hand, the Elementary Effects method requires less computational costs and provide anyway sensitivity data with a fair accuracy. Furthermore, this method needs elaboration of the results to proper realize which is their true meaning, while it provides

directly only the total order sensitivity. However, it has an important pro, which is the possibility to work by groups of variables. Since the objective functions that will be used in the model testing, reported in appendix A, present a large number of variables, this kind of analysis could help to reduce computational costs.

6.1.1 Sobol' sensitivity analysis

Let's first deal with the variance-based method and its results. As stated above, it gives to the final users the first and total order sensitivity index with quite fair computational costs. At higher costs, it gives also the second order index, referred to each possible couple of variables. Recalling eq. 5.20, that constrains the sum of all the sensitivity index to be equal to 1. Though, this is not true for the total sensitivity parameter, since it includes for each variable all its interactions with the other ones¹; it is important to remember these facts to properly understand and read the result of the analysis.

Variable	S_1	$S_{1,Confidence Lev.}$	S_{Tot}	$S_{Tot,Confidence Lev.}^2$
x_1	0.3980	0.0320	0.5485	0.0384
x_2	0.3919	0.0371	0.5527	0.0423
x_3	0.0037	0.0040	0.0063	0.0006
x_4	0.0028	0.0044	0.0059	0.0006
x_5	0.0060	0.0045	0.0064	0.0005
x_6	0.0039	0.0041	0.0063	0.0005
x_7	0.0013	0.0036	0.0063	0.0005
x_8	0.0032	0.0039	0.0062	0.0006
x_9	0.0044	0.0040	0.0063	0.0006
x_{10}	0.0049	0.0044	0.0067	0.0005
...

Table 6.1: Results of Sobol' sensitivity analysis for DTLZ1, from appendix A.2 on test function. It reports the values of the variables from the first to the tenth and for the first objective function only.

From table 6.1 it can be seen the result given by sensitivity analysis. Here are reported just 10 out of the 14 variables of problem DTLZ1, however it is quite evident which are the most influential variables: x_1 and x_2 . In fact, observing just the first two order indexes, they contribute for almost 80% of the total objective function variance, while dealing with the total sensitivity, each one contributes for about 55%, hence considering their interaction, one would almost completely get the behaviour of the real test function. Table reports also the *confidence level* of each index evaluation: here with confidence level it is denoted the maximum deviation from the mean value of a given prediction, which is calculated with a *confidence parameter* of 95%. Hence to properly measure the goodness of given results it is necessary to analyse at the same time both the sensitivity index and its confidence level. Running several times the code

¹Hence, summing up all of the total order sensitivity index will give a value larger than one, until the function is a linear relation

²The *Confidence Level* is the maximum deviation possible from the mean value of a given prediction, calculated with a selected *confidence parameter*.

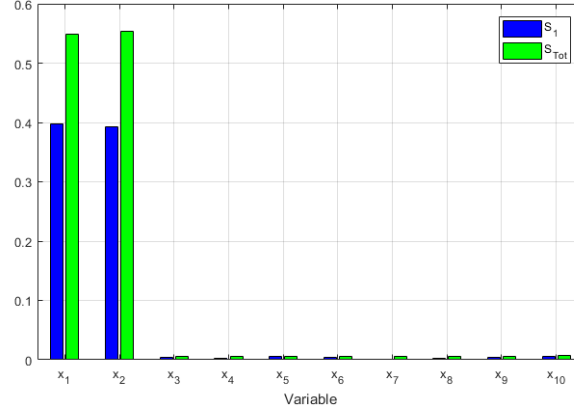


Figure 6.1: Histogram of Sobol' sensitivity analysis referred to data in tab. 6.1.

with different parameter inputs, as described in section 5.3.3, just modifying the sample dimension³ to perform the analysis, this value can drastically change. Selecting a small sample, all the confidence levels increase a lot, therefore taking away the meaning of the analysis. On the other hand, choosing a much larger number of points would not realize better and better sensitivity evaluation: depending on the number of function variables one needs to choose properly the dimension sample.

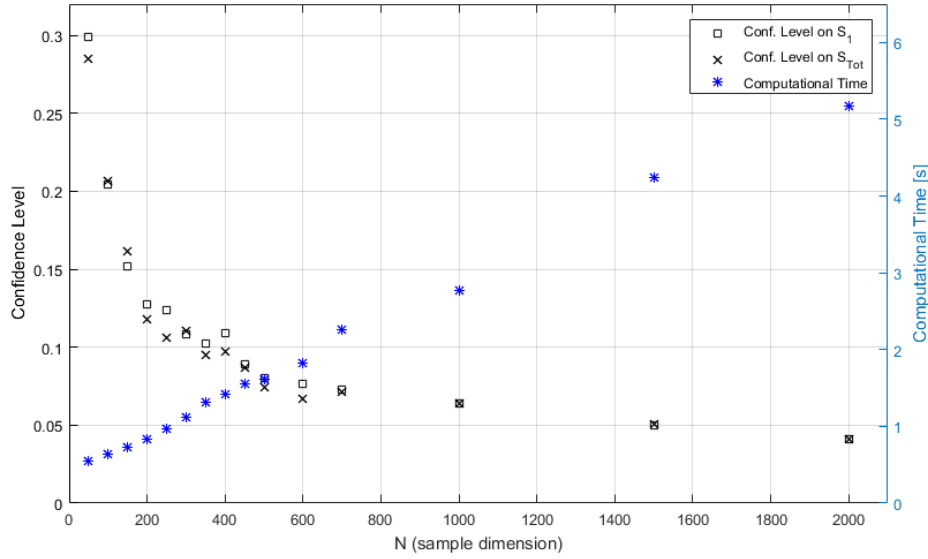


Figure 6.2: Variability of *Confidence Level* and *computational time* versus sample dimension for DTLZ2 problem. Black symbols refer to the Confidence Level left axis, while the blue ones refer to the Computational Time right axis. Average values for the indexes are: $S_1 = 0.4491$ and $S_{Tot} = 0.5558$.

Dealing with the test function DTLZ2, as reported in fig. 6.2, it is evident how much steep is the variation of confidence for adding points in small samples than in

³The *sample dimension* (N) term indicates the points selected for evaluating each variable variance. Dealing with a k -dimensional test function, the total number of points in the sample will be $k(N + 2)$, as already seen.

bigger ones. Doubling the sample size does not improve the estimation over the sensitivity index. On the other hand the computational time grows almost linearly with the sample dimension. Then it is better to choose a trade-off value to perform a good sensitivity estimation, without unnecessarily burden on the model, also depending on the complexity of evaluating objective functions.

Typically for the problems at hand this parameter has been set from 5 to 15 times the number of variables defining the function: as reported in A.2, DTLZ2 has 22 variables and a value of $N = 250 \div 350$ gives a good trade-off between accuracy and costs.

Interaction variables		S_2	$S_{2,Confidence Lev.}$
x_1	x_2	0.1483	0.0700
x_1	x_3	0.0157	0.0546
x_1	x_4	0.0187	0.0556
x_1	x_5	0.0172	0.0553
...
x_2	x_3	-0.0021	0.0515
x_2	x_4	-0.0020	0.0519
x_2	x_5	-0.0010	0.0513
...
x_3	x_4	-0.0003	0.0066
x_3	x_5	-0.0012	0.0066
...

Table 6.2: Results on second-order Sobol' sensitivity analysis for the first objective function of DTLZ1, from ch. A.2 on test function; just few interaction terms are reported here.

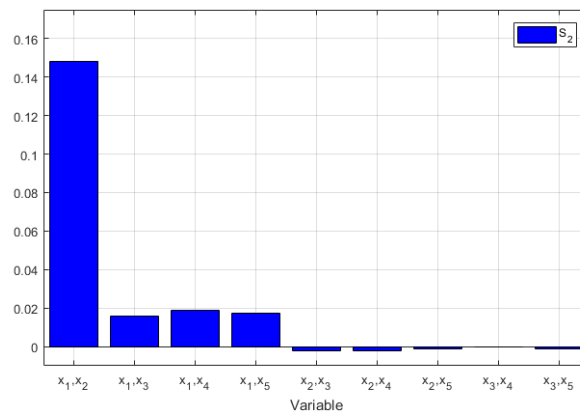


Figure 6.3: Histogram of second order Sobol' sensitivity analysis referred to data in tab. 6.2.

A further result can be obtained by the variance-based method of sensitivity analysis and it is the second-order index of the interaction between different variables by eq. 5.23, as reported in table 6.2. Also higher levels of interaction could be evaluated with

further equations, but these would require much more computational efforts for kind of later useless knowledge: in fact, the higher is the interaction the lower the contribute to the objective function variance. Moreover, higher level of interactions needs much larger sample set to provide fairly acceptable results: data reported on second-order table is generated with a sample size $N = 1500$, and it presents a confidence level more or less doubled with respect to data in table 6.1, based on a sample size of $N = 300$. This happens because the eq. 5.23 brings with itself the errors of first order indexes, leading to larger deviation.

However, observing first, second and total-order sensitivity indexes of the variables x_1 and x_2 and summing up the first two, the resulting value is, with quite good approximation, the total index. All the second order indexes have much lower values with respect to the first one, even if this could be not always true for other kind of functions. Although it can be found in table 6.2 negative values, these are results of numerical and approximation errors, again due to the second-order indexes equation; all of them can be view as null, considering also the relative confidence level.

All the Sobol' analysis performed on the test function described in appendix did not really required the second-order indexes calculation, or because their contribution is negligible and it does not bring useful knowledge to the behaviour of the chosen function, due to their definitions.

6.1.2 Morris sensitivity analysis

Morris analysis was first taken in consideration as sensitivity method to perform group analysis, on all those test functions with a large number of variables. Successive testing of variance-based and Elementary Effects methods, to compare their performance on the test functions, highlighted a little gain using the second one in terms of computational cost, both in the grouped and in single variables analysis, due to the simple process of evaluation. However, the first method can give further and more accurate information and so was chosen to describe all the test functions. Thus application of Morris method is only as mere comparison with results of the variance-based method to test group sensitivity analysis and concretely gets in touch with a screening method.

Variable	μ^*	μ	$\mu_{Conf. Lev.}^*$	σ
x_1	307.35	307.35	16.91	194.18
x_2	294.45	294.45	15.53	181.17
x_3	15.84	0.83	1.72	25.45
x_4	15.65	-1.33	1.65	24.62
x_5	14.70	0.25	1.73	24.44
x_6	16.20	0.13	1.60	24.57
x_7	17.16	0.74	1.94	27.99
x_8	16.30	0.59	1.66	24.92
x_9	15.58	0.57	1.70	24.74
x_{10}	15.49	0.10	1.82	25.37
...

Table 6.3: Results of Morris sensitivity analysis for the first objective of DTLZ1, from appendix A.2 on test function. Only the first ten variables are reported here.

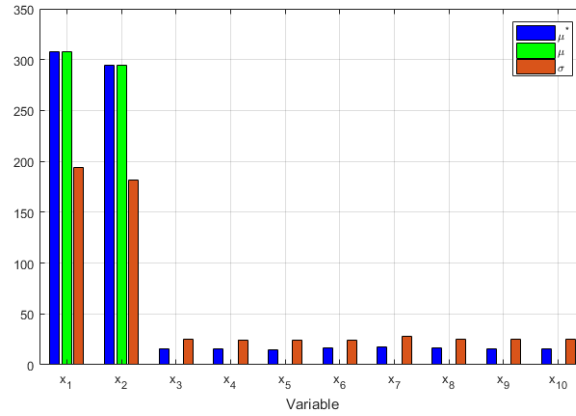


Figure 6.4: Histogram of Elementary Effects analysis referred to data in tab. 6.3.

Grouping 1				Grouping 2			
Group	Variables in the group	μ^*	$\mu^*_{Conf. L.}$	Group	Variables in the group	μ^*	$\mu^*_{Conf. L.}$
G_1	x_1, x_2	398.76	11.35	G_1	x_1, x_8	336.97	6.74
G_2	x_3, x_4	21.70	1.30	G_2	x_2, x_9	332.75	6.73
G_3	x_5, x_6	20.17	0.94	G_3	x_3, x_{10}	19.44	0.80
G_4	x_7, x_8	20.11	0.96	G_4	x_4, x_{11}	20.40	0.86
G_5	x_9, x_{10}	20.39	1.00	G_5	x_5, x_{12}	20.00	0.80
G_6	x_{11}, x_{12}	20.25	0.96	G_6	x_6, x_{13}	20.75	0.87
G_7	x_{13}, x_{14}	19.72	0.96	G_7	x_7, x_{14}	20.70	0.83

Table 6.4: Results of Morris sensitivity analysis by groups for the first objective of DTLZ1, from appendix A.2 on test function. Here there is a simple sequential coupling of the variables, but it is quite evident the difference on μ^* value with respect to the previous table 6.3 in the first group.

The reported table 6.3 and 6.4 describe Elementary Effects results for DTLZ1 test function, defined by 14 variables, respectively considering each variable alone and grouping them in couples. Let's briefly recall from section 5.3.2 the meaning of the calculated parameters: μ and μ^* describe the overall mean influence of the selected decision variable on the objective function variation, respectively summing up each contribute with its sign and considering its absolute value; σ instead describe the variance of the parameter μ , while $\mu^*_{Conf. Lev.}$ measure the confidence level of μ^* . Dealing with group sampling, it is useless measure each single influence with its sign and the relative variance, because it is function of two or more variables which has unknown behaviour taken singularly.

Looking at the analysis results and considering DTLZ1 test function A.2, in which the variables x_1 and x_2 brings always a positive contribute, it is quite outright find this behave on the first table: both variables in fact present equal values of μ^* and μ . Dealing instead with the other variables, one would immediately notice how μ parameter sets around zero and this, again, could be guessed a priori from the test function. Analysing finally the variability introduced in the objective function by

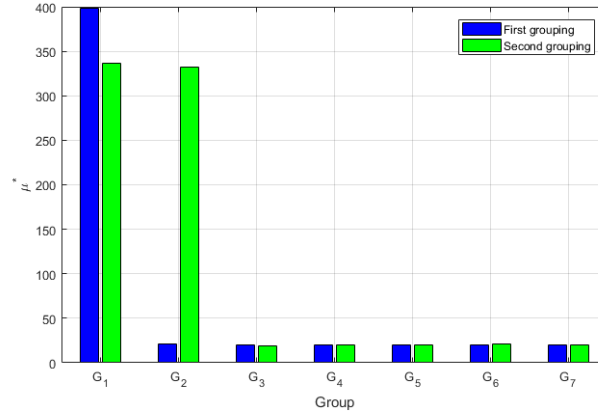


Figure 6.5: Histogram of Elementary Effects analysis by groups referred to data in tab. 6.4.

each single variable it is clear from μ^* values that first and second variables play a dominant role, as resulted also from the previous Sobol' analysis. The σ parameter then describes how much it is the variance of each variable; it could further help to recognize the influence of the relative variable on the objective function, since variance of x_1 and x_2 introduce in the system much more fluctuation than the whole contributes of the other variables.

Regarding the group analysis results, they confirm and strengthen the observation just made, though not bringing that much information by its own. However, looking at a single group analysis alone, *Grouping 1*, one would immediately obtain the most important information, which is the almost complete dependence of objective function from variables x_1 and x_2 .

Moreover, in table 6.4 are reported two different grouping sample, both generated by couple of variables. This example well describes which are the main features of the screening method coupled with groups: first of all, this gives a fair view over the contribution of variables to the objective function as said above; furthermore, performing rearrangement of variables in the groups it is possible to deeply investigate the contribution of each single variable with lower computational costs. Here, with the first grouping, one would not know if just one or both variables x_1 and x_2 have large influence on the function, but would immediately get that all the other variables are quite negligible. A second run with a new grouping is very important, because it is now possible to realize that both x_1 and x_2 are influential variables and with a similar weight, matching the results. Developing a reasoned grouping sample for much larger and complex problem, e.g. with 500 or more variables and computational demanding objective functions, can provide a complete overview on the behave of the function, without requiring too much resources. Note finally that in this method does not need to realize groups of the same size necessary: once identified which could be the main variables, some groups could contain just each of them and a further group contains all the other negligible factors.

Figure 6.6 reports, as for the variance-based method, the change on computational time and confidence interval versus the sample dimension. However, in this case performing the analysis for single variables and grouping them would bias the results, because also the μ^* value considered changes, therefore the relative mean value is reported in the figure description. As one could expect, the two behaviours obtained are

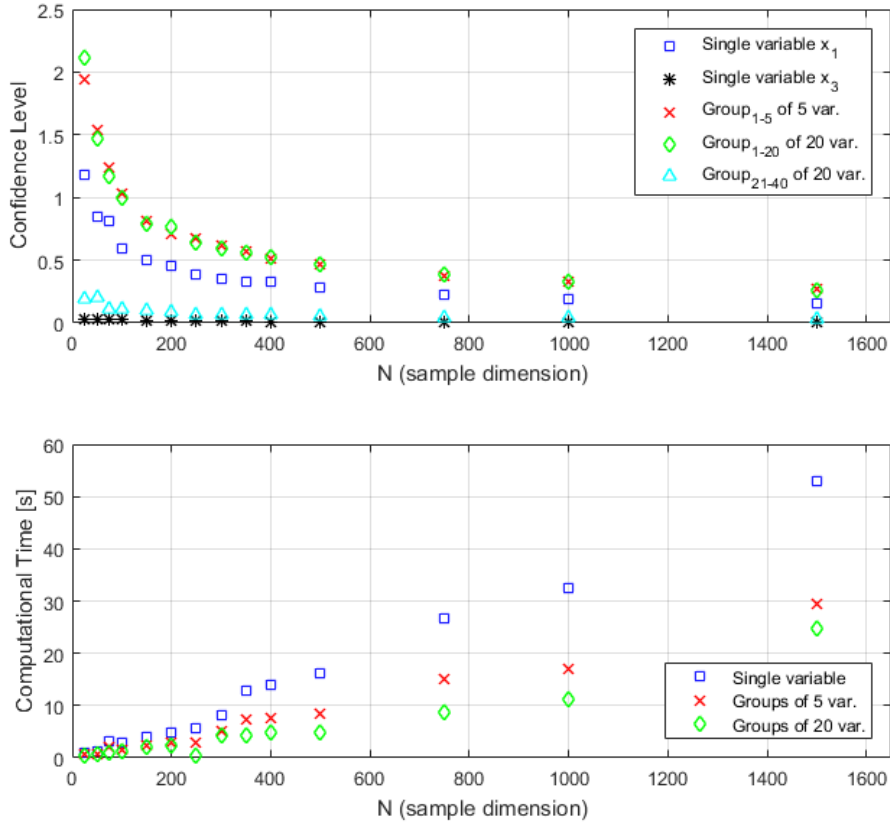


Figure 6.6: Variability of *Confidence Level* (upper) and *computational time* (lower) versus sample dimension for DTLZ6 problem. Average values of μ^* for single variable analysis, group of 5 and 20 variables are respectively: $\mu_1^* = 6.22$, $\mu_3^* = 0.11$, $\mu_{group\ 1-5}^* = 8.38$, $\mu_{group\ 1-20}^* = 8.64$ and $\mu_{group\ 21-40}^* = 0.48$.

very much similar to those observed above in the Sobol' analysis, even if here the test problem used is the first objective function of DTLZ6 A.2, defined by 100 variables, because it is much more suitable to several different grouping sizes. Observing the picture it is possible to notice a pretty large step in confidence level between singular variable and group analysis, which is present due to the chosen group selected: while groups which contain variables x_1 and x_2 grab also their large variable behaviour, groups that involve only negligible factors would show their limited variability and therefore a confidence level almost null, as happen to the confidence level of a single variable alone x_3 . Finally, observing grouping of the first 5 and 20 variables, the differences are almost null just because the factors, which are not x_1 and x_2 , would bring an almost null contribute.

Dealing with the coding aspects, this part of the implemented model has been realized using the *Sensitivity Analysis Library (SALib)* available in the open source programming language Python [24]. Previously the use of this library, several routines dealing with Variance-based sensitivity methods were written, based on the articles of Sobol' and Saltelli [52, 53, 56] as the library itself. Although, lacking the complete knowledge to evaluate the *confidence level* and some notion to realize the Sobol' sequence, briefly explained in [31], it has been chosen to use the library. On the other

hand, this makes possible to introduce and apply also the Elementary Effects method, which is quite more difficult to code, in particular dealing with the grouping aspects. Anyway, before its use, the entire library was studied and tested deeply to ensure the utility of all the routines and functions. Moreover, to properly take advantage of the information returned by the Morris and Sobol' methods it has been necessary to understand and store the analysis at each code run, which just displays on the local console the sensitivity indexes, without saving that information somewhere.

6.2 Response Surface modelling

Once realized the sensitivity analysis and obtained a good knowledge of the test function to model, it is possible to introduce the response surface method. Let's now describe in detail all the steps of algorithm 9, investigating deeply all its functions and how the arisen problems have been treated.

The first parameter to model the surface is the choice of the variables, their number and which one between all the objective function factors. To take this decision the sensitivity analysis sets in: the proper choice came out to be exactly the variables, which play a large role in the variability of the objective function. In this way, when calculating the surface coefficients, each of them can highly resemble the true behaviour of the original relation. On the other hand, generating surfaces including the variables which realize a negligible variation, would affect the proper estimation of important factors. Anyway, augmenting or decreasing the number of selected variables deeply affects the surrogate model: response surface is a method which works very well in low dimensional frame, because its fitting phase is almost never computed on-line, but it is rather developed by simple linear algebra 4.28, highly dependent on the number of coefficients to calculate and on the chosen sample points; introducing a further variable in the design would mean calculate all over again the entire response surface.

Even if the surface is often built with a little number of variables, it allows first of all to perform and realize a huge amount of test, varying the other parameters, hence obtaining a complete knowledge of the model behave over different settings: it is one of the key features to properly understand how the model works, before apply much more complex and computational expensive reality-based methods, requiring hours or even days of execution. Moreover, despite being a quite light surrogate with respect to many others, as the described Artificial Neural Network and Kriging filter are, it had proven to perform pretty well on the test problems, providing acceptable prediction compared with more sophisticated methods, as will be reported in the next chapter 7.

The second input in the surface building is the order of the polynomial surface: the implemented models realize first, second and third order complete surfaces and a partial third-order one, this last calculating only the mixed third order interaction coefficients (e.g. β_{123}). Set apart the first order model, which can properly realize surfaces describing only linear or quasi-linear relations and hence without alluring fitting capacities, the others provide good predictions. Once again, the tool helping this choice is the sensitivity analysis, in particular the Sobol' analysis in this case: even if it does not compute third order indexes, knowing first, second and total order sensitivity, one can easily cross them and select the proper desired order.

