Metamodel-Based Multidisciplinary Design Optimization for Automotive Applications

Ann-Britt Ryberg       Rebecka Domeij Bäckryd       Larsgunnar Nilsson

Division of Solid Mechanics
Linköping University
Linköping
September 2012
Technical Report LIU-IEI-R-12/003

Metamodel-Based Multidisciplinary Design Optimization for Automotive Applications

Ann-Britt Ryberg
Rebecka Domeij Bäckryd
Lars Gunnar Nilsson

Division of Solid Mechanics
Linköping University
Linköping

September 2012
Abstract

When designing a complex product, many groups are concurrently developing different parts or aspects of the product using detailed simulation models. Multidisciplinary design optimization (MDO) has its roots within the aerospace industry and can effectively improve designs through simultaneously considering different aspects of the product. The groups involved in MDO need to work autonomously and in parallel, which influence the choice of MDO method. The methods can be divided into single-level methods that have a central optimizer making all design decisions, and multi-level methods that have a distributed decision process.

This report is a comprehensive summary of the field of MDO with special focus on structural optimization for automotive applications using metamodels. Metamodels are simplified models of the computationally expensive detailed simulation models and can be used to relieve some of the computational burden during MDO studies. The report covers metamodel-based design optimization including design of experiments, variable screening, metamodels and their validation, as well as optimization methods. It also includes descriptions of several MDO methods, along with a comparison between the aerospace and automotive industries and their applications of MDO.

The information in this report is based on an extensive literature survey, but the conclusions drawn are influenced by the authors’ own experiences from the automotive industry. The trend goes towards using advanced metamodels and global optimization methods for the considered applications. Further on, many of the MDO methods developed for the aerospace industry are unsuitable for the automotive industry where the disciplines are more loosely coupled. The expense of using multi-level optimization methods is then greater than the benefits, and the authors therefore recommend single-level methods for most automotive applications.

**Keywords:** multidisciplinary design optimization (MDO), metamodel-based design optimization (MBDO), single-level optimization methods, multi-level optimization methods, automotive industry
Contents

1 Introduction .......................................................................................................................... 1
2 Background .......................................................................................................................... 3
3 Optimization Definitions ..................................................................................................... 5
  3.1 Optimization ................................................................................................................... 5
  3.2 Structural Optimization ................................................................................................. 6
  3.3 Multi-Objective Optimization ......................................................................................... 6
  3.4 Probabilistic-Based Design Optimization .................................................................... 7
  3.5 Multidisciplinary Design Optimization ....................................................................... 7
4 Metamodel-Based Design Optimization ............................................................................. 9
  4.1 Basics ............................................................................................................................ 10
     4.1.1 Process .................................................................................................................. 11
     4.1.2 Basic Statistics ....................................................................................................... 12
     4.1.3 Nomenclature ........................................................................................................ 13
  4.2 Design of Experiments ................................................................................................. 14
     4.2.1 Classical Experimental Designs ............................................................................. 14
     4.2.2 Experimental Designs for Complex Metamodels .................................................. 16
     4.2.3 Sampling Size and Sequential Sampling ............................................................... 20
  4.3 Variable Screening ......................................................................................................... 22
     4.3.1 One-Factor-at-a-Time Plans .................................................................................. 22
     4.3.2 Analysis of Variance .............................................................................................. 22
     4.3.3 Global Sensitivity Analysis .................................................................................... 23
  4.4 Metamodels .................................................................................................................... 25
     4.4.1 Polynomial Regression .......................................................................................... 27
     4.4.2 Moving Least Squares ........................................................................................... 28
     4.4.3 Kriging ................................................................................................................... 29
     4.4.4 Artificial Neural Networks ..................................................................................... 32
     4.4.5 Radial Basis Functions and Radial Basis Function Networks ................................ 35
     4.4.6 Multivariate Adaptive Regression Splines .............................................................. 39
     4.4.7 Support Vector Regression .................................................................................... 41
     4.4.8 Which Metamodel to Use? ..................................................................................... 43
  4.5 Metamodel Validation ...................................................................................................... 45
     4.5.1 Error Measures ...................................................................................................... 45
     4.5.2 Cross Validation ..................................................................................................... 48
     4.5.3 Jack-knifing and Bootstrapping ............................................................................. 51
     4.5.4 Generalized Cross Validation and Akaike's Final Prediction Error ....................... 51
  4.6 Optimization Methods ..................................................................................................... 52
     4.6.1 Optimization Strategies ......................................................................................... 52
     4.6.2 Optimization Algorithm Classification ................................................................. 56
     4.6.3 Gradient-Based Algorithms .................................................................................... 57
     4.6.4 Evolutionary Algorithms ....................................................................................... 60
     4.6.5 Particle Swarm Optimization .................................................................................. 63
     4.6.6 Simulated Annealing ............................................................................................. 64
     4.6.7 Multi-Objective Optimization ................................................................................ 66
5 Multidisciplinary Design Optimization Methods ............................................................ 73
  5.1 Problem Decomposition ................................................................................................. 73
     5.1.1 Terminology of Decomposed Systems .................................................................. 73
     5.1.2 Aspect- and Object-Based Decomposition ............................................................ 76
     5.1.3 Hierarchic and Non-Hierarchic Systems ............................................................... 76
     5.1.4 Coupling Breadth and Coupling Strength .............................................................. 77
1 Introduction

In a large scale industrial product development process, several design groups are responsible for different aspects or parts of the product. For a complex product, such as a car, the aspects or parts cannot be considered isolated entities as they mutually influence one another. The groups must therefore interact during the development. Traditionally, the goal of the design process has been to meet a certain number of requirements by repeated parallel development phases with intermediate synchronizations between the groups. Solving the problem using a traditional approach leads to a feasible solution, but probably not to an optimal one. The goal of multidisciplinary design optimization (MDO) is to find the optimal design for a complex problem using a formalized optimization methodology. To implement MDO as an industrial standard activity, the individual groups need to stay autonomous and work in parallel, which puts restrictions on the choice of methods.

The aim of this report is to explore the state-of-the-art within metamodel-based multidisciplinary design optimization, with special focus on automotive applications. The starting point is an extensive literature survey, but some of the parts are influenced by the authors’ own experiences from the automotive industry. The information is used to set up the framework for the authors’ continued research on the subject, which includes the development of an efficient MDO methodology for automotive development processes.

This document consists of several chapters. The first ones are devoted to a short background of the field and to the introduction of some important concepts. Chapters 4 and 5 constitute the main part of the report, and describe metamodel-based design optimization and MDO methods. These chapters can be read independently of each other and serve as summaries of the two areas. Chapter 6 briefly explains the product development process and MDO experiences within the automotive industry and it also compares the automotive and aerospace industries. The last chapter concludes the report by giving a summary of the two main chapters and recommendations regarding suitable methods.
Introduction
2 Background

Historically, MDO evolved as a new engineering discipline in the area of structural optimization, mainly within the aerospace industry, as described by Agte et al. (2010). Disciplines strongly interacting with the structural parts were included in the optimization problem, making the optimization multidisciplinary. The development has been heading towards incorporating the whole system in the MDO, i.e. also including design variables important for other disciplines than the structural ones.

Kroo and Manning (2000) describe the development of MDO in terms of three generations. Initially, all disciplines were integrated into a single optimization loop. As the MDO problem size grew, the second generation of MDO methods was developed. Analyses were distributed but coordinated by an optimizer. Both the first and second generations of MDO methods are so called single-level optimization methods, meaning that they rely on a central optimizer making all the design decisions. When MDO was applied to even larger problems involving several departments of a company, the need for distributing the decision making process became apparent. The third generation of MDO methods includes the so called multi-level optimization methods, where the optimization process as such is distributed. These different approaches are illustrated in Figure 2.1.

The major MDO users can be found within the aerospace and automotive industries according to Agte et al. (2010). Alexandrov (2005) concludes that the use of MDO in industry is smaller than was first expected as the problems MDO tries to solve are found to be very complex.

When designing complex products such as vehicles, detailed simulation models are required to evaluate and improve the design during the development. These detailed simulation models are often time-consuming to run. Furthermore, gradient information from the simulation models may be unavailable or spurious. In these cases, metamodel-based design optimization can be an alternative. Metamodels are simplified models of the detailed simulation models with smooth gradients and evaluations using metamodels are fast compared to evaluations using the detailed models. Metamodels are developed based on a series of runs of the detailed simulation models. One benefit of using metamodels within MDO is that the people responsible for different disciplines can work in parallel when developing the metamodels and verify their accuracy before the optimization process starts. Since metamodels are approximations of the detailed simulation models, an extra source of error is introduced and the challenge is to keep this error on an acceptable level for the problem at hand.
Background
3 Optimization Definitions

Different aspects of optimization related to MDO are introduced in this chapter and a general optimization problem is defined. Structural optimization is of interest to the automotive industry and is therefore given special attention. The concepts of multi-objective optimization and probabilistic-based design optimization are discussed. Finally, multidisciplinary design optimization is defined.

3.1 Optimization

A general optimization problem, or mathematical programming problem, can be formulated as:

\[
\begin{align*}
\min_{x} & \quad f(x) \\
\text{subject to} & \quad g(x) \leq 0 \\
& \quad h(x) = 0 \\
& \quad x_{\text{lower}} \leq x \leq x_{\text{upper}}
\end{align*}
\]  

(3.1)

The goal is to find the values of the \textit{design variables} \(x\) that minimize the \textit{objective function} \(f\). In general, the optimization problem has a number of inequality and equality \textit{constraints} that need to be fulfilled, represented by the vectors \(g\) and \(h\). The objective and constraint functions depend on the design variables \(x\) on which there are upper and lower limits, called \(x_{\text{upper}}\) and \(x_{\text{lower}}\), respectively. The design variables can be \textit{continuous} or \textit{discrete}, meaning that they can take any value, or only certain discrete values, between the upper and lower limits. Design points that fulfil all constraints are \textit{feasible}, while all other design points are \textit{unfeasible}. An \textit{unconstrained} optimization problem lacks constraints, as opposed to a \textit{constrained} optimization problem. The problem is a \textit{linear programming (LP)} problem if the objective and constraint functions are linear functions of the design variables, and a \textit{non-linear programming (NLP)} problem if the objective function or any of the constraint functions are non-linear. The formulation in Equation (3.1) also allows for maximization problems as \(\max f(x)\) can be replaced by \(\min (-f(x))\).

The general formulation can be recast into the simpler form:

\[
\begin{align*}
\min_{x} & \quad f(x) \\
\text{subject to} & \quad g(x) \leq 0
\end{align*}
\]  

(3.2)

In this latter formulation, the inequality constraints \(g\) contain all three types of constraints in the former formulation. This is achieved by replacing each equality constraint by two inequality constraints and by including these, together with the upper and lower limits on the design variables, in the constraint vector \(g\).

The solution of an optimization problem is called the optimum solution. Optimization problems can be solved by numerical techniques, consisting of iterative search processes that make use of information from past iterations. Different optimization methods are described in Section 4.6. When evaluating the objective and constraint functions in different design points during the solution process, one or several \textit{analyzers} are used. An analyzer can be an analytical function for a simple optimization problem, while it can be some kind of model that is described by governing equations for a more complex problem, e.g. a finite element model. It can also be a metamodel that describes the more complex model. For a vector of design variables \(x\), the analyzer(s) return a number of \textit{responses} denoted by \(y\). These responses can be combined into the objective and constraint functions for that specific vector of design variables.
3.2 Structural Optimization

Multidisciplinary design optimization evolved in the area of \textit{structural optimization}, which has been a field of intensive research since the 1960’s. According to Gallagher (1973, p. 7): “\textit{Structural optimization seeks the selection of design variables to achieve, within the limits (constraints) placed on the structural behaviour, geometry, or other factors, its goal of optimality defined by the objective function for specified loading or environmental conditions.}” Structural optimization is of great interest within the automotive industry, where the mass is typically minimized subject to a number of performance constraints.

Three types of structural optimization can be distinguished: size, shape, and topology optimization. In \textit{size optimization}, the design variables represent some kind of structural property, e.g. sheet thickness in the different parts of a car. In \textit{shape optimization} on the other hand, the design variables represent the shape of material boundaries. \textit{Topology optimization} is the most general form of structural optimization which is used to find where material should be placed to be most effective.

3.3 Multi-Objective Optimization

The optimization problem defined in Equation (3.2) is a \textit{single-objective optimization} problem. It has one objective function that is to be minimized. When solving \textit{multi-objective optimization (MOO)} problems, also called multi-criteria optimization problems, two or more objective functions are simultaneously being minimized. An optimization problem containing $m$ objective functions is formulated as:

$$
\min_{x} \quad f_1(x), \ldots, f_m(x)
$$

subject to $g(x) \leq 0$ \hspace{1cm} (3.3)

The simplest approach to solve a multi-objective optimization problem is to convert it into a single-objective optimization problem. There are two intuitive ways of doing this according to Haftka and Gürdal (1992). The first procedure is to minimize one of the objective functions, typically the most important one, and to treat all the others as constraints. The multiple objectives are then bypassed in the solution process. The second approach is to create one single objective function as a combination of the original objectives. Weight coefficients can then be used to mirror the relative importance of the original objective functions.

The drawback of the aforementioned methods is that one single optimum is found. If the designer wants to modify the relative importance of the objective functions in retrospect, the optimization process must be performed once again. An alternative is to find a number of \textit{Pareto optimal} solutions. A point is Pareto optimal if there is no other feasible point yielding a lower value of one objective function without increasing the value of at least one other objective function, as stated by for example Papalambros and Wilde (2000). The designer will then have a set of points to choose among, and the trade-off between the different objective functions can be performed after the optimization process has been carried out. An illustration for a problem with two objective functions can be found in Figure 3.1. Pareto optimal solutions can for example be found using evolutionary algorithms. The subject of multi-objective optimization in general and Pareto optimal solutions in particular is further elaborated upon in Section 4.6.7.
3.4 Probabilistic-Based Design Optimization

When designing a product, it can be of importance to deal with uncertainties in the design variables through performing probabilistic-based design optimization. This is in contrast to deterministic design optimization where uncertainties in the design variables are not considered. Zang et al. (2005) distinguish between two different branches within the area of probabilistic-based design optimization: robust design optimization and reliability-based design optimization.

In robust design optimization (RDO), a product that performs well and is insensitive to variations in the design variables is sought. This can be achieved by making a trade-off between the mean value and the variation of the product performance. In reliability-based design optimization (RBDO) on the other hand, the probability distribution of the product performance is calculated. The probability of failure is typically constrained to be below a certain level. Large variation in the performance of the product can thus be allowed as long as the probability of failure is low.

3.5 Multidisciplinary Design Optimization

Giesing and Barthelemy (1998, p. 2) provide the following definition of multidisciplinary design optimization: “A methodology for the design of complex engineering systems and subsystems that coherently exploits the synergism of mutually interacting phenomena.” In general, a better design can be found when considering the interactions between different aspects or parts of a product than when considering them as isolated entities, which is taken advantage of using MDO.

Traditionally, MDO is used to optimize a system, subsystem, or component in a product considering two or more disciplines simultaneously. A discipline can be said to be an aspect of the product, e.g. safety or aerodynamics within the automotive industry. Within one discipline, many different load cases can be considered. A load case is a specific configuration that is evaluated using an analyzer, e.g. a simulation of a crash scenario using a finite element model. The MDO methodology can just as well be applied to different load cases within one single discipline, and the problem is then not truly multidisciplinary. However, the idea of finding a better solution by taking advantage of the interactions between subproblems still remains.
A complex product cannot be fully understood by one single engineer, but by the collective
knowledge in all design groups. The product development process therefore needs to take advantage
of the skills that exist in the different groups and enable them to work on the problem in parallel. The
need for autonomy and parallelism must be taken into account when developing MDO methods.
Multidisciplinary design optimization is a tool that helps the engineers to explore the design space, as
stated by Alexandrov (2005). It should not be used to provide the complete design without human
intervention. The interaction between the designers and the MDO tool is fundamental.
4 Metamodel-Based Design Optimization

It is called *metamodel-based design optimization (MBDO)* when metamodels are used for the evaluations during the optimization process. There are several descriptions on MBDO, see for example Simpson et al. (2001), Queipo et al. (2005), Wang and Shan (2007), Forrester and Keane (2009), and Stander et al. (2010). This chapter is a summary of the most common definitions and methods, and the chapter is intended as background knowledge for metamodel-based MDO.

The design of complex products requires extensive investigations regarding the response of the product due to external loads. This could be done by physical experiments or computer simulations. In recent years, increased focus has been put on detailed computer simulations. However, these simulations can be very demanding from a computational point of view. Therefore, in many situations, e.g. during optimization of product performance, there is a need for a simplified model that could provide an efficient representation of the detailed and costly model of the product. These simplified models are called *surrogate models*. If the model is a surrogate for a detailed simulation model it is called a *metamodel*. Since this document focuses on optimization based on simulations, the term metamodel will be used throughout the text.

Metamodels are created by a mathematical description based on a dataset of input and the corresponding output from the detailed simulation model, see Figure 4.1. The mathematical description, i.e. metamodel type, suitable for the approximation could vary depending on the intended use or the underlying physics that the model should capture. Different datasets are appropriate for building different metamodels. The process of where to place the design points in the design space, i.e. the input settings for the dataset, is called design of experiments (DOE). Traditionally, the metamodels have been simple polynomials, but other metamodels that are better at capturing complex responses increase in popularity.

*Figure 4.1* The concept of metamodelling for a response depending on two design variables.

Before using the metamodels, it is important to know the accuracy of the model, i.e. how well the metamodel represents the underlying detailed simulation model. This could be done by studying different error measures. When the metamodel is found to be accurate enough, it can be used for optimization studies. Several methods exist for finding the optimal solution. Some of these methods will later be explained in more detail, as well as different metamodel types, DOEs, and error measures.
There are several reasons for using metamodels in optimization studies, see for example Wang and Shan (2007). One important reason is, as mentioned earlier, the computational time. In an optimization process, many design evaluations often need to be performed to find an optimum. If the detailed model could be replaced by a simple mathematical model, often thousands of evaluations could be performed in the same time as it would take to run only one detailed simulation. Roughly speaking, if accurate metamodels can be built from fewer detailed simulations than the number of evaluations required in the optimization process, the total cpu-time for the study will be reduced. In general, the detailed simulations needed to build the metamodels could be run in parallel instead of in sequence, as required by many optimization algorithms. Consequently, also the wall-clock time will be considerably reduced. The time saved will be most pronounced in optimization processes that require very many evaluations, e.g. multi-objective optimization and reliability-based design optimization. Another reason for using metamodel-based design optimization could, in fact, be the quality of the optimization results. Building metamodels may filter physical high frequency and numerical noise and hence make it easier to find the global optimum. Metamodels could also make it possible to use advanced optimization algorithms which are better suited for finding global optima but require many evaluations, see Section 4.6. In addition, metamodels render a view of the entire design space and might also make it easier to detect errors in the simulation model since the entire design region is analysed. When the metamodels are built, it is also inexpensive to rerun optimizations, e.g. with changed constraint limits. This makes it possible to investigate multiple scenarios almost without any additional cost. One further benefit with metamodels, when used in multidisciplinary design optimization, is the possibility for disciplinary autonomy. The different simulation experts can be responsible for establishing the metamodels for their respective disciplines and loadcases, and for the validity of these metamodels. The development of the metamodels can be done in parallel, making the work efficient. Concurrency and autonomy are two of the main drivers for the various MDO multi-level optimization methods proposed, and metamodels could thus serve as a kind of decomposition method that have similar positive effects.

The main drawback of using metamodels in optimization studies are the introduction of an additional source of error. The metamodels are approximations of the detailed simulation models and to be useful they need to be accurate enough. In general, the more evaluations from the detailed simulation model that are available, the more accurate metamodels can be built. The time to build the metamodels will, however, increase accordingly. There are many types of metamodels to choose from and many other decisions that need to be made in order to build a good metamodel. This means that additional knowledge is required among the people involved in the optimization work and that suitable software must be available.

4.1 Basics

Metamodel-based design optimization has been used in many engineering applications in a variety of ways. In this section, a general process is described followed by a very short description of some basic statistical terms and the general nomenclature used in the chapter. This introductory section is then followed by sections describing the steps of MBDO in more detail.
4.1.1 Process
The MBDO process can be summarized in distinctive steps, see for example Stander et al. (2010), Wang and Shan (2007), Simpson et al. (2001), and Figure 4.2.

A prerequisite for a successful MBDO study is to have a stable detailed model that accurately captures the behaviour of the product. A model could be regarded as stable if it could be run without crashes when the design variables are varied within the studied intervals, and when variations in output mainly are caused by input changes rather than numerical variations.

The first step is then to mathematically define the optimization problem, i.e. define the objective(s), the design variables and the range within which the variables could vary (= the design space) and, in most cases, define some constraints. Since the number of simulations needed to build an accurate metamodel, to a large extent, is based on the number of design variables, the logical next step is to identify the variables influencing the studied responses the most. This, the so called variable screening, can be done with a limited number of simulations using the detailed model. Next, a suitable DOE needs to be selected, the simulations run, and the responses extracted. The metamodels could then be built from the available dataset and the accuracy of the models should carefully be checked. Sometimes it is useful to build more than one metamodel for each response and choose the best one. When the metamodels are found to satisfy the requirements, the optimization can be performed. Based on the optimization results, one or more potential designs can be selected and verified using the detailed simulation model.

If, during the described process, the results of one step is not acceptable one needs to go back to a previous step and refine. If, e.g. the accuracy of the metamodels is not acceptable, additional designs might be added to the DOE and new simulations run. It could also be worthwhile to increase the level of detail of the metamodels by additional simulations using the detailed model in design regions found to be interesting. In this way the optimization process is often an iterative procedure.

The method described above assumes that one metamodel is used for the complete design space. This often requires the use of complex metamodels. Another method could be to use a simpler form of metamodel, e.g. a linear polynomial, that is sequentially built in an iterative process around the
found optimum. The metamodels are then only fitted to a part of the design space, called the region of interest, which is moved and shrunk as the iterative optimization process progresses.

### 4.1.2 Basic Statistics

Although this text concerns metamodels based on deterministic simulations, where a repetition of a detailed simulation with the same model and input is assumed to give the same responses, it is unavoidable to encounter some basic statistical concepts. A short summary of fundamental statistical terminology could therefore be useful.

A phenomenon is random, or stochastic, if individual outcomes are uncertain but there is a regular distribution of outcomes for a large number of repetitions. A random variable is consequently a variable whose value is an outcome from a random phenomenon. Many random variables are assumed to be normally distributed, i.e. follow the "bell-shaped" distribution known as the Gaussian function, see Figure 4.3. The probability of any outcome of a random phenomenon is the long term relative frequency, i.e. the proportion of the times the outcome would occur in a very long series of independent repetitions.

The expected value of a random variable is denoted by $E[x]$. It can be thought of as the “long term average” value attained by the random variable. The expected value of a random variable is also called its mean, $\mu_x$, and hence $E[x] = \mu_x$. The expected value of a discrete random variable is found from

$$
\mu_x = E[x] = \sum_{i=1}^{n} x_i P_x(x_i) = \frac{\sum_{i=1}^{n} x_i P_x(x_i)}{\sum_{i=1}^{n} P_x(x_i)}
$$

(4.1)

The expected value is thus the sum over all possible values $x_i$ of $x$, multiplied with its probability $P_x(x_i)$. Since all $P_x(x_i)$ add up to 1, the expected value could be viewed as a weighted sum with the weights $P_x(x_i)$, as indicated in the right part of Equation (4.1). In the case of a continuous variable, an integral of $x$ multiplied with its probability density function gives the expected value.

![Figure 4.3](image)

**Figure 4.3** Probability density function of the normal distribution, also called the Gaussian distribution.
The **variance** of a random variable $x$ is denoted by either $\text{Var}[x]$ or $\sigma_x^2$. The variance is defined by

$$\text{Var}[x] = \sigma_x^2 = E[(x - \mu_x)^2] = E[x^2] - (E[x])^2$$

(4.2)

The **covariance** is defined as

$$\text{Cov}[x, y] = E[(x - \mu_x)(y - \mu_y)] = E[(x - E[x])(y - E[y])]$$

(4.3)

where both $x$ and $y$ are random variables. The covariance is hence a measure of how much two variables change together and the variance is the special case of the covariance when the two variables are identical. The standard deviation $\sigma_x$ is the square root of the variance and gives another measure on how much variation there is from the expected value. A low standard deviation indicates that the data points tend to be very close to the mean, whereas a high standard deviation indicates that the data are spread out over a large range of values.

If the elements of a vector $\mathbf{x}$ are random variables $x_1, ..., x_k$, the **covariance matrix** $\Sigma$ is a matrix whose elements $\Sigma_{ij}$ is the covariance between the $i^{th}$ and $j^{th}$ elements of $\mathbf{x}$.

$$\Sigma_{ij} = \text{Cov}(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)]$$

(4.4)

The $k$ diagonal elements of $\Sigma$ are consequently the variances of all the $k$ variables.

The **correlation** $R$ between two variables $x_i$ and $x_j$ are defined by

$$R = \text{Cor}(x_i, x_j) = \frac{\text{Cov}(x_i, x_j)}{\sigma_{x_i}\sigma_{x_j}} = \frac{\text{Cov}(x_i, x_j)}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}$$

(4.5)

and varies between -1 and 1. The correlation indicates the linear relationship between two variables and the closer to either -1 (negative correlation) or 1 (positive correlation), the stronger the linear dependence is between the variables. As the correlation approaches zero, there is less of a relationship between the variables.

### 4.1.3 Nomenclature

In order to make the presentation clear, a consistent nomenclature is generally used throughout this chapter. Although the notation is expressed when used, the table below can serve as a helpful summary of the most frequently used symbols.

**Table 4.1** List of commonly used symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, \mathbf{x}$</td>
<td>Design variable, vector of design variables</td>
</tr>
<tr>
<td>$y, \mathbf{y}$</td>
<td>Response, vector of responses</td>
</tr>
<tr>
<td>$\hat{y}$</td>
<td>Estimated response from metamodel</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of designs / samples in a data set</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of design variables</td>
</tr>
<tr>
<td>$p$</td>
<td>Number of regression coefficients</td>
</tr>
</tbody>
</table>
4.2 Design of Experiments

In order to build a metamodel, a dataset of input (design variable settings) and corresponding output (response values) is needed. The theory on where these design points should be placed in the design space in order to get the best possible information from a limited sample size is called design of experiments (DOE). The classical experimental designs are primarily used for screening purposes and as a base for building polynomial metamodels. When the dataset is used to fit a more complex metamodel, other experimental designs are preferred.

4.2.1 Classical Experimental Designs

The theories of design of experiments originate from planning physical experiments. The idea is to gain as much information as possible from a limited number of experiments. The methods focus on planning the experiments so that the random error from the physical experiments has minimum influence in the approval or disapproval of a hypothesis. Popular designs include factorial or fractional factorial designs, central composite designs, Box-Behnken designs, Plackett-Burman designs, Koshal designs, and D-optimal designs, see e.g. Myers et al. (2008).

The information gained from the experiments is often used to identify the influence on the response caused by variable changes. The result of changing one single variable is called main effect and the result of changing more than one variable at the same time is called interaction effect. Commonly, an approximate polynomial model of the true response is developed, see Section 4.4.1.

Factorial design is a $l^k$ grid of designs where $l$ is the number of levels in one dimension and $k$ is the number of variables, also called factors. The most common are $2^k$ designs for evaluating main effects and interactions, and $3^k$ designs for evaluating main and quadratic effects as well as interactions. The size of the designs increases exponentially with the number of factors and therefore fractional factorial designs ($l^{k−r}$) are often used when experiments are costly and many factors are required.

The reduction of the design size means, however, that some effects and/or interactions are aliased with each other, i.e. cannot be estimated independently. It is therefore important to choose a fractional factorial design that allows for independent estimations of the main effects and interactions that are assumed to be important, as described by Myers et al. (2008). A fractional factorial design can always be augmented by additional points to a higher resolution fraction, where more main effects and interactions can be estimated independently, or to a full factorial where all main effects and interactions can be estimated.

When there are many factors, the system is often assumed to be dominated by main effects and low order interactions (scarcity-of-effects principle). Often $2^k$ or $2^{k−r}$ designs are used to identify important factors, i.e. variable screening. One specific family of fractional factorial designs frequently used for screening is the two-level Plackett-Burman designs. Some of these designs are saturated, i.e. the number of design points is equal to one more than the number of factors to be estimated. Saturated fractional factorial designs allow independent estimation of all main effects if the interactions are negligible.

Another class of small designs is the family of Koshal designs which are saturated for fitting any polynomial model of order $d$ ($d = 1, 2, ...$). The first order model is simply a one-factor-at-a-time design which could be used to estimate the main effects. The Koshal design for fitting a second order model, see Figure 4.4, includes ten points and all ten coefficients of the second order model could be
estimated. It should, however, be noted that since all the points are needed to estimate the model parameters no information is left to check the model accuracy (lack-of-fit).

To fit a linear model, two levels of each variable are needed. If instead, a second order polynomial should be used, a minimum of three levels are needed for each variable. A $3^k$ or $3^k-r$ design can be used but requires often too many design points. The most common class of designs for fitting a second order model with limited number of design points is instead the central composite designs (CCDs). The CCD is a two level ($2^k$ or $2^k-r$) factorial design, augmented by $n_c$ centre runs and axial runs, see Figure 4.4. The distance from the centre to the axial points, $\alpha$, and the number of centre runs are selected to get different properties of the design. It should be noted that $n_c > 1$ is not relevant for studies with deterministic simulations. Another popular design is the Box-Behnken design (BBD) which is formed by $n_c$ centre runs and blocks of $2^2$ designs at which the other factors are held constant. Since the BBD does not have any design points at the vertices of the hypercube, the BBD is not a good choice if predictions of the response at the extremes are important.

\begin{align*}
\text{a) Full factorial,} & \quad n = 27 \\
\text{b) CCD,} & \quad n = 15 \\
\text{c) BBD,} & \quad n = 13 \\
\text{d) Koshal,} & \quad n = 10
\end{align*}

\text{Figure 4.4 Experimental designs in three variables for fitting second order models, starting with a) full factorial design requiring many evaluations and then examples of three more economical designs, b) central composite design, c) Box-Behnken design, and d) Koshal design.}

Different criteria can be used to evaluate the experimental designs. Some criteria focus on good estimation of model parameters while others focus on good prediction in the design region. The most well-known and often used criterion is the D-optimality, which focuses on good model parameter estimation, but also A-, G-, V-, and I-optimality could be studied as described by Myers et al. (2008).

A design is said to be D-optimal if the determinant of the so-called moment matrix $|M|$ is maximized.

\begin{equation}
|M| = \frac{|X^T X|}{n^p}
\end{equation}

where $X$ is the model matrix which has $n$ rows, one for each design point, and $p$ columns, one for each coefficient to be estimated (see Section 4.4.1 for more details). The D-efficiency could be used to compare designs of different sizes and is comparing the design at hand against a D-optimal one.

\begin{equation}
D_{\text{eff}} = \left( \frac{|X^T X|_{\text{design}}}{|X^T X|_{D-\text{optimal}}} \right)^{1/p}
\end{equation}
Generating a D-optimal design is an optimization task in which a computer algorithm chooses the best set of design points in order to maximize \( |X^T X| \). The total number of design points to be used and the model that should be fitted are given as input. Often the algorithm chooses the best possible subset from a candidate set, which usually is a full factorial design. Another method is to start from a random design of the correct size and then adjust the positions of the design points.

The D-optimality criterion can not only be used when generating a DOE from scratch but also when augmenting an existing design with additional points.

### 4.2.2 Experimental Designs for Complex Metamodels

As mentioned earlier, the classical experimental designs focus on reducing the effect of noise in physical experiments. They also tend to spread the sample points around the border and only put a few points in the interior of the design space. The DOE for computer experiments needs to consider the fact that computer models are deterministic, i.e., will give the same result for a specific set of input each time, assuming numerical noise is negligible. This means that repeated runs are not needed. Often many design variables are studied over a large design space and generally a complex metamodel should be fitted. There seem to be a consensus among scientists that a proper experimental design for these cases should be **space-filling**, which aims to spread the design points within the complete design space. This is desired when the form of the metamodel is unknown and when interesting phenomena can be found in different regions of the design space. Space-filling designs allow a large number of levels for each variable with a moderate number of experimental points. These designs are especially useful in conjunction with non-parametric metamodels (such as neural networks) and Kriging (see Section 4.4).

The first space filling design, the **Latin hypercube sampling (LHS)**, was proposed by McKay et al. (1979) and is a constrained random design. For each of the \( k \) variables the range of each variable is divided into \( n \) non-overlapping intervals of equal probability. One value from each interval is selected at random but with respect to the probability density in the interval. The \( n \) values of the first variable are then paired randomly with the \( n \) values of the second variable. These \( n \) pairs are combined randomly with the \( n \) values of the third variable to form \( n \) triplets, and so on, until \( n \) \( k \)-tuplets are formed, see Swiler and Wyss (2004) for a detailed description. This result in an \( n \times k \) sampling plan matrix \( S \), where the \( k \) columns describe the levels of each variable, and the \( n \) rows describe the variable settings for each design, see Figure 4.5.

Mathematically, this could be described as a basic sampling plan matrix \( X \) with elements

\[
X_{ij} = \frac{\pi_j(i) - U_{ij}}{n}, \quad 1 \leq i \leq n, \quad 1 \leq j \leq k
\]  

(4.8)

where \( \pi_j(1), \ldots, \pi_j(n) \) are independent uniform random permutations of the integers 1 to \( n \) and \( U_{ij} \) are independent uniformly distributed random variables between 0 and 1, independent of \( \pi_j \). Each element of \( X \) is then mapped according to its marginal distribution to get the final sampling plan \( S \)

\[
S_{ij} = F_{X_{ij}}^{-1}(X_{ij})
\]  

(4.9)

where \( F_{X_{ij}}^{-1} \) represent the inverse of the target cumulative distribution function for variable \( j \).
A common variant of LHS is the *median Latin hypercube sampling (MLHS)* or *lattice sample* which has points from the centre of the $n$ intervals and hence

$$ X_{ij} = \frac{\pi_j(i) - 0.5}{n}, \quad 1 \leq i \leq n, \quad 1 \leq j \leq k \tag{4.10} $$

Figure 4.5 Latin hypercube sampling for two variables at five levels, one normally distributed variable and the other uniformly distributed.

In order to generate a better space filling design, the LHS can be taken as a starting design and the values of each column in the $n \times k$ matrix are then permuted to optimize some criterion. One approach is to maximize the minimum distance between any two points (i.e. any two rows) with the help of an optimization algorithm. Another method is to minimize the discrepancy, which is a measure of non-uniformity of the design points on an experimental domain. Different discrepancy measures exist but the most popular ones are based on the $L_2$ norm. It has been shown by Iooss et al. (2010) that modifying the LHS based on minimizing the discrepancy leads to a better space-filling design compared to one where the minimum distance is maximized.

**Orthogonal arrays (OAs)** could be used to improve the LHS. An orthogonal array of strength $t$ is a matrix of $n$ rows and $k$ columns with elements from a set of $q$ symbols ($q \geq 2$), such that in any $n \times t$ submatrix each of the $q^t$ possible rows occurs the same number $\lambda$ of times. Consequently, $n = \lambda q^t$. The array is denoted $OA(n, k, q, t)$ and is said to be of size $n$ with $k$ constraints and $q$ levels. The number $\lambda$ is called the index of the array. The LHS described by Equation (4.10) is thus an OA of strength 1 with $\lambda = 1$ and $q = n$, i.e. $OA(n, k, n, 1)$. 
Queipo et al. (2005) give two reasons why the OA might not be used directly; lack of flexibility and point replicates. Given a desired sample size \( n \) for a set of variables \( k \) at a required number of levels \( q \) with a specific strength \( t \), the OA might not exist. In addition, OA designs that after screening are projected onto a subspace of the most important variables can, in the general case, result in replication of points, which is not desired for deterministic simulations. See for example the case in which column three and five in the OA in Figure 4.6 are eliminated, which will result in only four different designs which all are replicated.

One of the methods where OAs are used to improve the LHS is the randomized orthogonal array. If the elements of an OA are called \( A_{ij} \), where \( 0 \leq A_{ij} \leq q - 1 \), the randomized orthogonal arrays corresponding to Equations (4.8) and (4.10) are

\[
X_{ij} = \frac{\pi_j(A_{ij}) + U_{ij}}{q}
\]  

(4.11)

and

\[
X_{ij} = \frac{\pi_j(A_{ij}) + 0.5}{q}
\]  

(4.12)

respectively, according to Owen (1992). \( \pi_j \) are independent permutations of 0, ... , \( q - 1 \), all \( q! \) permutations being equally probable and \( U_{ij} \) are independent uniformly distributed random variables between 0 and 1, independent of \( \pi_j \). In this way the design space is divided into subspaces and not more than one design point is placed in each subspace.

**Orthogonal array-based Latin hypercubes**, as described by Tang (1993), is an LHS with the design space divided into subspaces and not more than one design point placed in each subspace. This is done by replacing the elements in each column of an OA in a special way so that the resulting matrix is an LHS.
**Figure 4.7** Comparison between different space-filling DOEs with two variables and four design points. 

- **a)** MLHS - "one point in every row and column"
- **b)** Randomized OA - "one point in every subspace"
- **c)** OA-based LHS - "one point in every row, column and subspace"

In addition to the various LHS methods several other space-filling methods exist. When \( n \) points are chosen within the design space so that the minimum distance between them are maximized a **maximin** or sphere-packing design is obtained, as described by Johnson et al. (1990). For small \( n \) this will generally result in the points lying on the exterior of the design space and that the interior is filled as the number of points becomes larger. Another of the so called distance-based designs is the **minimax** design, where the maximum distance between any design points is minimized. In this case, the designs will generally lie in the interior of the design space also for small numbers of \( n \).

**Figure 4.8** Comparison of maximin and minimax designs with seven points in two variables. 

- **a)** Maximin - "maximize \( R \)"
- **b)** Minimax - "minimize \( R \)"

**a)** Maximin: the design space is filled with spheres with maximum radius \( \Rightarrow \) no design point is too close to another design point. 

**b)** Minimax: the design space is covered by spheres with minimum radius \( \Rightarrow \) no design point is too far from another design point.
**Hammersley sequence sampling (HSS)** described by Kalagnanam and Diwekar (1997) and **uniform designs (UD)** described by Fang et al. (2000) belong to a group called low-discrepancy sequences. The discrepancy is a measure of the difference from a uniform distribution and could be measured in several ways. While LHS is uniform only in a one-dimensional projection, these methods tend to be more uniform in the entire design space. In HSS, the low discrepancy sequence of Hammersley points is used to sample the \( k \)-dimensional space. The UD, on the other hand, has similarities with LHS. In the UD, the points are always selected from the centre of cells in the same way as for the MLHS described in Equation (4.10). In addition to the one-dimensional balance of all levels for each factor in the LHS, the UD also requires \( k \)-dimensional uniformity. The most popular UD, the UUD, could for example be obtained by selecting the design with the smallest discrepancy out of all possible designs according to Equation (4.10).

In addition to the different space-filling designs, different criteria-based designs could be constructed if certain information about the metamodel to be fitted is available a priori, which is not always the case. In an **entropy design** the purpose is to maximize the expected information gained from an experiment, while the **mean squared error design** minimizes the expected mean squared error. See Koehler and Owen (1996) for more details about these designs and other designs previously mentioned.

### 4.2.3 Sampling Size and Sequential Sampling

Several factors are important for determining how well the metamodel will fit the true response. Two of the important factors are the number of design points used for fitting the model and their distribution in the design space. In order to build a polynomial metamodel, there is a fixed minimum number of design points required, depending on the number of variables. However, it is usually desirable to use a larger sampling size than the minimum required, i.e. to use **oversampling**, to be able to improve the accuracy and also have the potential to estimate how good the metamodel is. For non-parametric metamodels, such as neural networks, there is no such minimum sample size, although the accuracy of the metamodel will be limited if the sampling size is too small. Also, the more complex response the metamodel should capture the larger sample size it requires.

The minimum sample size, \( n_{\text{min}} \), needed to fit a linear or a full quadratic metamodel is

\[
 n_{\text{min}} = 1 + k
\]

and

\[
 n_{\text{min}} = 1 + 2k + \frac{k(k - 1)}{2} = \frac{(k + 1)(k + 2)}{2}
\]

respectively, where \( k \) is the number of variables. These design points must be unique (no replicates) and contain at least two levels for each variable for the linear model and three levels for each variable for the quadratic model.

The accuracy of a metamodel is generally improved by increasing the number of design points. But for low order polynomial metamodels this is only valid up to a certain limit. Thereafter, increasing the number of points does not contribute much to the approximation accuracy. Stander et al. (2010) state that this limit is very roughly at 50% oversampling. Additionally, Shi et al. (2012) have found
that an increased sample size might not improve the metamodel much if there is a large uncertainty in the data.

Detailed simulation models are often time-consuming to run. The question in practice is therefore often how many design points that are needed to fit a reasonably accurate metamodel. It has been proposed by Gu and Yang (2006) and Shi et al. (2012) that a minimum of $3k$ sampling points, where $k$ equals the number of variables, are needed to build a reasonably accurate metamodel. An initial sampling size of between $3k$ and $4k$ could therefore be sensible, at least if $k$ is not too large. Note that this number is less than what is needed to build a quadratic model with all interactions. It is, however, difficult to know the appropriate sampling size beforehand. Therefore sequential sampling can be used to avoid issues with too many, i.e. unnecessary time-consuming, or too few design points giving low metamodel accuracy. A limited number of designs could then be used as a starting point and if required, additional points could be added later.

Sequential sampling is typically based on some optimality criteria for experimental designs. When information from previously fitted metamodels is used in the sequential sampling, the sampling is said to be adaptive. Many different sequential approaches have been proposed, see Jin et al. (2002) and Forrester and Keane (2009). Some of the adaptive approaches select a new sample set based on an existing model fitted to an existing set. Kriging models, for example, provide an estimate of the prediction error at an unobserved point. This estimate is called mean squared error (MSE) and is the base for some approaches. The entropy approach, which maximizes the information obtainable from the new set, and the IMSE approach, which minimizes the integrated MSE, are two of the adaptive methods that can be used for Kriging models. Another one is the MSE approach, which chooses the point with the largest MSE. This is a special case of the entropy criteria where only one point is chosen. More details can be found in Jin et al. (2002).

For other models, where an estimate of the prediction error is not provided, cross validation (CV) can be used to estimate the prediction error, see Section 4.5.2. Based on the existing sample set with $n$ points, the prediction error in point $x$ can be estimated by a leave-one-out error, i.e.

$$e(x) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_{-i}(x) - \hat{y}(x))^2} \quad (4.15)$$

where $\hat{y}(x)$ denotes the prediction of the response for $x$ on the metamodel created based on all $n$ existing sample points and $\hat{y}_{-i}(x)$ denotes the prediction of the response for $x$ using the metamodel created based on $(n - 1)$ existing sample points with the $i^{th}$ point omitted ($i = 1, 2, \ldots, n$). With the CV approach, the point with the largest prediction error according to Equation (4.15) is selected as the new sample point. The idea is hence similar to the MSE approach.

The maximin distance approach is not adaptive and works consequently with all metamodels. Given an existing sample set, the idea is to select the new sample set so that the minimum distance between any two points in the complete set is maximized. An adaptive version of this approach is to scale the distances based on the importance, identified from the existing metamodel, of the different variables. This approach is expected to lead to a better uniformity of the projection of sample points into the space made of the important variables and therefore improve the quality of the information obtained.
One issue with adaptive sequential sampling could arise when several responses from one detailed model are studied. In this case, several metamodels are fitted, based on the same set of design points, but ideally different points should probably be selected for different metamodels.

It has been shown by Jin et al. (2002) that the performance of sequential sampling approaches, in general, is comparable to the single-stage approach and that no adaptive sequential sampling approach consistently outperforms the approaches without adaption.