Given this input, the method starts to build the response surface: first of all it retrieves from the problem at hand the domain and the surface bounds of each variable,

the second one providing information about the region in which will be built the surface. This last parameter could be user defined, but for all the test functions of this work it had been chosen to realize a single surface over the entire domain space; while for complex function this could provide low quality results, dealing with the problem at hand the fitting resulted quite good and hence for all the tests the setting had been accepted.

Now it is possible to define the sample points: in previous chapter 5.2 and in section of response surface method 4.4 has been named and described several sampling techniques and just few of them were adopted. To provide a good description of the border region factorial design combined with axis sample and tri-factorial design interchange: since p -factorial design in k variables build a sample of p^k points, which becomes rapidly large just when $p = 3$, the tri-factorial design gets used only when the chosen variables to describe the surface are less or equal than 4. In the other cases the two-factorial design is used, assisted by the axis sample, adding just $2k$ points; the axis sample assists only the factorial design simply because in tri-factorial it introduces points that are already sampled, thanks to its features. The following choice consists in choosing a method between random, quasi-random and Latin Hypercube sampling. The random sample 5.2.1 can be straight away discarded for the quasi-random one, because as described in the comparison in 5.2.5: while the first method realizes cluster and empty spots in the domain and so fails in filling it uniformly, quasi-random sampling manages to do it with no further computational costs. Instead, dealing with Latin Hypercube sampling, it builds easily good samples for discrete random variables, while in presence of continuous ones it needs to be sided with the random sampling inside the selected interval.

Few tests had been run to investigating the behave of surface with quasi random and Latin Hypercube sampling, proving that for little samples they give similar results. However, enlarging the sample dimension the quasi-random technique realizes better performance because all the points are more or less at the same distance, while Latin Hypercube shows some clustering effect, due to the randomness inserted in the intervals. Then the tests had been repeated, this time selecting the mid-point of each interval, realizing better samples which could be compared with the quasi-random ones. Anyway, the method selected to define the samples was the quasi-random technique realizing Sobol' sequences due to its light computational costs, since it just requires the starting parameter list and few further details to perform the sample.

Once generated all the samples and merged them together, the users can select the number of points which will realize the surface settings. The first choice available is an exact fitting over the sample points, which requires to have the same number of surface coefficients and sample points; this fit could provide good results when the designed surface is a second or third-order one and the number of factors defining it is large. Another choice is to input the total desired number of points defining the fitting sample set, knowing the number of β parameters to calculate. Finally the simplest choice select all the sampled points to define the fitting set. When this is not the user's choice, one by one the points in the sample set are randomly selected and moved to the fitting set, until it does not reach the required dimension.

The successive step in the Response Surface method deals with the building of the sample for the original test function, starting from the one just realized. This

step requires to introduce for each vector of sample point the values taken by the non-modelled variables. As described above in 4.4.1, all the literature [3, 33, 34, 43] and the articles [13, 58] treats this variables fixing their values in the mid-point of the relative domain. However, this setting had revealed to be misleading: the surfaces such built in different test problems shown opposite fitting behaviours. Several three objective functions in A.2 presented amusing results, quite near to those obtained with more complex methods and to the true Pareto Front. On the other hand two objective test problems in A.1 was far from the true Pareto Front and most of the others algorithms; trying to modify the parameters such as number of surface variables and surface order did not realize any valuable improvement. Later, accurate analysis of the objective functions shown that the key problem was related to the non-modelled (later called also *secondary*) variables setting and the definition of test problems: several three objective tests' Pareto Fronts occur when all the *secondary* factors take value 0.5, which is exactly the mid-point of their domain, while two objective ones behave differently.

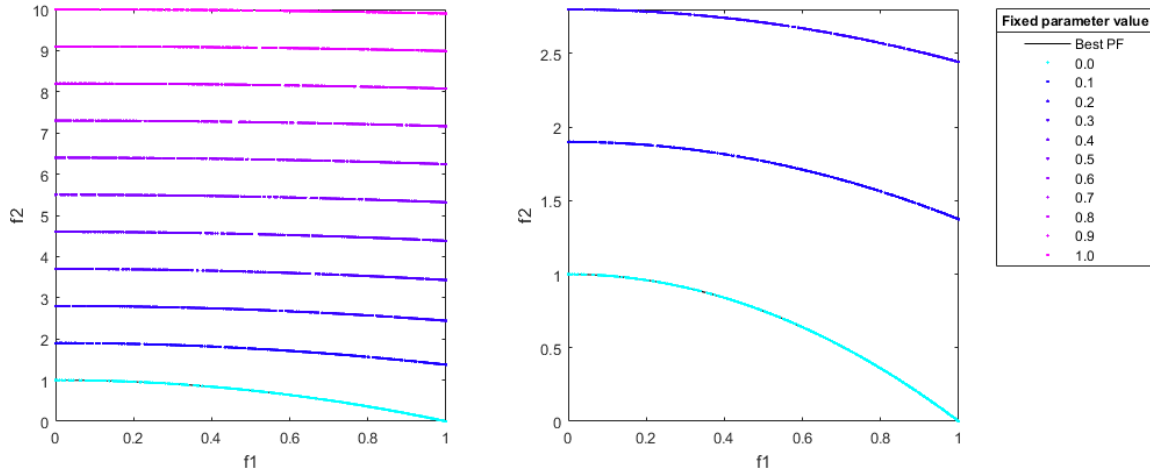


Figure 6.7: Surface Responses for different values of *secondary* factors fixed parameter in ZDT2 test problem. Left: all surfaces. Right: zoom over the best surfaces.

Then it was necessary to use a strategy to tackle this problem and improve the performance in this different scenarios. A first idea let the *secondary* variables behave randomly, however this setting lead to even worse result: in this way one loses the little control on those non-modelled variable; moreover, introducing large variability into the output data by these secondary variables, since the fitting phase does not involve at all these factors, this variability would be spread over the surface variables and hence augments the fitting error in most of the cases. The second idea was much more successful and in line with the literature. It consists in covering the entire domain of secondary variables changing the fixed factor at each surface building, hence generating various surfaces, as those in figures 6.7 and 6.8: starting the fixed parameter value from the lower bound of each non-modelled variable and augmenting it at each iteration by a set quantity defined from the user and by the delta between its upper and lower bound (e.g. in the figures this quantity is equal to 0.1 of the entire domain), one finds out several surfaces and hence the relative Pareto Front. Analysing their positions, it is possible to identify which fixed value performs better than the others and so focus around that value to further investigate, iterating the procedure and decreasing the size of fixed parameter admitted. For example, in the three objective test this procedure

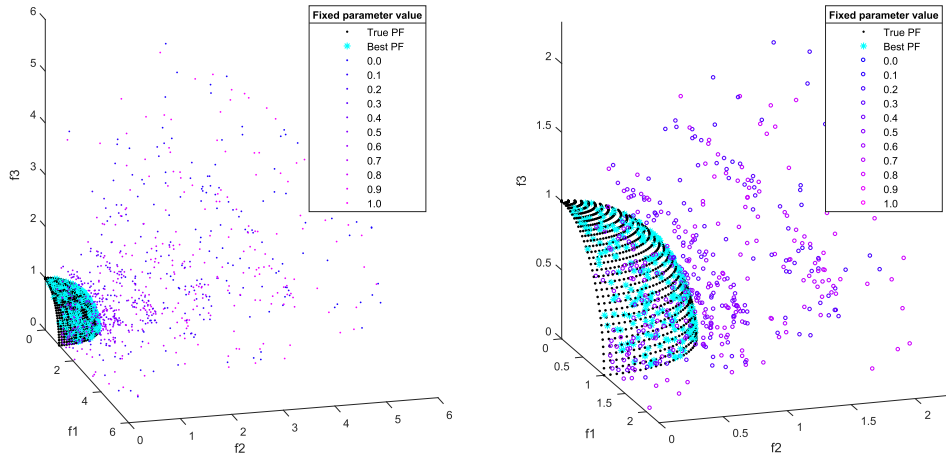


Figure 6.8: Surface Responses for different values of *secondary* factors fixed parameter in DTLZ2 test problem. Left: all surfaces. Right: zoom over the best surfaces.

lead to the best surface at fixed value 0.5, so a further analysis could involve values in a restricted interval as $[0.45, 0.55]$. Anyway, it is always necessary to observe carefully the results, because the features of the surface could realize prediction values outside the co-domain of the original function due to numerical approximation and the errors of fitting procedure. It could also occur that the Pareto Front of the response surface overcome the true one, but this is again due to this causes and it should not be seen as an error; this also because often the real Pareto Front it is not even known, so the Response Surface could lead to cheap but useful information on where to search more accuracy or with different model.

Before realizing the fit of the β parameters it is necessary to translate the entire sample from the natural reference frame, in which it has been built, to the global reference frame. This procedure let display all the successive results in the original reference, in which will be calculated also the surface parameter. Even if this could seem quite simple, obvious and needless, it helps making the final results easier to read and compare, dealing with other kind of surrogate models or optimization procedures.

Finally, built the surface sample and the test function sample and once calculated the output response, the β coefficient get evaluated with the least square method. The resulting vector values need to be associated with each relative coefficient of first, second and third order term and known term. This can be done value by value, or building the matrices of each order, which can be easier used later in evaluating the surface on a given point; dealing with the matrices of second and third order (since first order matrix results to be a column vector), it is necessary to highlight that elements outside the diagonal are coupled and hence need to be modified before entering in the matrix.

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{bmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \beta_{11}x_1^2 + 2\beta_{12}x_1x_2 + \beta_{22}x_2^2 \quad (6.1)$$

Thus in the second order matrix, entries outside the diagonal coefficients as β_{12} and β_{21} lead to the same contribute, hence they need to be divided by two. Similarly,

in the third order matrix, which is a 3-dimensional matrix, entries which characterize second order self-interactions (e.g. β_{112}) will be divided by 3, since appears three times, meanwhile terms which deal with mixed interactions (e.g. β_{123}), need to be divided by 6; instead, entries of pure self-interaction (e.g. β_{111}) appear only once in the main diagonal.

Contrary to the sensitivity analysis, the coding relative of Response Surface methodology has been entirely developed in all its routines and functions, with the only exception of the quasi-random Sobol' sequence used to perform the random sampling. The main feature that this code achieves and that differentiates it from the literature ones is the treating of non-modelled variables; it does not enhance the surface fitting but let the surrogate model reach much better optimization results in certain cases, as described later on in chapter 7.

6.3 Genetic algorithm application

As stated in the introduction of this chapter, the choice of the evolutionary algorithm performing the optimum searching process on the response surface fell upon the second version of *Non-dominated Sorting Genetic Algorithm (NSGA-II, 3.2)*, hence involving elitism. This choice was done due to the huge simplicity achieved through the previous phase of the model: response surfaces are often described by few decision variables and do not requires specific features, so the optimum search become much easier than original functions and does not involve particular problems. *NSGA-II* displays good behave on various kind of functions and moreover it is recognized as a standard approach in the scientific community.

The parameter settings for the algorithm are quite similar to those realized in [7, 62], excluding the coefficients of recombination, mutation and so on, which have not been modified from those set in the chosen algorithm. Population size is set to 100 individuals and the maximum number of generations depends problem by problem and it is reported in the table 6.5 below. The function tolerance, which checks the objective function values through consecutive generation, to stop the algorithm when it does not realize further improvements, is set to 10^{-4} .

The method built often gets to the Pareto Front before reaching the maximum number of generations and hence stops due to function tolerance; this fact further highlights the model capabilities, which manage to build an extremely simple surrogate, though still effective and suitable to the optimum searching task.

Dealing with the implementation of this last step of the model, it was realized in *MATLAB*[®] and it uses either developed and already written routines. The main code, which is the Genetic Algorithm *NSGA-II*, was already implemented in the software and did not need any further development. Anyway, its code had been carefully studied to verify the described properties and analogies with the descriptive section of *NSGA* and elitism 3.2, as well as to get how to define population size and the other parameters. Moreover, it was necessary to let *MATLAB* software read the Python's results: it required the translation of stored outcomes, which were the β coefficients. Once defined them, the surface function has been built, independently on the number of factors involved, number of objective functions and surface order; the chosen method to organize the β values was the matrix one, because it realizes the simplest possible

Problem	Max. number of generations
ZDT1	20
ZDT2	20
ZDT3	40
ZDT4	40
ZDT6	20
DTLZ1	100
DTLZ2	70
DTLZ3	80
DTLZ5	80
DTLZ6	50
DTLZ7	100

Table 6.5: Maximum number of generations for each test problem.

behaviour related with the input vector. To perform the successive metric analysis of the results and to retrieve the Pareto Fronts, it was used functions developed by authors of article [62]. Instead, to compare the obtained results, it was available the datasets resulting again from the above article; once retrieved the useful data, it was developed the routines realizing several analyses, as the information to build the box-plot and further more.

Chapter 7

Result and comparison on Test Function

The analysis of results on test functions reported in A is described in the following. First thing to notice and already mentioned, both the two- and three-objective problems present a peculiar behaviour dealing with variables from the second and third one respectively. In fact these problems reach the Pareto front when all the so-called *secondary* variables assume the same values. These are 0 in most of the two-objective test functions and 0.5, the mid-point of the domain, for three-objective tests. With the response surface built, the optima conditions can be reached quite effectively, obtaining good fits of the functions at lower costs. Therefore, these test problems do not evaluate completely the performance of the surrogate model realized. Another analysis of the literature to search for further test functions has been done, though it does not find any better problem. Several tests were found grouped in the *PlatEMO* package for MATLAB [60]¹. However, none of those proved to provide different characteristics and specific behaviours with the Response Surface Methodology developed. So, only the aforementioned problems had been used to test the realized surrogate, despite the highlighted issues.

Using these functions it is possible to study the behaviour of the response surface submitted to the evolutionary search process of the genetic algorithm. But to further investigate the true fitting qualities of this method, in the following chapter 8 an analysis which deals with a real dataset will be run.

Dealing with results on test functions, let's describe step by step all the choice taken, from the sensitivity analysis to the genetic algorithm.

7.1 Sensitivity analysis results

Sensitivity analysis, in particular Variance-based method, highlights that almost the entire variability of the test functions is produced by the first decision variable alone in two-objective problems and by the combination of first and second decision variables in three-objective problems. Moreover, most of the first test functions for two objective problems are defined involving just one variable, as in A.1. So their expressions often can be neglected in the sensitivity analysis due to their obvious results. Instead, three-

¹*PlatEMO* is an open source platform developed in MATLAB including several evolutionary multi-objective algorithms and test problems [2]

objective tests present functions which often involve several factors contemporaneously, but in DTLZ7 A.2.

ZDT3			DTLZ2		
Variable	S_1	S_{Tot}	Variable	S_1	S_{Tot}
x_1	0.9109	0.9050	x_1	0.4549	0.5775
x_2	0.0007	0.0011	x_2	0.4245	0.5492
x_3	0.0011	0.0011	x_3	-0.0015	0.0022
x_4	0.0011	0.0011	x_4	-0.0046	0.0022
x_5	0.0011	0.0011	x_5	-0.0014	0.0022
x_6	0.0010	0.0011	x_6	0.0011	0.0024
x_7	0.0012	0.0011	x_7	-0.0059	0.0022
x_8	0.0011	0.0011	x_8	-0.0004	0.0023
x_9	0.0009	0.0011	x_9	-0.0013	0.0026
x_{10}	0.0004	0.0011	x_{10}	0.0041	0.0022
...

Table 7.1: Results of Sobol' sensitivity analysis for second objective function of ZDT3 and first one of DTLZ2

Observing table 7.1, it is possible to notice this behaviour observing the variables from the second one and third one in the respective test problems. These factors do not introduce in the respective functions any influential variability compared to the previous variables.

Taking advantage of these results in building the response surfaces, they lead to realize surrogate models which are described by just one and two factors respectively for two and three objective test functions. In fact, attempts in realizing the surfaces with larger or lower number of factors produced models quite far from the original ones. Introduction of further variables in the surface model leads the most important factors to share part of their variability with other variables within the fitting procedure. Such models produce anyway quite good prediction. However they do not perform as well as those using only the main variable(s) found by sensitivity analysis. On the other hand, as it can be simply deduced, a smaller number of factors in the surface cannot properly describe the true behaviour of test function at all.

7.2 Response Surface configuration

Once identified the main factors defining the function at hand, one should decide the surface order. As explained in the previous chapter, in this choice sensitivity analysis plays again a central role: observing first and total sensitivity indexes, one can directly evaluate the contributes of interactions. So, without requiring second order indexes, which introduce high computational cost, it is quite simple to choose between a first order model or an upper order one. Later, to select between second or third order it had resulted more convenient to realize and to study the surfaces than performing second order Sobol' analysis. In fact the cost of realizing a surface parameter estimation is quite lower than those required by the sensitivity analysis. This until it does not get

much higher the involved order or the number of factors. Therefore, looking at the error estimation on a random generated sample, it is possible to discern which is the better choice between different order of surfaces. Dealing with test problems at hand, all of them had been modelled by third order surface, either using *3full* and *3simple* order, described in 6.2. In fact they require similar computational cost (little higher than a second order model), but they also provide very effective results, independently of the non-modelled fixed factor chosen.

Surface Order	ZDT2		ZDT4	
	f_1	f_2	f_1	f_2
First	4.100E-15	-2.344E-04	3.642E-15	-2.044E+00
Second	-2.726E-15	3.888E-15	-1.578E-15	1.267E+00
Third Simple	-1.579E-15	7.443E-14	-4.692E-15	-1.835E-01
Third Full	1.710E-14	1.383E-15	-1.801E-14	-3.727E-03

Table 7.2: Relative errors for response surfaces of different orders in two-objective test functions ZDT2 and ZDT4.

Surface Order	DTLZ2			DTLZ6		
	f_1	f_2	f_3	f_1	f_2	f_3
First	1.97E+00	1.97E+00	8.54E-02	4.57E-01	4.57E-01	2.32E-02
Second	-5.14E-01	-5.14E-01	-1.65E-02	-1.73E-01	-1.73E-01	-5.63E-03
Third Simple	-7.48E-02	-7.48E-02	-7.61E-04	-7.20E-02	-7.20E-02	-3.16E-04
Third Full	-1.04E-01	-1.04E-01	-5.22E-04	-1.02E-02	-1.02E-02	-4.37E-04

Table 7.3: Relative errors for response surfaces of different orders in three-objective test functions DTLZ2 and DTLZ6.

As it can be seen in tables 7.2 and 7.3 which report the fitting errors on tests functions, these last are best fitted by both third order simple and full models. Both f_1 functions of two-objective tests present for example a larger error in third order full model. It happens because the full model involves third order self-interactions, which are not present in the simple one. However the test function just involves the simple x_1 variable alone, in fact the first order model in this case provides the best performance. Anyway, all the errors in f_1 functions set around the machine error and these predictions can be considered almost exact.

Dealing with errors on test functions, the problem DTLZ4 A.2 has been omitted from any further analysis. This problem introduces a parameter $\alpha = 100$ in the objective functions as exponent of the two main variables x_1 and x_2 . Since these factors enter in the test functions as arguments of sine and cosine, the parameter α leads to a huge variability of results for little different inputs. The response surface method is not able to reproduce such large discrepancy on results, generating completely erroneous predicting values. So the DTLZ4 problem has been rejected from the set of test functions. Anyway, it highlights a limit in the model built and in particular in its fitting ability.

7.3 Final results

The objective of this work is to build up a meta-heuristic model to perform the optimization process, then the final step consists in evaluating its overall performances. Due to the problem at hand, the optimization deals with locating the Pareto front through the searching features of *NSGA-II*, exploiting the Response Surface previously built.

In this section will be reported just 2 example of the data collected during the analysis, but all the plots of results can be found in the appendix B. Instead, a complete comment of all the test functions will be done, dealing with the pros and cons of the developed method.

Since this model realizes an optimization process, to evaluate its qualities it is necessary to compare the resulting data against other methods. The algorithms chosen as comparison are those used in article [62], algorithms which work also on the same test problems. Before the presentation of results, let's give a brief description of the other meta-heuristic models involved:

ASEMOO: it is based on evolutionary genetic optimization [29, 32]. It initializes a set of starting solutions using a Latin Hypercube Sampling, which gets completely evaluated by objective function. Successively, a Kriging model based on this set is realized and it is then used to perform evolutionary optimization. Then, when more points are needed, the algorithm evaluates in the original function those just found by the optimization of the Kriging model and it adds them to the database. Finally the Kriging model is once again created from the new dataset and the process continues in loop.

GeDEA-II: it is a multi-objective real-coded evolutionary algorithm, developed in [7] as an upgrade of GeDEA [61]. It is based on basic steps of evolutionary strategy, but it takes advantage of genetic diversity between individuals to improve later generation fitness values. GeDEA-II exploits simplex-crossover, shrink mutation and other typical characteristic of evolutionary algorithm, which combined together work extremely well.

GeDEA-II-K: this is a further development of GeDEA-II, realized in [62]. It combines the previous two algorithms, ASEMOO and GeDEA-II, matching Kriging filter and genetic diversity. The resulting algorithm outperforms on all test functions the other two, obtaining towering features, however at a higher computational cost.

ParEGO²: it is again an evolutionary genetic optimization, developed in [37]. It does not take advantage of any surrogate model, so it evaluates directly the objective function.

Article [62], which uses these models, introduces the *Adimensional Direct Evaluation Number* (*ADEN*). As the name suggests, it limits the number of direct evaluation of the test functions and it is defined as the ratio between the direct evaluation number and

²This algorithm is described and reported to the completeness of the analysis. Despite its comparison with the other algorithms is always performed, just few pictures show its results. This happens due to the fact that it worsen the clearness of the other results.

the design space dimension. So, given a model with k -decision variables, the maximum number of direct evaluation will be equal to $k \cdot ADEN$.

Also the data collected to perform comparison come from authors of article [62]. However, the tests performed in the cited article were based on different numbers of decision variables and $ADEN$. On the other hand, optimization using the response surface does not require any direct evaluation of the test function. In fact, once the surface is built, the original function does not appear any more. Indeed, during the fitting procedure also the surface requires the evaluation of sample points obviously. So the sample set has been bounded in size to 400 points, letting the users to decide if there will be used the maximum number of point generated or less. The bound has been set to this value because 400 is also the maximum direct evaluation number in the tests of the article. There data were recorded at $ADEN$ equals to 6 and 10 and the design space dimension was set to 6, 25, 40. Finally, the data selected to be compared with response surface results are the best available for each meta-heuristic model and in each test problem.