### 4.3 Variable Screening

As mentioned earlier, the number of simulations needed to build a metamodel, very much depends on the number of design variables. Eliminating the variables which are not influencing the results can therefore substantially reduce the computational cost. The process of studying the importance of different variables, identifying the ones to be included and eliminating the ones that do not influence the responses is called variable screening. Several screening methods exist, see e.g. Viana et al. (2010). One of the simplest screening techniques is called one-factor-at-a-time plans. Another category of screening techniques are variance-based. One simple and commonly used variance-based approach uses a factorial or fractional factorial design followed by an analysis of variance as described by Myers et al. (2008). An alternative variance-based method gaining popularity is Sobol’s global sensitivity analysis (Sobol’, 2001). The first technique is used to separately identify the main and interaction effects that account for most of the variance in the response while the second method provides the total effect (main and interaction effects) of each variable. Both these techniques are described in more detail below.

#### 4.3.1 One-Factor-at-a-Time Plans

The one-factor-at-a-time plans evaluate the effect of changing one variable at a time (compare with linear Koshal designs). This is a very inexpensive approach but it does not estimate the interaction effects between variables. Therefore, variants of this method that account for interactions have been proposed. One example is Morris method (Morris, 1991), which to the cost of additional runs, tries to determine whether the variables have effects that are (a) negligible, (b) linear and additive, or finally (c) non-linear or involved in interactions with other variables. Based on repeated random one-factor-at-a-time simulations, the distributions of elementary effects for all variables are calculated. For a given value of input $x$, the elementary effect of variable $i$ is determined by

$$d_i(x) = \frac{[y(x_1, x_2, ..., x_i-1, x_i + \Delta, x_{i+1}, ..., x_k) - y(x)]}{\Delta}$$

where $\Delta$ is the difference in the $i^{th}$ variable between two simulations. A distribution of elementary effect with a large mean indicates a variable with important influence on the response while a distribution with large spread indicates a variable whose influence is involved in interactions or whose effect is non-linear.

#### 4.3.2 Analysis of Variance

The analysis of variance (ANOVA) procedure is based on the idea that the metamodel is fitted using regression analysis, as is the case with polynomial metamodels, see Section 4.4.1. These metamodels are defined by determining the size of the regression coefficients, i.e. the coefficients for each term in the model. The results from the ANOVA are often presented in a table that give information on
which variables and interactions that are significant, i.e. which regression coefficients $\beta_j$ that are not equal to zero with a defined level of certainty. The ANOVA process is described in Table 4.2. The workflow is from left to right and the $F_0$ or $P$-values are ultimately compared with relevant limits to judge the significance of a model term. If $F_0$ is larger than the $F$-statistic $F_{a,DoF,n-p}$, the corresponding regression coefficient is non-zero with $100(1 - \alpha)$% certainty. The $P$-value is a measure on how much there is against the hypothesis that the regression coefficient $\beta_j$ is equal to zero (null hypothesis). The smaller the $P$-value the more evidence there is against this hypothesis. Consequently, a large $F_0$ gives a low $P$-value which indicates that the coefficient is significant. Often used limits for the $P$-value are $P < \alpha = 0.05$ or $P < \alpha = 0.01$.

Table 4.2 Description of the ANOVA-procedure where $n =$ number of observations, $p =$ number of regression coefficients, $p = k + 1$ where $k =$ number of variables, $b_j$ is the estimate of the $j^{th}$ regression coefficient $\beta_j$, and $C_{ij}$ is the diagonal element of $(X'X)^{-1}$ corresponding to $b_j$. $X$ is the model matrix which has $n$ rows, one for each design point, and $p$ columns, one for each coefficient to be estimated (see Section 4.4.1 for more details).

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>Sum of Squares</th>
<th>Degrees of Freedom</th>
<th>Mean Square</th>
<th>$F_0$</th>
<th>$P$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression Coefficients</td>
<td>$SS = \frac{b_j^2}{C_{jj}}$</td>
<td>1</td>
<td>$MS = \frac{SS}{DoF}$</td>
<td>$F_0 = \frac{MS}{MS_{err}}$</td>
<td>From table or program ($F_0$, DoF + DoF$_{err}$ as input)</td>
</tr>
<tr>
<td>Error</td>
<td>$SS_{err} = SS_{tot} - \sum_{j=1}^{p} SS$</td>
<td>$n - p$</td>
<td>$MS_{err} = \frac{SS_{err}}{DoF_{err}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>$SS_{tot} = \sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}$</td>
<td>$n - 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The confidence intervals for the estimated regression coefficients $b_j$ ($j = 0, 1, ..., p$) can also be calculated and give the limits within which the coefficients $\beta_j$ lie with $100(1 - \alpha)$% certainty. The importance of a variable, i.e. if it should be included in the model or not, is judged both by the magnitude of the related estimated regression coefficients $b_j$ and by the level of confidence that the regression coefficient $\beta_j$ is non-zero. The significance of the variables can be visualized in a bar chart of the magnitudes of the coefficients $b_j$ with the confidence interval for each coefficient indicated by an error bar. If the terms are normalized with the design space so that the choice of units becomes irrelevant, Stander et al. (2010) states that the relative bar lengths give an estimate of the importance of the variables while the error bars represent the contribution to noise or poorness of fit by the variables.

### 4.3.3 Global Sensitivity Analysis

The global sensitivity analysis (GSA) procedure includes the calculation of global sensitivity indices, also called Sobol’ indices. These indices are sensitivity measures for arbitrary complex metamodels and estimate the effect of input variables on the model response as described by Sobol’ (2001). It has been found by Reuter and Liebscher (2008) that the method can be used to identify relevant input variables for non-linear non-monotonic problems where simple ANOVA methods may fail.
If the model under investigation is described by the function $y = f(x)$, where $x = (x_1, x_2, \ldots, x_k)^T$ is an input vector of $k$ variables, the model can be decomposed into terms of increasing dimension according to

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

where $1 \leq i_1 < \ldots < i_s \leq k$. The response $y$ is characterized by its variance $D$, and this variance can be divided into partial variances in the same way

$$D = \sum_{s=1}^{k} \sum_{i_1 < \ldots < i_s} D_{i_1 \ldots i_s} = \sum_{i=1}^{k} D_i + \sum_{1 \leq i < j \leq k} D_{ij} + \ldots + D_{12 \ldots k}$$

Each partial variance can then be used to evaluate its global sensitivity index.

$$s_{i_1 \ldots i_s} = \frac{D_{i_1 \ldots i_s}}{D}, \quad 1 \leq i_1 < \ldots < i_s \leq k,$$

Each of these indices represents a sensitivity measure which describes the amount of the variance $D$ that is caused by the main or interaction effect of the corresponding variable or variables. Hence, if all of them are added together the sum is 1. All of the partial sensitivity indices (main and interaction effects) related to the single variable $x_i$ ($i = 1, \ldots, k$) can be added to a total sensitivity index $s_{i \text{tot}}$ to evaluate the total effect of $x_i$. The total sensitivity index can be used to rank the importance of the variables $x_i$ for a response $y$ and identify insignificant input variables. This is done by estimating all $s_{i \text{tot}}$ values and rank the variables according to these values. In order to quantify which amount of the variance $D$ that is caused by a single variable $x_i$, the corresponding $s_{i \text{tot}}$ can be normalized

$$\text{norm}_i s_{i \text{tot}} = \frac{s_{i \text{tot}}}{\sum_{j=1}^{k} s_{j \text{tot}}}$$

so that the sum of all $\text{norm}_i s_{i \text{tot}}$ becomes 1.

The global sensitivity indices can be calculated through different integrals of $f(x)$, as described by Sobol’ (2001). To evaluate these integrals, the Monte Carlo approach is normally used. The idea behind this numerical integration technique is as follows. Consider a deterministic function $y = f(x)$, where $x = (x_1, x_2, \ldots, x_k)^T$ is a random vector with uniform distribution on the unit hypercube $[0, 1]^k$. Estimating the mean $\mu = E(y)$ of the random variable $y$ is then equivalent to finding the integral of $y = f(x)$. The simplest way of doing this is to draw $n$ samples, $x_1, x_2, \ldots, x_n$, independently from the uniform distribution Unif$[1 0]^k$ and to estimate $\mu$ by

$$\mu = \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

Consequently, the more samples that are drawn, the better the estimate becomes.
The total effect of an input variable $x_i$ can be estimated using the Monte Carlo approach according to Sobol’ (2001). Consider two independent points $(1)x = (1)x_i, (1)x_{\neq i}$ and $(2)x = (2)x_i, (2)x_{\neq i}$, where $x_{\neq i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$. In order to estimate $s_i^{tot}$ two computations of the model is needed for each Monte Carlo trial, $f^{(1)}(x)$ and $f^{(1)}(x_{\neq i}, (2)x_i)$. The total effect of an input variable $x_i$ can then be estimated as

$$s_i^{tot} = 1 - s_{\neq i} = 1 - \frac{\hat{D}_i}{\bar{D}}$$

$$= 1 - \frac{\frac{1}{n} \sum_{r=1}^{n} f^{(1)}(x^r) f^{(1)}(x_{\neq i}, (2)x^r) - \left(\frac{1}{n} \sum_{r=1}^{n} f^{(1)}(x^r)\right)^2}{\frac{1}{n} \sum_{r=1}^{n} f^{2}(x^r) - \left(\frac{1}{n} \sum_{r=1}^{n} f^{(1)}(x^r)\right)^2}$$

where the hats are indicating that the values are estimated using the Monte Carlo approach.

When the sensitivity indices are calculated based on metamodel evaluations, the variance related to the approximation of the detailed simulation model is included. The variance of the discrepancies between the response from the detailed model and the response from the metamodel at the experimental points can be calculated and denoted by $\bar{D}_{unex}$. This variance, which is unexpected for the detailed model, can then according to Reuter and Liebscher (2008) be used to modify the normalized total sensitivity index

$$\text{norm } s_i^{tot} = \frac{s_i^{tot}}{\sum_{j=1}^{k} s_j^{tot}} \frac{\hat{D}}{\bar{D} + \hat{D}_{unex}}$$

where $\hat{D}$ and $\bar{D}_{unex}$ are computed using the metamodel.

Since the calculation of the global sensitivity indices with the Monte Carlo approach only involves functional evaluations of different sets of input variables, the method is not restricted to any special type of metamodel, as in the case of ANOVA. However, the accuracy of the indices depends on the number of Monte Carlo evaluations that are used to calculate the indices.

### 4.4 Metamodels

A metamodel is a mathematical approximation of a detailed and usually computationally costly simulation model, i.e. a model of a model. The metamodels can be used as surrogates for the detailed model when a large number of evaluations are needed, as in optimization, and when it is too time-consuming to run the detailed model for each evaluation. When running a detailed simulation model, a vector of input (design variable values), $x$, results in a vector of output (response values), $y$. The detailed model can therefore be seen as a function $f : \mathbb{R}_k^{k} \to \mathbb{R}_l^{l}$ which means that the function $f$ maps the set of $k$ real numbers (design variables) into another set of $l$ real numbers (responses).

$$y = f(x)$$

For each scalar response $y$, a metamodel can be built to approximate the true response as

$$\hat{y} = s(x)$$
where $s(x)$ is the mathematical function defining the metamodel and which maps the design variables $x$ to the predicted response $\hat{y}$. In general, this approximation is not exact and the predicted response $\hat{y}$ will differ from the observed response $y$ from the detailed model, i.e.

$$y = \hat{y} + \varepsilon = s(x) + \varepsilon$$

(4.26)

where the error $\varepsilon$ consequently represents the approximation error.

A metamodel for a single response is built from a dataset of input $x_i$ and corresponding output $y_i = f(x_i)$, where $i = 1, \ldots, n$ and $n$ is the number of designs used to fit the model. Consequently, $n$ evaluations of the detailed model with different variable settings $x_i = (x_{i1}, x_{i2}, \ldots, x_{ik})^T$ of the $k$ design variables are required to build the metamodel.

Several mathematical formulations can be used for the metamodels. Some of them are suitable for global approximations, i.e. can be used for representing the complete design space, while others are more suitable for local approximations of a part of the design space. Metamodels can **interpolate** the responses from the detailed simulations or **approximate** the responses depending on the formulation. In the case of deterministic simulations, interpolating metamodels might be preferred, if the numerical noise is negligible. However, an interpolating metamodel is not necessarily better than an approximating one at predicting the response between the fitting points, see Figure 4.9.

**Figure 4.9** Comparison of metamodels. **a)** Interpolating metamodel. **b)** Approximating metamodel. **c)** Interpolating and approximating metamodels in comparison with the true response. Note that the approximating metamodel is closer to the true response in the middle of the design space.

Different approaches are used to build the metamodels. Parametric techniques are based on an a-priori chosen functional relationship between the design variables and the response. The metamodel is fitted to the dataset of design variables and corresponding responses from the detailed model by determining the coefficients of the chosen function. Examples of metamodels built in this way are polynomial and Kriging models. Non-parametric techniques are used to build different types of neural network models. These techniques do not have an a-priori functional form, instead they use an a-priori method for constructing an approximating function based on the available dataset. This is done by the use of various types of simple local models in different regions, which are then combined to build an overall model.

In the following sections, a number of different well-known and often used metamodels are presented and their main characteristics as well as the basic idea behind their derivations are outlined.
4.4.1 Polynomial Regression

Polynomial metamodels are often referred to as response surface models and used in response surface methodology (RSM). The RSM is described by Myers et al. (2008) as a set of statistical and mathematical methods for developing, improving, and optimizing processes and products. The models are developed using regression, which is the process of fitting a regression model \( y = s(x, \beta) + \varepsilon \) to a dataset of \( n \) variable settings \( x \), and corresponding responses \( y_i \).

The method of least squares chooses the regression coefficients \( \beta \) so that the quadratic error is minimized, i.e. solve the regression (or data fitting) problem

\[
\min_{\beta} \sum_{i=1}^{n} \varepsilon_i^2 = \min_{\beta} \sum_{i=1}^{n} (y_i - s(x_i, \beta))^2 \tag{4.27}
\]

A regression model can be of different forms, not necessarily a polynomial one. However, the most frequently used class in linear regression where \( s(x, \beta) \) is linear in \( \beta \), consists of low order polynomials. For example can the following models be used to fit a metamodel in \( k \) design variables

\[
y = s(x, \beta) + \varepsilon = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \varepsilon \tag{4.28}
\]

\[
y = s(x, \beta) + \varepsilon = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{ij} x_i x_j + \varepsilon \tag{4.29}
\]

\[
y = s(x, \beta) + \varepsilon = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{ij} x_i x_j + \sum_{i=1}^{k} \beta_{ii} x_i^2 + \varepsilon \tag{4.30}
\]

These models are first-order (4.28), first-order with interaction (4.29), and second-order (quadratic) polynomial models (4.30), respectively. The models presented here only include second order interaction effects, i.e. effects that involve two variables. In the general case also higher order interaction effects can be included. The least square estimators of the regression coefficients \( \beta \) are denoted by \( \hat{\beta} \) and the process of finding these estimates is easily described in matrix notation. The regression model may be written as

\[
y = X \beta + \varepsilon \tag{4.31}
\]

where

\[
y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} \cdots & x_{1k} & x_{11} x_{12} \cdots & x_{1(k-1)} x_{1k} & x_{21}^2 \cdots & x_{1k}^2 & \cdots \\ 1 & x_{21} \cdots & x_{2k} & x_{21} x_{22} \cdots & x_{2(k-1)} x_{2k} & x_{21}^2 \cdots & x_{2k}^2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} \cdots & x_{nk} & x_{n1} x_{n2} \cdots & x_{n(k-1)} x_{nk} & x_{n1}^2 \cdots & x_{nk}^2 & \cdots \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f^T(x_1) \\ f^T(x_2) \\ \vdots \\ f^T(x_n) \end{bmatrix} \tag{4.32}
\]
Thus, $y$ is a vector of the $n$ responses, $X$ is an $n \times p$ model matrix consisting of the variable settings expanded to model form, $\beta$ is a vector of the $p$ regression coefficients and $\epsilon$ is a vector of the $n$ errors. In the model matrix, each row corresponds to one design point and each column to one regression coefficient.

The fitted regression model becomes

$$\hat{y} = Xb$$  \hspace{1cm} (4.33)

where

$$b = (X^TX)^{-1}X^Ty$$  \hspace{1cm} (4.34)

Thus, if the $\beta_i$'s are replaced by $b_i$'s and the error terms are omitted in Equations (4.28), (4.29), and (4.30) above, the linear, linear with interaction, and quadratic metamodels are obtained. The response in an unknown point $x_u$ can hence be found as

$$\hat{y}(x_u) = f^T(x_u)b$$  \hspace{1cm} (4.35)

where $f(x_u)$ is a vector with elements corresponding to a row of the model matrix $X$ for $x_u$.

The polynomial metamodels will in general not interpolate the fitting data. One exception is when the fitting set is so small that there is just enough data available to determine all the regression coefficients in the model. However, such small fitting sets are generally not recommended. Low order polynomial metamodels will capture the global trends of the detailed simulation model, but will in many cases not be a good representation of the complete design space. These metamodels are therefore mainly used for screening purposes and in iterative optimization procedures where a sequence of metamodels are built in a smaller and smaller region of the design space around the proposed optimum, see more details in Section 4.6.1.

### 4.4.2 Moving Least Squares

Polynomial metamodels can give large errors for highly non-linear responses but give good approximations in small regions where the response is less complex. These features are taken advantage of in the method of moving least squares (MLS). The mathematical description of a MLS metamodel can according to Breitkopf et al. (2005) be formulated as

$$\hat{y}(x) = \sum_{i=1}^{p} f_i(x) b_i(x) = f^T(x)b(x)$$  \hspace{1cm} (4.36)

where $f$ is a vector of basis functions (polynomials) for the metamodel and $b$ is a vector of coefficients. The number of coefficients $p$ depends on the order of approximation.

For a specific value of $x$, a polynomial is fitted according to the least squares method, where the influence of surrounding points is weighted depending on their distance to $x$. Hence, compared to Equation (4.35) for polynomial metamodels, the MLS model has coefficients $b$ that depend on the location in the design space, i.e. depend on $x$. Thus, one polynomial fit is not valid over the entire domain as for normal polynomial metamodels. Instead, the polynomial is valid only locally around the point $x$ where the fit is made. However, this will not lead to an interpolating model in the general case.
The coefficients \( b_i(x) \) are determined by a weighted least squares method, minimizing the weighted error between the response from the detailed model \( y(x) \) and the estimated value from the metamodel \( \hat{y}(x) \)

\[
\min \sum_{i=1}^{n} w_i \varepsilon_i^2 = \min_b \sum_{i=1}^{n} w(||x_i - x||) \{f^T(x_i - x)b - y(x_i)\}^2
\]

(4.37)

where \( n \) is the number of fitting designs and \( x_i \) the input of design \( i \). The weights \( w_i \geq 0 \) ensure the continuity and locality of the approximation. \( w_i \) takes its maximum value at the point \( x_i \). The weight \( w_i \) is decreasing within a fixed region around the point, called the domain of influence of \( x_i \), and vanishes outside this region. The weight functions, including the size of the domain of influence, play an important role by governing the way the coefficients \( b_i(x) \) depend on the location of the studied point \( x \).

The solution to Equation (4.37) gives

\[
b(x) = (X^TWX)^{-1}X^Wy
\]

(4.38)

where

\[
X = X(x) = \begin{bmatrix} f^T(x_1 - x) \\ \vdots \\ f^T(x_n - x) \end{bmatrix}, \quad W = W(x) = \begin{bmatrix} w(||x_1 - x||) & 0 \\ 0 & \vdots & \ddots & 0 \\ 0 & \vdots & \ddots & w(||x_n - x||) \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}
\]

(4.39)

which can be compared to Equations (4.34) and (4.32) for a polynomial model. The MLS metamodel for an unknown point \( x_u \) can then be written as

\[
\hat{y}(x_u) = f^T(x_u)b(x_u) = f^T(x_u)(X^TW(X_u)X(X_u))^{-1}X^TW(X_u)y
\]

(4.40)

Note that since \( b \) is a function of \( x \), a new MLS model needs to be fitted for every new evaluation. Furthermore, in order to construct the metamodel, enough fitting points need to fall within the domain of influence. The number of influencing fitting designs can be adjusted by changing the weight functions, or rather the radius of the domain of influence. The denser the design space is sampled, the smaller the domain of influence might be, and the more accurate the metamodel becomes.

### 4.4.3 Kriging

**Kriging** is named after the South African mining engineer D. C. Krige, and this method for building metamodels has been used in many engineering applications. *Design and analysis of computer experiments (DACE)* is a statistical framework for dealing with Kriging approximations to complex and/or expensive computer models presented by Sacks et al. (1989). The idea behind Kriging is that the deterministic response \( y(x) \) can be described as

\[
y(x) = f(x) + Z(x)
\]

(4.41)

where \( f(x) \) is a known polynomial function of the design variables \( x \) and \( Z(x) \) is a stochastic process (random function). This process is assumed to have mean zero, variance \( \sigma^2 \) and a non-zero covariance. The \( f(x) \) term is similar to a polynomial model described in Section 4.4.1 and provides a
"global" model of the design space while the \( Z(x) \) term creates "local" deviations so that the Kriging model interpolates the \( n \) sampled data points. In many cases, \( f(x) \) is simply a constant term and the method is then called \textit{ordinary Kriging}. If \( f(x) \) is set to 0, implying that the response \( y(x) \) has mean zero, the method is called \textit{simple Kriging}. In matrix notation, the general \textit{universal Kriging} model fitted to \( n \) points can be written as

\[
\hat{y} = Xb + Z
\]

where \( X \) is the model matrix defined in Equation (4.32), \( b \) is a vector of the estimated regression coefficients and \( Z \) is a vector of the stochastic process with mean zero and covariance

\[
\text{Cov} \left( Z(x_i), Z(x_j) \right) = \sigma^2 R([R(x_i, x_j)]) = \\
\begin{bmatrix}
\sigma^2 \\
\vdots \\
\sigma^2 \\
\end{bmatrix} \begin{bmatrix}
1 & \cdots & R(x_1, x_n) \\
\vdots & \ddots & \vdots \\
R(x_n, x_1) & \cdots & 1 \\
\end{bmatrix}
\]

(4.43)

\( \sigma^2 \) is the process variance and \( R(x_i, x_j) \) is the correlation function between the evaluated sample points \( x_i \) and \( x_j \). This makes \( R \) an \( n \times n \) symmetric matrix with a unit diagonal. The correlation function \( R \) controls the smoothness of the resulting Kriging model and the influence of nearby points by quantifying the correlation between observations. Many different correlation functions could be used. According to Stander et al. (2010) two commonly applied functions are the exponential and the Gaussian correlation functions, i.e.

\[
R(x_i, x_j) = \prod_{r=1}^{k} e^{-\theta_r |x_i^r - x_j^r|}
\]

(4.44)

and

\[
R(x_i, x_j) = \prod_{r=1}^{k} e^{-\theta_r |x_i^r - x_j^r|^2}
\]

(4.45)

respectively. \( |x_i^r - x_j^r| \) is the distance between the \( i^{th} \) and \( j^{th} \) sample point of variable \( x^r \), \( k \) is the number of variables, and \( \theta_r \) is the correlation parameter for variable \( x^r \). In general, a different \( \theta_r \) for each variable is used which yields a vector \( \theta \) with \( k \) elements. In some cases a single correlation parameter for all variables gives sufficiently good results and the model is then said to be isotropic. The parameter \( \theta_r \) is essentially a width parameter which affects how far the influence of a sample point extends, see Forrester and Keane (2009). A low \( \theta_r \) means that all points will have a high correlation \( R \), with \( Z(x^r) \) being similar across the sample, while a high \( \theta_r \) means that there is a significant difference between the \( Z(x^r) \)'s for different sample points. The elements of \( \theta \) could therefore be used to identify the most important variables provided that a suitable scaling of the design variables is used.

In order to build a Kriging metamodel, both the regression coefficients \( b \) as well as the correlation parameters \( \theta \) need to be determined. Optimum values of \( b \) are found from

\[
b = (X^T R^{-1} X)^{-1} X^T R^{-1} y
\]

(4.46)
where $X$ is the model matrix, $y$ is a vector of the $n$ observed responses from the detailed model, and $R$ is still unknown. Note the similarities with Equation (4.34) for a polynomial model. The optimum values of $\theta$ are determined by solving the non-linear optimization problem of maximizing the log-likelihood function

$$
\max_{\theta} \quad L(\theta) = -\frac{1}{2} \left[ n \ln(\hat{\sigma}^2) + \ln|R| \right]
$$

subject to $\theta_r > 0, r = 1, \ldots, k$

where $|R|$ is the determinant of $R$. Both $R$ and $\hat{\sigma}^2$ are functions of $\theta_r$. The estimate of the variance is given by

$$
\hat{\sigma}^2 = \frac{(y - Xb)^T R^{-1} (y - Xb)}{n}
$$

An equivalent problem to the maximizing problem (4.47) is to minimize $\hat{\sigma}^2 |R|^{1/n}$ for $\theta > 0$. As can be observed, this is a $k$-dimensional optimization problem, which requires significant computational time if the sample data set is large. In addition, the correlation matrix can become singular if the sample points are too close to each other, or if the sample points are generated from particular DOE. A small adjustment of the $R$-matrix could avoid ill-conditioning but might result in a metamodel that does not interpolating the observed responses exactly.

When $b$ and $\theta$ (and hence $R$) are determined, the best linear unbiased predictor (BLUP) of an unknown point $x_u$ can be written as

$$
\hat{y}(x_u) = f^T(x_u)b + r^T(x_u)R^{-1}(y - Xb)
$$

where $f(x_u)$ is a vector corresponding to a row of the model matrix $X$ for $x_u$, $b$ is the vector of estimated regression coefficients, $r(x_u) = [R(x_u, x_1), R(x_u, x_2), \ldots, R(x_u, x_n)]^T$ is a vector of correlation functions between the unknown point and the $n$ sample points, $R$ is the matrix of correlation functions for the fitting sample, and $y$ is a vector of the observed responses in the fitting sample. The term $(y - Xb)$ is a vector of residuals for all fitting points when the stochastic term of the model is disregarded.

According to Simpson et al. (2001), special choices of correlation functions could give metamodels which approximate the fitting data. Although this is not normally used, the Kriging method is flexible due to the choice of different correlation functions, and it is well suited for global approximations of the complete design space. Kriging models also provide an estimate of the prediction error in an unobserved point directly, see Equation (4.50), a feature that can be used in adaptive sequential sampling approaches as presented in Section 4.2.3. According to Sacks et al. (1989), the estimate of the prediction error (mean squared error) of an unknown point can be evaluated as

$$
s^2(x_u) = \hat{\sigma}^2 \left( 1 - [f^T(x_u) \quad r^T(x_u)] \begin{bmatrix} 0 & X^T \\ X & R \end{bmatrix}^{-1} \begin{bmatrix} f(x_u) \\ r(x_u) \end{bmatrix} \right)
$$

with notations as previously defined.

When working with noisy data, an interpolating model might not be desirable. The original interpolating Kriging model can then be modified by adding a regularization constant to the diagonal
of the correlation matrix so that the model does not interpolate the data. In this case, the error estimation in Equation (4.50) needs to be modified accordingly, see Forrester and Keane (2009).

4.4.4 Artificial Neural Networks

Artificial neural networks (ANNs) are intended to respond to stimulus in a fashion similar to the biological nervous systems. One of the attractive features of these structures is the ability to learn associations between data. An artificial neural network, or often just called neural network (NN), may therefore be used to approximate complex relations between a set of input and output, and can thus be used as a metamodel.

An NN is composed of small computing elements called neurons, assembled into an architecture. Based on the input \( x = (x_1, x_2, \ldots, x_k)^T \), the output \( y_m \) from a single neuron \( m \) is evaluated as

\[
y_m(x) = f \left( b_m + \sum_{i=1}^{k} w_{mi} x_i \right) = f(a)
\]  

(4.51)

where \( f \) is the transfer or activation function, \( b_m \) is the bias value, and \( w_{mi} \) the weight of the corresponding input \( x_i \) for neuron \( m \). A schematic description is presented in Figure 4.10. The input \( x \) to the neuron are either variable values or output from previous neurons in the network. The connection topology of the architecture, the weights, the bias, and the transfer function used, determine the form of the neural network.

![Figure 4.10](image)

Figure 4.10 Schematic illustration of neuron \( m \) in a neural network, where input is variables or output from previous neurons.

One very common architecture is the multi-layer feedforward neural network (FFNN), see Figure 4.11, in which the information is only passed forward in the network and no information is fed backward. The transfer function in the hidden layers of an FFNN is often a sigmoid function, i.e.

\[
f(a) = \frac{1}{1 + e^{-a}}
\]  

(4.52)

which is an S-shaped curve ranging from 0 to 1 and \( a \) is defined in Equation (4.51). For the input and output layers, a linear transfer \( f(a) = a \) is usually used with bias added to the output layer but not to
the input layer. This means that a simple neural network with only one hidden layer of \( M \) neurons could be of the form

\[
\hat{y}(x) = b + \sum_{m=1}^{M} \frac{w_m}{1 + e^{-(b_m + \sum_{i=1}^{k} w_{mi} x_i)}}
\]  

(4.53)

where \( b \) is the bias of the output neuron, \( w_m \) is the weight on the connection between the \( m^{th} \) hidden neuron and the output neuron, \( b_m \) is the bias in the \( m^{th} \) hidden neuron, and \( w_{mi} \) is the weight on the connection between the \( i^{th} \) input and the \( m^{th} \) hidden neuron.

\[ a = b_m + \Sigma w_{mi} x_i \] and usually,

- for input and output layers
  \[ f(a) = a \]
- for hidden layers
  \[ f(a) = 1 / (1+e^a) \]

**Figure 4.11** Schematic illustration of a feedforward neural network architecture with multiple hidden layers.

Another common type of neural network is the radial basis function network which is described in more detail in Section 4.4.5.

There are two distinct steps in building a neural network. The first is to choose the architecture and the second is to train the network to perform well with respect to the training set of input (design variable values) and corresponding output (response values). The second step means that the free parameters of the network, i.e. the weights and biases in the case of an FFNN, are determined. This is a non-linear optimization problem in which some error measure is minimized.

If the steepest descent algorithm, see Section 4.6.3, is used for the optimization, the training is said to be done by **back-propagation**, which means that the weights are adjusted in proportion to

\[
\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_{ji}}
\]  

(4.54)

according to Rumelhart et al. (1986). The studied error measure \( E \) is the sum of the squared differences between the target output and the actual output from the network over all \( n \) points in the training set

\[
E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2
\]  

(4.55)
The adjustments of the weights start at the output layer and is thus based on the difference between
the response from the NN and the target response from the training set. For the hidden layers,
where there is no specified target value \( y_i \), the adjustments of the weights are instead determined
recursively based on the sum of the changes at the connecting nodes multiplied with their respective
weights. In this way the adjustments of the weights are distributed backwards in the network and
hence the name back-propagation.

It has been shown by Hornik et al. (1989) that FFNNs with one hidden layer can approximate any
continuous function to any desired degree of accuracy, given a sufficient number of neurons in the
hidden layer and the correct interconnection weights and biases. In theory, FFNN metamodels thus
have the flexibility to approximate very complex functions, and these metamodels are therefore well
suited for global approximations of the design space.

The decision of the appropriate number of neurons in the hidden layer or layers is not trivial.
Generally, the correct number of neurons in the hidden layer(s) is determined experimentally, i.e. a
number of candidate networks are constructed and the one judged to be the best is then selected.
Only one hidden layer is often used. Although FFNNs with one hidden layer theoretically should be
able to approximate any continuous function, only one hidden layer is not necessarily optimal. One
hidden layer may require many more neurons to accurately capture complex functions than a
network with two hidden layers. In a network with two hidden layers, it might be easier to improve
an approximation locally without making it worse elsewhere, according to Chester (1990).

Evidently, if the number of free parameters is sufficiently large and the training optimization is run
long enough, it is possible to drive the training error as close to zero as preferred. However, that is
not desirable since it can lead to \textit{overfitting} instead of a model with good prediction capabilities. An
overfitted model does not capture the underlying function properly. It rather describes the noise
instead of the underlying relationship and can also give poor predictions even for noise-free data, see
Figure 4.12. Overfitting generally occurs when a model is excessively complex, i.e. when having too
many parameters relative to the number of observations in the training set. On the other hand, if the
network model is not sufficiently complex, the model can also fail in capturing the underlying
function leading to \textit{underfitting}, see Figure 4.12. Given a fixed amount of training data, it is beneficial
to reduce the number of weights and biases as well as the size of them in order to avoid overfitting.

\[ \text{a) Underfitting} \quad \text{a) Overfitting} \]

\[ \text{Figure 4.12 Examples of models with poor prediction capabilities due to a) underfitting, where the}
\text{model is not complex enough, and b) overfitting, where the model is excessively complex.} \]
Regularization means that some constraints are applied to the construction of the NN model in order to reduce the prediction error in the final model. For FFNN models, regularization may be done by controlling the number of hidden neurons in the network. Another way is to impose penalties on the weights and biases or to use a combination of both methods as described by Stander et al. (2010). A fundamental problem when modelling noisy and/or using very limited data is to balance between the goodness of fit and the choice of how tough the constraints forced on the model by regularization should be.

4.4.5 Radial Basis Functions and Radial Basis Function Networks

Radial basis function (RBF) methods for interpolating scattered multivariate (multiple variables) data were first studied by the geodesist Roland Hardy and a description could be found in Hardy (1990). Radial basis functions depend only on the radial distance from a specific point \( x_i \) such that

\[
\phi(x, x_i) = \phi(||x - x_i||) = \phi(r)
\]  

(4.56)

where \( r \) is the distance between the points \( x \) and \( x_i \). The RBFs can be of many forms but are always radially symmetric. The Gaussian function and Hardy’s multiquadrics are commonly used and expressed as

\[
\phi(r) = e^{-\frac{r^2}{2c^2}}
\]  

(4.57)

and

\[
\phi(r) = \sqrt{r^2 + c^2}
\]  

(4.58)

respectively, where \( c \) is a shape parameter that controls the smoothness of the function, see also Figure 4.13.

**Figure 4.13** Examples of radial basis functions.

An RBF metamodel consists of a linear combination of radially symmetric functions to approximate complex responses, which can be expressed as

\[
\hat{y} = s(x) = \sum_{i=1}^{n} w_i \phi(||x - x_i||) = \mathbf{w}^T \phi
\]  

(4.59)
The metamodel is thus represented by a sum of \( n \) RBFs, each associated with a sample point \( x_i \), representing the centre of the RBF, and weighted by a coefficient \( w_i \). The coefficients \( w_i \), i.e., the unknown parameters that need to be determined when building the metamodel, can be collected in a vector \( w \). The vector \( \Phi \) contains the evaluations of the RBF for all distances between the studied point \( x \) and the sample designs \( x_i \).

Radial basis function approximations are often used in combination with interpolation, i.e., the parameters \( w_i \) are chosen, if possible, such that the approximation matches the responses in the sampled dataset \((x_i, y_i)\) where \( i = 1, \ldots, n \). These conditions together with Equation (4.59) result in a square \( n \times n \) linear system of equations in \( w_i \)

\[
y = Bw
\]

(4.60)

where \( y \) is the vector of responses, \( w \) is the vector of unknown coefficients, and \( B \) is the \( n \times n \) symmetric interpolation matrix. The number of RBFs is thus equal to the number of samples in the dataset. The elements of the interpolation matrix \( B \) contain evaluations of the RBF for the distances between all the fitting points

\[
B_{ij} = \phi(\|x_i - x_j\|)
\]

(4.61)

The equation system (4.60) can be solved by standard methods, using matrix decompositions, for small \( n \). Special methods need to be applied when \( n \) becomes too large, as described by Dyn et al. (1986), since the interpolation matrix often is full and ill-conditioned.

When the number of basis functions \( n_{RBF} \) is smaller than the sample size \( n_s \), the model will be approximating. Similarly to the polynomial regression model, the optimal weights in the least squares sense is analytically obtained as

\[
w = (B^T B)^{-1} B^T y
\]

(4.62)

where \( B \) is an \( n_s \times n_{RBF} \) matrix with elements \( B_{ij} \) as described in Equation (4.61) for \( i = 1, \ldots, n_s \) and \( j = 1, \ldots, n_{RBF} \) and \( x_j \) represents the centre of the basis functions.

The shape parameter \( c \) in Equations (4.57) and (4.58) plays an important role since it affects the conditioning of the problem. When \( c \rightarrow \infty \), the elements of the interpolation matrix \( B \) approach constant values and the problem becomes ill-conditioned. In a physical sense, the shape parameter \( c \) controls the width of the functions and thereby the influence of nearby points. A large value of \( c \) gives a wider affected region, i.e., points further away from an unknown point will have an effect on the prediction of the response at that point. A small value of \( c \), on the other hand, means that only nearby points will influence the prediction. Consequently, the selection of \( c \) also influences the risk of over- or underfitting, see Figure 4.14. If the value is chosen too small, overfitting will occur, i.e., every sample point will influence only the very close neighbourhood. On the other hand, if the value is selected too large, underfitting will appear and the model loses fine details. So, while the correct choice of \( w \) will ensure that the metamodel can reproduce the training data, the correct estimate of \( c \) will enable a smaller prediction error in unknown points.

Gaussian RBFs have a desirable feature in that the prediction error easily can be evaluated at any \( x \) in the design space by
\[ s^2(x) = 1 - \Phi^T \Phi \]  \hspace{1cm} (4.63)

according to Forrester and Keane (2009). This is very useful in sequential sampling, see Section 4.2.3.

**a) Underfitting**

![Underfitting Diagram](image)

**b) Overfitting**

![Overfitting Diagram](image)

**Figure 4.14** Examples of models with poor prediction capabilities due to **a)** underfitting, where the width of the RBFs is too large, and **b)** overfitting, where the width of the RBFs is too small.

A Kriging metamodel can be seen as a special case of an RBF metamodel combined with an additional low order polynomial. In fact, a simple \( f(x) = 0 \) in Equation (4.41) and isotropic \( \theta_r = \text{constant} \) in Equation (4.45) Kriging model with Gaussian correlation functions has the same form as an RBF model with Gaussian basis functions.

RBF metamodels can also be seen as artificial neural networks with activation functions in the form of RBFs, as mentioned in Section 4.4.4. An RBF network has a defined three-layer architecture with the single hidden layer built of non-linear radial units, each responding only to a local region of the design space. The input layer is linear and the output layer performs a biased weighted sum of the hidden layer units and creates an approximation over the entire design space, see Figure 4.15. The RBF network model is sometimes complemented with a linear part corresponding to additional direct connections from the input neurons to the output neuron.

![RBF Network](image)

**Figure 4.15** Schematic illustration of an RBF network with Gaussian activation functions.
Gaussian functions and Hardy’s multiquadrics, respectively, as defined in Equations (4.57) and (4.58), are commonly used RBFs. The activation of the $m^{th}$ RBF is determined by the Euclidean distance

$$ r = \sqrt{\sum_{i=1}^{k} (x_i - w_{mi})^2} $$

between the input vector $x = (x_1, \ldots, x_k)^T$ and the RBF centres $w_m = (w_{m1}, \ldots, w_{mk})$ in the $k$-dimensional space. For a given input vector $x$, the output from an RBF network with $k$ input neurons and a hidden layer consisting of $M$ RBF units (but without a linear part) is given by

$$ \hat{y} = b + \sum_{m=1}^{M} w_m f(a_m) $$

(4.65)

where, in the case of a Gaussian model,

$$ f(a_m) = e^{-\|a_m\|^2} = e^{-w_{m0} \sum_{i=1}^{k} (x_i - w_{mi})^2} $$

(4.66)

This means that the hidden layer parameters $w_m = (w_{m1}, \ldots, w_{mk})$ represent the centre of the $m^{th}$ radial unit, while $w_{m0}$ determines its width. The parameters $b$ and $w_{11}, \ldots, w_{M1}$ are the bias and weights of the output layer, respectively. All these parameters and the number of neurons $M$ need to be determined when building the RBF network metamodel. Note the similarities between the RBF metamodel in Equation (4.59) and the RBF network metamodel in Equation (4.65).

In the same way as a feedforward neural network can approximate any continuous function to any desired degree of accuracy, an RBF network with enough hidden neurons can too. An important feature of the RBF networks which differs from the FFNNs, is that the hidden layer parameters, i.e. the parameters governing the RBFs, can be determined by semi-empirical, unsupervised training techniques. This means that RBF networks can be trained much faster than FFNNs although the RBF network may require more hidden neurons than a comparable FFNN, see Stander et al. (2010).

The training process for RBF networks are generally done in two steps. First, the hidden layer parameters, i.e. the centre and width of the radial units, are set. Then, the bias and weights of the linear output layer are optimized, while the basis functions are kept fixed. In comparison, all of the parameters of an FFNN are usually determined at the same time as part of a single optimization procedure (training), as described in Section 4.4.4. The optimization in the second step of the RBF network training is done to minimize some performance criterion, e.g. the mean sum of squares of the network errors on the training set (MSE), see Equation (4.83). If the hidden layer parameters are kept fixed, the performance function MSE is a quadratic function of the output layer parameters and its minimum can be found as the solution to a set of linear equations. The possibility of avoiding time consuming non-linear optimization during the training is one of the major advantages of RBF networks compared to FF networks.

Commonly, the number of RBFs are chosen to be equal to the number of samples in the training dataset ($M = n$), the RBF centres are set at the fitting designs ($w_m = x_m, m = 1, \ldots, n$), and the widths of the radial units are all selected equal. In general, the widths are set to be a multiple $s_w$ of the average distance between the RBF centres so that they overlap to some degree and hence give a
relatively smooth representation of the data. Sometimes the widths instead are individually set to the distance to the $n_w (<< n)$ closest neighbours so that the widths become smaller in areas with many samples close to each other. This gives a model which preserves fine details in densely populated areas and interpolates the data in sparse areas of the design space, and could therefore be beneficial in sequential optimization where the metamodel is refined iteratively around the optimum solution.

When building an RBF network metamodel, the goal is to find a smooth model that captures the underlying functional response without fitting potential noise, i.e. avoid overfitting. However, for noisy data, the exact RBF network that interpolates the training dataset is typically a highly oscillatory function, and this needs to be addressed when building the model. Similarly as can be done for an FFNN or a Kriging model, regularization can be applied to adjust the output layer parameters in the second phase of the training. This will then yield a model that no longer passes through the fitting points. However, more effective is probably to properly select the hidden layer parameters, i.e. the width and centres of the RBF units, in the first step of the training. Regularization in the second step can never compensate for large inaccuracies in the model parameters. Another way of constructing an approximating model is to reduce the number of RBFs. This could be done by starting with an empty subset of basis functions and adding, one at a time, the basis function which reduces some error metric the most. The selection is done from the $n$ possible basis functions, which are centred around the observed data points $x_i$, and the process is continued until no significant decrease in the studied error metric is observed.

Since the accuracy of the metamodel strongly depends on the hidden layer parameters, it is important to estimate them well. Instead of just selecting the values, the widths can be found by looping over several trial values of $s_w$ or $n_w$ and finally selecting the best RBF network. The selection can for example be based on the generalized cross validation error which is a measure of goodness of fit that also takes the model complexity into account, see Section 4.5.4. Another solution to find the best possible RBF network metamodel can be to include the widths as adjustable parameters along with the output layer parameters in the second step of training. However, this requires a non-linear optimization in combination with a sophisticated regularization, and one of the benefits with the RBF networks, the speed of training, will be lost.

4.4.6 Multivariate Adaptive Regression Splines

*Multivariate adaptive regression splines (MARS)* is a non-parametric regression procedure introduced by Jerome Friedman that automatically models non-linearities and interactions but is normally not interpolating the fitting data, see Friedman (1991). The approximation does not have a predefined form but is constructed based on information derived from the fitting data. MARS builds the metamodel from a set of coefficients $a_m$ and basis functions $B_m$ that are adaptively selected through a forward and backward iterative approach.