7.3.1 Analysis of two-objective test functions

The problem reported here is ZDT1. The response surface shows a good accuracy in fitting the true Pareto front, however it cannot overlap this last, as ASEM00, GeDEA-II and Ge-DEA-II-K manage to do. Comparing in fig. 7.1 all the runs and the single run (respectively plots (a) and (b)), it is possible to highlight that each single run stands on the same line and does not vary a lot, as it happens for the previous three algorithms and not for the ParEGO. So the variability of results is pretty limited, as it can be seen both from the box-plots in fig. 7.2 and from the relative data in tab. 7.4 of normalized Hyper-Volume and D-metric. The D-metric box-plots show also that all the algorithms but the Response Surface have a little variability on results. This fact can be likely associated to the evolution searching process. Response surface is quite a deterministic algorithm and moreover in two-objective test functions the model involves just one variable to better fit the data. So its results cannot vary that much. Meanwhile, the other algorithms are much more based on the genetic optimization, which can produce quite different results at each run. This could lead a Pareto front always with a good Hyper-Volume value, but which presents also some element far from the true front. So its presence would make the D-metric change more than the HV at each run. In this case such behaviour is not evident because the best three algorithms always reach the true Pareto front without issues.

Let's now give the results description for the other two-objective test functions, whose plots and data can be found in appendix B.

ZDT2 It presents a much higher accuracy of the results for the response surface method, as it can be seen in fig. B.1. This statement is even strengthened by the box-plots in fig. B.2. It shows that Hyper-Volume of the response surface as the other best ones approaches 1, while the D-metric measure is the lowest and so the best one, since it describes the overall proximity to the true Pareto front.

ZDT3 Conversely to the previous test, this problem shows a higher variability of the Pareto front. The response surface defined by a single variable cannot reproduce faithfully this behaviour, but neither a response surface with more factors

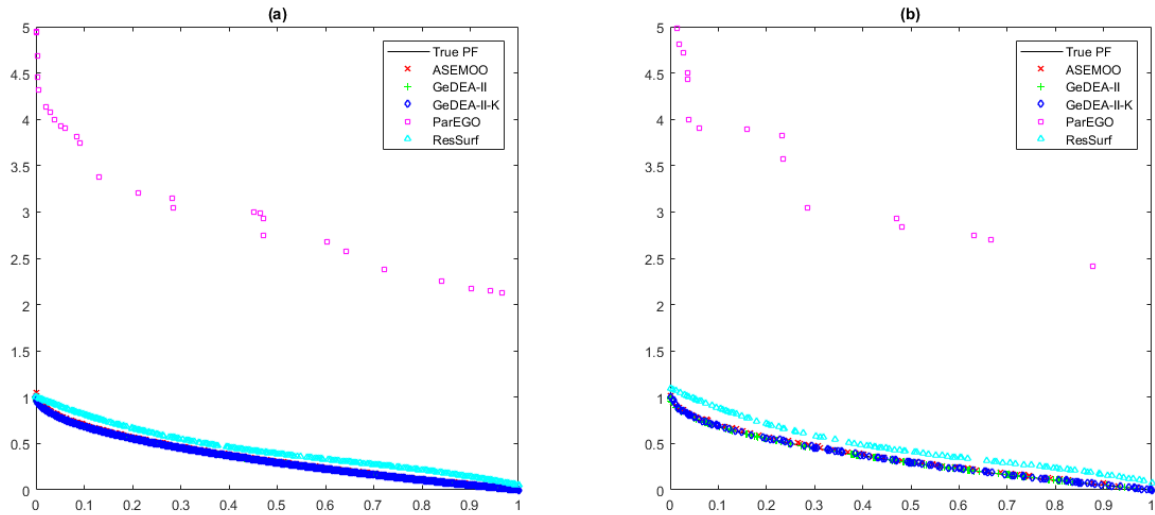


Figure 7.1: Test function ZDT1: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

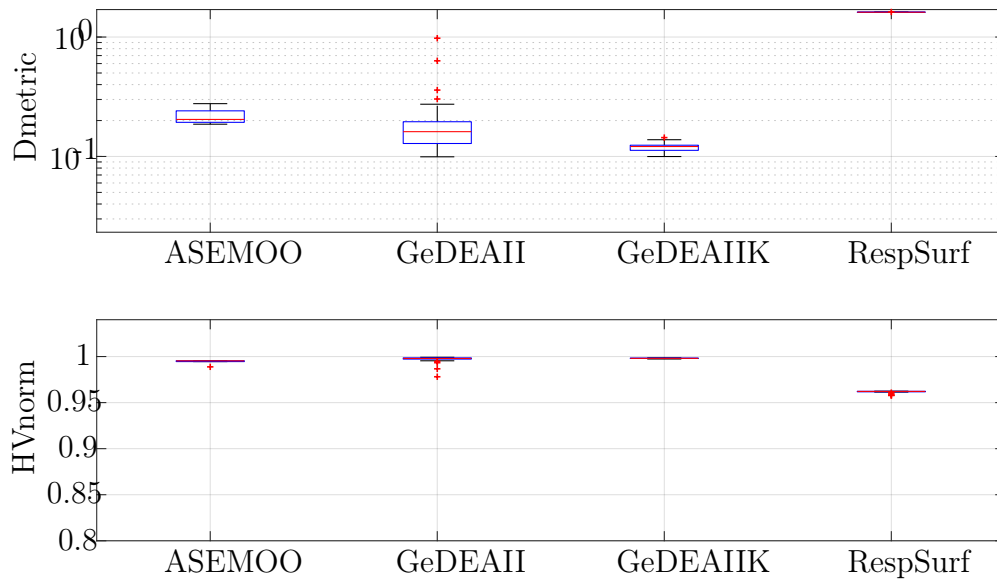


Figure 7.2: Test function ZDT1: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1,4)).

does. The figure B.1 and data reported displays the best parameter configuration possible, as it happens in the other cases. Even if comparison with the best algorithms shows a pretty worse result, the response surface performs anyway quite better than the ParEGO and it is quite near to the true Pareto Front. This fact is measured also in the normalized Hyper-Volume box-plot B.2, while dealing with D-metric it is evident the better results obtained by both ASEM00 and GeDEA-II-K.

ZDT4 In this successive problem there are reported two response surface curves in figure B.5. The best one (the lower, in yellow) refers to response surface with

	D metric				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.2177	0.2113	0.1197	34.7358	1.6148
Median	0.2043	0.1613	0.1212	35.3521	1.6140
Perc. 25%	0.1932	0.1284	0.1126	32.9628	1.6135
Perc. 75%	0.2410	0.1954	0.1243	36.2711	1.6157
Whisker low	0.1863	0.0994	0.0999	31.0943	1.6128
Whisker up	0.2769	0.9776	0.1439	38.4544	1.6224

	HV normalized				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.9942	0.9967	0.9981	0.2274	0.9616
Median	0.9953	0.9979	0.9980	0.2268	0.9622
Perc. 25%	0.9945	0.9972	0.9979	0.2196	0.9617
Perc. 75%	0.9955	0.9986	0.9983	0.2341	0.9623
Whisker low	0.9888	0.9781	0.9973	0.2036	0.9578
Whisker up	0.9955	0.9992	0.9986	0.2496	0.9624

Table 7.4: Test function ZDT1: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (1, 4)).

fixed parameter value = 0.5, while in the other one it takes value 0.45, as the legend reports. Dealing with data in the ZDT4 box-plot and table, they refer only to the best curve. This comparison it is done because the best curve outperforms the true Pareto front, due to fitting error. These are not due to the fitting data set, but likely due to the fact that only one factor defines the surface and so it cannot reproduce very well the true Pareto front behaviour. On the other hand, the second response surface curve shows how much far it does place with such a little variation of the *fixed parameter value*.

Before the analysis of box-plot data, the set of points performing better than the true Pareto front has been handled. They were modified to realize a negative contribution to the Hyper-Volume. Dealing with the D-metric, this is not necessary since it always measures the discrepancy of points from the true Pareto front. Finally, observing the box-plot it is possible to see that both D-metric and HV are anyway very good. Notice that the reference point is placed far from the true Pareto front to display results also from the ParEGO algorithm. In fact also ASEMOO, whose results lay over the second response surface ones, has an HV measure almost equal to 1, as the best algorithms³.

ZDT6 In this last two-objective test function, response surface is able to retrieve just a little part of the Pareto front, again due to fitting problem, as reported in

³It is possible also to notice that the Pareto front plot B.5 is reported in a logarithmic y-scale to display all the results in a proper way. Otherwise, it would not be possible to observe clearly that response surface performs better than the true Pareto front. However, even if there are no other plots in logarithmic scale and they cannot highlight this particular behaviour, data show that this behaviour does not happen in other test functions.

figure B.7. However, it obtains results that are exactly on the true Pareto front. Though on the other hand, due to the little front build, the performances of D-metric and normalized HV are quite low. Even if HV box-plot marks response surface as the worst model, the D-metric shows that it does not perform so bad, again because of the accurate result of the ending part of the true Pareto front.

7.3.2 Analysis of three-objective test functions

Let's first deal with the test problem DTLZ2, whose plots and data are here reported. Observing fig. 7.3 it is straight evident that summing all runs of response surface method, it realizes a good covering of the true Pareto front. Although, also on the single run the algorithm performs well and it retrieves several points on the front. Since all the test methods obtain good results on this test problem⁴, the reference point for evaluating the Hyper-Volume measure can be set in proximity of the true Pareto front, so it can highlight also small variation of the HV values. In fact, analysing the box-plots 7.4, it is possible to observe that GeDEA-II-K, Response Surface and GeDEA-II retrieve similar results, but the first one almost sets itself to $HV_{norm} = 1$ in all the runs. In the meanwhile the other two display little variations, anyway they reach the top value. Instead, while ASEM00 performs little worse, ParEGO places its HV value much far from the other. Dealing now with the D-metric, the response surface performs almost as well as GeDEA-II-K. Such result means that response surface locates its solution very much near to the true Pareto front, but it cannot cover the front with the same quality. Anyway this performance shows that Response Surface method retrieves very satisfactory results and, as it will be seen later, these are even better in three-objective test functions than in the two-objective ones.

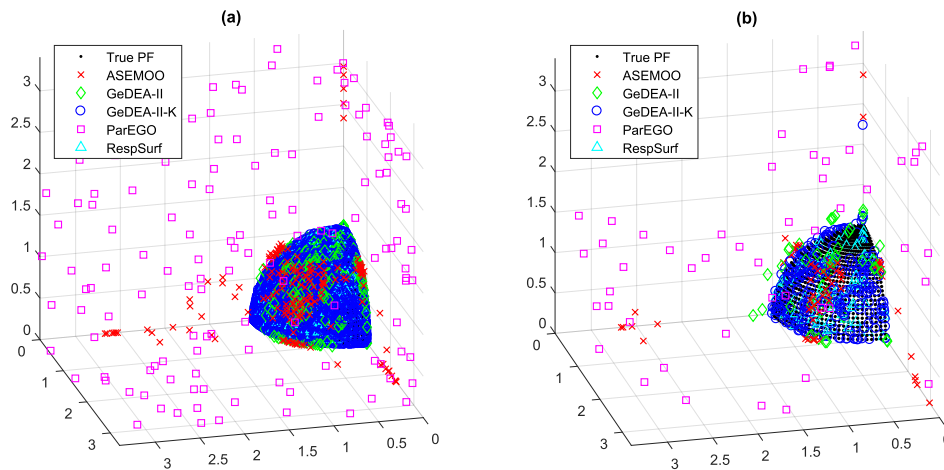


Figure 7.3: Test function DTLZ2: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

Finally, let's analyse the results for all the other three-objective tests:

DTLZ1 The algorithms tested on this problem are just the ASEM00 and the GeDEA-II, because it was used only in [7]. Here as in almost all the three-objective prob-

⁴Compared to the previous ZDT4 two-objective test function, but in particular compared to most three-objective ones

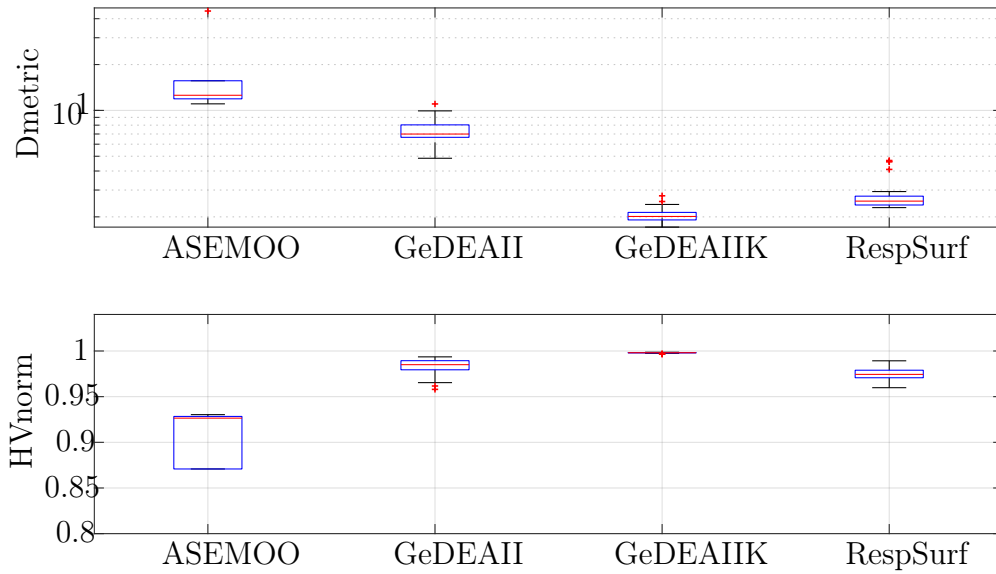


Figure 7.4: Test function DTLZ2: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at $(3, 3, 3)$).

	D metric				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	18.111	7.3972	2.0632	77.606	2.7919
Median	12.570	6.9942	2.0120	77.847	2.5374
Perc. 25%	11.912	6.6569	1.9096	75.472	2.3927
Perc. 75%	15.653	8.0356	2.1394	79.962	2.7349
Whisker low	11.042	4.8505	1.7129	69.871	2.2997
Whisker up	44.921	11.0240	2.7500	83.623	4.6883

	HV normalized				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.8742	0.9827	0.9980	0.2087	0.9745
Median	0.9265	0.9850	0.9981	0.2034	0.9744
Perc. 25%	0.8708	0.9794	0.9979	0.1930	0.9707
Perc. 75%	0.9284	0.9894	0.9983	0.2142	0.9790
Whisker low	0.6627	0.9580	0.9965	0.1735	0.9598
Whisker up	0.9303	0.9936	0.9986	0.2560	0.9893

Table 7.5: Test function DTLZ2: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at $(3, 3, 3)$).

lems, response surface method performs remarkably well. As figure B.9 reports, ASEMoo obtains results very far from the Pareto front and not even in all its runs (single run figure does not report any ASEMoo point). GeDEA-II instead retrieves better solutions but these are still not that good. As it can be seen in

fig. B.10, which shows a zoom in the neighbourhood of the true Pareto front, GeDEA-II obtains just one point in its proximity. On the other hand response surface completely covers it really well. Analysing the performance of the various algorithms through box-plot data fig. B.11 and tab. B.5, both GeDEA-II and Response Surface show high values of the HV, but the last one gets a much better result on the D-metric. The same values of Hyper-Volume for the former algorithms occur due to the fact that the reference point is set far from the true Pareto front, to obtain non-null data for the ASEMoo box-plot. Although it is clear just from the figures the better results of response surface against GeDEA-II.

DTLZ3 This problem shows a similar frame to the former one. ASEMoo and ParEGO algorithms obtain results very far from the true Pareto front, while GeDEA-II and GeDEA-II-K manage to get better prediction, but still they cannot achieve the true front. Once again, response surface reaches true Pareto and it fills quite well the front. These behaviour are plain from the box-plots figure B.14 and data. Not only the response surface reaches the normalized Hyper-Volume best value 1, but it shows also lower level of the D-metric (much lower than GeDEA-II and GeDEA-II-K). Finally, the behave in the HV box-plots occur once again due to the far located reference point.

DTLZ5 Here the response surface method fits well the true Pareto front and this is plain from both picture B.15 and its zoom B.16. However, the evaluation of HV is once again altered by the low performing algorithms ASEMoo and ParEGO, which require a far reference point to appear in the box-plot picture. Anyway, observing the D-metric which reports an unbiased analysis, this time the response surface is outperformed by the GeDEA-II-K. It reaches the best accuracy in fitting the true Pareto front, while GeDEA-II and response surface settle more or less on the same level.

DTLZ6 The same behaviour observed in the former problem repeats itself similarly here. GeDEA-II-K gives the best fitting of the true Pareto front, followed by Response Surface and then GeDEA-II. However, in this case the other two algorithms reach results much nearer to the true front as reported in figure B.18. So the reference point can be set quite nearer than the previous case. Hyper-Volume measures highlight anyway that the former three algorithms reach $HV = 1$, so it means that all of them give very good results on the test function. D-metric instead proves again that GeDEA-II-K results realize better performance, this time followed by Response Surface and then by GeDEA-II.

DTLZ7 Finally, tests on this last problem give results not that satisfying. While GeDEA-II-K covers completely the true Pareto front and little less does GeDEA-II, Response Surface sets its results just above the front but it cannot follow exactly its behaviour, as reported in fig. B.20. However, it sets itself more or less near to the Pareto front as the ASEMoo, which in turn performs better than ParEGO. This is the exact classification displayed in both D-metric and normalized HV box-plot B.21. Anyway also in this case the response surface shows quite good performance.

Chapter 8

Analysis of the Response Surface on an experimental dataset

This test case was realized to measure the abilities of the response surface in fitting a generic source of data. The dataset on which this analysis is based represents a computer based experiment and it is realized using a computational fluid dynamics software.

The available dataset deals with the baseline geometry of a subsonic airfoil. The profile can be divided in two regions: a morphing part and a fixed one, as figure 8.1 displays. The morphing part is the object in analysis and its profile is parametrized with a B-spline with Constant Arc Length technique. Its upper and lower sides are separately parametrized by cubic splines, each one with 6 control points, with non-periodic uniform knot sequence. The arc length is kept constant by a procedure controlling the movement of the parameter points. Fixing the arc length allows to introduce a constrain on the deformation. The decision variables of this problem are the y coordinate of control points (CP) 3 and 4 in the upper part ($y3Up$, $y4Up$), 1 and 2 in the lower part ($y1Low$, $y2Low$) and both x and y coordinates of the leading-edge control point (xLE , yLE), whose keeps the arc length constant. The other points reported in fig. 8.2 are used just as control points.

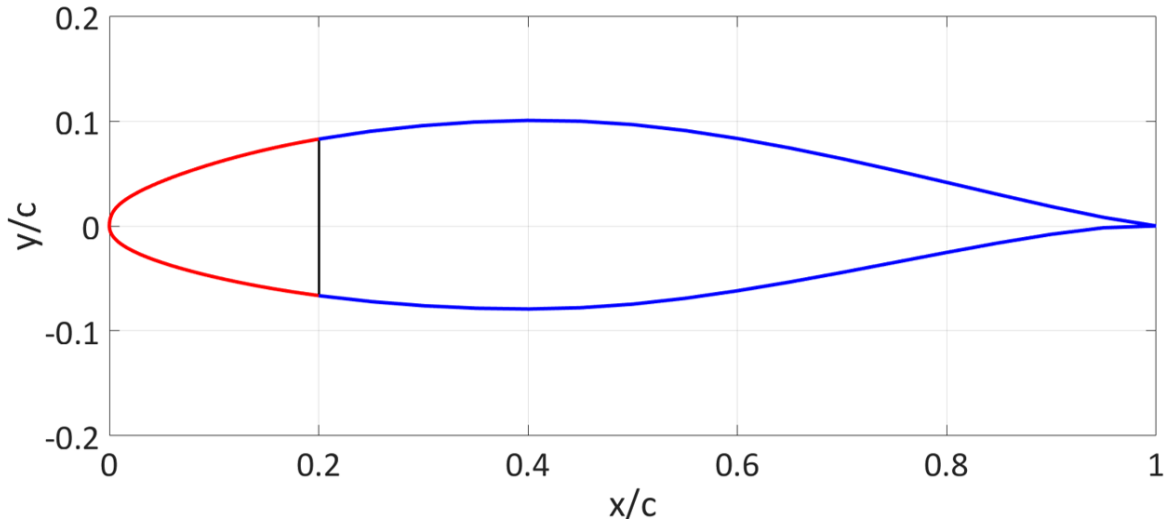


Figure 8.1: Airfoil profile divided in the morphing and fixed region. Deformable part sets in $[0, 0.20]$ of the chord.

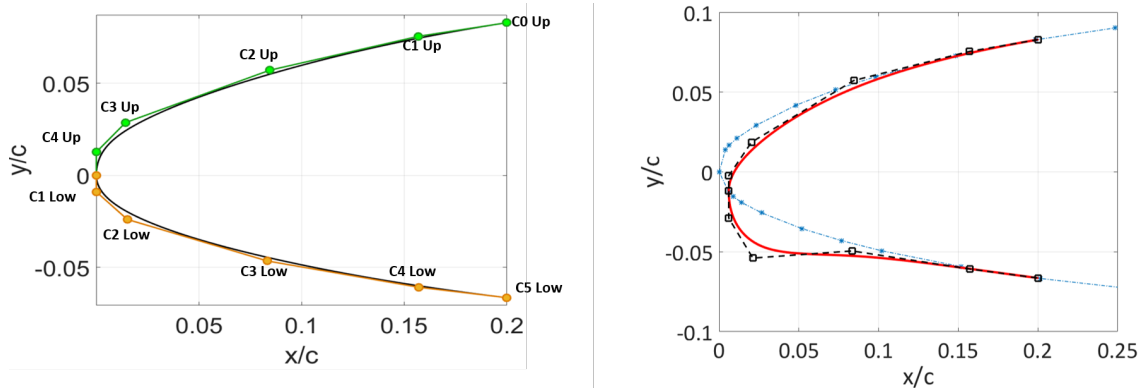


Figure 8.2: Morphing part of the airfoil. Left: profile parameters and their location. Right: example of the spline fitting of the profile parameters.

The objective functions are the lift and drag coefficients (c_L and c_D) of the profile. Objective of the original optimization task is to minimize the drag and maximize the lift.