A spline is a continuous function of piecewise polynomial real functions. The connection points between the polynomials are called knots and the highest order of polynomial used gives the order of the spline. A MARS model is built from truncated power functions representing $q^{th}$ order splines

\[
\begin{align*}
  b_q^+ (x - t) &= \left[+ (x - t) \right]^q = \max \{0, (x - t)^q\} \\
  b_q^- (x - t) &= \left[- (x - t) \right]^q = \max \{0, -(x - t)^q\} = \max \{0, (t - x)^q\}
\end{align*}
\]  

(4.67)
where \( t \) is the truncation location, i.e. the knot location, and \( q \) is the order of the spline. The subscript “+” indicates that the argument, i.e. the value within the squared brackets, is positive. For \( q > 0 \), the spline is continuous and has \( q - 1 \) continuous derivatives. Often \( q = 1 \) is recommended and the splines then become "hinge functions" as can be seen in Figure 4.16. The resulting MARS model will then have discontinuous derivatives but could be modified according to a description in Friedman (1991) to have continuous first order derivatives.

The MARS metamodel can be written as

\[
\hat{y}(x) = a_0 + \sum_{m=1}^{M} a_m B_m(x) \tag{4.68}
\]

which could be seen as a weighted sum of basis functions.

The coefficients \( a_m \) are estimated through least-squares regression of the basis functions \( B_m(x) \) to the responses \( y_i \) (\( i = 1, \ldots, n \)) in the fitting set. Each basis function \( B_m \) is either a one-sided truncated function \( b \) as described by Equation (4.67), or a product of two or more of these functions

\[
B_m(x) = \prod_{j=1}^{J_m} \left[ s_{jm} \cdot (x_{v(j,m)} - t_{jm}) \right]^q_+ \tag{4.69}
\]

where \( J_m \) is the number of factors in the \( m^{th} \) basis function, i.e. the number of functions \( b \) in the product. The parameter \( s_{jm} = \pm 1 \) and indicates the "left" or "right" version of the function, \( x_{v(j,m)} \) denotes the \( v^{th} \) variable where \( 1 \leq v(j,m) \leq k \) and \( k \) is the total number of variables, and \( t_{jm} \) is the knot location for each of the corresponding variables. As previously, \( q \) indicates the power of the function.

**Figure 4.16** A mirrored pair of hinge functions with the knot at \( x = t \).

Building a MARS metamodel is done in two steps. The first step starts with \( a_0 \), which is the mean of the response values in the fitting set. Basis functions \( B_m \) and \( B_{m+1} \) are then added in pairs to the model, choosing the ones that minimize a certain measure of lack of fit. Each new pair of basis functions consists of a term already in the model multiplied with the "left" and "right" version of a truncated power function \( b \), respectively. The functions \( b \) are defined by a variable \( x_v \) and a knot location \( t \). When adding a new pair of basis functions, the algorithm must therefore search over all combinations of the existing terms of the metamodel (to select the term to be used), all variables (to select the one for the new basis function), and all values of each variable (to find the knot location). For each of these combinations, the best set of coefficients \( a_m \) is found through least square
regression of the model response $\hat{y}$ to the response from the fitting set $y$. The process of adding terms to the model is continued until a pre-defined maximum number of terms are reached or until the improvement in lack of fit is sufficiently small. This so called forward pass usually builds a model that overfits the data. The second step of the model building is therefore a backward pass where model terms are removed one by one, deleting the least effective term until the best metamodel is found. The lack of fit for the models is calculated using a modified form of generalized cross validation (see Section 4.5.4), which takes both the error and complexity of the model into account. More details about this could be found in Friedman (1991). The backward pass has the advantage that it can choose to delete any term except $a_0$. The forward pass can only add pairs of terms at each step, which are based on the terms already in the model.

A lot of searches need to be done during the model building. However, Jin et al. (2001) state that one of the advantages of the MARS metamodel, compared to Kriging, is the reduction in computational cost associated with building the model.

### 4.4.7 Support Vector Regression

**Support vector regression (SVR)** comes from the theory of support vector machines (SVM), which original algorithm was developed by Vladimir Vapnik and co-workers at AT&T Bell Laboratories in the 1990s. See e.g Smola and Schölkopf (2004) for more details about SVR and its background.

When it comes to metamodels, SVR can be seen to have similarities with other methods. The SVR metamodel can be described by the typical mathematical formulation

$$\hat{y}(x) = b + w \cdot Q(x) = b + \sum_{m=1}^{M} w_m Q_m(x) \quad (4.70)$$

Hence, a sum of basis functions $Q = [Q_1(x), \ldots, Q_M(x)]^T$ with weights $w = [w_1, \ldots, w_M]^T$ added to a base term $b$. This can be compared to the RBF models described by e.g. Equation (4.59). The parameters $b$ and $w_m$ are to be estimated, but in a different way than the counterparts in RBF and Kriging. The basis functions $Q$ in the SVR model could also be seen as a transformation of $x$ into some feature space in which the model is linear, see Figure 4.17.

![Figure 4.17](image)

**Figure 4.17** SVR metamodel in one design variable with support vectors marked with dark dots and the designs disregarded in the model build marked with light dots. The non-linear SVR model is reduced to a linear SVR model by the mapping $Q$ from input space into feature space and the support vectors contribute to the cost by the $\varepsilon$-insensitive loss function.
One of the main ideas with SVR is that a margin $\varepsilon$ is given within which a difference between the fitting set responses and the metamodel prediction is accepted. This means that the fitting points that lie within the $\pm \varepsilon$ band (called the $\varepsilon$-tube) are ignored, and the metamodel is defined entirely by the points called support vectors that lie on or outside this region, see Figure 4.17. This can be useful when the fitting data has an element of random error due to numerical noise etc. A suitable value of $\varepsilon$ might then be found by a sensitivity study. In practical cases, however, the dataset is often not large enough to afford not to use all of the samples when building the metamodel. In addition, the time needed to train an SVR model is longer than what is required for many other metamodels.

Estimating the unknown parameters of an SVR metamodel is an optimization problem. The goal is to find a model that has at most a deviation of $\varepsilon$ from the observed $y_i$ ($i = 1, \ldots, n$) and at the same time minimizes the model complexity, i.e. makes the metamodel as flat as possible in feature space, see Smola and Schölkopf (2004). Flatness means that $w$ should be small, which can be ensured by minimizing the vector norm $\|w\|^2$. Since it might be impossible to find a solution that approximates all $y_i$ with precision $\pm \varepsilon$ and that better predictions might be obtained if the possibility of outliers are allowed, slack variables $\xi^+$ and $\xi^-$ can be introduced, see Figure 4.17, and the optimization problem can then be stated as

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} (\xi_i^+ + \xi_i^-) \\
\text{subject to} & \quad y_i - w \cdot Q(x_i) - b \leq \varepsilon + \xi_i^+ \\
& \quad w \cdot Q(x_i) + b - y_i \leq \varepsilon + \xi_i^- \\
& \quad \xi_i^+, \xi_i^- \geq 0
\end{align*}
\] (4.71)

This problem is a trade-off between model complexity and the degree to which errors larger than $\varepsilon$ are tolerated. This trade-off is governed by the user defined constant $C > 0$, and this method of tolerating errors is known as the $\varepsilon$-insensitive loss function, see Figure 4.17. Other loss functions are also possible. The $\varepsilon$-insensitive loss function means that no loss will be associated to the points inside the $\varepsilon$-tube, while points outside will have a loss that increases linearly with a rate determined by $C$. A small constant will lead to a flatter prediction, i.e. more emphasis on minimizing $\|w\|^2$, usually with fewer support vectors. A larger constant will lead to closer fitting of the data, i.e. more emphasis on minimizing $\sum(\xi^+ + \xi^-)$, usually with a larger number of support vectors. Although there might be an optimum value of $C$, the exact choice is not critical according to Forrester and Keane (2009). It is therefore sufficient to try a few values of $C$ of varying orders of magnitude and choose the one which gives the lowest error measure.

**Table 4.3** Kernel functions for SVR where $c$, $\vartheta$, and $\kappa$ are constants.

<table>
<thead>
<tr>
<th>Kernel function</th>
<th>Mathematical description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$k(x_i, x_j) = x_i \cdot x_j$</td>
</tr>
<tr>
<td>homogeneous polynomial of degree $d$</td>
<td>$k(x_i, x_j) = (x_i \cdot x_j)^d$</td>
</tr>
<tr>
<td>inhomogeneous polynomial of degree $d$</td>
<td>$k(x_i, x_j) = (x_i \cdot x_j + c)^d, c \geq 0$</td>
</tr>
<tr>
<td>Gaussian</td>
<td>$k(x_i, x_j) = \frac{\exp(-|x_i - x_j|^2)}{c}$</td>
</tr>
<tr>
<td>Hyperbolic tangent</td>
<td>$k(x_i, x_j) = \tanh(\vartheta + \kappa(x_i \cdot x_j))$</td>
</tr>
</tbody>
</table>
In most cases, the optimization problem described by Equation (4.71) is more easily solved in its dual form, and is therefore written as the minimization of the corresponding Lagrangian function $L$. In optimum, the partial derivatives of $L$ with respect to its primal variables $w$, $b$, $\xi^+$, and $\xi^-$ must vanish, which leads to the optimization problem in dual form

$$
\text{max} \quad -\frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-)k(x_i, x_j) - \varepsilon \sum_{i=1}^{n} (\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) \\
\text{subject to} \quad \sum_{i=1}^{n} (\alpha_i^+ - \alpha_i^-) = 0 \\
(\alpha_i^+ - \alpha_i^-) \in [0, C]
$$

(4.72)

where $\alpha_i^+$ and $\alpha_i^-$ are dual variables (Lagrange multipliers) and $k(x_i, x_j) = Q(x_i) \cdot Q(x_j)$ represents the so-called kernel function. This problem can be solved using a quadratic programming algorithm to find the optimal choices of the dual variables. The kernel functions need to have certain properties and possible choices include linear and Gaussian functions etc. as seen in Table 4.3 and described by Smola and Schölkopf (2004).

The partial derivative of $L$ with respect to $w$ being zero yields $w = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)Q(x_i)$. This means that, Equation (4.70) can be rewritten and give the response in an unknown point $x_u$ as

$$
\hat{y}(x_u) = w \cdot Q(x_u) + b = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)Q(x_i) \cdot Q(x_u) + b = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)k(x_i, x_u) + b
$$

(4.73)

The optimization problem in Equations (4.71) and (4.72) corresponds to finding the flattest function in feature space, not in input space. The base term is still unknown but could be determined from

$$
b = y_j - w \cdot Q(x_j) + \varepsilon = y_j - \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)k(x_i, x_j) + \varepsilon \quad \text{if} \quad 0 < \alpha_i^- < C
$$

(4.74)

$$
b = y_j - w \cdot Q(x_j) - \varepsilon = y_j - \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)k(x_i, x_j) - \varepsilon \quad \text{if} \quad 0 < \alpha_i^+ < C
$$

(4.75)

which means that $b$ can be calculated for one or more $\alpha_i^+$ that fulfil the conditions. Better results are obtained for $\alpha_i^+$ not too close to the bounds according to Forrester and Keane (2009). The set of equations could also be solved via linear regression.

It can be seen that SVR methods produce RBF networks with all width parameters set to the same value and centres corresponding to the support vectors. The number of basis functions, i.e. hidden layer units, $M$ in Equation (4.65), is thus the number of support vectors.

### 4.4.8 Which Metamodel to Use?

A number of metamodel types have been described in the previous sections. The presentation is not by any means complete, as other methods and variants of the presented methods exist. The question arises regarding which metamodel to use. As can be understood from the preceding presentation, the different metamodels have their unique properties and consequently there is no universal model that always is the best choice. Instead, the suitable metamodel depends on the problem at hand. It is
for example important to decide whether the model should be a global approximation valid over the entire design space or if it should be a local approximation. A basic knowledge about the complexity of the response the metamodel should capture is useful when choosing between metamodel types. Another decision that needs to be taken is whether noise is assumed to be present in the fitting set or not. An interpolating model might be the best choice in the noise-free case, while an approximating model may be better when noise is present. However, it should be noted that there is no guarantee that an interpolating model gives better predictions in unknown points compared to an approximating one, even if there is no noise present.

If several metamodels are built based on the same fitting set, the selection of the best one is not a trivial task. Methods described in Section 4.5 can be used to assess the accuracy of the metamodels and consequently make it possible to compare them and guide the selection. These methods are similar to the ones used when selecting architectures for neural networks etc.

Many comparative studies have been made over the years to guide the selection of metamodel types. A study by Jin et al. (2001) compared polynomial, Kriging, MARS, and RBF models. The authors concluded that RBF metamodels are the best choice in most cases, especially for small fitting sets. When non-linearity is not pronounced, polynomial and Kriging models were found to be good choices for problems with small or large number of variables, respectively. For the most difficult problems with highly non-linear responses and many variables, the MARS model was found to be the most accurate if the sample set was large enough, otherwise RBF performed the best. It was also noted that Kriging models are very sensitive to noise as they generally interpolate the fitting data.

In more recent studies, SVR models have shown promising results. In a comparison with polynomial, Kriging, RBF, and MARS metamodels, it was found by Clarke et al. (2005) that SVR had the best overall performance regarding accuracy and robustness. In another study by Li et al. (2010) where artificial neural network, RBF network, SVR, Kriging, and MARS metamodels were compared for stochastic problems, it was found that SVR performed best in terms of prediction accuracy and robustness, followed by Kriging. For more complicated problems with higher dimension as well as larger and heterogeneous error, it was found that RBF can serve as an alternative. In contrast to these studies, Kim et al. (2009) compared MLS, Kriging, SVR, and RBF metamodels and found that Kriging and MLS gave more accurate metamodels compared to RBF and SVR models. Thus, it is not possible to draw any decisive conclusions regarding the superiority of any of the presented metamodels. As noted from the previous presentation, there are often several parameters that must be tuned when building a metamodel. This means that results can differ considerably depending on how well these parameters are tuned and consequently also depend on the software used.

Instead of selecting only the metamodel believed to be the best, another idea is to use several models. The time for running the detailed simulation model to determine the responses in the DOE is often much longer compared to building the metamodels and use them for optimization (hours or days compared to minutes). It can therefore be worthwhile to perform the optimization repeatedly with different metamodels, probably leading to more than one candidate design. In addition, the most accurate metamodel does not always lead to the best design and this method can therefore help to avoid overlooking potential good solutions.

Several different metamodels can also be combined. The idea is that the combined model should perform at least as well as the best individual metamodel but at the same time protect against the worst individual metamodel. A **weighted average surrogate (WAS)** makes a weighted linear
combination of $m$ metamodels in the hope of cancelling prediction errors through a proper selection of the weights.

\[
\hat{y}_{\text{WAS}}(x) = \sum_{i=1}^{m} w_i(x) \hat{y}_i(x) = w^T(x) \hat{y}(x)
\] (4.76)

and

\[
\sum_{i=1}^{m} w_i(x) = 1^T w(x) = 1
\] (4.77)

where $\hat{y}_i(x)$ is the response predicted by the $i^{th}$ metamodel and $w_i(x)$ is the weight associated with the $i^{th}$ metamodel at design point $x$. A metamodel that is judged to be more accurate should be assigned a large weight, and a less accurate metamodel should have a lower weight resulting in a smaller influence on the predictions. The evaluation of the accuracy is done with different measures of goodness of fit and could be either global or local. When weights are selected based on some global measure, the weights are fixed in space, i.e. $w_i(x) = C_i$ for all $x$. This is for example done by Goel et al. (2007a) using the generalized mean squared cross validation error. If the weights are based on some local measure, the weights are instead functions of space, i.e. $w_i = w_i(x)$. Different metamodels could thus have the largest influence on the prediction in different areas of the design space. This is demonstrated by Zerpa et al. (2005) using prediction variance.

Another way of combining metamodels can also be used if there are enough samples in the fitting set. A multi-surrogate approximation (MSA) is created by first classifying the given samples into clusters based on their similarities in the design space. Then, a proper local metamodel is identified for each cluster and a global metamodel is constructed using these local metamodels, as described by Zhao and Xue (2011). This method is particularly useful when sample data from various regions of the design space are of different characteristics, e.g. with and without noise.

### 4.5 Metamodel Validation

The accuracy of a metamodel is influenced by the metamodel type as well as the quality and quantity of the dataset from which it is built. There is not one single measure that can describe the goodness of the model. Instead, there are several measures and methods that could be used for assessing the accuracy of a metamodel and comparing it to others. It is also important to know the intended use of the metamodel to judge if it is acceptable. Initially, when identifying important design variables and interesting areas of the design space, the demands on metamodel accuracy is not as high as later in the process when potential trade-offs between competing requirements should be evaluated. The method used to check the accuracy of a metamodel has to be decided on a case to case basis. A balancing between the effort that is needed for the validation and the gain in knowledge of the accuracy must also be made.

#### 4.5.1 Error Measures

In general, the accuracy of a metamodel can be evaluated by its residuals, i.e. the difference between the simulated value, $y_i$, and the predicted value from the metamodel, $\hat{y}_i$. Small residuals mean that the model reflects the dataset more accurately than if the residuals were larger. Several different...
error measures can be evaluated based on these residuals, as described by Topuz (2007) and presented in more detail below.

The coefficient of determination $R^2$ is a measure of how well the metamodel is able to capture the variability in the dataset and is defined as

$$R^2 = 1 - \frac{SS_{err}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = \frac{SS_{reg}}{SS_{tot}} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

(4.78)

where $n$ is the number of design points and $\bar{y}, \hat{y}_i, y_i$ represent the mean, the predicted, and the actual response as defined in Figure 4.18. The total sum of squares, $SS_{tot}$, equals the sum of the regression sum of squares $SS_{reg}$ (the explained variability, i.e. the variability of the model predictions), and the residual sum of squares $SS_{err}$ (the unexplained variability, i.e. the variability of the model errors),

$$SS_{tot} = SS_{err} + SS_{reg}$$

(4.79)

The closer to 1 the $R^2$ value is, the better, since a value of 1.0 indicates a perfect fit. However, a high $R^2$ value can be deceiving if it is due to overfitting which, in turn, means that the model will have poor prediction capabilities between the fitting points. Another occasion when the $R^2$ value could be misleading is when the response is insensitive to the studied variables, i.e. the metamodel equals the mean value of the observed responses. In this case $R^2$ will be close to 0 even for a well fitted model, see Figure 4.19.

Some metamodels are interpolating the dataset, which means that there are no residuals and $R^2$ equals 1.0, see Figure 4.20. For the deterministic simulation case without random error or numerical noise, this is of course the ideal. But there is no guarantee that these interpolating metamodels are predicting the response between the known points better than other models. In some cases, numerical noise is also present and it can then be beneficial to filter the response by a non-interpolating model.
**Figure 4.19** Different cases when $R^2$ can be misleading. **a)** Overfitted model where $R^2 = 1$. **b)** The response is insensitive to the variable, i.e. $y = \bar{y}$ and $R^2 = 0$.

**Figure 4.20** Examples of interpolating metamodels, i.e. $R^2 = 1$.

Since it is insufficient to study the $R^2$ value, a good way of validating the metamodel is to use additional points, which are not used for fitting the model, and evaluate the errors in these points. The error measures evaluated in these $m$ validation points, the validation set, can for example be maximum absolute error (MAE), average absolute error (AAE), mean absolute percentage error (MAPE), mean squared error (MSE), and root mean squared error (RMSE).

\[
MAE = \max_{i=1}^{m} |y_i - \hat{y}_i|, \quad i = 1, ..., m \quad (4.80)
\]

\[
AAE = \frac{\sum_{i=1}^{m} |y_i - \hat{y}_i|}{m} \quad (4.81)
\]

\[
MAPE = \frac{\sum_{i=1}^{m} |y_i - \hat{y}_i|}{\sum_{i=1}^{m} y_i} \times 100\% \quad (4.82)
\]

\[
MSE = \frac{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2}{m} \quad (4.83)
\]

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2}{m}} \quad (4.84)
\]
The lower these error measures are the more accurate the metamodel is. The AAE, MAPE, MSE and RMSE give a measure on the overall accuracy while the MAE is a measure on the local accuracy of the model. RMSE is the most commonly used metric but can be biased as the residuals are not relatively measured. If the dataset contains both high and low response values, it might be desirable to equally account a small error on a small response as a larger error on a larger response. The MAPE measure takes this aspect into consideration. If validation of the metamodel is done by studying error measures for a validation set, it is important that the validation set is large enough and spread out over the design domain to give a reliable picture of the accuracy. It is also important according to Loos et al. (2010) that the points in the validation set are not placed too close to the fitting points since it could lead to a too optimistic evaluation of the metamodel.

The $R^2$ value can also be evaluated for the validation set and be another measure of the accuracy of the metamodel. In the same way as the $R^2$ value can be evaluated both for the fitting set and the validation set, the other standard error measures mentioned previously can as well. When the error measures are evaluated for the fitting set, they indicate how well the metamodel is representing the fitting data, but does not tell how well it predicts the response in other points. All these measures are consequently not very meaningful for interpolating metamodels when they are evaluated for the fitting points. However, the error measures can also be used in the method described next, and are consequently not limited to only one accuracy check method. It is therefore important to know how the measures are obtained to understand their meaning.

### 4.5.2 Cross Validation

Another way of assessing the quality of a metamodel and comparing it to others is called cross validation (CV), see Meckesheimer et al. (2002). The methodology makes it possible to compare interpolating metamodels with non-interpolating ones. With this approach, the same dataset is used for fitting and validating the model. When the simulation time is long and the available data is limited, it can be desirable to use the complete dataset for fitting the metamodels and not potentially lower the accuracy by leaving out a part of the set for validation.

In p-fold CV, the dataset of $n$ input-output data pairs is split into $p$ different subsets. The metamodel is then fitted $p$ times, each time leaving out one of the subsets. The omitted subset is used to evaluate the error measures of interest. A variation of the method is the leave-k-out CV, in which all possible $\binom{n}{k}$ subsets of size $k$ are left out, and the metamodel is fitted to the remaining set. Each time, the relevant error measures are evaluated at the omitted points. This approach is computationally more expensive than the p-fold CV. However, for the special case where $k = 1$, called leave-one-out CV, an estimation of the prediction error can inexpensively be computed for some metamodels, e.g. polynomial, Kriging, and RBF models. The generalization error, i.e. prediction error, for a leave-one-out calculation when the error is described by the MSE is represented by

$$\text{MSE}_{CV} = \frac{1}{n} \sum_{i=1}^{n} e_i^2 = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \hat{y}_i^{(-i)} \right)^2$$

(4.85)

where $\hat{y}_i^{(-i)}$ represents the prediction at $x_i$ using the metamodel constructed utilizing all sample points except $(x_i,y_i)$, e.g. see Forrester and Keane (2009).
In Figure 4.21 the different cross validation methods are illustrated for a simple example and it is easily understood that CV can be expensive if many metamodels should be fitted.