In building the response surface model it was introduced the *secondary variables fixed factor*. It sets the values of all the non-modelled variables to better evaluate the output of the sample set. It is useless in such frameworks, since the data was retrieved previously and for a different purpose. So all the sample points are already and completely defined. In fact, to take advantage of the surface model implementation realized, it would be necessary to perform previously a designed experiment to build the sample set. Moreover, it is not enough to collect data separating factors in modelled and non-modelled variables and successively to perform the sample. To correctly define the chosen variables generating the surface, it would be necessary first of all to perform a sensitivity analysis to highlight which variables realize the main contributes. Anyway, dealing with the data at hand, the entire sample and its output were already defined and so none of the features of the design of experiment above described can be done. Also the sensitivity analysis cannot be performed, because data do not present in the desired shape.

Let's now analyse two different methods which evaluate the performances of the response surface on the available dataset. Due to the fact that it is not possible to perform sensitivity analysis, when building the response surface it is not evident which are the best variables to use. So the result will show the best combination of variables found between all the possible ones to model the problem.

Anyway, to perform all the following analysis a brief modification of the routine of the response surface was necessary. Here it is no more possible to define the values of the *secondary* variables, since all the points are already defined with their relative lift and drag outputs values.

8.1 Fitting and testing sets

This way of testing is partly borrowed from Artificial Neural Network fitting procedure, described previously in 4.3. Here the database is simply divided into two subsets: a set of sample points, which provides the information to perform the fit, and a testing set. This last is used once the fitting phase is concluded and its objective is to measure the quality of the response surface just generated.

The testing set size is kept varying between 10% and 20% the size of the starting database. At each run of the algorithm, one by one points are randomly selected from the original database and moved to the testing set. In this way, no repetition of the points can occur and once the procedure ends both fitting and testing sets are available.

The following phase consists in the examination of all the possible response surface configuration. In fact, here the final user can choose both the variables defining the surface and its total order. To analyse in an exhaustive way the surface performance, all the possible combinations of parameters need to be tested.

Here are reported just few box-plots of the test results. Further figures for this way of testing and also for the following two test models can be found in appendix C.

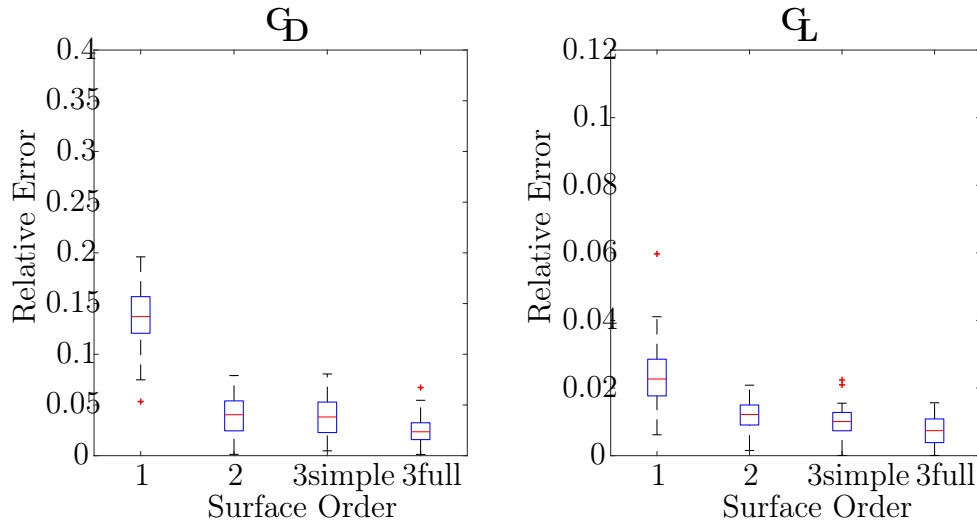


Figure 8.3: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by 5 variables: $y3Up$, $y4Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

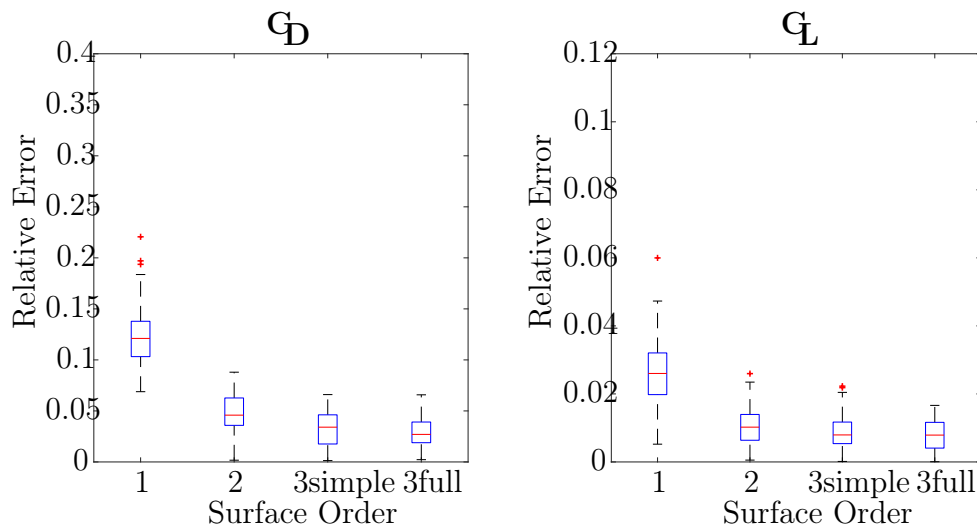


Figure 8.4: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by 6 variables: $y3Up$, $y4Up$, $y1Low$, $y2Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

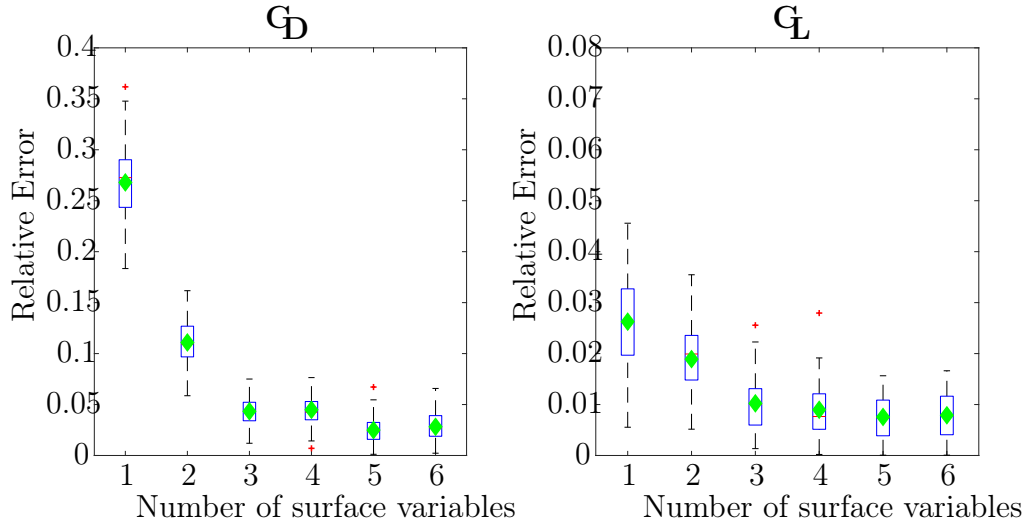


Figure 8.5: Mean error and box-plot comparison on response surfaces built by different number of variables by fit&test model on full third order. Mean error values are displayed by the green diamond. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

From the graphs it is evident that the surface quality fitting is quite good. Lift coefficients resulting from response surfaces are almost always acceptable in all the tests. On the other hand, predictions on drag can be considered satisfying for response surfaces of second and upper order and involving at least 2-3 variables. Dealing with the width of the box-plots and excluding results of the 1-variable response surfaces, predictions are not much spread and different from run to run.

Finally let's observe the comparison of *full third order* response surfaces generated by different number of variables. It can be seen how using from three to six variables does not enhance a lot the prediction performances. Instead, surfaces described by 5 variables shows the best behave with less spread results, in particular in Drag coefficient. This last seems to have a much harder behaviour to replicate.

8.2 One point excluded test

In this second testing method, performance are evaluated just in one point per surface building. Here at each algorithm run, from the original database a point gets excluded and the response surface is built from the new fitting set. Once defined the β coefficients, the response surface is evaluated on the excluded point and performance information are retrieved.

A further test, which can be run in this framework, deals with the building a local response surface. Once selected the excluded point, it is possible to filter the original database and consider a new sample set containing only points in the neighbourhood of the excluded one. To properly define the new sample set, some parameters are fixed: the neighbourhood set needs to contain at least 100 sample points independently on the surface order. Considered the total spread of the domain, a generic point in the database $\mathbf{x}_i = (x_{i1}, \dots, x_{i6})$ is a neighbour of the excluded point $\mathbf{x}_E = (x_{E1}, \dots, x_{E6})$ if and only if it falls inside a ball centred in the excluded point and with a diameter length equal to $p_1 = 15\%$ the entire domain in each direction. If such set does not

contain enough points, the diameter is enlarged to $p_2 = 25\%$ of the entire domain.

$$\mathbf{x}_i \in nbb_{\mathbf{x}_E} \iff \begin{cases} \mathbf{x}_i \in B_{\mathbf{x}_E} \left(\frac{p_1}{2} (\mathbf{U} - \mathbf{L})^1 \right) & \text{if } |B_{\mathbf{x}_E}(r)| \geq 100 \\ \mathbf{x}_i \in B_{\mathbf{x}_E} \left(\frac{p_1}{2} (\mathbf{U} - \mathbf{L}) \right) & \text{otherwise} \end{cases}$$

Again, all the possible configurations of surface order and chosen variables need to be investigated to exhaustively evaluate the model performance.

8.2.1 Global response surface result

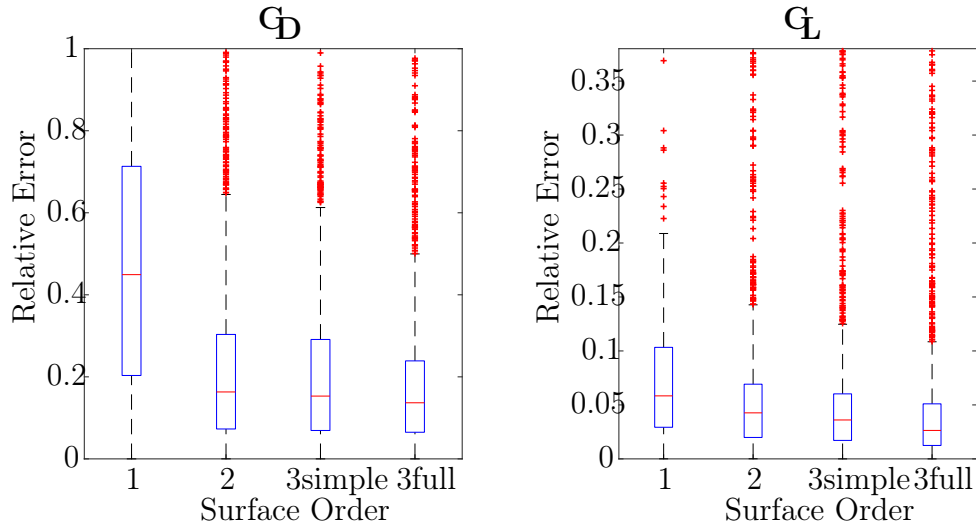


Figure 8.6: Box-plots of relative errors on global response surface defined by six variables: $y3Up$, $y4Up$, $y1Low$, $y2Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

Outcomes on global defined surfaces of this second method work on the entire dataset and collect error data from each single database point. Here it is evident the presence of more erroneous data, since each box-plot is a lot wider. These data realize larger influence on the mean error values. Anyway, the mean error is much higher than the median value, because box-plots exclude outliers from their evaluation process. So the resulting mean values obtained by *full third order* response surfaces with different number of variables are much higher than those obtained in the previous method. Although, taking into account the above considerations, it is possible to validate the obtained predictions. In fact the model itself it is way cheaper than many other that could be also used as first source of rough data.

8.2.2 Local response surface result

Local defined response surfaces show little better results both on Drag and Lift predicted values. Anyway, restricting the surface building on such a tiny region of the domain to obtain this small gain does not pay off the computational cost. However, it

¹ \mathbf{U} and \mathbf{L} are the vectorial forms of the upper and lower bounds. Hence the radius of the ball in the i -th direction is given by $\frac{p_1}{2} (U_i - L_i) \cdot 0.15$.

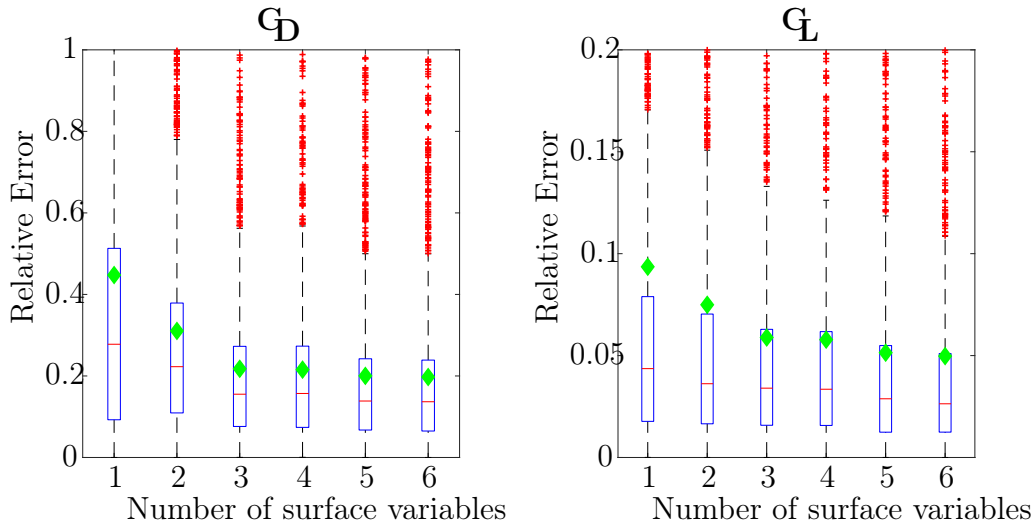


Figure 8.7: Mean error and box-plot comparison on globally defined response surfaces, built by different number of variables. Mean error values are displayed by the green diamond. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

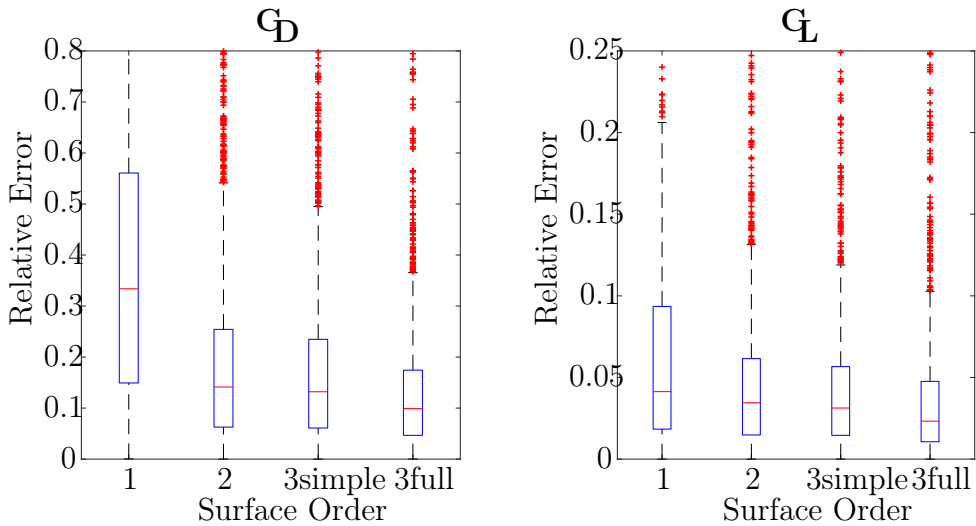


Figure 8.8: Box-plots of relative errors on local response surface defined by six variables: $y3Up$, $y4Up$, $y1Low$, $y2Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

could happen for different test functions (e. g. with larger variability) that function results from global to local response surfaces lead to valuable gains in prediction with a reduction of the test domain.

8.3 Brief conclusions on response surface methods

From the data collected and in particular thanks to the first method results it can be stated that Response Surface methodology performs quite well on a general dataset. Although, here it cannot be used the introduced method of the *fixed secondary variables factor* where best results are found for 5-variables surfaces, tests show acceptable errors for this method also in other configurations. Further development could involve higher

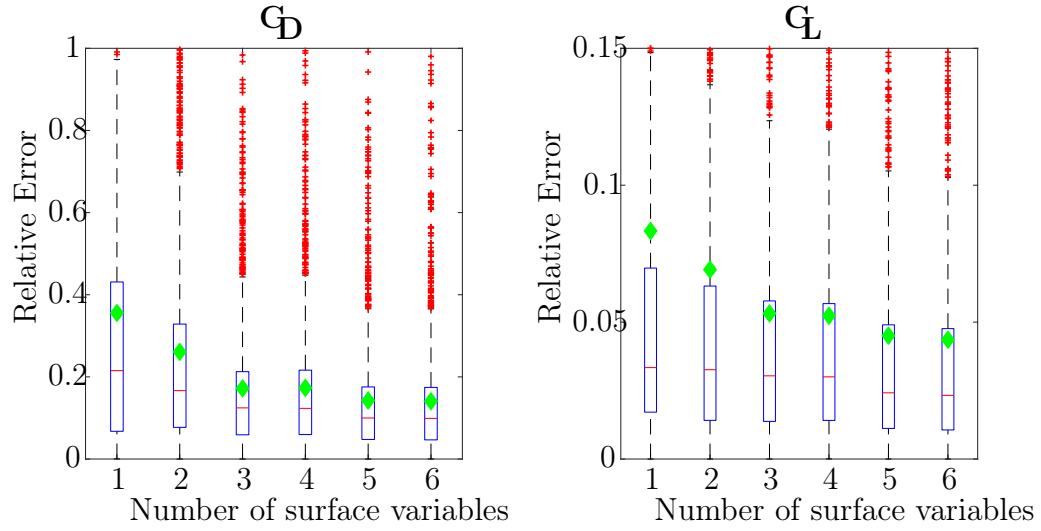


Figure 8.9: Mean error and box-plot comparison on local defined response surfaces, built by different number of variables. Mean error values are displayed by the green diamond. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

order fitting or different parameters estimation. However, any new implementation needs to realize better fitting capacity, since the method here presented is a good trade-off between prediction fidelity and computational cost.

Chapter 9

Conclusions

This study provides an insight into multi-objective optimization models. It describes some fundamental parts of this framework: starting from the general optimization concepts, it introduces the multi-objective treatment, which involves several other aspects as the *No free lunch theorem* and the *curse of dimensionality*. Following this, it describes the basic structures and aims of evolutionary optimization methods and surrogate models, reporting for both of them some examples and describing their development. Last phase of the theoretical part introduces the sensitivity analysis which seldom escorts a surrogate model. Successively it is described the implementation of the overall method, followed by the tests and the analysis of its results on multi-objective problems and on an experimental database. These last part works as a sort of validation phase for the model, that proves to be quite effective and robust.

Dealing with the mutual help of sensitivity analysis and response surface methodology, this coupling works really well and it provided useful results. Even if with the test functions at hand the main factors are evident, the sensitivity leads to analytical results. Following indications of this tool and just using the main variables highlighted to develop each response surface, the model realizes the best performance possible. In this way the response surface not only provides better fitting features than any other surface described by more or less variables, but it also obtains good optimization results.

Summing up the results obtained in the test problems, it is possible to state that the method shows satisfactory performance. In fact it manages to retrieve good solutions in almost all the tests and to provide a full coverage of the true Pareto front. Despite it does not work well on DTLZ4 problem due to its large variation, it generates accurate prediction for almost all the other functions. This fact is achieved also thanks to the *fixed parameter* which has been introduced once the first results were obtained and studied. It helps to build a response surface that partly considers also the *secondary variables*¹ which do not entry in the model and thus in the optimization task. Building several surfaces that test different configurations of the secondary factor enhances the possibility to later obtain a better result during the optimization. This particular approach has revealed helpful in retrieving solutions near the Pareto front in two-objective test functions. While in three-objective ones all the secondary variables fixed to their mid-domain values, as literature suggests, led to the Pareto front by luck at first, in most of other problems the results were far from the optimum with the

¹Recall previous chapter 4.4 and 6.2 to complete definition of the *secondary variables* and of the *fixed parameter*.

secondary variables set to their mid-domain values. Thanks to this *fixed parameter* it has been identified the set of values that allows the response surfaces of these problems to approach the true Pareto front. So finally, it has been developed an optimization method that retrieves really good results over all the test functions at hand and which seems to require also little computational cost.

The test phase on the experimental dataset gives positive outcomes as well. Despite the nature of the dataset, the fitting performance reached by the response surface is quite good. In fact the data were obtained before the method implementation and its sample points do not build up the typical dataset, which is defined to fit the response surface coefficients. Moreover, it was neither possible to take advantage of the *fixed parameter* to set the secondary variable. Neither the sensitivity analysis was performed to highlight the most important factors, once again due to the database nature. So, the problem has been tackled by testing all the possible configurations of response surfaces: changing its order, the number of model variables and their combinations. These tests retrieve satisfying results for the fitting abilities of the response surface. Hence, also this feature of the model is partly validated, even if further tests should be performed to completely analyse its behaviour in different framework.

The comparison with other optimization algorithms highlights once again that the performance of the developed model are really satisfying. Although its results sometimes are matched and exceeded by some of the other algorithms, these however are much more structured and computational expensive. This statement cannot be proved in this work, due to the lack of complete knowledge over the other tested algorithms. Just one of them is an open source algorithm, but it is also the one that realizes most of the time the worst results. On the other hand, only the data referring to the other algorithms were provided. So it had not been possible to perform a computational cost analysis, which would have lead to a more complete model validation. Anyway, observing the nature of the comparison algorithms, their building-block described in the relative literature results accurate but also computational expensive. Moreover, also other kind of multi-objective optimization test problems and database to study fitting abilities would have been necessary, to provide further results on different frameworks. However, all the features of the model had been tested and it proved to manage very well various situations.

In conclusion, this work leads to the construction of a new meta-heuristic surrogate assisted model to perform multi-objective optimization. Tests proved the effectiveness of the realized model. After an analysis also on a real application, it could be also used to perform cheap optimization on reality-based complex problem, before the search of accurate optimum solutions with other expensive codes. Moreover, always in combination with sensitivity analysis, it could be used to retrieve information on the problem at hand, through a complete investigation of the search space. However it is necessary to realize further analysis on this method to test its features on different kind of function. Last, the model could be implemented with higher order of the response surface, a different fitting procedure and other new features to enhance its abilities. Anyway as already stated, this method solves cheaply the optimization tasks and its performance is quite satisfying, compared to other test algorithms.