**a) Linear metamodel fitted to all 4 points**

![Graph showing linear metamodel fitted to all 4 points](image)

**b) p-fold CV, p = 2**

![Graph showing p-fold CV, p = 2](image)

**c) Leave-one-out CV**

![Graph showing leave-one-out CV](image)

**d) Leave-k-out CV, k = 2**

![Graph showing leave-k-out CV, k = 2](image)

**Figure 4.21** Comparison of different CV methods for a linear metamodel with four available data points. 

- **a)** Metamodel fitted to all points.  
- **b)** P-fold CV with two datasets, i.e. two metamodels fitted and two points available for error estimation for each model.  
- **c)** Leave-one-out CV, i.e. four metamodels fitted and one point available for error estimation for each model.  
- **d)** Leave-k-out CV with $k = 2$, i.e. six metamodels fitted and two points available for error estimation for each model.

The **Prediction Error Sum of Squares (PRESS)** is an error measure often used in regression analysis. It provides a summary measure of the fit of a model to a sample of observations and is used as an indication of the predictive power of the model. In principle, PRESS is calculated by using each possible subset of $n - 1$ responses as the fitting dataset, and the remaining response as the validation set, i.e. leave-one-out CV. However, PRESS can also easily be computed from a single polynomial regression model fitted to all $n$ points according to Myers et al. (2008), thus avoiding the work of fitting many metamodels. The square root of PRESS divided by $n$ is the root mean square prediction error, which is also used sometimes. These two measures can be evaluated as

$$
PRESS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i^{(-i)})^2 = \sum_{i=1}^{n} \left( \frac{y_i - \hat{y}_i}{1 - h_{ii}} \right)^2
$$

(4.86)

and
\[ RMSE_{PRESS} = \sqrt{\frac{PRESS}{n}} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \]  

(4.87)

respectively. The notations follow Equation (4.85) and \( h_{ii} \) is the \( i \)th diagonal element of the “hat” matrix, \( H = X(X^TX)^{-1}X^T \), that maps the simulated responses \( y \) to the fitted responses \( \hat{y} \), i.e.

\[ \hat{y} = Hy \]  

(4.88)

This is consequently a very efficient way of calculating a leave-one-out error for a polynomial metamodel. Myers et al. (2008) also state that if the \( PRESS \) value is available, it is possible to approximate the \( R^2 \) for prediction, which represents the ability of the model to detect the variability in predicting new responses. This statistic can be evaluated in different ways

\[ R^2_{prediction} = 1 - \frac{PRESS}{SS_{tot}} = 1 - \frac{PRESS}{\sum_{i=1}^{n} (y_i - \bar{y})^2} = 1 - \frac{PRESS}{\sum_{i=1}^{n} y_i^2 - \frac{1}{n} (\sum_{i=1}^{n} y_i)^2} \]  

(4.89)

For a polynomial regression model, the \( PRESS \), \( RMSE_{PRESS} \), and \( R^2_{prediction} \) measures defined above, give information on how well the model predicts responses in unknown points, i.e. they are prediction errors. In contrast, if the error measures presented in Section 4.5.1 are calculated for a model fitted to all available data points, the only information obtained is how well the model describes the fitting data, i.e. they are fitting errors.

As mentioned earlier, it is possible to inexpensively estimate the prediction error also for Kriging and RBF models. According to Martin and Simpson (2004), the vector of leave-one-out errors for a Kriging model fitted to all \( n \) points could be evaluated as

\[ e = Q(R^{-1}y - R^{-1}Xb) \]  

(4.90)

where \( R \) is the correlation matrix, \( y \) is the vector of observed responses, \( b \) is the vector of estimated regression coefficients, \( X \) is the model matrix, and \( Q \) is a diagonal matrix with elements that are the inverse of the diagonal elements of \( R^{-1} \). Using this in the first part of Equation (4.86) yields the PRESS-value.

For an RBF metamodel on the form \( \hat{y}(x) = b + \sum_{i=1}^{n_{RBF}} w_i f_i(x) \), Goel and Stander (2009) state that the vector of leave-one-out errors can be evaluated as

\[ e = (\text{diag}(P))^{-1}Py \]  

(4.91)

where \( y \) is the vector of observed responses and \( P \) is the projection matrix which is defined by

\[ P = I - F(F^TF + \Lambda)^{-1}F^T \]  

(4.92)

\( F \) is the design matrix constructed using the response of the radial functions at the design points such that \( F_{i,j} = 1, F_{i,j+1} = f_i(x_j), i = 1, \ldots, n \) and \( j = 1, \ldots, n_{RBF} \). \( \Lambda \) is a diagonal matrix where \( \Lambda_{ii}, i = 1, \ldots, n_{RBF} \), is the regularization parameter associated with the \( i \)th weight as briefly mentioned at the end of Section 4.4.5.
It has been shown by Meckesheimer et al. (2002) that $k = 1$ in leave-k-out CV, i.e. leave-one-out, provides a good prediction error estimate for RBF and low-order polynomial metamodels. For Kriging models, instead the recommendation is to choose $k$ as a function of the sample size, e.g $k = 0.1n$ or $k = \sqrt{n}$.

The leave-one-out CV is a measure of how sensitive the metamodel is to lost information at its data points. An insensitive metamodel is not necessarily accurate and an accurate model is not necessarily insensitive to lost information. The leave-one-out CV is therefore not sufficient to measure metamodel accuracy and validation with an additional dataset is therefore recommended by Lin (2004). Small fitting sets, which are common in reality, are not really suitable for CV according to Stander et al. (2010). Data distribution could change considerably when even a small portion of the dataset is removed and used as a validation set. In addition, the CV approach is often expensive since it, in general, involves fitting of several metamodels for the same response. Nevertheless, CV could be the only practical way of obtaining information regarding the predictive capabilities of the metamodels in cases where the simulation budget is restricted and the detailed simulations are very cpu-expensive.

4.5.3 Jack-knifing and Bootstrapping

Leave-one-out CV is sometimes called jackknifing. However, in its original meaning, jackknifing is a technique to estimate the bias of a statistic while CV is used to estimate the prediction error. Similar to leave-one-out CV, the jackknife method is based on fitting a metamodel $n$ times, each time omitting one of the $n$ sample points. The statistic of interest is computed for each of these metamodels, and the average of the values are then compared to the same statistic from a metamodel fitted to all $n$ points in order to estimate the bias of the latter. If jackknifing is used to estimate the bias of the fitting error, it is also possible to obtain an estimation of the prediction error, but Sarle (1997) states that this process is more complicated than the leave-one-out CV.

Another method with similarities to CV is bootstrapping. In its simplest form, instead of repeatedly analyzing subsets of data, subsamples of data are analyzed. These, so called bootstrap samples, are random samples with repeated draws from the full dataset and are of the same size as the original dataset. This means that the bootstrap samples probably will contain some sample points more than once while others will be left out. Many such bootstrap samples are drawn and a metamodel is fitted for each bootstrap sample while the complete dataset is used as a validation set. Based on the errors calculated for these metamodels and the fitting error from a metamodel fitted to all $n$ points, the prediction error of the latter can be estimated. Many versions of bootstrap methods exist and they have been shown to work better than CV in many cases, see e.g. Efron (1983) and Lendasse et al. (2003).

4.5.4 Generalized Cross Validation and Akaike’s Final Prediction Error

Overfitting of a metamodel can lead to a model with a very small fitting error but a large prediction error. Overfitting generally occurs when a model is excessively complex, i.e. when having too many parameters relative to the number of observations in the fitting set. Some measures of goodness of fit have therefore been developed that take both the residual error and the model complexity into account. One of these methods is the generalized cross validation (GCV) described by Craven and Wahba (1979) and another one is the final prediction error (FPE) defined by Akaike (1970). These error measures are evaluated for metamodels of different complexity fitted to the same data set.
The model with the lowest value should then be chosen as the one with the appropriate complexity. For a metamodel with a mean squared fitting error $MSE$, Stander et al. (2010) state that the corresponding GCV and FPE measures are defined by

$$MSE_{GCV} = \frac{MSE}{\left(1 - \frac{\nu}{n}\right)^2}$$

and

$$MSE_{FPE} = MSE \frac{1 + \frac{\nu}{n}}{1 - \frac{\nu}{n}} = MSE \frac{n + \nu}{n - \nu}$$

respectively. $n$ is the number of fitting points, which should be large, and $\nu$ is the number of (effective) model parameters. In the original forms, valid for linear or unbiased models without regularization, $\nu$ is the number of model parameters. Otherwise, e.g. for neural network models, $\nu$ should be the number of effective model parameters which could be estimated in different ways.

### 4.6 Optimization Methods

Optimization can be defined as a procedure for achieving the best solution to a specific problem while satisfying certain restrictions. The basic terminology for optimization was presented in Section 3.1. A general optimization problem was formulated in Equation (3.1) and then simplified to Equation (3.2), which is repeated here for convenience.

$$\min_{x} f(x)$$

subject to $g(x) \leq 0$

(4.95)

where $f$ and $g$ are functions of the design variables $x = (x_1, x_2, ..., x_k)^T$. The objective function $f$ is the quantity to be minimized (or maximized) and the constraint functions $g$ represent the restrictions.

#### 4.6.1 Optimization Strategies

Optimization can be performed using the detailed simulation model, or using its metamodel representation. The first method is called **direct optimization** and is suitable for inexpensive simulations and/or optimization algorithms that require relatively few evaluations to find the solution. The detailed simulations are in many cases computationally expensive, and a single simulation could take hours to run. In these cases, it can be beneficial to first build metamodels and then perform metamodel-based design optimization. Since evaluations on metamodels are very fast compared to evaluations using detailed simulations, it is not as important to have an efficient optimization algorithm for metamodel-based design optimization as it is when performing direct optimization. The focus is therefore then more often put on selecting a robust method that will find the global optimum and not only a local one. The main focus in this report is MBDO. However, the optimization algorithms presented later are not restricted to either of the optimization methods, even if they could be more or less suitable.

The main issue with MBDO is the error that is introduced when approximating the detailed simulations with metamodels. For the method to work properly, the metamodels need to accurately capture the detailed simulation models. If that is the case, it has been found that MBDO is more
Metamodel-based design optimization can be performed using different strategies. If there is only a fixed limited simulation budget available, the best idea is probably to use as many simulations as could be afforded to build the metamodels from a DOE of sampling points selected in one single stage. After proper validation, the metamodels from this DOE are hopefully found to be accurate enough to be used for optimization in a single stage strategy. Another strategy is to do the sampling of points sequentially. A limited number of points are then chosen for each iteration, and more and more refined metamodels are built and used for optimization in a sequential strategy. This approach has the advantage that the iterative process can be stopped as soon as the metamodels or optimum points have sufficient accuracy. Both the above mentioned strategies are good for design exploration but require flexible metamodels that can adjust to an arbitrary number of points and capture complex responses. Polynomial metamodels are therefore not suitable in these cases. Since the metamodels for both these strategies are built to have approximately the same accuracy within the complete design space, these methods are preferred over the ones described in the following when constructing a Pareto optimal front, see Section 3.3.

The sequential strategy with domain reduction is similar to the sequential strategy described above. However, in each iteration the subregion where the new points are selected, also called the region of interest, is reduced in size and moved within the design space to close in on the optimum point. In sequential adaptive metamodelling, all the available points are used for building global metamodels. The approach requires a flexible metamodel that can capture complex responses and is a good method for converging to an optimum. Another very popular method, which has proven to work well in the past, is the sequential response surface method in which only the points in the current iteration is used to build a local (often linear) polynomial metamodel. Despite its simplicity, this method can in fact work remarkably well and outperform the other approaches since global metamodels often are insufficiently accurate according to Duddeck (2008). However, the method is only suitable for convergence to one single optimum and should not be used to construct a Pareto optimal front or to do any other type of design exploration, since the metamodels only are valid locally within the design space. Another drawback is that many iterations can be required to find the optimum point for complex responses. In Figure 4.22 and Figure 4.23, the different strategies are described schematically and compared for the case of a response depending on only one variable.
**Figure 4.22** Schematic illustration of different optimization strategies to find the global maximum. The dark dots indicate current and light dots previous sampling points. The light and dark stars indicate true and estimated optimum respectively. **a)** Single stage strategy in which more points normally give better accuracy. **b)** Sequential strategy in which the first crude estimation is improved iteratively by adding points in the whole design space.
Figure 4.23 Schematic illustration of different optimization strategies to find the global maximum. Notations follow the previous figure, and additionally, white dots indicate disregarded sampling points. **a)** Sequential adaptive metamodeling in which the global metamodel is iteratively refined by adding points in a subregion around the estimated optimum. **b)** Sequential response surface method in which a simple local metamodel is iteratively built in a subregion around the estimated optimum. Note that the given example requires more iterations to converge.
4.6.2 Optimization Algorithm Classification

There are several different algorithms that can be used when solving a specific optimization problem, regardless of the chosen direct or metamodel-based strategy. A local optimization algorithm only attempts to find a local optimum, and there is no guarantee that this optimum also is the global one unless very specific conditions are fulfilled. Thus, if the response is complex enough to have several local optima, different results can be obtained depending on the starting point. Most local optimization algorithms are gradient-based, i.e. they make use of gradient information to find the optimum solution, see e.g. Venter (2010). These techniques are popular because they are efficient, can solve problems with many design variables, and typically require little problem-specific parameter tuning. On the other hand, in addition to only finding local optima, they have difficulty solving discrete optimization problems (in which at least one design variable only can take discrete values) and may be susceptible to numerical noise. When using a local optimization algorithm, a simple way of dealing with multiple local optima in the design space is to use a multi-start approach, in which multiple local searches are performed from different starting points.

In most cases, the global optimum is requested, and a global optimization algorithm has a better chance of finding the global or near global optimum. Global optimization methods can be classified into two main categories: deterministic methods and stochastic (or heuristic) methods, see Younis and Dong (2010). Deterministic methods solve an optimization problem by generating a deterministic sequence of points converging to a globally optimal solution. Such methods behave predictable and given the same input, the algorithm will follow the same sequence of states and give the same result each time. The deterministic methods quickly converge to the global optimum but require the problem to have certain mathematical characteristics that often do not exist. Details regarding these methods are therefore not presented.

The stochastic methods are based on random generation of points that are used for non-linear local optimization search procedures as described by Younis and Dong (2010). The methods are typically inspired by some phenomenon from nature, and have the advantage of being robust and well suited for discrete optimization problems. Compared to the deterministic methods they usually have fewer restrictions on the mathematical characteristics of the problem, can search large design spaces, and do not require any gradient information. On the other hand, they cannot guarantee that an optimal solution is ever found and they often require many more objective function evaluations. Stochastic optimization methods are therefore particularly suitable for MBDO since the evaluations using metamodels are fast. According to Venter (2010), other drawbacks associated with stochastic methods include poor constraint-handling abilities, problem-specific parameter tuning, and limited problem size. Typical stochastic optimization algorithms include genetic algorithms, evolutionary strategies, particle swarm optimization, simulated annealing etc., which could be categorized according to Figure 4.24. These methods will be presented in more detail in subsequent sections.

Since the different optimization algorithms have different benefits, hybrid optimization algorithms can be used in which the merits of different methods are taken advantage of. One example can be to initially perform a global optimization to find the vicinity of the global optimum, and then use a local optimization algorithm to identify the optimum with greater accuracy.

Classification of algorithms can also be done according to whether gradient information is used or not. Non-derivative methods, also called zeroth order algorithms, only make use of the functional values, while derivative methods also takes the gradients into account. Derivative, or gradient-based,
methods can be divided into first and second order methods depending on whether only first order derivatives are used or if also second order derivatives are considered.

Figure 4.24 Classification of global optimization methods including examples of stochastic algorithms.

4.6.3 Gradient-Based Algorithms

Gradient-based algorithms typically use an iterative two-step method to reach the optimum as described by Venter (2010). The first step is to use gradient information to find the search direction and the second step is to move in that direction until no further progress can be made or until a new constraint is reached. The second step is known as the line search and provides the optimum step size. The two-step process is repeated until the optimum is found, see Figure 4.25. Depending on the scenario, different search directions are required. For unconstrained problems and constrained problems without active or violated constraints, a search direction that will improve the objective function is desired. Any such search direction is referred to as a usable direction. If one or more constraints are violated, a search direction that will overcome the constraint violations is desired. For constrained optimization problems with one or more active constraints and no violated constraints, a search direction that is both usable and feasible (do not violate any constraints) is required.

a) Unconstrained optimization 

b) Constrained optimization

Figure 4.25 Schematic picture of a gradient-based optimization algorithm for the case with two design variables. The response values are indicated by the iso-curves and the star represents the optimum solution for a) unconstrained optimization and b) constrained optimization. The unfeasible region violating the constraints is marked by the shaded areas.
Different gradient-based algorithms differ mostly in the logics used to determine the search direction. In the **steepest descent** method, the search direction in iteration $i$ is based only on the gradient of $f(x^i)$ and the search direction is the direction in which the objective function $f(x^i)$ locally decreases the most. If an exact line search is made in each iteration, then two consecutive search directions will be orthogonal to each other. The steepest descent method works well for many problems. However, the method may experience slow convergence for some problems due to the zig-zag shaped trail of points with successively smaller steps. The method is called **steepest ascent** when used for maximization problems. Many other algorithms exist and can be reviewed in detail in books dedicated to the topic, e.g. Lundgren et al. (2010).

For most optimization problems, the gradient information is not readily available but can be obtained using a finite difference technique. However, this way of obtaining the gradients is expensive and typically dominates the total computing time required to complete the optimization. Some numerical simulations can provide gradient information directly. If such gradients are available, those should preferably be used since they are usually obtained at significantly lower computational cost and are often more accurate than the finite difference gradients. In non-linear dynamic simulations, such as crash or metal-forming, the derivatives of the response functions are often severely discontinuous due to contact forces and friction. The response, and thus the derivatives, may also be highly non-linear due to the chaotic nature of impact phenomena and the gradients may therefore not reveal much of the overall behaviour. For these reasons, it can be advantageous for the optimization process to use metamodels that smoothes the responses, see Stander et al. (2010).

When gradient information is available, the **Karush-Kuhn-Tucker (KKT) conditions** can be used to determine if a local optimum has been found for a constrained optimization problem. The KKT conditions are derived from the Lagrangian function of the constrained optimization problem in Equation (4.95) that can be written as

$$L(x) = f(x) + \lambda^T g(x)$$

where $\lambda$ contains one Lagrangian multiplier for each constraint. The KKT conditions (together with some regularity conditions) provide the necessary conditions for a local optimum and can be summarized as:

1. The optimum design point $x^*$ must be feasible, i.e. $g(x^*) \leq 0$  
2. The gradient of the Lagrangian must vanish at $x^*$, i.e. $\nabla f(x^*) + \lambda^T \nabla g(x^*) = 0$ where $\lambda \geq 0$
3. For each inequality constraint $\lambda^T g(x^*) = 0$

If the optimization problem is convex, the local optimum is also the global optimum. An optimization problem is convex if the objective function $f(x)$ is a convex function and the feasible region, defined by the constraints $g(x) \leq 0$, is a convex set. A graphical illustration of the notation convex set and convex function are found in Figure 4.26. More details about KKT criteria and convexity can be found, e.g., in Lundgren et al. (2010).

Many different local gradient-based algorithms are available for solving non-linear constrained optimization problems. The **sequential quadratic programming (SQP)** algorithms are probably the most popular ones for engineering optimization applications as noted by Venter (2010). As for most optimization methods, SQP is not a single algorithm, but rather a conceptual method from which many specific algorithms have evolved. A quadratic programming (QP) problem has a quadratic
objective function and linear constraints. The optimal solution may thus be found anywhere within the region or on its boundary where one or more of the constraints are active. This type of problems can easily be solved if the problem is convex, but is otherwise much harder to solve. The basic idea of a general SQP algorithm is to approximate an arbitrary non-linear optimization problem as a QP subproblem, solve that subproblem, and then use the solution to construct a new subproblem. This construction is then iteratively repeated until the sequence converges to a local optimum.

**a) Convex set**

![Convex set diagram](image)

**b) Convex function**

![Convex function diagram](image)

**Figure 4.26** Definition of convexity. **a)** In a convex set, all points on a line connecting any two points in the set are also in the set. **b)** A convex function is a function where all points on or above the curve form a convex set.

There are other gradient-based algorithms that do not rely on line search to progress; one of them is the **leap-frog optimizer for constrained problems (LFOPC)**, see Snyman (2000). The idea of this method is to see \( f(x) \) as the potential energy of a unit mass particle at point \( x(t) \) and time \( t \), where \( x = (x^1, \ldots, x^k)^T \). The approach is to find the minimum of the function \( f(x) \) by studying an associated dynamic problem of motion of the particle in a \( k \)-dimensional conservative force field, as described by Snyman (1982). In that field, the total potential and kinetic energy of the particle is conserved. The method requires the solution of the equations of motion of the particle subject to initial conditions on the position and velocity. The algorithm computes an approximation of the trajectory followed by the particle in the force field using the so-called leap-frog (Euler forward - Euler backward) method. An interfering strategy that reduces the kinetic energy whenever the particle appears to move uphill is applied. The consequence is a systematic reduction in potential energy \( f(x) \) that forces the particle to a local minimum \( x^* \). The LFOPC algorithm uses a penalty function formulation to incorporate the constraints into the optimization problem. Violations of the constraints are thus multiplied by a penalty value and added to the objective function. The penalty parameter value is first moderate, but is later increased to more strictly penalize the remaining active constraints.
4.6.4 Evolutionary Algorithms

*Evolutionary algorithms (EAs)* try to mimic biological evolution and are inspired by Darwin's principle of survival of the fittest. During the 1960s, different implementations of the basic idea were developed in different places. The algorithms are based on several iterations of a principal evolution cycle as described by Eiben and Smith (2003), see Figure 4.27. The process starts with a random population of candidate designs. The response value representing the objective function gives the fitness of each design in the population. Based on this fitness, some of the better candidates are chosen to seed the next generation. By applying recombination and/or mutation to these so called parents, a set of new candidates, the offspring, is formed. The offspring then compete, based on their fitness and possibly age, with the parents for a place in the next generation. This process can be iterated until a candidate with sufficient fitness is found or until a previously defined computational limit is reached. Different variants of evolutionary algorithms follow the same basic cycle. They differ only in details related to a number of components, procedures and operators that must be specified in order to define a particular EA:

1. **Representation**
   The candidate solutions are defined by a set of design variable settings and possibly additional information. These, so called genes, need to be represented in some way for the EA. This could, e.g be done by a string of binary code, a string of integers, or a string of real numbers.