Appendix A

Test Functions

The problems here reported was first introduced by Zitzler, Deb, Thiele and Laumanns in [67] and [12]. These test function offer an easy visualization of the results, hence the Pareto Front, and involve multiple variables and objective functions. The authors aimed to develop problems which were easy to construct, scalable in the number of variables and also objectives, with easily understandable Pareto Front, either continuous or discrete, widely distributed.

A.1 Two-objectives test functions

This test problems were presented in [67] and their main features are the difficulties to converge to the optimal Pareto Front and to maintain the population diversity. Each function reported below is structured in the following manner, build by the basic functions f_1 , g , h :

$$\begin{aligned} \text{Minimize} \quad & \mathbf{F}(\mathbf{x}) = (f_1(x_1), f_2(\mathbf{x})) \\ \text{subject to} \quad & f_2(\mathbf{x}) = g(x_2, \dots, x_m)h(f_1(x_1), g(x_2, \dots, x_m)) \\ \text{where} \quad & \mathbf{x} = (x_1, x_2, \dots, x_m) \end{aligned} \quad (\text{A.1})$$

The first objective function f_1 is always defined just by the x_1 variable (moreover often $f_1 = x_1$), while g function of all the remaining $m - 1$ variables. The number of variables describing the problem was set to $m = 30$ in the original paper [67] but it is possible to enlarge or reduce arbitrarily the space dimension to complicate or simplify the problem.

ZDT1

Test function *ZDT1* has a convex Pareto optimal front:

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ g(\mathbf{x}) &= 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \quad (\text{A.2})$$

The original function has $m = 30$ and $x_i \in [0, 1] \forall i$, while the implemented version has $m = 100$. The Pareto optimal front is formed with $g(\mathbf{x}) = 1$.

ZDT2

Test function *ZDT2* is the non-convex counterpart to *ZDT1*:

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ g(\mathbf{x}) &= 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g} \right)^2 \end{aligned} \tag{A.3}$$

The original function has $m = 30$ and $x_i \in [0, 1] \forall i$, while the implemented version has $m = 100$. The Pareto optimal front is formed with $g(\mathbf{x}) = 1$.

ZDT3

Test function *ZDT3* represents the discreteness feature; its Pareto optimal front consists of several non-contiguous convex parts:

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ g(\mathbf{x}) &= 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} - \left(\frac{f_1}{g} \right) \sin(10\pi f_1) \end{aligned} \tag{A.4}$$

The original function has $m = 30$ and $x_i \in [0, 1] \forall i$, while the implemented version has $m = 100$. The Pareto optimal front is formed with $g(\mathbf{x}) = 1$. The introduction of the sine function in causes discontinuity in the Pareto optimal front. However, there is no discontinuity in the parameter space.

ZDT4

Test function *ZDT4* contains 21^9 local Pareto optimal fronts and, therefore, tests the ability of Evolutionary Algorithm to deal with multi-modality:

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ g(\mathbf{x}) &= 1 + 10(m-1) + \sum_{i=2}^m [x_i^2 - 10 \cos(4\pi x_i)] \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{A.5}$$

The original function has $m = 10$ with $x_1 \in [0, 1]$ and $x_2, \dots, x_m \in [-5, -5] \forall i$, while the implemented version has $m = 100$. The global Pareto optimal front is formed with $g(\mathbf{x}) = 1$, the best local Pareto optimal front with $g(\mathbf{x}) = 1.25$ (note that not all local Pareto optimal sets are distinguishable in the objective space).

ZDT6

Test function *ZDT6* includes two difficulties caused by the non-uniformity of the search space: first, the Pareto optimal solutions are non-uniformly distributed along the global Pareto front (the front is biased for solutions for which $f_1(\mathbf{x})$ is near one); second, the

density of the solutions is lowest near the Pareto optimal front and highest away from the front:

$$\begin{aligned} f_1(\mathbf{x}) &= 1 - \exp(-4x_i) \sin^6(6\pi x_i) \\ g(\mathbf{x}) &= 1 + 9 \left(\sum_{i=2}^m \frac{x_i}{m-1} \right)^{0.25} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g} \right)^2 \end{aligned} \quad (\text{A.6})$$

The original function has $m = 10$ and $x_i \in [0, 1] \forall i$, while the implemented version has $m = 100$. The Pareto optimal front is formed with $g(\mathbf{x}) = 1$ and is non-convex.

A.2 Three-objective test functions

The three-objective problems reported below are particular expression of the general M -objectives test. In fact the original definition in [12] introduce an arbitrary number of objective function, and the relative problem becomes more difficult as this number increases. The way of building the test function and the difficulties inherent resemble those introduced above in A.1. Although larger problem can be defined, here are reported only problems with three objectives. This gives the possibility to graphically represent the results and intuitively understand them.

DTLZ1

Test function *DTLZ1* contains $11^k - 1$ local Pareto optimal fronts, each of which can attract an multi-objective evolutionary algorithm [12]. To increase the difficulty of the problem, the number of variables and the frequency of the cosine function were doubled when compared to those suggested in [12].

$$\begin{aligned} f_1(\mathbf{x}) &= \frac{1}{2}(1 + g(\mathbf{x}_M))x_1x_2 \\ f_2(\mathbf{x}) &= \frac{1}{2}(1 + g(\mathbf{x}_M))x_1(1 - x_2) \\ f_3(\mathbf{x}) &= \frac{1}{2}(1 + g(\mathbf{x}_M))(1 - x_1) \\ g(\mathbf{x}_M) &= 100 \left[|\mathbf{x}_M| + \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 - \cos[20\pi(x_i - 0.5)] \right] \end{aligned} \quad (\text{A.7})$$

where $n = 14$ and $x_i \in [0, 1]$.

DTLZ2

Test function *DTLZ2* is the Generic sphere problem.

$$\begin{aligned} f_1(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \cos\left(\frac{x_2\pi}{2}\right) \\ f_2(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \sin\left(\frac{x_2\pi}{2}\right) \\ f_3(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \sin\left(\frac{x_1\pi}{2}\right) \\ g(\mathbf{x}_M) &= \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 \end{aligned} \quad (\text{A.8})$$

where $n = 22$ and $x_i \in [0, 1]$. The number of variables suggested in [12] for this test problem is 12.

DTLZ3

Test function *DTLZ3* is similar to test function *DTLZ2*, except for the function g , which introduces $3^k - 1$ local Pareto optimal fronts, and only one global Pareto optimal front.

$$\begin{aligned} f_1(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \cos\left(\frac{x_2\pi}{2}\right) \\ f_2(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \sin\left(\frac{x_2\pi}{2}\right) \\ f_3(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \sin\left(\frac{x_1\pi}{2}\right) \\ g(\mathbf{x}_M) &= 100 \left[|\mathbf{x}_M| + \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 - \cos[20\pi(x_i - 0.5)] \right] \end{aligned} \quad (\text{A.9})$$

where $n = 22$ and $x_i \in [0, 1]$. The number of variables suggested in [12] for this test problem is 12.

DTLZ4

Test function *DTLZ4* is a modified version of *DTLZ2*, since it features a different meta-variable mapping.

$$\begin{aligned} f_1(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1^\alpha\pi}{2}\right) \cos\left(\frac{x_2^\alpha\pi}{2}\right) \\ f_2(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1^\alpha\pi}{2}\right) \sin\left(\frac{x_2^\alpha\pi}{2}\right) \\ f_3(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \sin\left(\frac{x_1^\alpha\pi}{2}\right) \\ g(\mathbf{x}_M) &= \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 \end{aligned} \quad (\text{A.10})$$

where $n = 22$, $\alpha = 100$ and $x_i \in [0, 1]$. The number of variables suggested in [12] for this test problem is 12.

DTLZ5

Test function *DTLZ5* features a different mapping compared to the one of *DTLZ2*. This problem will test a multi-objective evolutionary algorithm ability to converge to a curve and will also allow an easier way to visually demonstrate the performance of the algorithm.

$$\begin{aligned} f_1(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \cos\left(\frac{x_2\pi}{2}\right) \\ f_2(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \sin\left(\frac{x_2\pi}{2}\right) \\ f_3(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \sin\left(\frac{x_1\pi}{2}\right) \\ g(\mathbf{x}_M) &= \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 \\ \text{with } \theta_2 &= \frac{\pi}{4(1 + g)}(1 + 2gx_2) \end{aligned} \quad (\text{A.11})$$

where $n = 22$ and $x_i \in [0, 1]$. The number of variables suggested in [12] for this test problem is 12.

DTLZ6

Test function *DTLZ6* is a modified, harder-to-optimize version of the above test problem. The number of decision variables was dramatically increased when compared to the original one.

$$\begin{aligned}
 f_1(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \cos\left(\frac{x_2\pi}{2}\right) \\
 f_2(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \cos\left(\frac{x_1\pi}{2}\right) \sin\left(\frac{x_2\pi}{2}\right) \\
 f_3(\mathbf{x}) &= (1 + g(\mathbf{x}_M)) \sin\left(\frac{x_1\pi}{2}\right) \\
 g(\mathbf{x}_M) &= \sum_{x_i \in \mathbf{x}_M} (x_i)^{0.1} \\
 \text{with } \theta_2 &= \frac{\pi}{4(1+g)}(1 + 2gx_2)
 \end{aligned} \tag{A.12}$$

where $n = 100$ and $x_i \in [0, 1]$. The number of variables suggested in [12] for this test problem is 12.

DTLZ7

Test function *DTLZ7* features 2^{M-1} disconnected local Pareto optimal regions in the search space. It is chosen to test the multi-objective evolutionary algorithm ability in finding and maintain stable and distributed sub-populations in all four disconnected global Pareto-optimal regions.

$$\begin{aligned}
 f_1(\mathbf{x}) &= x_1 \\
 f_2(\mathbf{x}) &= x_2 \\
 f_3(\mathbf{x}) &= [1 + g(\mathbf{x}_M)]h \\
 g(\mathbf{x}_M) &= 1 + \frac{9}{|x_i|} \sum_{x_i \in \mathbf{x}_M} x_i \\
 \text{with } h &= M - \sum_{i=1}^{M-1} \left[\frac{f_i}{1+g} [1 + \sin(3\pi f_i)] \right]
 \end{aligned} \tag{A.13}$$

where $n = 100$ and $x_i \in [0, 1]$. Once again, the number of decision variables was dramatically increased when compared to the original one, suggested in [12] for this test problem, and equal to 22.

Appendix B

Plots and Data of Test Functions

Here the reader can find all the results on test problems. Plots and data already presented in chapter 7 are not repeated. However, the test problems are divided in two sections relative to their number of objective functions. For each problem there are reported:

1. Pareto front plots for all runs and a single run of all the test algorithms. Some problems present also a further Pareto front plot to highlight results in the neighbourhood of the true front. This is the case of B.2 with fig. B.13, where the best results of ASEM00 and ParEGO are much far from the true Pareto.
2. The box-plots of D-metric and normalized Hyper-Volume for all the test algorithms, obtained from the results variation of their single runs.
3. The table which reports the mean values and the characteristics of the previous box-plots for D-metric and normalized Hyper-Volume on all the test algorithms.

B.1 Results of two-objective problems

ZDT2

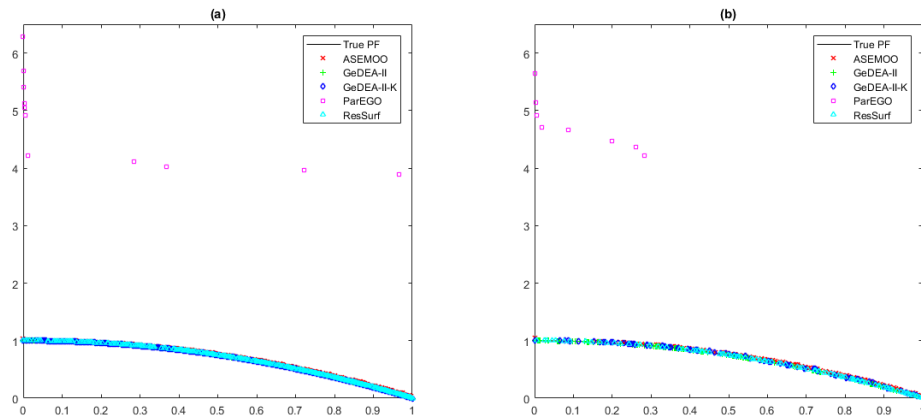


Figure B.1: Test function ZDT2: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

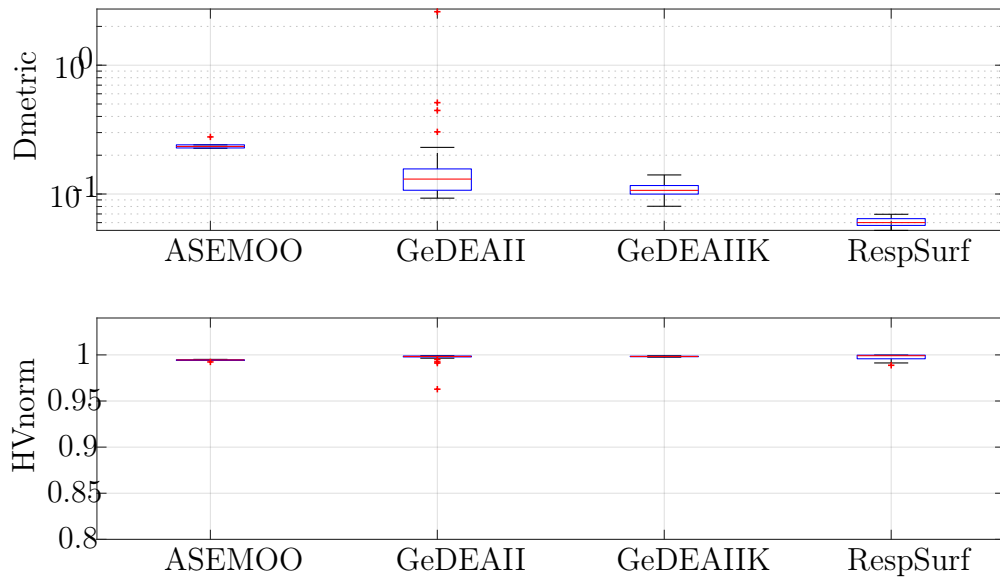


Figure B.2: Test function ZDT2: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1, 6)).

D metric					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.2401	0.2407	0.1078	42.5630	0.0602
Median	0.2340	0.1306	0.1067	43.0106	0.0600
Perc. 25%	0.2278	0.1069	0.0998	41.4886	0.0570
Perc. 75%	0.2409	0.1566	0.1164	43.1841	0.0644
Whisker low	0.2265	0.0929	0.0803	40.4812	0.0522
Whisker up	0.2775	2.6024	0.1406	44.3363	0.0696

HV normalized					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.9942	0.9966	0.9983	0.3166	0.9975
Median	0.9945	0.9983	0.9983	0.3125	0.9991
Perc. 25%	0.9940	0.9978	0.9980	0.3028	0.9958
Perc. 75%	0.9946	0.9988	0.9985	0.3323	0.9997
Whisker low	0.9924	0.9628	0.9975	0.2909	0.9887
Whisker up	0.9951	0.9991	0.9990	0.3423	0.9998

Table B.1: Test function ZDT2: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (1, 6)).

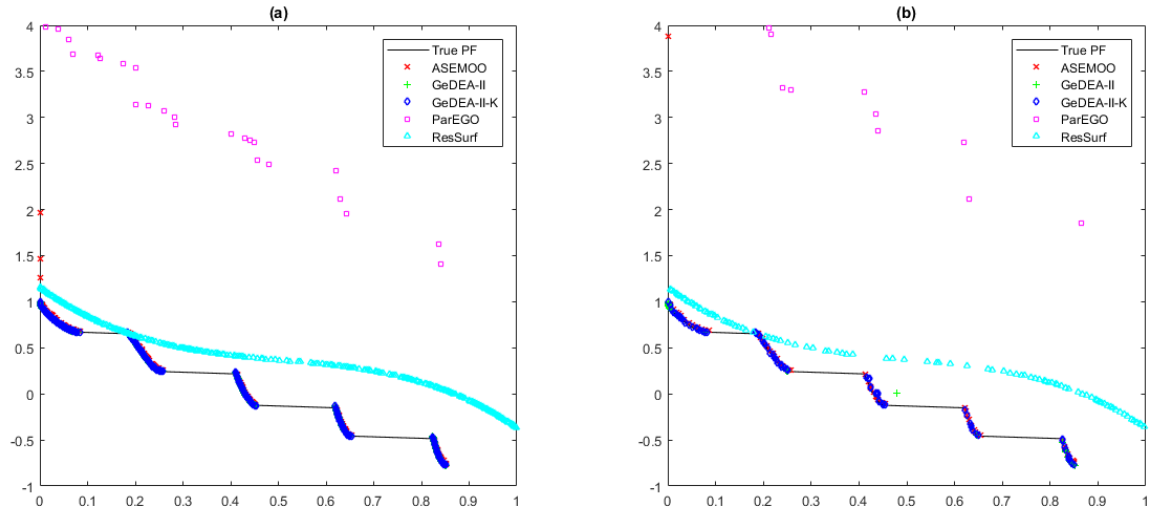
ZDT3

Figure B.3: Test function ZDT3: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

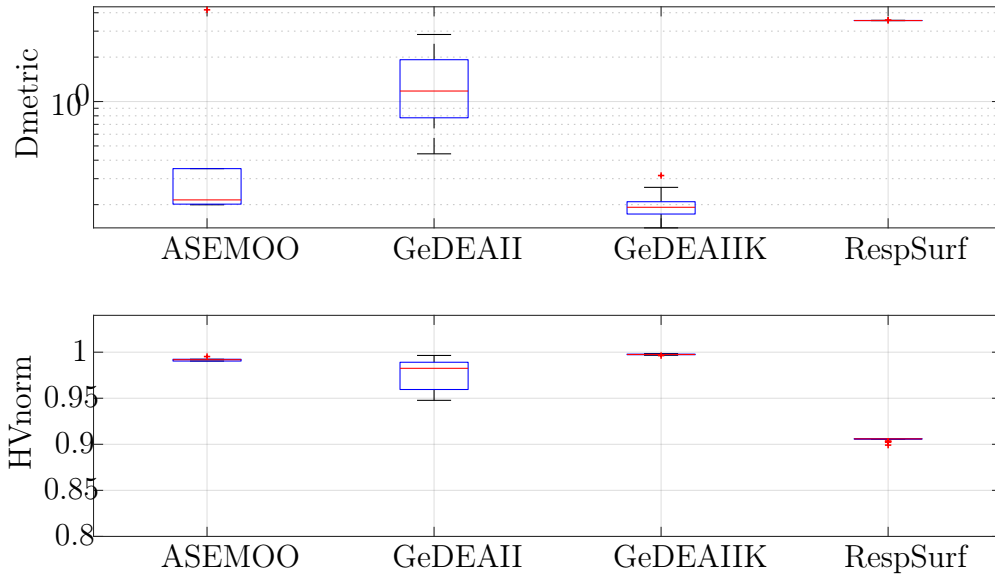


Figure B.4: Test function ZDT3: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1, 4.5)).

	D metric				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.89448856	1.376560997	0.194158349	29.48717305	3.542727923
Median	0.215326776	1.178176132	0.19186676	30.11026466	3.53973088
Perc. 25%	0.201492497	0.775718055	0.17266714	28.01847132	3.538851389
Perc. 75%	0.350786151	1.920959139	0.209299497	30.62127223	3.544852194
Whisker low	0.199490354	0.44196404	0.139116769	23.39608568	3.536473088
Whisker up	4.184508803	2.848019594	0.314880099	33.63030292	3.560232243

	HV normalized				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.953437573	0.976703063	0.997530336	0.341933701	0.905293547
Median	0.99185563	0.982498627	0.997498745	0.341331007	0.905926659
Perc. 25%	0.990388913	0.959500756	0.997275989	0.332788412	0.905751633
Perc. 75%	0.992328872	0.989079328	0.997876686	0.349953517	0.906007409
Whisker low	0.758922773	0.947746544	0.99630598	0.319158847	0.89912409
Whisker up	0.995273623	0.996462845	0.998462966	0.374332924	0.906134788

Table B.2: Test function ZDT3: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (1, 4.5)).

ZDT4

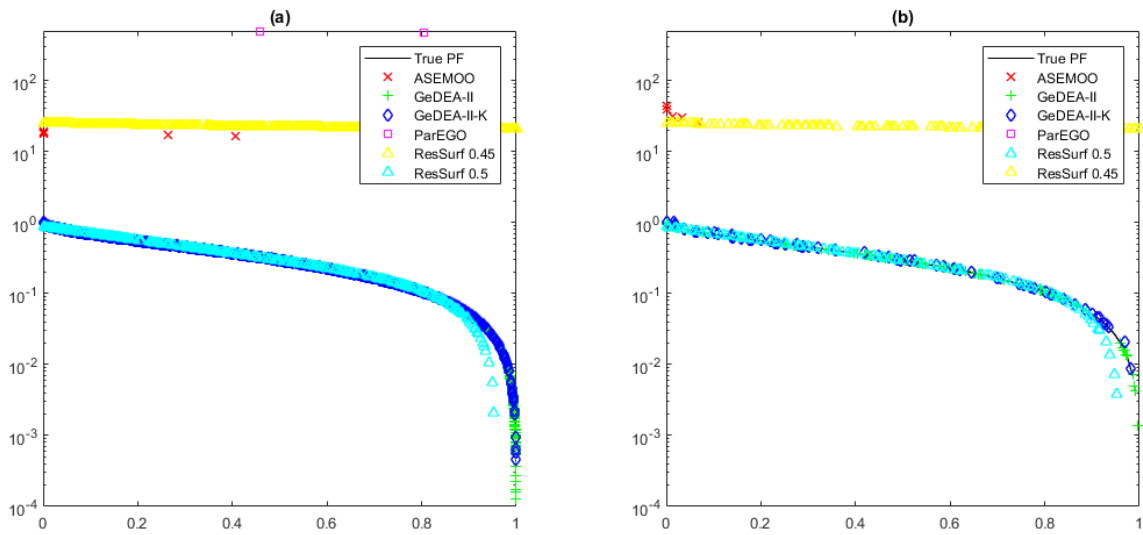


Figure B.5: Test function ZDT4: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

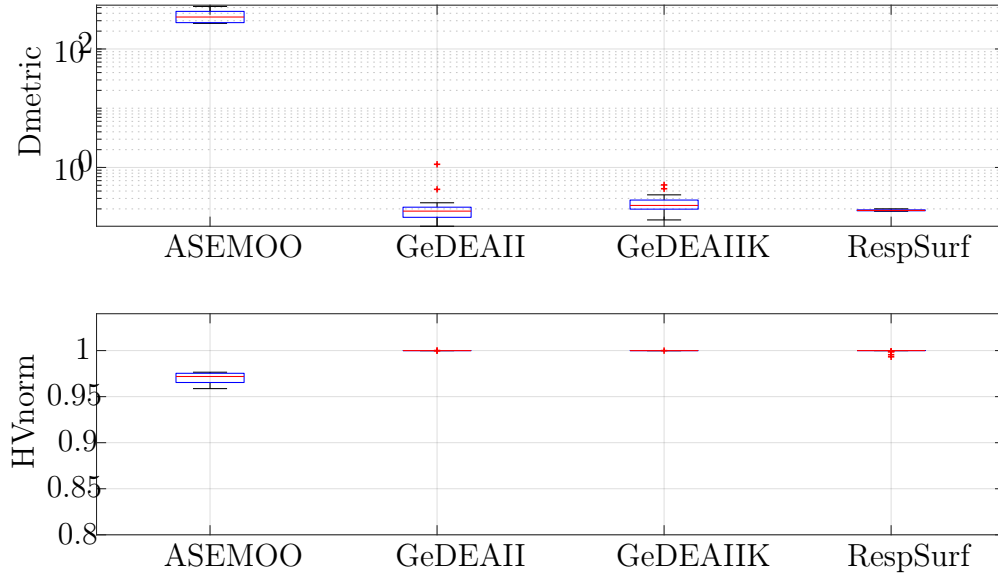


Figure B.6: Test function ZDT4: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1, 750)).

	D metric				
	ASEM00	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	365.6865	0.2135	0.2489	8765.37	0.1896
Median	346.0383	0.1833	0.2286	8786.71	0.1881
Perc. 25%	278.3476	0.1441	0.1980	8520.63	0.1857
Perc. 75%	430.8727	0.2137	0.2818	8969.81	0.1926
Whisker low	270.4160	0.1019	0.1302	8098.32	0.1822
Whisker up	522.4064	1.1319	0.5076	9275.59	0.2005

	HV normalized				
	ASEM00	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.9699	1.0000	1.0000	0.2795	0.9995
Median	0.9719	1.0000	1.0000	0.2766	1.0000
Perc. 25%	0.9653	1.0000	1.0000	0.2643	1.0000
Perc. 75%	0.9753	1.0000	1.0000	0.2911	1.0000
Whisker low	0.9587	0.9999	0.9999	0.2514	0.9932
Whisker up	0.9765	1.0000	1.0000	0.3269	1.0000

Table B.3: Test function ZDT4: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (1, 750)).

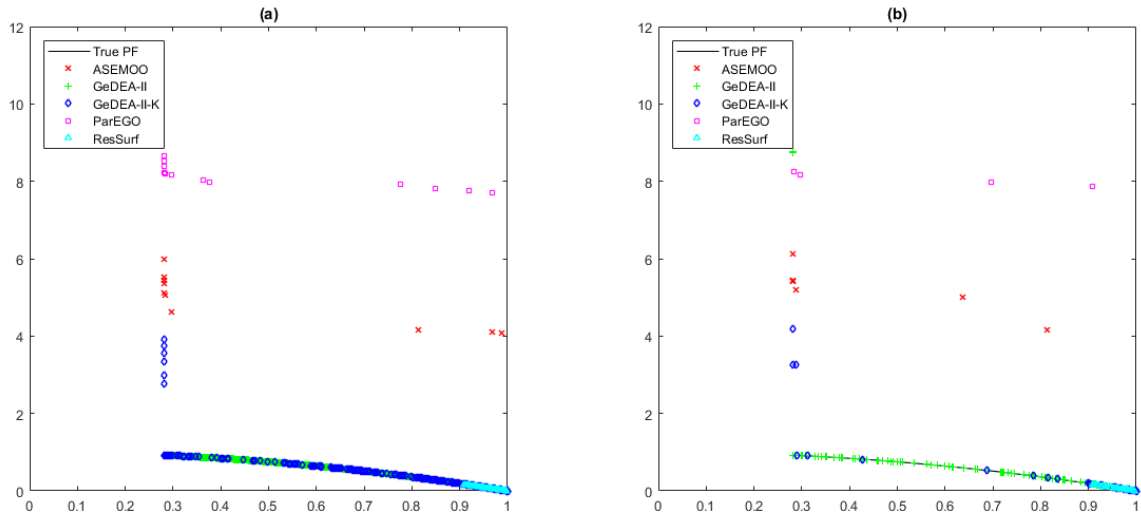
ZDT6

Figure B.7: Test function ZDT6: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

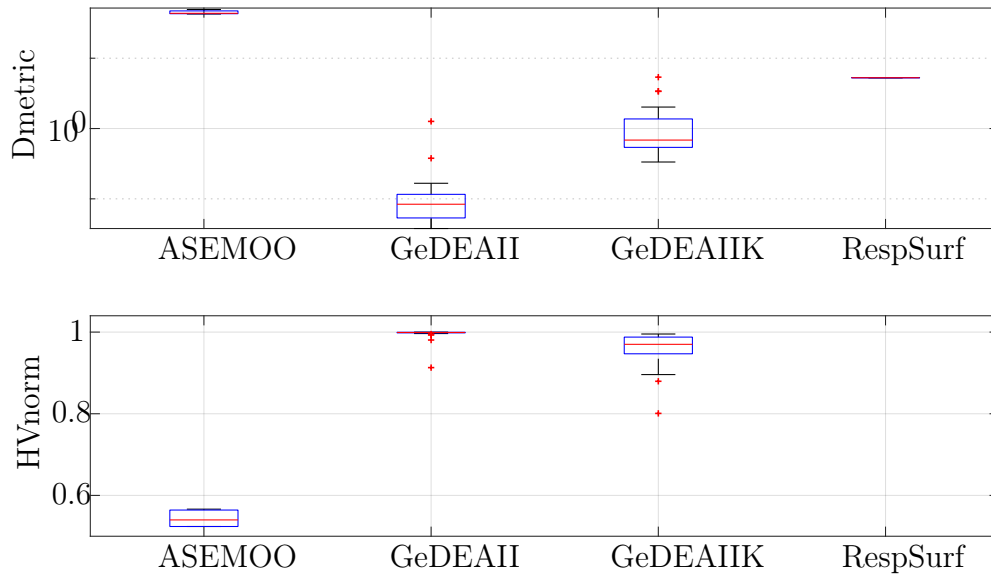


Figure B.8: Test function ZDT6: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1, 10)).

D metric					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	44.8098	0.1321	1.1659	87.063	5.2797
Median	43.5201	0.0838	0.6850	87.243	5.2797
Perc. 25%	42.9681	0.0535	0.5395	86.423	5.2794
Perc. 75%	47.0851	0.1159	1.3689	87.538	5.2799
Whisker low	42.5276	0.0378	0.3341	85.476	5.2790
Whisker up	49.2375	1.2630	5.3784	89.171	5.2806
HV normalized					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.5186	0.9952	0.9566	0.1949	0.1323
Median	0.5400	0.9990	0.9701	0.1948	0.1323
Perc. 25%	0.5239	0.9981	0.9469	0.1899	0.1323
Perc. 75%	0.5641	0.9996	0.9877	0.1990	0.1323
Whisker low	0.3776	0.9130	0.8009	0.1825	0.1323
Whisker up	0.5663	0.9999	0.9953	0.2097	0.1323

Table B.4: Test function ZDT6: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at $(1, 10)$).

B.2 Results of three-objective problems

DTLZ1

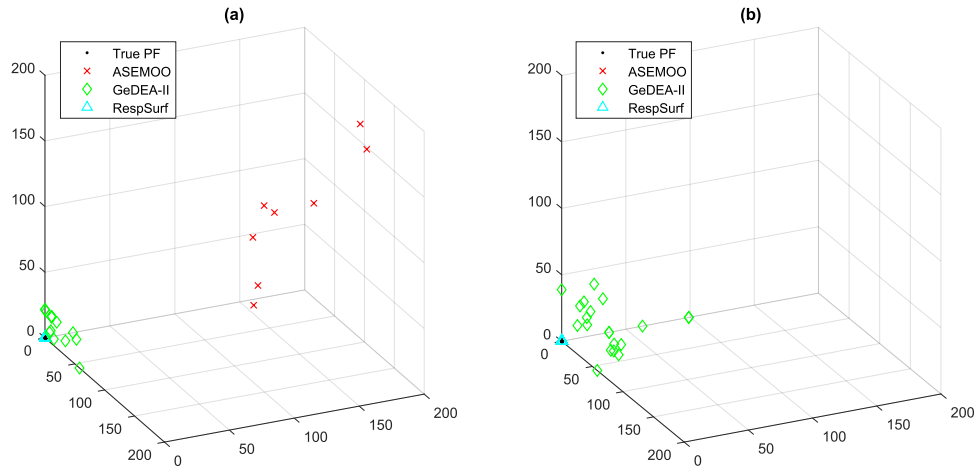


Figure B.9: Test function DTLZ1: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

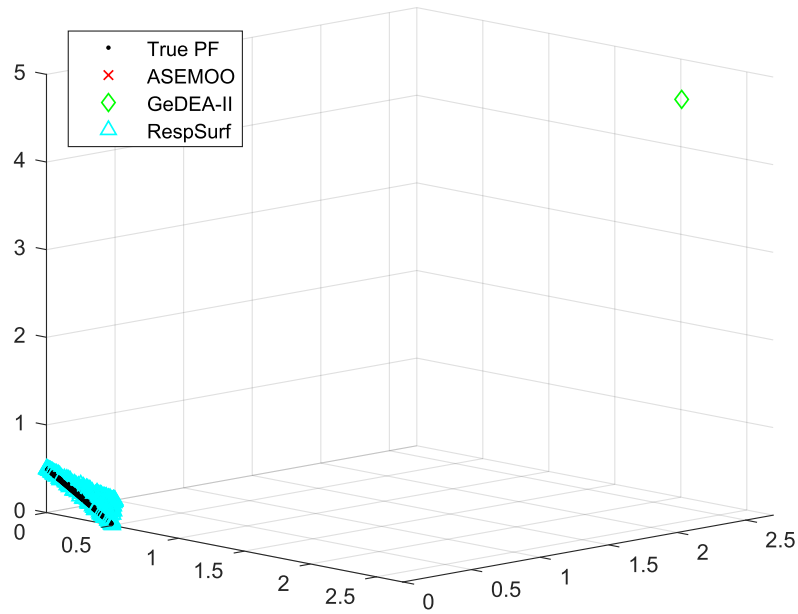


Figure B.10: Test function DTLZ1: zoom on the Pareto font.

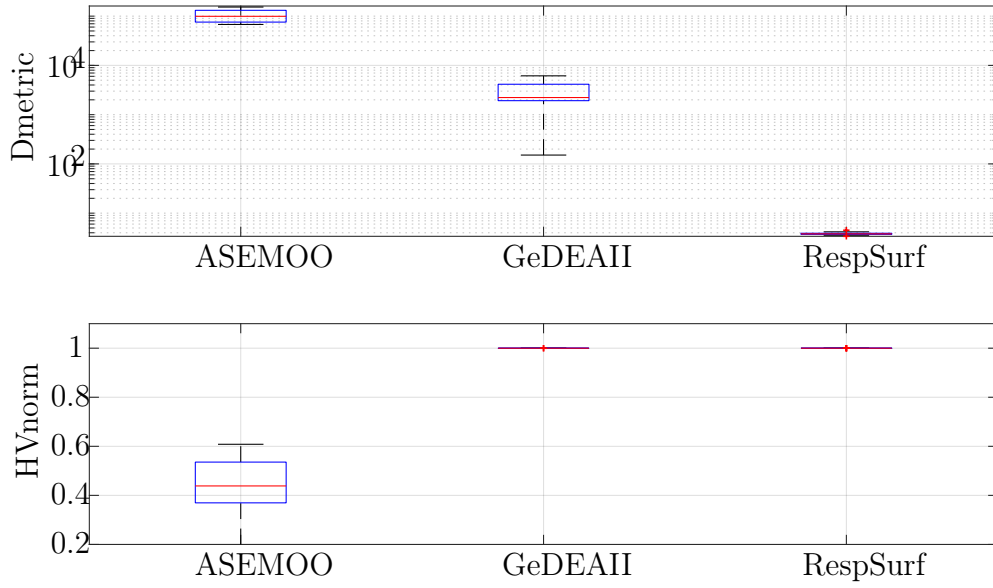


Figure B.11: Test function DTLZ1: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (1000, 1000, 1000)).

	D metric		
	ASEM00	GeDEA-II	RespSurf
Mean value	1.04E+05	2.90E+03	3.8303
Median	9.93E+04	2.22E+03	3.7964
Perc. 25%	7.57E+04	1.92E+03	3.7111
Perc. 75%	1.31E+05	4.14E+03	3.9132
Whisker low	6.78E+04	1.51E+02	3.3884
Whisker up	1.53E+05	6.14E+03	4.5058
	HV normalized		
	ASEM00	GeDEA-II	RespSurf
Mean value	0.4241	1.0000	1.0000
Median	0.4385	1.0000	1.0000
Perc. 25%	0.3695	0.9999	1.0000
Perc. 75%	0.5357	1.0000	1.0000
Whisker low	0.1543	0.9998	1.0000
Whisker up	0.6083	1.0000	1.0000

Table B.5: Test function DTLZ1: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (1000, 1000, 1000)).

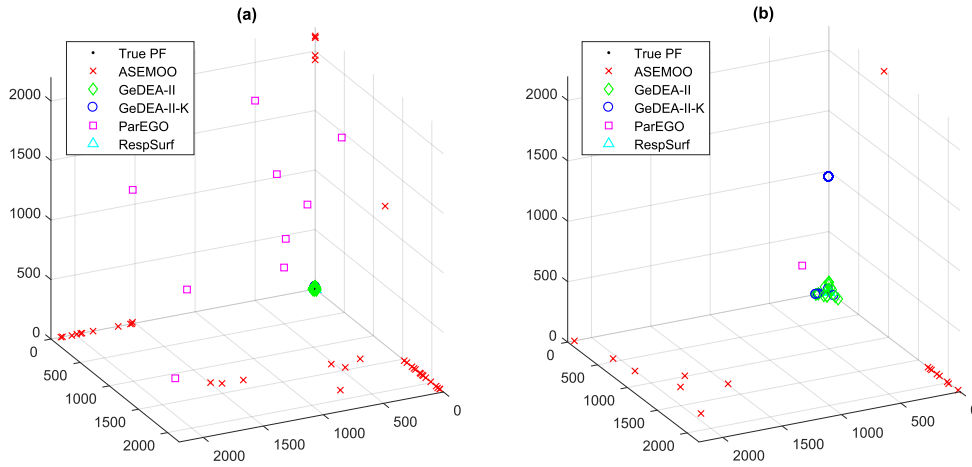
DTLZ3

Figure B.12: Test function DTLZ3: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

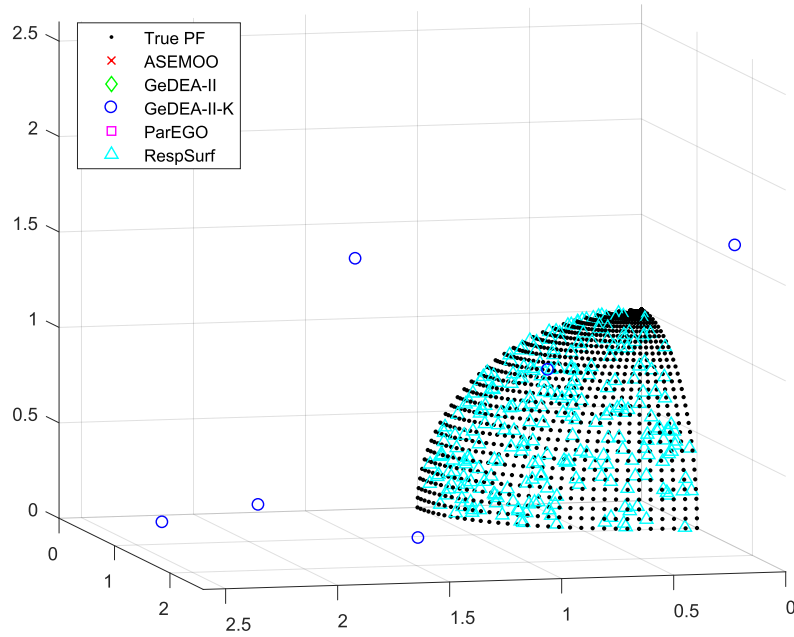


Figure B.13: Test function DTLZ3: zoom on the Pareto front.

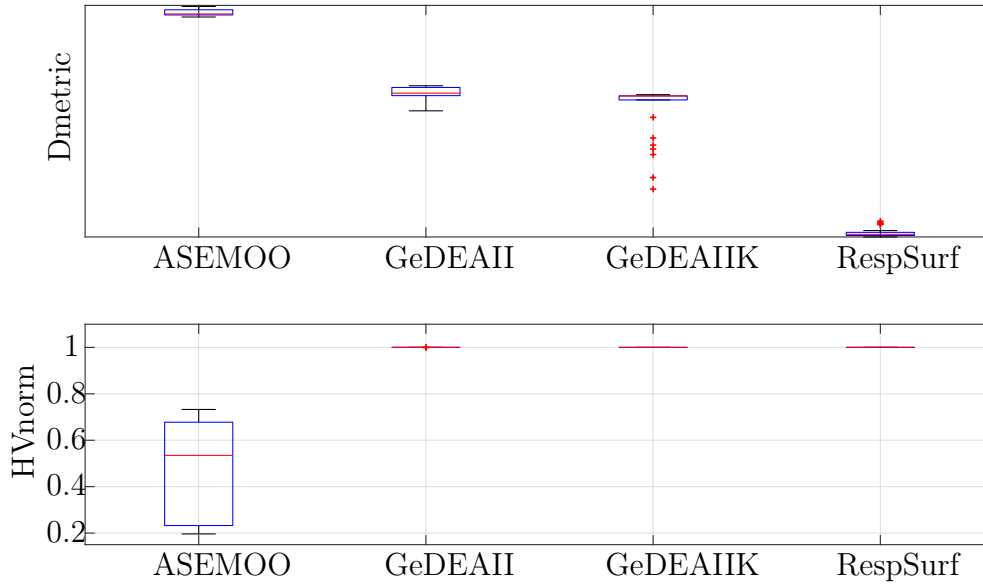


Figure B.14: Test function DTLZ3: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (2500, 2500, 2500)).

D metric					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	63927.9	1720.8	1115.7	103517.4	2.8946
Median	59581.6	1618.1	1425.7	105797.1	2.6212
Perc. 25%	56901.1	1445.4	1187.4	99002.9	2.4849
Perc. 75%	72008.7	2098.3	1427.8	109771.4	2.8688
Whisker low	51772.7	724.0	20.5	87051.7	2.3254
Whisker up	83721.4	2267.3	1513.0	113028.1	4.7685

HV normalized					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.4848	1.0000	0.9999	0.0546	1.0000
Median	0.5349	1.0000	1.0000	0.0491	1.0000
Perc. 25%	0.2325	1.0000	0.9999	0.0338	1.0000
Perc. 75%	0.6778	1.0000	1.0000	0.0764	1.0000
Whisker low	0.1962	1.0000	0.9998	0.0136	1.0000
Whisker up	0.7327	1.0000	1.0000	0.1095	1.0000

Table B.6: Test function DTLZ3: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (2500, 2500, 2500)).

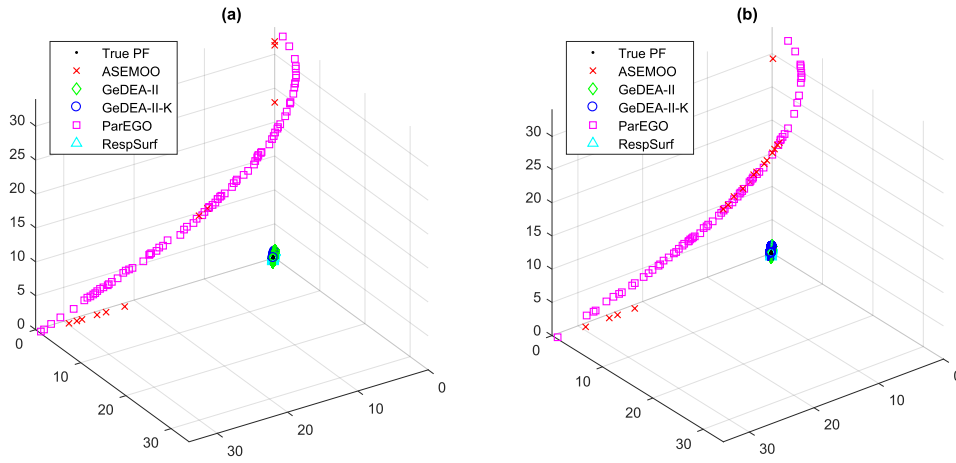
DTLZ5

Figure B.15: Test function DTLZ5: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

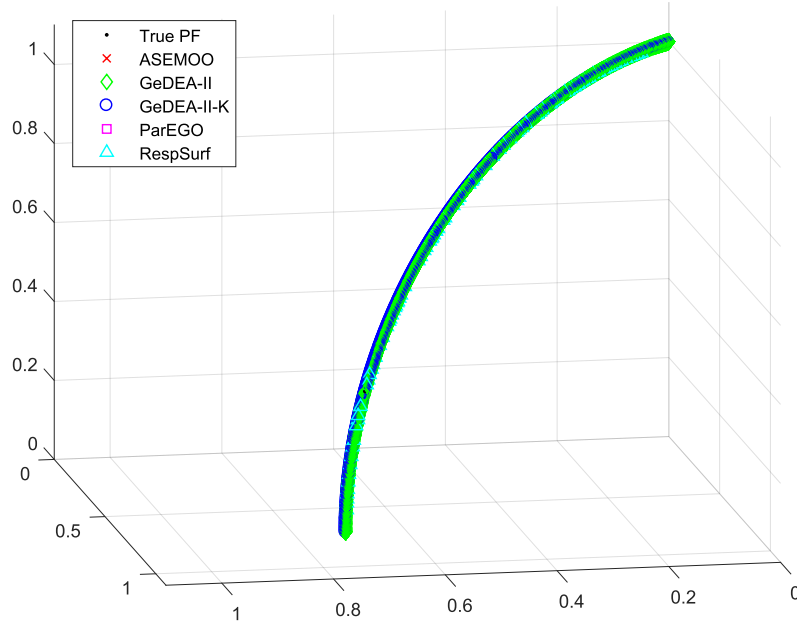


Figure B.16: Test function DTLZ5: zoom on the Pareto font.