2. **Fitness Function**
   The fitness function assigns a quality measure to the candidate solutions. This is normally the objective function or a simple transformation of it. If penalty functions are used to handle constraints the fitness is reduced for unfeasible solutions.

3. **Population**
   A set of individuals or candidate designs forms a population. The number of individuals within the population, i.e. the population size, needs to be defined.

4. **Parent Selection Mechanism**
   The role of parent selection is to distinguish among individuals based on their quality and to allow the better ones to become parents of the next generation. This selection is typically probabilistic so that high-quality individuals get higher chance of becoming parents than those with low quality. Nevertheless, low-quality individuals often still have a small chance of getting selected to avoid the algorithm from being trapped in a local optimum.

5. **Variation Operators**
   The role of variation operators are to create new individuals (offspring) from old ones (parents), i.e. generate new candidate designs. Recombination, also called crossover, is applied to two or more selected candidates and results in one or more new candidates. Mutation is applied to one candidate and results in one new candidate. Both operators are stochastic and the outcome depends on a series of random choices. Several different versions exist for the various representations.

6. **Survivor Selection Mechanism**
   The role of survivor selection, also called replacement, is to select the individuals that will be allowed in the next generation based on their quality. Survivor selection is often deterministic, for instance ranking the individuals and selecting the top segment from parents and offspring (fitness biased) or selecting only from the offspring (age biased).
In general, evolutionary algorithms are divided into genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Genetic algorithms are often implemented into commercial software and some also include evolution strategies. These algorithms are therefore presented in more detail and the differences between them are outlined.

John H. Holland at University of Michigan is considered to be the pioneer of genetic algorithms (GAs) (Holland, 1992), which are the most widely known type of evolutionary algorithms. There are several genetic algorithms that differ in representation, variation, and selection operators. What can be considered a classical GA has a binary representation, fitness proportionate parent selection, a low probability of mutation, emphasis on genetically inspired recombination to generate new candidate solutions, and parents replaced by offspring in the next generation. This algorithm is commonly referred to as simple GA or canonical GA. Mutation is typically done by bit flip and recombination in the form of 1-point crossover, see Figure 4.28. It has been argued that real-coded GAs often give better results than binary-coded GAs. However, it is very problem dependant and can actually be more related to the selected crossover and mutation operators.

**Figure 4.27** The basic evolution cycle followed by evolutionary algorithms.

**a) 1-point crossover**

```
Parents
\[ \begin{align*}
  x &= (x_1, x_2, x_3)^T \\
  \text{Parents} &= 01101 11000 01000 \\
  \text{Parents} &= 10110 01010 01100 \\
\end{align*} \]

Offspring
\[ \begin{align*}
  x &= (x_1, x_2, x_3)^T \\
  \text{Offspring} &= 01101 11010 01100 \\
  \text{Offspring} &= 10110 01000 01000 \\
\end{align*} \]
```

**b) Bit-flip mutation**

```
Parent
\[ \begin{align*}
  x &= (x_1, x_2, x_3)^T \\
  \text{Parent} &= 10110 01010 01100 \\
\end{align*} \]

Offspring
\[ \begin{align*}
  x &= (x_1, x_2, x_3)^T \\
  \text{Offspring} &= 10110 00010 01100 \\
\end{align*} \]
```

**Figure 4.28** Typical variation operators used in simple GA for three variable designs in a binary string representation. **a)** Recombination with 1-point crossover where the crossover point is randomly selected. **b)** Mutation with bit-flip mutation where each bit is flipped (from 1 to 0 or 0 to 1) with a low probability. The number of flips therefore varies between individuals.
Evolution strategies (ES) also belong to the EA family and were developed by Ingo Rechenberg and Hans-Paul Schwefel at Technical University of Berlin, see Beyer and Schwefel (2002). In the original ES algorithm, one parent individual is subjected to mutation to form one offspring and the best of these two individuals is chosen to form the next generation. Development of the method has now lead to more complex algorithms. General ES have a real valued representation, random parent selection, and mutation as the primary operator for generating new candidate solutions. After creating $\lambda$ offspring and calculating their fitness, the best $\mu$ are chosen deterministically, either from the offspring only, called $(\mu, \lambda)$ selection, or from the union of parents and offspring, called $(\mu + \lambda)$ selection. Often $(\mu, \lambda)$ selection is preferred, especially if local optima exist. The value $\lambda$ is typically much higher than the value $\mu$, a ratio of 1 to 7 is recommended by Eiben and Smith (2003). Mutation is commonly done by Gaussian perturbation and recombination is either discrete or intermediary, see Figure 4.29. Most ES are self-adaptive which means that some parameters are included in the representation of the individuals and co-evolve with the solutions so that the algorithm performs better. A comparison between GAs and ES is presented in Table 4.4.

a) Discrete and intermediary recombination

\[
\mathbf{x}^{p1} = (x_1^{p1}, x_2^{p1}, x_3^{p1}, \sigma_1^{p1}, \sigma_2^{p1}, \sigma_3^{p1})^T
\]

Parents

\[
\begin{array}{cccccc}
3.0 & 2.5 & 1.7 & 0.1 & 0.3 & 0.5 \\
3.2 & 2.2 & 1.5 & 0.3 & 0.1 & 0.3 \\
\end{array}
\]

variables strategy parameters

\[
\mathbf{x}^{p2} = (x_1^{p2}, x_2^{p2}, x_3^{p2}, \sigma_1^{p2}, \sigma_2^{p2}, \sigma_3^{p2})^T
\]

Discrete recombination

$x_1^{p1}$ or $x_1^{p2}$ chosen randomly

Offspring

\[
\begin{array}{cccccc}
3.0 & 2.5 & 1.5 & 0.2 & 0.2 & 0.4 \\
\end{array}
\]

Intermediary recombination

$(\sigma_1^{p1} + \sigma_1^{p2})/2$

\[
\sigma_i^p = \sigma_i^p \cdot N_i(0,1)
\]

Parent

\[
\begin{array}{cccccc}
3.2 & 2.2 & 1.5 & 0.3 & 0.1 & 0.3 \\
\end{array}
\]

\[
\mathbf{x}^p = (x_1^p, x_2^p, x_3^p, \sigma_1^p, \sigma_2^p, \sigma_3^p)^T
\]

Offspring

\[
\begin{array}{cccccc}
3.3 & 2.1 & 1.5 & 0.2 & 0.1 & 0.4 \\
\end{array}
\]

\[
\sigma_i^p = \sigma_i^p \cdot e^{\tau \cdot N(0,1)} + \tau \cdot N(0,1)
\]

\[\tau, \tau'\] are parameters set by the user

b) Mutation by Gaussian perturbation

Figure 4.29 Typical variation operators used in ES for three variable designs with different mutation step sizes $\sigma_i$ for each variable. a) Recombination is commonly discrete for the variable part and intermediary for the strategy parameter part. Typically, global recombination is used where the parents are drawn randomly from the population for each position $i$. This means that more than two individuals are commonly contributing to the offspring. b) Mutation by Gaussian perturbation means that each variable is changed a small amount randomly drawn from a normal distribution. $N(0,1)$ denotes a draw from a normal distribution with mean 0 and standard deviation 1 and $N_i(0,1)$ denotes a separate draw from the normal distribution for each variable $i$. 
Table 4.4 Overview of typical features of genetic algorithms and evolution strategies according to Eiben and Smith (2003).

<table>
<thead>
<tr>
<th></th>
<th>Genetic algorithms</th>
<th>Evolution strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical representation</td>
<td>Strings of a finite alphabet</td>
<td>Strings of real numbers</td>
</tr>
<tr>
<td>Role of recombination</td>
<td>Primary variation operator</td>
<td>Secondary variation operator</td>
</tr>
<tr>
<td>Role of mutation</td>
<td>Secondary variation operator</td>
<td>Primary and sometimes the only variation operator</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Random, biased by fitness</td>
<td>Random, uniform</td>
</tr>
<tr>
<td>Survivor selection</td>
<td>All individuals replaced or deterministic, biased by fitness</td>
<td>Deterministic, biased by fitness</td>
</tr>
</tbody>
</table>

In general, GAs are considered more likely to find the global optimum while ES are considered faster. A general recommendation is therefore to use a GA if it is important to find the global optimum, while ES should be used if speed is important and a "good enough" solution is acceptable. However, the results depend on the algorithm settings and the problem at hand, and there is no generally accepted conclusion on the superiority of any of the algorithms. Instead, the merits of both algorithms could be taken advantage of if they are used together, as described by Hwang and Jang (2008).

Constraints are often enforced by using penalty functions that reduce the fitness of unfeasible solutions. Preferably, the fitness is reduced in proportion to the number of constraints that are violated. A good idea is often also to reduce the fitness in proportion to the distance from the feasible region. The penalty functions are sometimes set so large that unfeasible solutions will not survive. Occasionally the penalty functions are allowed to change over time and even adapt to the progress of the algorithm. There are also other techniques to handle constraints. One of them is to use a repair function that modifies an unfeasible solution into a feasible one.

### 4.6.5 Particle Swarm Optimization

Swarm algorithms are based on the idea of swarm intelligence, i.e., the collective intelligence that emerges from a group of individuals, and are inspired by the behaviour of organisms that live and interact in nature within large groups. One of the most well-known algorithms is *particle swarm optimization (PSO)* which imitates, for example, a flock of birds. Hence, swarm algorithms are population-based algorithms like the evolutionary algorithms.

Particle swarm optimization was introduced by James Kennedy and Russell Eberhart after studying the social behaviour of birds, as described by Kennedy and Eberhart (1995). To search for food, each member of a flock of birds determines its velocity based on their personal experience as well as information gained through interactions with other members of the flock. The same ideas apply to PSO, in which the population, called swarm, converges to the optimum using information gained from each individual, referred to as particle, and from the information gained by the swarm as a whole. A basic PSO algorithm has a very simple formulation that is easy to implement and modify. The algorithm starts by initializing a swarm of particles with randomly chosen velocity and position within the design space. The position of each particle is then updated from one iteration to the next using the simple formula

\[
x_i^{q+1} = x_i^q + v_i^q \Delta t
\]

(4.98)
where $i$ refers to the $i$th particle in the swarm, $q$ to the $q$th iteration and $v_{i,q}$ to the velocity. The time increment $\Delta t$ is typically set to be one and the velocity vector is updated in each iteration using

$$v_{i,q+1} = w v_{i,q} + c_1 r_1 \frac{(p_i - x_{i,q})}{\Delta t} + c_2 r_2 \frac{(p_g - x_{i,q})}{\Delta t}$$

(4.99)

where $w$ is the inertia parameter, $r_1$ and $r_2$ are random numbers between 0 and 1, $c_1$ and $c_2$ are the trust parameters, $p_i$ is the best point found so far by the $i$th particle, and $p_g$ is the best point found by the swarm. The user thus needs to select and/or tune the values of $w$, $c_1$ and $c_2$, and decide on the number of particles in the swarm, as well as how many iterations that should be performed. The inertia parameter $w$ controls the search behaviour of the algorithm. Larger values (around 1.4) result in a more global search while smaller values (around 0.5) result in a more local search as stated by Venter (2010). The $c_1$ trust parameter indicates how much the particle trusts itself while $c_2$ specifies how much the particle trusts the group. Recommended values according to Venter (2010) are $c_1 = c_2 = 2$. Finally, $p_g$ can be selected to represent either the best point in a small subset of particles or the best point in the whole swarm.

The original PSO algorithm has been developed and enhanced, and different versions have been applied to different types of optimization problems. Constraints can be handled by some kind of penalty method, as described in Section 4.6.4. Another simple approach is to use strategies that preserve feasibility. Hu et al. (2003) describes a method where each particle is initialized repeatedly until it satisfies all constraints and where the particles then search the whole space but only keep the feasible solutions in their memory.

4.6.6 Simulated Annealing

Simulated annealing (SA) is a global stochastic optimization algorithm that mimics the metallurgical annealing process, i.e. heating and controlled cooling of a metal to increase the size of its crystals and reduce their defects. The algorithm was developed by Scott Kirkpatrick and co-workers, and exploits the analogy with a metal that cools and freezes into a minimum energy crystalline structure, see Kirkpatrick et al. (1983). In SA, the objective function of the optimization problem is seen as the internal energy of the metal during annealing. The idea is to start at a high temperature that is slowly reduced so that the system goes through different energy states in the search of the lowest state representing the global minimum of the optimization problem. When annealing metals, the initial temperature must not be too low and the cooling must be done sufficiently slowly to avoid the system from getting stuck in a meta-stable non-crystalline state representing a local minimum of energy. The same principles apply to simulated annealing in the process of finding the solution to an optimization problem.

The strength of the SA algorithm is its ability to deal with highly non-linear, chaotic, and noisy objective functions, as well as a large number of constraints. On the other hand, a major drawback stated by Younis and Dong (2010) is the lack of clear trade-off between the quality of a solution and the time required to locate the solution, which leads to long computation times. Different modifications to the original algorithm have been proposed to improve the speed of convergence. One of these is the very fast simulated re-annealing (VFSR) algorithm presented by Lester Ingber. This algorithm is also known as adaptive simulated annealing (ASA), see Ingber (1996).
Simulated annealing algorithms can, in general, be described by the following steps according to Stander et al. (2010):

1. **Initialisation**
   The search starts at iteration $q = 0$ by identifying a starting design, also called starting state, $x^{(0)}$ from the set of all possible designs $X$ and calculating the corresponding energy $E^{(0)} = E(x)$. The set of checked points $X^{(0)} = \{x^{(0)}\}$ now includes only the starting design. The temperature is initialized at a high value $T^{(0)} = T_{max}$ and a cooling schedule $C$, an acceptance function $A$, and a stopping criterion are defined.

2. **Sampling**
   A new sampling point $x' \in X$ is selected using a sampling distribution $D(X^{(q)})$, and the corresponding energy $E' = E(x')$ is calculated. The set of checked points $X^{(q+1)} = X^{(q)} \cup \{x'\}$ now contains $q + 2$ designs.

3. **Acceptance check**
   A random number $\zeta$ is sampled from the uniform distribution $[0, 1]$ and
   
   $$x^{(q+1)} = \begin{cases} x' & \text{if } \zeta \leq A(E', E^{(q)}, T^{(q)}) \\ x^q & \text{otherwise} \end{cases}$$

   where $A$ is the acceptance function that determines if the new point is accepted. The most commonly used acceptance function is the Metropolis criterion

   $$A(E', E^{(q)}, T^{(q)}) = \min \{ 1, e^{-\frac{(E' - E^{(q)})}{T^{(q)}}} \}$$

4. **Temperature update**
   The cooling schedule $T^{(q+1)} = C(X^{(q+1)}, T^{(q)})$ is applied to the temperature. It has been proven that a global minimum will be obtained if the cooling is made sufficiently slowly.

5. **Convergence check**
   The search is ended if the stopping criterion is met, otherwise $q = q + 1$ and the search continues at step 2. Typically, the search is stopped when there is no noticeable improvement over a number of iterations and/or when the number of iterations has reached a predefined value and/or when the temperature has fallen to a desired level.

It is obvious that the efficiency of the algorithm depends on the appropriate choices of the mechanisms to generate new candidate states $D$, the cooling schedule $C$, the acceptance criterion $A$, and the stopping criterion. The choices of $D$ and $C$ are typically the most important issues in defining an SA algorithm and they are strongly interrelated. The next candidate design $x'$ is usually selected randomly in the neighbourhood of the current design $x$ with the same probability for all neighbours. The size of the neighbourhood is typically selected based on the idea that the algorithm should have more freedom when the current energy is far from the global optimum. Larger step sizes are therefore allowed initially. However, a more complicated, non-uniform selection procedure is used in adaptive simulated annealing to allow much faster cooling rates, see Stander et al. (2010). The basic idea of the cooling schedule is to start at a high temperature and then gradually drop the temperature to zero. The primary goal is to quickly reach a temperature where low energies are preferred but where it is still possible to explore different areas of the design space. Thereafter, the SA algorithm lowers the temperature slowly until the system freezes and no further changes occur.
Simulated annealing algorithms generally handle constraints by penalty methods similar to the ones described in Section 4.6.4, i.e. the energy for unfeasible solutions is increased so that the probability of selecting such designs is reduced.

*Hill-climbing* is a very simple optimization technique used to find a local maximum. It resembles a gradient-based algorithm, but does not require any gradient information. New candidate designs are iteratively tested in the region of the current design and adopted if they are better. This enables the algorithm to climb uphill until a local maximum is found. A similar technique could, of course, be used to find a local minimum. Simulated annealing differs from these simple algorithms in that new candidate solutions can be chosen at a certain probability even if they are worse than the previous one, i.e. have higher energy. A new worse solution is more likely to be chosen early in the search when the temperature is high and if the difference in energy is small. Simulated annealing therefore goes from being similar to a random search initially, with the aim of finding the region of the global optimum, to being very similar to "Hill-climbing" in order to locate the minimum more exactly. Simulated annealing can also be seen as a GA with a population of one individual and a changing mutation rate, as noted by Andersson (2000).

### 4.6.7 Multi-Objective Optimization

Real-world applications of optimization often include more than one objective. Many of the previously mentioned algorithms have therefore been extended to handle multi-objective optimization (MOO) problems. Some of these algorithms will be presented briefly in the following.

A typical MOO problem with $m$ objective functions is defined by Equation (3.3) and is repeated below.

$$\min_x \quad f_1(x), \ldots, f_m(x)$$

subject to $g(x) \leq 0$  \hspace{1cm} (4.102)

Typically, an MOO problem does not have a single optimal solution. Instead, there is a set of solutions that reflect the trade-off among objectives as, described in Section 3.3. For a single-objective optimization problem, it is easy to compare solutions and identify the best one. However, for MOO problems, special considerations are required to compare different designs and a non-dominating concept is therefore often used. A solution is said to be *non-dominated*, or Pareto optimal, if there exist no other solution that could improve any of the objectives without worsening at least one of the other objectives. The set of all Pareto optimal solutions is called the *Pareto optimal set* and the representation of this set in the objective space is called the *Pareto optimal front*. The Pareto optimal front is consequently a curve, a surface, or a hyper-surface for the respective cases of two, three, or more conflicting objectives. The size of the population needed to accurately capture the Pareto optimal front grows exponentially with the number of objectives.

As indicated in Section 3.3 and described by Marler (2004), Andersson (2000), and Hwang et al. (1980), MOO methods can be divided into categories depending on when the decision maker articulates his or her preference regarding different solutions. The alternatives are never, before, during, or after the optimization process. Many different approaches exist for each of these categories.
In many cases, the decision maker cannot define explicitly what he or she prefers, and then a method that does not need an articulation of preference can be used. However, these methods output only one point from the Pareto optimal set which has to be accepted by the decision maker. One example of such a formulation is the \textit{global criterion method} in which all objective functions are combined to form a single objective function, i.e.

$$\min_x F_p = \left[ \sum_{i=1}^{m} \left( \frac{f_i(x^*) - f_i(x)}{f_i(x^*)} \right)^p \right]^\frac{1}{p}$$

subject to $g(x) \leq 0$

where $f_i(x^*)$ is the minimum of the $i^{th}$ objective function and $p$ is a parameter between one and infinity. A special case is the \textit{min-max method} in which $p = \infty$ and thus the largest of the terms in the global criterion is minimized. The solution $F^* = (f_1(x^*), \ldots, f_m(x^*))^T$ is called the \textit{utopian solution} and is rarely feasible, see Figure 4.30. Depending on the value of the parameter $p$, different optima will be obtained. In fact, the selection of $p$ can be seen as a way of articulating preference since the size of $p$ reflects the emphasis that is placed on the largest components of the summation. Selecting $p = 1$ means that all terms are equally important, while a larger $p$ implies that more weight is given to the larger terms.

The most common way of conducting MOO is probably by using methods with a priori articulation of preference. This can simply be done by assigning weights to the different objectives in the global criterion method resulting in a \textit{weighted global criterion method}. However, the easiest and perhaps most widely used method is the \textit{weighted sum method} formulated as

$$\min_x \sum_{i=1}^{m} w_i f_i(x)$$

subject to $g(x) \leq 0$

where the individual weights $w_i$ are positive numbers whose sum often is set to unity, i.e. $\Sigma w_i = 1$. Many other methods exist and some of them are described in e.g. Marler (2004), Andersson (2000), and Hwang et al. (1980).

Methods with progressive articulation of preference are interactive and the decision maker gives input to the optimization algorithm simultaneously during the search. The idea behind these methods is that the decision maker is unable to a priori give preference information due to the complexity of the problem, but will be able to give some information on preference as the search moves on and the decision maker learns more about the problem. These methods will not be covered here, but more information can be found in Hwang et al. (1980) or Andersson (2000).

In many cases, it is hard for the decision maker to articulate his or her preferences before the optimization process starts. In these situations, it can be effective to allow the decision maker to choose from a set of solutions. In the methods with a posteriori articulation of preference, the idea is to search the solution space for a set of Pareto optimal points and present them to the decision maker. A major advantage of these methods is that different alternatives can be explored without having to rerun the optimization. On the other hand, generating the Pareto optimal set can be computationally expensive. Another disadvantage is that there might be so many solutions to choose
from that it is very hard for the decision maker to select the most satisfactory one. There are several ways to obtain a sample set of points on the Pareto optimal front. One approach is to perform multiple optimization runs, each time obtaining a new point. Another approach is to use an evolutionary algorithm, such as a genetic algorithm, that finds multiple points in one single optimization run.