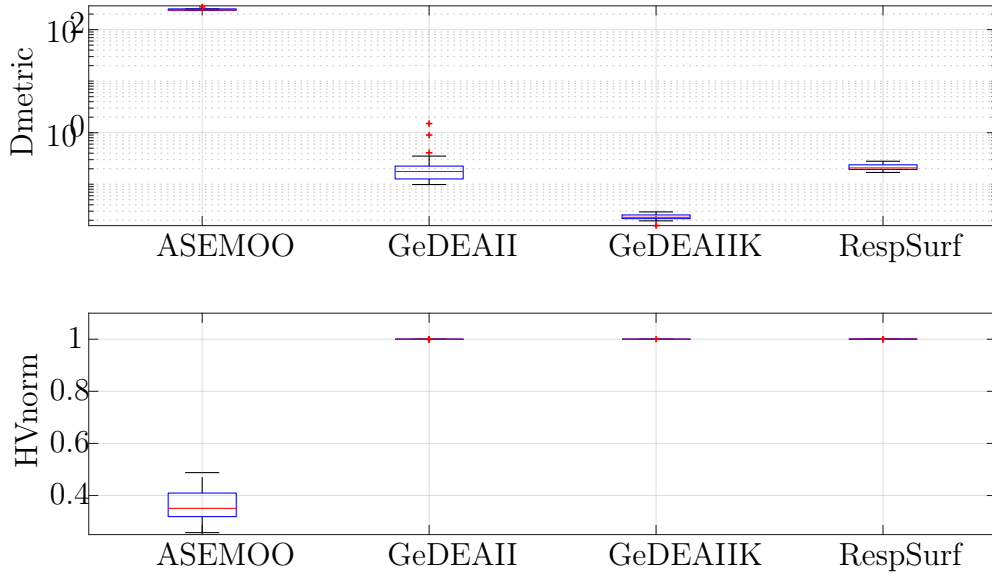


Figure B.17: Test function DTLZ5: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at $(32, 32, 32)$).

D metric					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	246.99	0.2524	0.0233	364.19	0.2141
Median	239.44	0.1765	0.0229	364.19	0.2075
Perc. 25%	238.23	0.1267	0.0215	362.51	0.1940
Perc. 75%	252.18	0.2248	0.0254	366.65	0.2382
Whisker low	237.04	0.0987	0.0158	357.53	0.1694
Whisker up	275.43	1.5032	0.0291	368.09	0.2794

HV normalized					
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.3637	1.0000	1.0000	0.1287	1.0000
Median	0.3507	1.0000	1.0000	0.1290	1.0000
Perc. 25%	0.3190	1.0000	1.0000	0.1276	1.0000
Perc. 75%	0.4093	1.0000	1.0000	0.1301	1.0000
Whisker low	0.2574	0.9998	1.0000	0.1251	1.0000
Whisker up	0.4878	1.0000	1.0000	0.1310	1.0000

Table B.7: Test function DTLZ5: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at $(32, 32, 32)$).

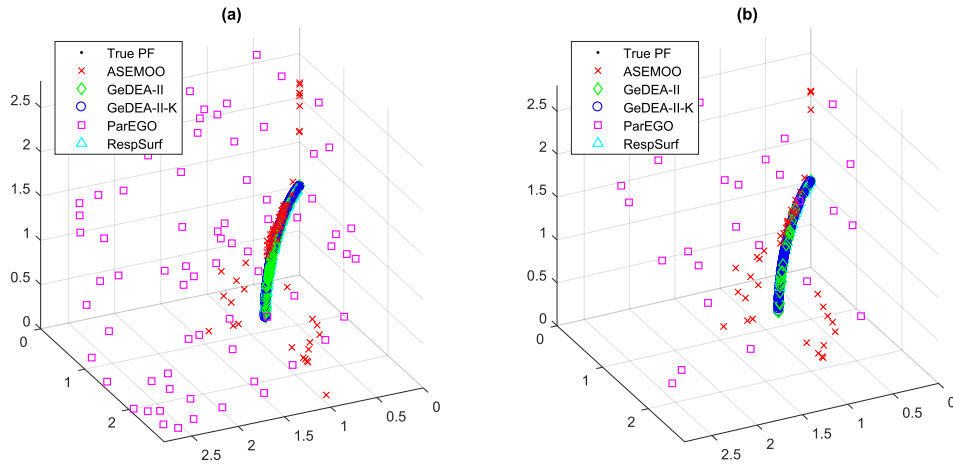
DTLZ6

Figure B.18: Test function DTLZ6: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

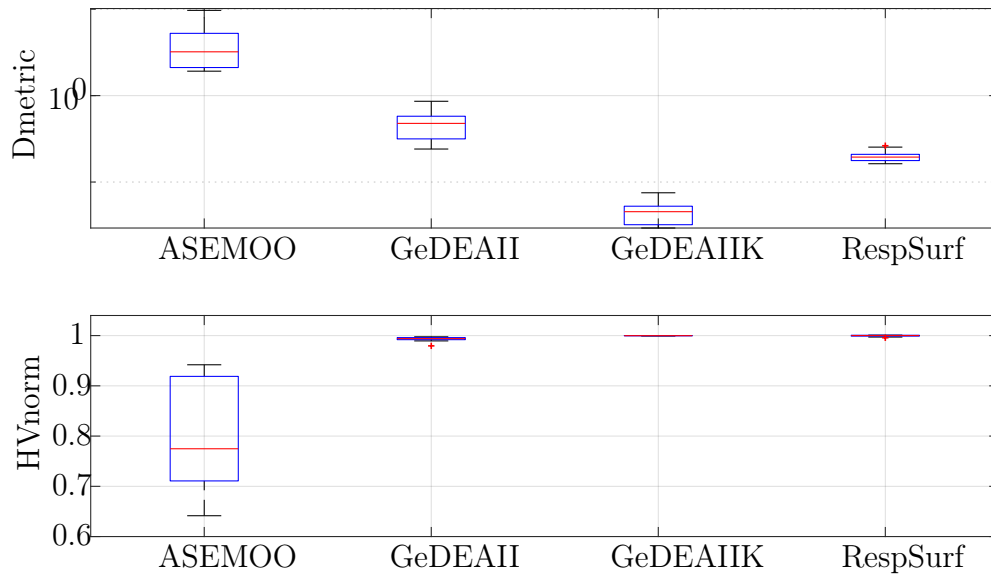


Figure B.19: Test function DTLZ6: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at $(32, 32, 32)$).

	D metric				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	4.2337	0.4689	0.0446	22.370	0.1978
Median	3.2121	0.4772	0.0454	22.298	0.1944
Perc. 25%	2.1129	0.3157	0.0321	22.036	0.1772
Perc. 75%	5.2544	0.5774	0.0525	23.129	0.2092
Whisker low	1.9176	0.2409	0.0294	19.665	0.1629
Whisker up	9.6931	0.8600	0.0751	23.577	0.2631

	HV normalized				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.7938	0.9936	0.9998	0.2263	0.9997
Median	0.7748	0.9940	0.9999	0.2208	1.0000
Perc. 25%	0.7108	0.9919	0.9997	0.2148	0.9990
Perc. 75%	0.9188	0.9960	1.0000	0.2344	1.0000
Whisker low	0.6415	0.9795	0.9992	0.1894	0.9957
Whisker up	0.9419	0.9976	1.0000	0.2691	1.0000

Table B.8: Test function DTLZ6: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at $(3, 3, 3)$).

DTLZ7

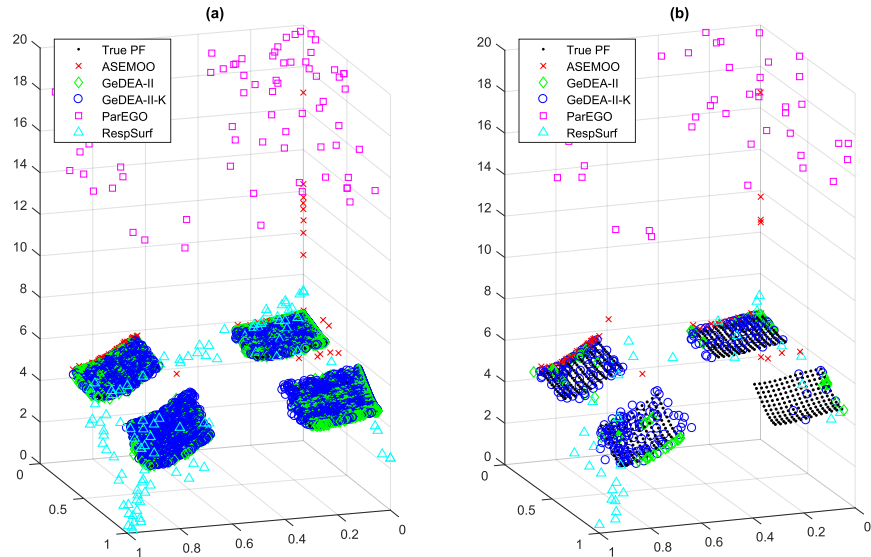


Figure B.20: Test function DTLZ7: Pareto fronts for all runs (a) and single run (b) and for all the optimization algorithms.

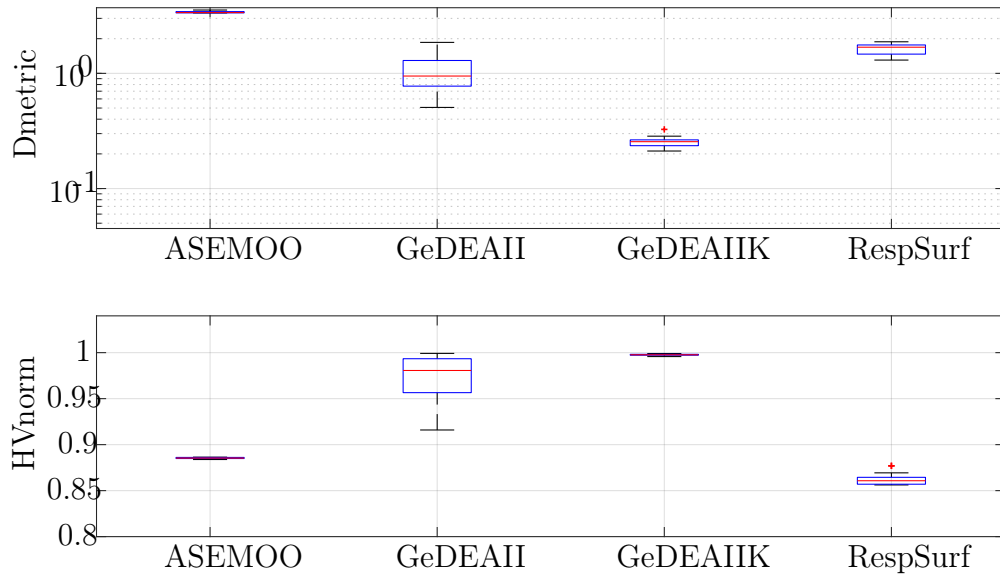


Figure B.21: Test function DTLZ7: box-convergence history of D-metric and normalized Hyper-Volume (with reference point at (18, 18, 18)).

	D metric				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	3.4030	1.0215	0.2526	49.289	1.6401
Median	3.3708	0.9478	0.2552	48.834	1.6868
Perc. 25%	3.3513	0.7746	0.2356	47.390	1.4690
Perc. 75%	3.4360	1.2934	0.2646	51.507	1.7642
Whisker low	3.3454	0.5062	0.2117	45.411	1.3041
Whisker up	3.5468	1.8577	0.3262	53.670	1.8799

	HV normalized				
	ASEMOO	GeDEA-II	GeDEAII-K	ParEGO	RespSurf
Mean value	0.8854	0.9747	0.9977	0.2816	0.8616
Median	0.8854	0.9806	0.9977	0.2870	0.8608
Perc. 25%	0.8849	0.9565	0.9972	0.2567	0.8570
Perc. 75%	0.8861	0.9934	0.9982	0.3019	0.8645
Whisker low	0.8838	0.9159	0.9957	0.2320	0.8562
Whisker up	0.8864	0.9992	0.9990	0.3223	0.8770

Table B.9: Test function DTLZ7: box-plot statistics of D-metric and normalized Hyper-Volume (with reference point at (18, 18, 18)).

Appendix C

Box-plots of errors on the experimental dataset

Here are reported the results of Response Surface method over the experimental database of chapter 7. These box-plots below, with those in the relative chapter, describe all the tested configuration of the surface.

C.1 Fitting and testing sets box-plots

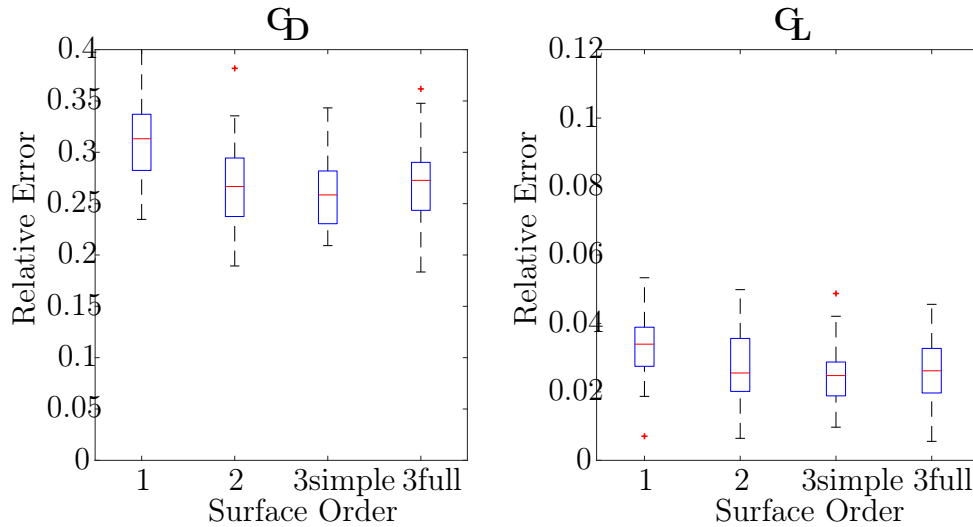


Figure C.1: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by a the single variable $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

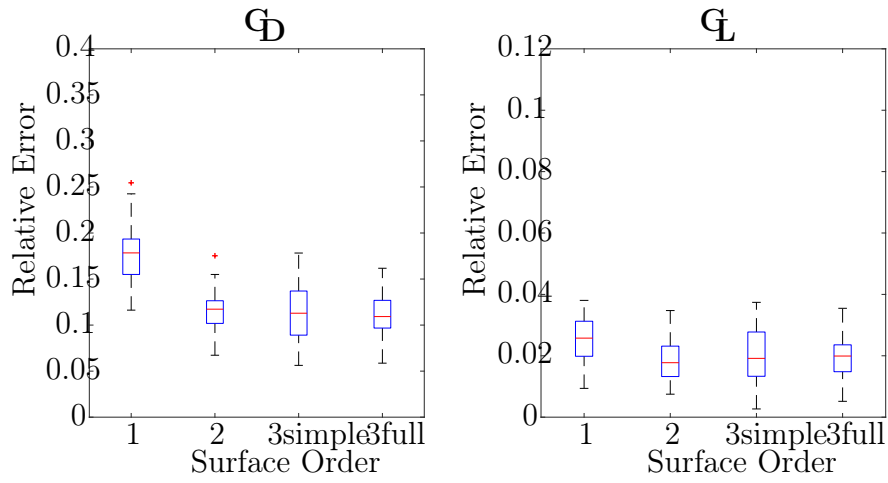


Figure C.2: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by 2 variables: $y3Up$ and $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

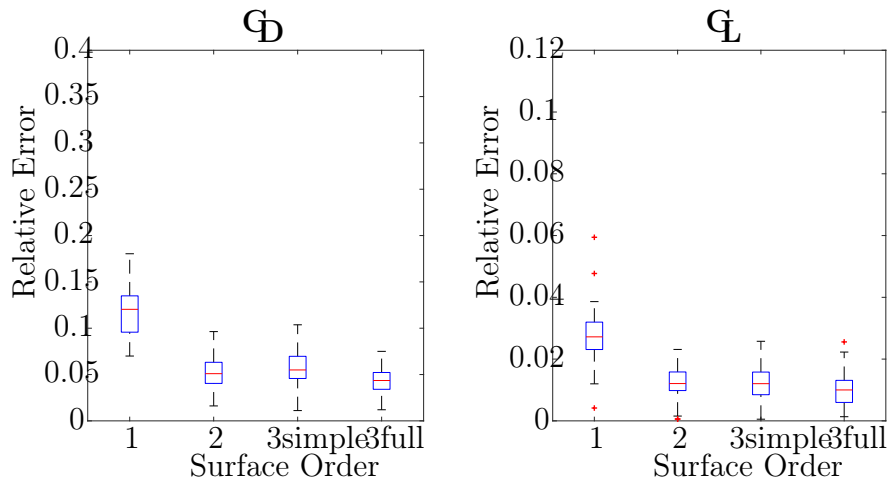


Figure C.3: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by 3 variables: $y3Up$, $y1Low$ and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

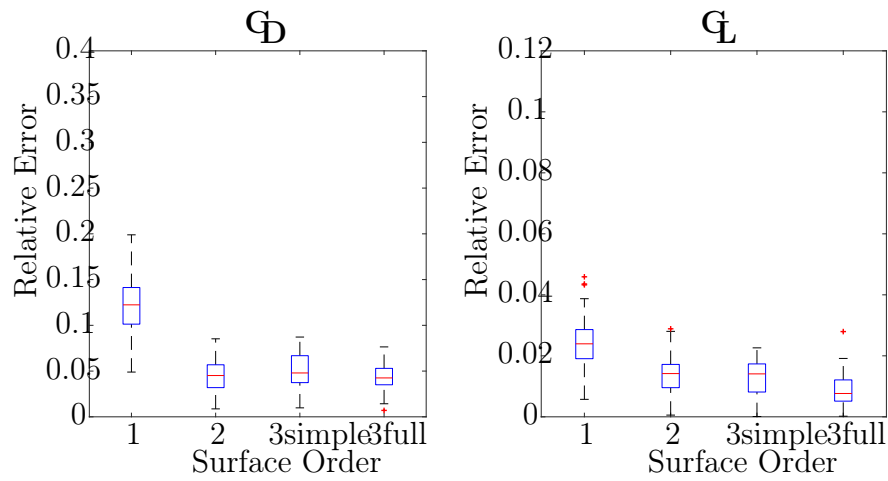


Figure C.4: Box-plots of relative errors of fit&test method over 50 runs with response surface defined by 4 variables: $y3Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

C.2 One point excluded global tests box-plots

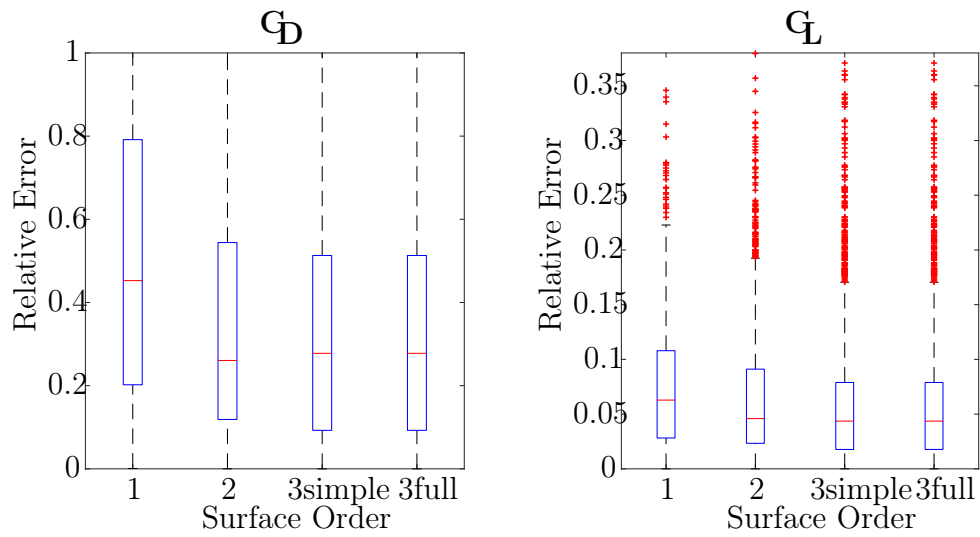


Figure C.5: Box-plots of relative errors on global response surface defined by a the single variable $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

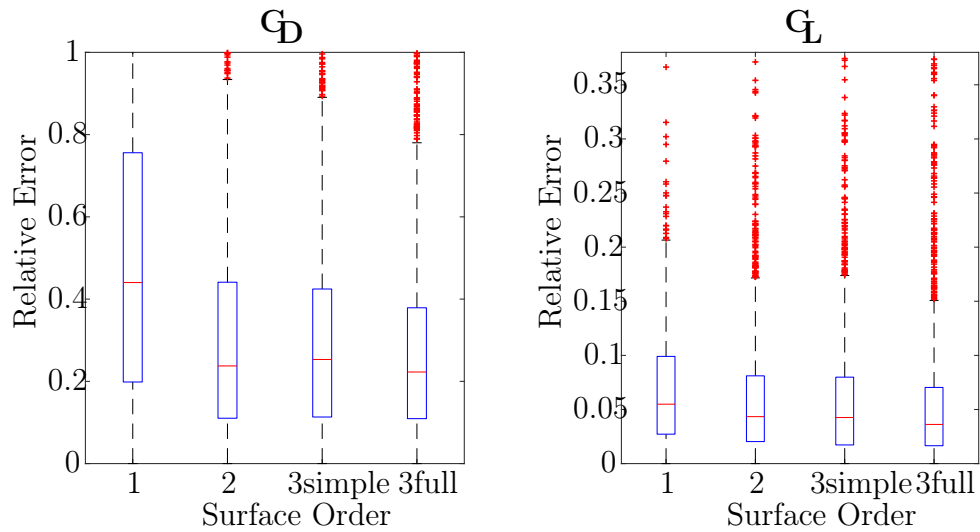


Figure C.6: Box-plots of relative errors on global response surface defined by 2 variables: $y3Up$ and $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

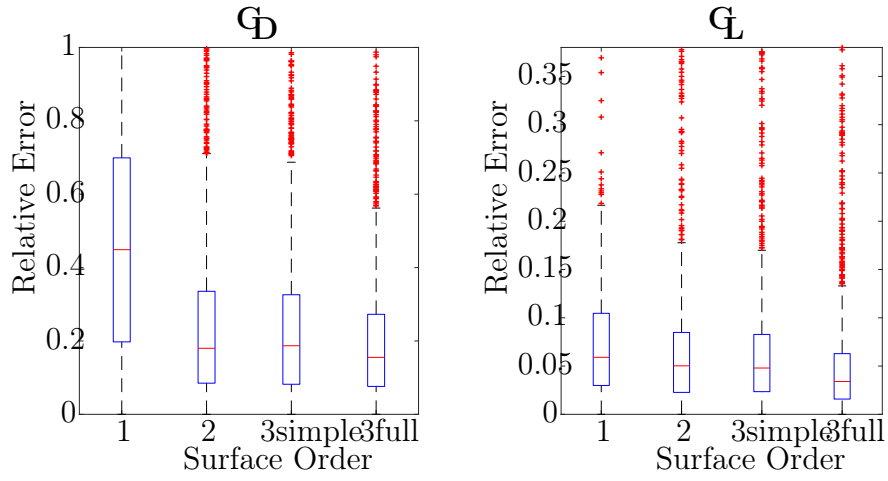


Figure C.7: Box-plots of relative errors on global response surface defined by 3 variables: $y3Up$, $y1Low$ and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

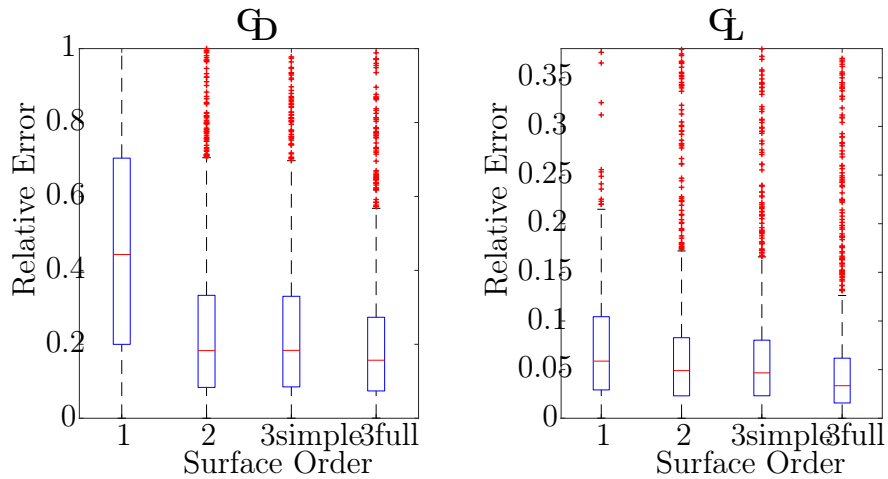


Figure C.8: Box-plots of relative errors on global response surface defined by 4 variables: $y3Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

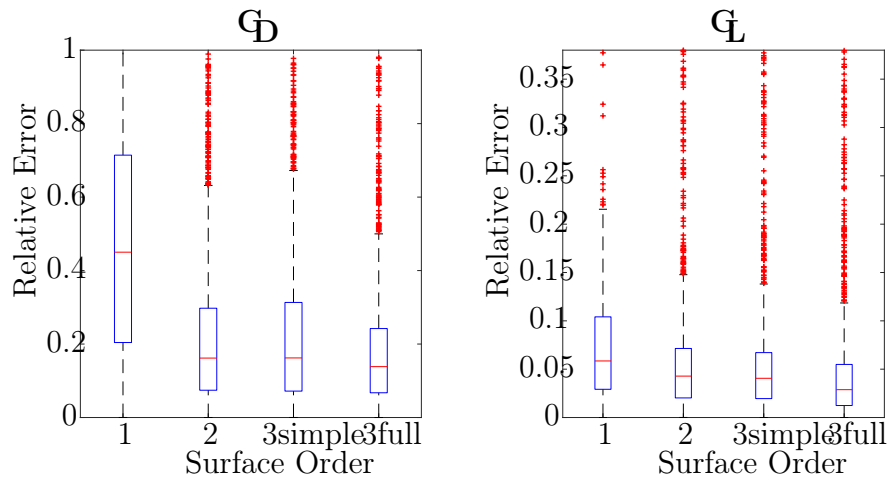


Figure C.9: Box-plots of relative errors on global response surface defined by 5 variables: $y3Up$, $y4Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

C.3 One point excluded local tests box-plots

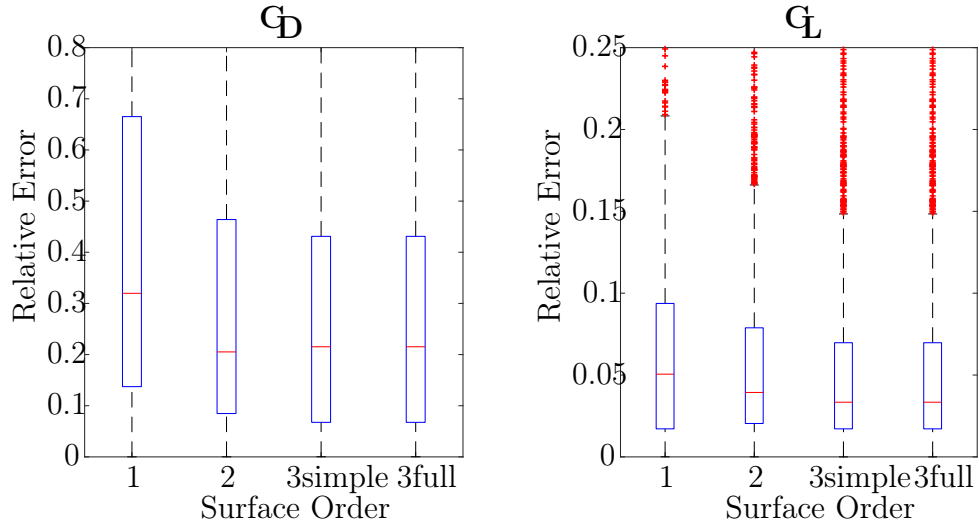


Figure C.10: Box-plots of relative errors on local response surface defined by a the single variable $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

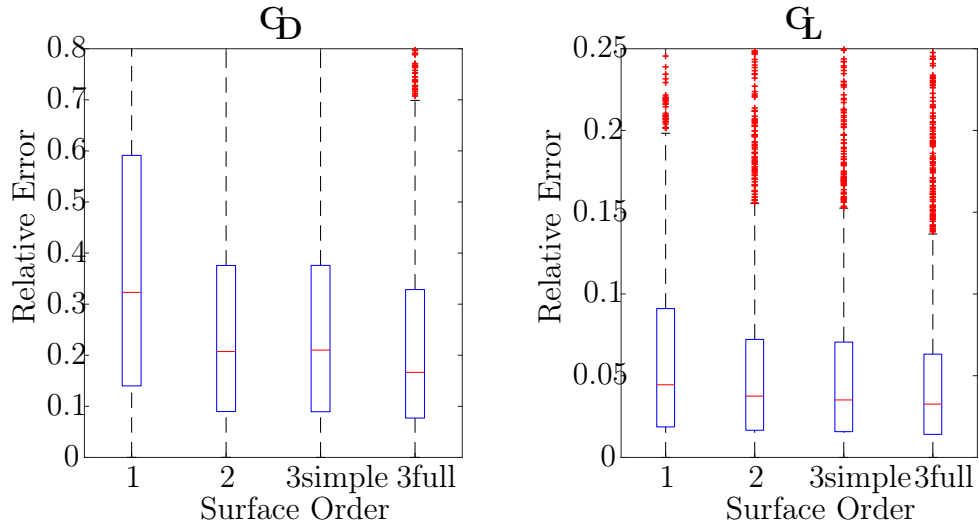


Figure C.11: Box-plots of relative errors on local response surface defined by 2 variables: $y3Up$ and $y1Low$. Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

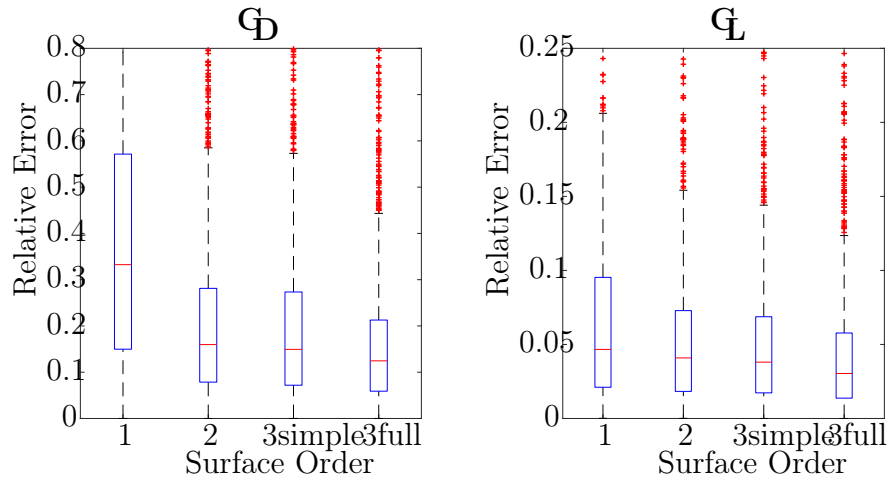


Figure C.12: Box-plots of relative errors on local response surface defined by 3 variables: $y3Up$, $y1Low$ and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

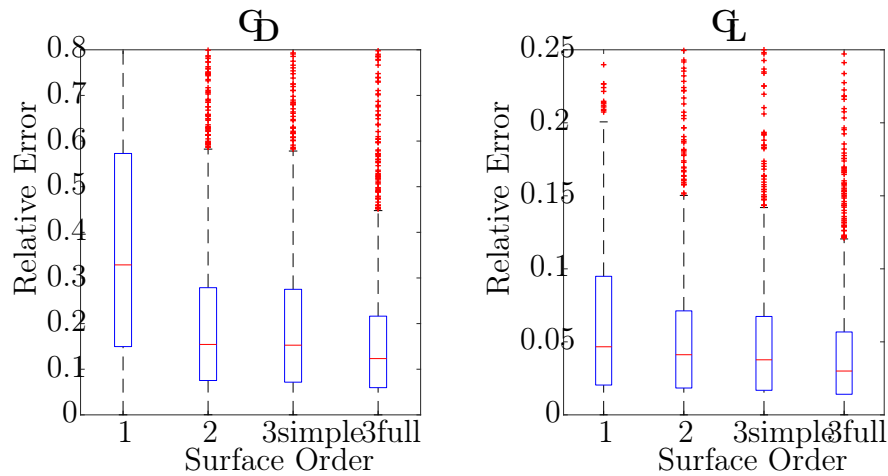


Figure C.13: Box-plots of relative errors on local response surface defined by 4 variables: $y3Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

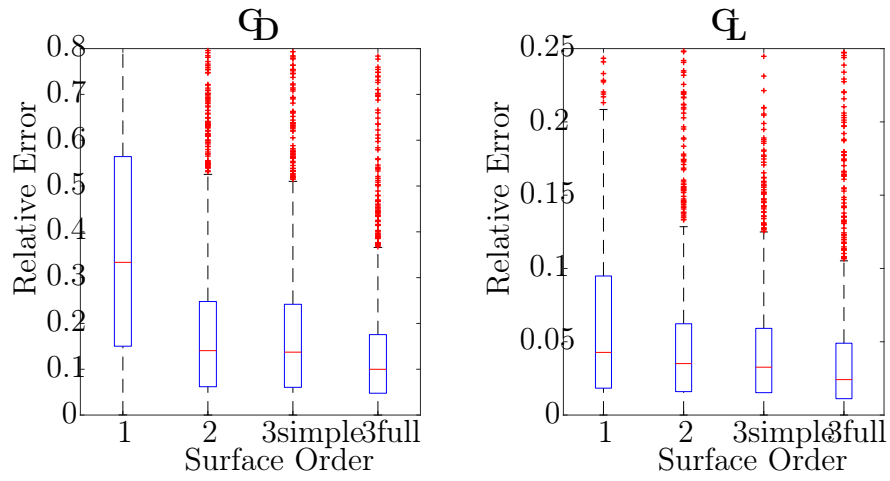


Figure C.14: Box-plots of relative errors on local response surface defined by 5 variables: $y3Up$, $y4Up$, $y1Low$, xLE and yLE . Left: relative error on Drag (c_D). Right: relative error on Lift (c_L).

Bibliography

- [1] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Computational Statistic*, 2:433–459, 2010.
- [2] Computer Science Technology College Anhui University. Platemo on-line resources. [Online; accessed 16-November-2017].
- [3] G. E. P. Box and N. R. Draper. *Response Surface, Mixtures, and Ridge Analyses*. Wiley, 2007.
- [4] D. G. Cacuci. *Sensitivity and uncertainty analysis*. Chapman and Hall, 2003.
- [5] Francesca Campolongo, Jessica Cariboni and Andrea Saltelli. An effective screening design for sensitivity analysis of large models. *Environmental Modelling and Software*, 22:1509–1518, 2007.
- [6] J. Cohoon, S. Hegde, W. Martin and D. Richards. Punctuated equilibria: A parallel genetic algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.
- [7] C. Comis Da Ronco and E. Benini. A simplex crossover based evolutionary algorithm including the genetic diversity as objective. *Applied Soft Computing*, 13:2104–2123, April 2012.
- [8] Davi Sampaio Correia, Cristiene Vasconcelos Gonçalves, Jr. Sebastião Simões da Cunha and Valtair Antonio Ferraresi. Comparison between genetic algorithms and response surface methodology in gmaw welding optimization. *Journal of Materials Processing Technology*, 160:70–76, 2005.
- [9] Jhilly Dasgupta, Jaya Sikder and Durbadal Mandal. Modeling and optimization of polymer enhanced ultrafiltration using hybrid neural-genetic algorithm based evolutionary approach. *Applied Soft Computing*, 55:108–126, 2017.
- [10] Danilo R.B. de Araújo, Carmelo J.A. Bastos-Filho and Joaquim F. Martins-Filho. An evolutionary approach with surrogate models and network science concepts to design optical networks. *Engineering Applications of Artificial Intelligence*, 43:67–80, 2015.
- [11] Luís Filomeno de Jesus Fernandes and Edson Alves da Costa Júnior. Estimation of dominant mode parameters in power systems using correlation analysis. *Electric Power Systems Research*, 148:295–302, 2017.

- [12] K. Deb, L. Thiele, M. Laumanns and E. Zitzler. Scalable multi-objective optimization test problems. *Congress on Evolutionary Computation (CEC 2002)*, pages 825–830, 2002.
- [13] Shailesh Dewangan, Soumya Gangopadhyay and Chandan Kumar Biswas. Multi-response optimization of surface integrity characteristics of edm process using grey-fuzzy logic-based hybrid approach. *Engineering Science and Technology, an International Journal*, 18:361–368, 2015.
- [14] M. Dorigo, V. Maniezzo and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern.*, 26:29–41, 1996.
- [15] M. Eusuff, K. E. Lansey and F. Pasha. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering Optimization*, 38:129–154, 2006.
- [16] Shu-Kai S. Fan, Yun-Chia Liang and Erwie Zahara. A genetic algorithm and a particle swarm optimizer hybridized with nelder-mead simplex search. *Computers and Industrial Engineering*, 50:401–425, 2006.
- [17] Toomaj Foroud, Abbas Seifi and Babak Amin Shahidi. Assisted history matching using artificial neural network based global optimization method - applications to brugge field and a fractured iranian reservoir. *Journal of Petroleum Science and Engineering*, 123:46–61, 2014.
- [18] Alexander I. J. Forrester, András Sóbester and Andy J. Keane. *Engineering Design via Surrogate Modelling*. John Wiley and Sons, 2008.
- [19] Xiao gen Zhou, Gui jun Zhang, Xiao hu Hao, Dong wei Xu and Li Yu. Enhanced differential evolution using local lipschitz underestimate strategy for computationally expensive optimization problems. *Applied Soft Computing*, 48:169–181, 2016.
- [20] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer, 2010.
- [21] G. Giftson Samuel and C. Christofer Asir Rajan. Hybrid: Particle swarm optimization-genetic algorithm and particle swarm optimization-shuffled frog leaping algorithm. *Electrical Power and Energy Systems*, 65:432–442, 2015.
- [22] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computational and Operational research*, 13:533–549, 1986.
- [23] Eled Hazan and Ariel Schwartzman. Lecture 2, theoretical machine learning, February 2016. [Online; accessed 21-November-2017].
- [24] Jon Herman, Will Usher et al. SALib: open source scientific tools for Python, 2013–2017. [Online; accessed 16-November-2017].
- [25] Shih-Cheng Horng and Shin-Yeu Lin. Evolutionary algorithm assisted by surrogate model in the framework of ordinal optimization and optimal computing budget allocation. *Information Sciences*, 233:214–229, 2013.

- [26] Rong-Hwa Huang and Tung-Han Yu. An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. *Applied Soft Computing*, 57:642–656, 2017.
- [27] L. Ingberg. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25:33–54, 1996.
- [28] M. J. W. Janseni. Analysis of variance designs for model output. *Computer Physics Communications*, 117:35–43, 1999.
- [29] S. Jeong, M. Murayama and K. Yamamoto. Efficient optimization design method using kriging mode. *J. Aircr.*, 42:413–420, 2005.
- [30] Ping Jiang and Xuejiao Ma. A hybrid forecasting approach applied in the electrical power system based on data preprocessing, optimization and artificial intelligence algorithms. *Applied Mathematical Modelling*, 40:10631–10649, 2016.
- [31] Stephen Joe and Frances Y. Kuo. Notes on generating sobol’ sequences, 2008.
- [32] D. R. Jones, M. Schonlau and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.
- [33] A. I. Khuri and S. Mukhopadhyay. Response surface methodology. *WIREs Computational Statistics*, 2:128–149, 2010.
- [34] André I. Khuri. *Response Surface Methodology and Related Topics*. World Scientific Publishing Co., 2006.
- [35] Min-Jae Kim, Jaewon Lim, Jang-Ho Seo and Hyun-Kyo Jung. Hybrid optimization strategy using response surface methodology and genetic algorithm for reducing cogging torque of spm. *Journal of Electrical Engineering and Technology*, 6:202–207, 2011.
- [36] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi. Optim. simulated annealing. *Science*, 220:671–680, 1983.
- [37] J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput.*, 10:50–66, 2006.
- [38] Slawomir Koziel and Leifur Leifsson. *Surrogate-Based Modeling and Optimization*. Springer, 2013.
- [39] Enying Li and Hu Wang. An alternative adaptive differential evolutionary algorithm assisted by expected improvement criterion and cut-hdmr expansion and its application in time-based sheet forming design. *Advances in Engineering Software*, 97:96–107, 2016.
- [40] Heping Liu and Saeed Maghsoodloo. Simulation optimization based on taylor kriging and evolutionary algorithm. *Applied Soft Computing*, 11:3451–3462, 2011.
- [41] Dorigo M. Optimization, learning and natural algorithms. *Department of Electronics, Polytechnic of Milan, Italy*, 1992.

- [42] M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33:161–174, 1991.
- [43] R. H. Myers, D. C. Montgomery and C. M. Anderson-Cook. *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, 2016.
- [44] Özgür Yeniay. Comparative study of algorithms for response surface optimization. *Mathematical and Computational Applications*, 19:93–104, 2014.
- [45] Elmar Plischke, Emanuele Borgonovo and Curtis L. Smith. Global sensitivity measures from given data. *European Journal of Operational Research*, 226:536–550, 2013.
- [46] P. M. Reed, D. Hadka, J. D. Herman, J. R. Kasprzyk and J. B. Kollat. Evolutionary multiobjective optimization in water resources: The past, present, and future. *Advances in Water Resources*, 51:438–456, 2013.
- [47] B. C. Routara, A. K. Sahoo, A. K. Parida and P. C. Padhi. Response surface methodology and genetic algorithm used to optimize the cutting condition for surface roughness parameters in cnc turning. *Procedia Engineering*, 38:1893–1904, 2012.
- [48] R. Roy, S. Himduja and R. Teti. Recent advances in engineering design optimisation: Challenges and future trends. *CIRP Annals - Manufacturing Technology*, 57:697–715, 2008.
- [49] Narayana Saibaba and Pulipati King. Modelling and optimization of dye removal process using hybrid response surface methodology and genetic algorithm approach. *Fundamentals of Renewable Energy and Applications*, 2013.
- [50] A. Saltelli, Â. Guimarães Pereira, J. P. Van der Sluijs and S. Funtowicz. What do i make of your latinorum? sensitivity auditing of mathematical modelling. *Int. J. Foresight and Innovation Policy*, 9(2/3/4):213–234, 2013.
- [51] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana and S. Tarantola. *Global Sensitivity Analysis. The Primer*. John Wiley and Sons, 2008.
- [52] Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145:280–297, 2002.
- [53] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto and Stefano Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer Physics Communications*, 181:259–270, 2010.
- [54] Andrea Saltelli and Marco Ratto Stefano Tarantola, Francesca Campolongo. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley and Sons, 2004.

- [55] Jie Shao, Leiquan Wang, Zhicheng Zhao, Fei Su and Anni Cai. Deep canonical correlation analysis with progressive and hypergraph learning for cross-modal retrieval. *Neurocomputing*, 214:618–628, 2016.
- [56] I. M. Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55:271–280, 2001.
- [57] I. M. Sobol and Kucherenko S. Derivative based global sensitivity measures and their link with global sensitivity indices. *Mathematics and Computers in Simulation*, 79:3009–3017, 2009.
- [58] G. Steenackers, F. Presezniak and P. Guillaume. Development of an adaptive response surface method for optimization of computation-intensive models. *Computers and Industrial Engineering*, 57:847–855, 2009.
- [59] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.
- [60] Ye Tian, Ran Cheng, Xingyi Zhang and Yaochu Jin. Platemo: A matlab platform for evolutionary multi-objective optimization. *Neural and Evolutionary Computing*, 2017.
- [61] A. Toffolo and E. Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11:151–167, February 2003.
- [62] G. Venturelli, E. Benini and Ł. Łaniewski-Wołk. A Kriging-assisted multiobjective evolutionary algorithm. *Applied Soft Computing*, 58:155–175, 2017.
- [63] Wikipedia. Crossover (genetic algorithm) — Wikipedia, the free encyclopedia, 2017. [Online; accessed 16-November-2017].
- [64] Wikipedia. Low-discrepancy sequence — Wikipedia, the free encyclopedia, 2017. [Online; accessed 16-November-2017].
- [65] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1, 1997.
- [66] Hui Xiong and Pengjian Shang. Weighted multifractal cross-correlation analysis based on shannon entropy. *Commun Nonlinear Sci Numer Simulat*, 30:268–283, 2016.
- [67] E. Zitzler, K. Deb and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.