Different MOO algorithms are compared using criteria based on two properties; convergence and diversity. An effective algorithm not only needs to identify the Pareto optimal front, i.e. have a good convergence, but it also has to be able to represent different regions of the front, i.e. maintain diversity.

The simplest and most straight-forward of the multiple run approaches is to use Equation (4.104) and vary the weights in order to obtain different points. However, it might be hard to choose the weights so that the points get evenly spread on the Pareto front. Another drawback is that not all Pareto optimal solutions can be found if the Pareto optimal front is non-convex, as described by Hwang et al. (1980). To be able to capture points on the non-convex part of the Pareto optimal front, a weighted $L_p$-norm problem can be solved instead, as described by Andersson (2000).

$$\min_x \left( \sum_{i=1}^{m} \left[ w_i f_i(x) \right]^p \right)^{1/p}$$  \hspace{1cm} (4.105)

subject to $g(x) \leq 0$

where $p$ is an integer satisfying $1 \leq p \leq \infty$. This is a generalization of the weighted sum formulation in Equation (4.104). With an appropriate selection of $p$, all the Pareto optimal points can be found. However, the proper choice of $p$ is a priori unknown.

In order to avoid some of the difficulties for problems with a non-convex Pareto optimal front, the $\epsilon$-constraint method can be used. The idea is to keep only one of the objectives and reformulate the other objectives as constraints.

$$\min_x f_i(x)$$

subject to $f_j \leq \epsilon_j, \quad j = 1, \ldots, m \text{ and } j \neq i$ \hspace{1cm} (4.106)

Different points on the Pareto optimal front will be found by progressively changing the constraint values $\epsilon_j$. By first calculating the extremes of the Pareto optimal front, the ranges of the different objective functions can be identified and the constraint values $\epsilon_j$ selected appropriately. The method enables an even spread of points as long as the Pareto optimal front is continuous. A graphical comparison between the weighted sum method and the $\epsilon$-constraint method for a problem with two objectives and a non-convex Pareto optimal front is found in Figure 4.30.

As mentioned previously, population-based algorithms are very attractive for MOO problems as many Pareto optimal solutions can be found in one single optimization run. **Multi-objective evolutionary algorithms (MOEAs)** are therefore often used for solving MOO problems. There exist two approaches to acquire points on the Pareto optimal front, as described by Marler (2004). Either the algorithm searches for and stores the Pareto optimal points in a separate set as they appear, or it forces the general population to evolve into an approximation of the Pareto optimal set.
Figure 4.30 Comparison of two methods to find Pareto optimal points for a case with two objectives forming a non-convex Pareto optimal front. **a)** In the weighted sum method, the Pareto optimal points are identified at the locations where a straight line (whose normal is defined by the weights of the objectives) only tangents the set of feasible solutions and never intersects it. This means that the dashed part of the Pareto optimal front will never be found and that the rest of the Pareto optimal front might be unevenly sampled. **b)** In the $\epsilon$-constraint method, the Pareto optimal points are identified by selecting one objective (in this case $f_2$) while the other objectives are transformed into constraints (in this case $f_1 \leq \epsilon$). After identifying $f_1^*$ and $f_2^*$, it is easy to select the different $\epsilon$-values to get a reasonably evenly sampled Pareto optimal front.

Most MOEAs apply Pareto-based **ranking** schemes that were introduced by Goldberg (1989). The different solutions are assigned rank in an iterative procedure, where a rank of one is considered the best rank. The process starts with running a non-domination check on all individuals in the population and assigning rank one to the non-dominated ones. The non-dominated individuals are then removed and the non-domination check is re-run and rank two is assigned to the new non-dominated solutions. This procedure of removing non-dominated individuals, re-running the non-domination check, and assigning increased rank to the newly found non-dominated solutions are continued until all individuals have been assigned rank, see Figure 4.31.

Figure 4.31 Illustration of the concept of rank for the case of two objectives that should be minimized. The solutions in the Pareto optimal set are assigned rank one and the Pareto optimal front is indicated by the line. All solutions with higher rank are dominated by the solutions with lower rank, i.e. the solutions in the Pareto optimal set are non-dominated.
Another commonly used approach in MOEAs is *elitism*, which is the process of artificially keeping high fitness individuals to preserve favourable genetic information. The idea is to improve convergence but it may also yield reduced diversity in the population. Nevertheless, it has been shown by Zitzler et al. (2000) that elitism is an important factor for a successful MOEA.

Often, MOEAs tend to create clusters around a limited set of Pareto optimal points, i.e. converge to niches. This phenomenon is called *genetic drift*. *Niche techniques* are used to force the development of multiple niches and limit the growth of any single niche. *Fitness sharing* is a common niche technique, as described by Marler (2004). The basic idea is to penalize the fitness of points in crowded areas and hence reduce their probability of surviving to the next generation. One problem with the fitness sharing approach is that it relies on a user defined parameter defining the sharing distance, and choosing the value of this parameter might not be obvious.

One popular MOEA is the *non-dominated sorting genetic algorithm (NSGA-II)* developed by Deb et al. (2002). As noted by Zhou et al. (2011), many of today’s MOEAs share the basics with NSGA-II, but other approaches exist. The NSGA-II uses a fast non-dominated sorting procedure and an elitist-preserving approach, as well as a parameter-free crowding distance niche technique to preserve diversity. A schematic picture of the algorithm can be seen in Figure 4.32. The basic steps of the algorithm are as follows:

1. Randomly initialize a parent population of size $N$. Evaluate the population, i.e. calculate objective and constraint values. Rank the population using non-domination criteria. Compute the crowding distance, i.e. a measure of relative closeness to other solutions in the objective space, which is used to differentiate between solutions of the same rank, see Figure 4.33.

2. Employ genetic operators, i.e. selection, crossover, and mutation, to form a child population of size $N$. Evaluate the child population.

3. Combine the parent and child populations. Assign rank and calculate the crowding distance for each individual.

4. Apply elitism by selecting the $N$ best individuals from the combined population based on rank and crowding distance. These individuals will form the parent population in the next iteration.

5. If the termination criterion is not met, go to step 2.

Since the population in MOEAs is finite, some Pareto optimal solutions might be lost during the search and replaced by other solutions. Pareto sub-optimal points can therefore be part of the final solution. This problem is called *Pareto drift*. A remedy for this problem is to keep an external archive of unlimited size for the Pareto optimal solutions. This has successfully been implemented together with the NSGA-II algorithm by Goel et al. (2007b).

In MOEAs, constraints are often handled by penalty methods, i.e. the approach is the same as previously described for single-objective optimization algorithms. An alternative method is used in *Constrained NSGA-II* and described by Deb et al. (2002). This constraint handling technique is based on the selection operator called binary tournament selection where two individuals are picked from the population and the better one is chosen. There exist three different selection scenarios in which (a) none, (b) one, or (c) both of the solutions are feasible, i.e. fulfil all the constraints. If none of the solutions are feasible, the one with the smaller overall constraint violation should be chosen. In the
case of only one feasible solution, that solution should be chosen. If both solutions are feasible, the selection should be based on rank and crowding distance as described previously. This can be implemented in the NSGA-II algorithm by simply modifying the domination criteria. A solution \( i \) is said to constrained-dominate a solution \( j \), if any of the following conditions is true.

1. Solutions \( i \) and \( j \) are both unfeasible, but solution \( i \) has a smaller overall constraint violation.
2. Solution \( i \) is feasible and solution \( j \) is not.
3. Solutions \( i \) and \( j \) are both feasible, but solution \( i \) dominates solution \( j \).

The rest of the NSGA-II procedure as described earlier remains the same.

**Figure 4.32** A schematic picture of the NSGA-II procedure. The parent and child populations are combined, and the best solutions are selected based on rank and crowding distance to form the next generation. In this example, all solutions of rank 1 and 2 are selected, but only the ones of rank 3 that are in the least crowded region of the objective space.

**Figure 4.33** Illustration of the crowding distance and how a solution is selected in NSGA-II. Each solution is assigned a rank \( r \) and a crowding distance \( d \). The crowding distance is calculated based on the distances to the nearest solutions of the same rank for all \( m \) objectives. The rank and crowding distance are the basis for the elitist selection where a low rank is always preferred and a large crowding distance is favoured when the rank is equal.
5 Multidisciplinary Design Optimization Methods

The aim of this chapter is to describe and compare a selected number of MDO methods documented in the literature. The methods are divided into two main categories: single-level and multi-level optimization methods. Using single-level methods, the optimization process is performed by one single optimizer, while the optimization process is distributed using multi-level methods. Single-level methods can either integrate the optimization process with the analyses or let the optimization process communicate with distributed analyses, referred to as the first and second generations of MDO methods in Chapter 2. In both cases, all design decisions are made by the optimizer. Multi-level methods, on the other hand, distribute the optimization process as well as the analyses resulting in distributed design decisions. This group of methods developed as the third generation of MDO methods also referred to in Chapter 2.

When choosing a method for solving a specific problem, the nature of the problem and the environment in which the problem is to be solved must be taken into account. Large-scale problems that involve several departments of a company have to be decomposed in one way or another, excluding the first generation of MDO methods. Before studying specific single-level and multi-level methods, the implications of problem decomposition will be given special attention.

5.1 Problem Decomposition

When solving large-scale MDO problems, some kind of problem decomposition is required. There are two main motivations for decomposing a problem according to Kodiyalam and Sobieszczanski-Sobieski (2001), namely concurrency and autonomy. Concurrency is achieved through distribution of the problem so that human and computational resources can work on the problem in parallel. Autonomy can be attained if individual groups responsible for certain parts of the problem are granted freedom to make their own design decisions and to govern methods and tools.

For single-level optimization methods, decomposition is achieved through distributing the analyses. The problem can then be solved efficiently, but autonomy will be restricted as all design decisions are made by the optimizer. For multi-level optimization methods, the whole optimization process is decomposed, making it possible to solve the problem efficiently and give the individual groups freedom to make their own design decisions.

5.1.1 Terminology of Decomposed Systems

A unified terminology for decomposed systems, needed when discussing different MDO methods, is presented in this section. The analysis of the original problem can be summarized in Figure 5.1. The vector of design variables, indicated by $x$, is sent to an analyzer. The analyzer solves the governing equations and computes the values of the objective function, $f$, and the constraint functions, $g$, that are used to drive the optimization routine.

The original problem can be decomposed into a number of subproblems. Each subproblem has a number of variables, indicated by the vector $x_j$ for subproblem $j$. The union of the variables in all subproblems is the original set of design variables $x$. The variables in the different subproblems are in general not disjoint. Variables that are unique to a specific subproblem are called local variables, denoted by the vector $x_{lj}$ for subproblem $j$. The collection of local variables in all subproblems is termed $x_l$. There will also be a number of shared variables that are present in at least two subproblems. $x_{sj}$ indicates the vector of shared variables in subproblem $j$, where each component is
present in at least one other subproblem. The union of shared variables in all subproblems is denoted by $x_s$. An illustration can be found in Figure 5.2.

Figure 5.1 Analysis of the original system. A number of variables are given as input and the objective and constraint functions for that specific set of variables are received as output.

Figure 5.2 a) Illustration of variables in three subproblems. The variables $x_1$, $x_2$, and $x_3$ are not disjoint. b) Illustration of local and shared variables. The intersection of $x_{s1}$ and $x_{s2}$ are shared variables present in both subproblem 1 and subproblem 2, while the intersection of $x_{s1}$, $x_{s2}$, and $x_{s3}$ are shared variables present in all three subproblems.

When a problem is decomposed, it is necessary to handle the connections between the resulting subproblems. We define **coupling variables** as output from one subproblem needed as input to another subproblem. The vector $y_{ij}$ consists of a number of variables output from subproblem $j$ and input into subproblem $i$. $y_{ij}$ denotes all coupling variables output from subproblem $j$ and $y_{j\ast}$ all coupling variables input to subproblem $j$. The collection of all coupling variables is indicated by the vector $y$.

Analysis of the decomposed system involves fulfilling the governing equations of each subproblem and finding consistent values of the coupling variables, illustrated in Figure 5.3. Each subproblem $j$ contributes to the original objective and constraint functions through $f_j$ and $g_j$. Consistency of coupling variables means that the input $y_{ij}$ used for subproblem $i$ is the same as the output $y_{ij}$ obtained from subproblem $j$. This is referred to as multidisciplinary feasibility by Cramer et al. (1994), but since feasibility in an optimization context refers to a solution that fulfils the constraints, the term **multidisciplinary consistency** is used throughout this text. **Individual discipline consistency**,
also renamed from the definition by Cramer et al. (1994), refers to the situation when the governing
equations of each subproblem are fulfilled, but the coupling variables are not necessarily consistent.
This term will be used when defining the individual discipline feasible formulation in Section 5.2.2.

![Diagram of the decomposed system](image)

**Figure 5.3** Analysis of the decomposed system. The local, shared, and coupling variables are given as input to each subproblem, resulting in output that can be used by the optimization routine.

A consistent nomenclature is used throughout Chapter 5. The symbols are defined when they first appear, but are also summarized in Table 5.1 for convenience.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Variables</td>
</tr>
<tr>
<td>$x_l$</td>
<td>Local variables (in all subspaces)</td>
</tr>
<tr>
<td>$x_s$</td>
<td>Shared variables (in all subspaces)</td>
</tr>
<tr>
<td>$x_j$</td>
<td>Variables in subspace $j$</td>
</tr>
<tr>
<td>$x_{lj}$</td>
<td>Local variables in subspace $j$</td>
</tr>
<tr>
<td>$x_{sj}$</td>
<td>Shared variables in subspace $j$</td>
</tr>
<tr>
<td>$y$</td>
<td>Coupling variables</td>
</tr>
<tr>
<td>$y_{ij}$</td>
<td>Coupling variables output from subspace $j$ and input into subspace $i$</td>
</tr>
<tr>
<td>$y_{j*}$</td>
<td>Coupling variables output from subspace $j$</td>
</tr>
<tr>
<td>$y^*$</td>
<td>Coupling variables input into subspaces for non-consistent designs All subscripts of $y$ are possible</td>
</tr>
<tr>
<td>$f$</td>
<td>Objective function for a single-level optimization method or system objective function for a multi-level optimization method</td>
</tr>
<tr>
<td>$g$</td>
<td>Constraint functions for a single-level optimization method</td>
</tr>
<tr>
<td>$f_j$</td>
<td>Part of objective function from subspace analyzer $j$ for a single-level optimization method or objective function in subspace $j$ for a multi-level optimization method</td>
</tr>
<tr>
<td>$g_j$</td>
<td>Part of constraint functions from subspace analyzer $j$ for a single-level optimization method or constraint functions in subspace $j$ for a multi-level optimization method</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of subspaces</td>
</tr>
<tr>
<td>$x_{j,m}$</td>
<td>Component $m$ of $x_j$, where $x_j$ can be replaced by any other vector to indicate a certain component of that vector</td>
</tr>
</tbody>
</table>
5.1.2 Aspect- and Object-Based Decomposition

A system can be decomposed in different ways, as described by Sobieszczanski-Sobieski and Haftka (1987). Aspect-based decomposition refers to dividing the system into different disciplines. The system will then naturally consist of two levels: one top level and one for all the disciplines. An example from the automotive industry can be found in Figure 5.4.

![Figure 5.4 Example of aspect-based decomposition in the automotive industry.](image)

**Object-based decomposition** simply means dividing the entire system into its constituent subsystems, which in turn can be divided into smaller subsystems or components. A system decomposed by object can have an arbitrary number of levels. In Figure 5.5, an example of object-based decomposition in the automotive industry can be seen.

![Figure 5.5 Example of object-based decomposition in the automotive industry.](image)

5.1.3 Hierarchic and Non-Hierarchic Systems

While the previous section focused on how a system is decomposed, this section deals with how decomposed systems communicate. There is a clear distinction in communication pathways between **hierarchic** and **non-hierarchic systems** according to Sobieszczanski-Sobieski (1988). In hierarchic systems, communication only occurs vertically between parent and child, while there is no such communication restriction in non-hierarchic systems, see Figure 5.6. Non-hierarchic systems can be converted into hierarchic systems by introducing additional constraints.
5.1.4 Coupling Breadth and Coupling Strength
Decomposing MDO problems can be more or less efficient. The terms *coupling breadth* and *coupling strength* can be employed to classify MDO problems in order to gain an understanding of the effectiveness of decomposition, as described by Agte et al. (2010). The coupling breadth is defined by the number of coupling variables and the coupling strength is a measure of how much a change in a coupling variable, output from one subproblem, affects the subproblem that it is input to.

For visualization purposes, the coupling breadth can be plotted against the coupling strength, see Figure 5.7. Agte et al. (2010) discuss how to look upon problems in the four different corners of the graph from an MDO perspective. The discussion focuses specifically on the suitability of multi-level optimization methods. The existing methods are particularly suitable for problems in the upper left corner that have a strong but narrow coupling. Decomposition is least complicated in the lower left corner where the subproblems are weakly coupled. Problems in the lower right corner have many but weak couplings of which some may be neglected in order to obtain an effective decomposition. In the upper right corner on the other hand, subproblems are so widely and strongly coupled that it may be preferable to merge them.

![Figure 5.7 Coupling breadth versus coupling strength.](image-url)

5.2 Single-Level Optimization Methods
Common for single-level optimization methods is a central optimizer that makes all design decisions. The two methods presented here are distinguished by the kind of consistency that is maintained during the optimization.
5.2.1 Multidisciplinary Feasible

The most common and basic single-level optimization method is the multidisciplinary feasible (MDF) formulation, described by Cramer et al. (1994). The method is also called All-in-One by Kodiyalam and Sobieszczanski-Sobieski (2001), and Single-NAND-NAND by Balling and Sobieszczanski-Sobieski (1994). The latter name consists of three parts. The first part expresses that the method has a single optimization level, while the second and third parts define how the method functions at the system and disciplinary levels, respectively. NAND is an abbreviation for nested analysis and design.

In the MDF formulation, the optimizer is responsible for finding the optimal design. The optimizer requests the values of the objective and constraint functions for different sets of design variables from the system analyzer. The system analyzer enforces multidisciplinary consistency, i.e. finds a consistent set of coupling variables, for every set of design variables. This is typically done iteratively using either fixed-point iteration or Newton’s method, as explained by Balling and Sobieszczanski-Sobieski (1994). Fixed-point iteration is most straightforward to implement and a simple example shows the fundamental idea. For a problem with two subspaces, initial values of the coupling variables input to the first subspace are given. The coupling variables output from the first subspace are computed and input to the second subspace. Thereafter, the second subspace computes the output coupling variables and sends them to the first subspace. This procedure continues until convergence. Newton’s method is more complicated to implement and involves derivatives. Both methods can experience convergence problems and which one that performs best depends on the problem at hand. Haftka et al. (1992) argue that fixed point iteration is preferred over Newton’s method except for problems with very large coupling strength and low coupling breadth. A schematic picture of the MDF method is presented in Figure 5.8. Multidisciplinary consistency is achieved when \( y_{ij} = y_{ij}^* \) for all \( i \neq j \), where the coupling variables sent to a subspace analyzer are indicated by a superscript plus sign. There are no restraints on communication pathways, and the MDF method is therefore non-hierarchic. This is not entirely clear in Figure 5.8, where there are only vertical communication pathways, but can be realized when considering the role of the system analyzer.

---

**Figure 5.8** Illustration of the multidisciplinary feasible formulation with three subspaces.
The MDF optimization formulation is the same as the original optimization formulation defined in Equation (3.2), but it is repeated below for convenience.

$$\min_{x} f$$
subject to  $$g \leq 0$$  \hspace{1cm} (5.1)

Allison et al. (2005a) describe a number of drawbacks with the MDF formulation associated with efficiency and robustness, of which some are mentioned here. Parallelism is limited when the system analyzer tries to achieve multidisciplinary consistency. The optimizer may fail to find the optimal design if the system analyzer has convergence problems. These shortcomings motivate the development of alternative methods.

### 5.2.2 Individual Discipline Feasible

The **individual discipline feasible (IDF)** formulation is an alternative single-level approach proposed by Cramer et al. (1994). Balling and Sobiesczanski-Sobieski (1994) call the method Single-SAND-NAND, where SAND stands for simultaneous analysis and design, NAND for nested analysis and design, and the naming convention was defined in Section 5.2.1.

In the IDF formulation, the subspace analyzers are decoupled to enable parallelism. This is achieved by letting the optimizer control the coupling variables and treat them as design variables. The optimizer sends the design variables together with estimations of the coupling variables to the subspace analyzers. The subspace analyzers enforce individual discipline consistency and send back updated coupling variables as well as contributions to the global objective and constraint functions to the optimizer. The iterative process needed to find multidisciplinary consistent designs at every call from the optimizer is consequently avoided. An additional constraint is introduced for every coupling variable to drive the optimization process towards multidisciplinary consistency at optimum. The IDF formulation is illustrated in Figure 5.9. As in the previous section, the coupling variables sent to the subspace analyzers are indicated by a superscript plus sign and $$y_{ij} \neq y_{ij}^+$$ before multidisciplinary consistency is reached.

![Figure 5.9 Illustration of the individual discipline feasible formulation with three subspaces.](image)

The optimization formulation can be summarized in the equation below, where $$x$$ is the union of all local and shared variables, $$y$$ is the collection of all coupling variables, and $$n$$ is the number of subspaces.
The IDF formulation avoids finding consistent multidisciplinary designs at every set of design variables, thereby enabling parallelism and avoiding the convergence problems associated with the MDF formulation. As an additional variable and constraint are introduced for every coupling variable, the method is most efficient for problems with low coupling breadth. Allison et al. (2005a) use an example problem that allows for variable coupling strength to show that IDF is more suitable than MDF for strongly coupled problems.

5.3 Multi-Level Optimization Methods

The single-level optimization methods presented in the previous sections have a central optimizer making all design decisions. Distribution of the decision making process is enabled using multi-level optimization methods, where a system optimizer communicates with a number of subspace optimizers. Several multi-level optimization methods have been presented in the literature and some of the most well known ones are investigated in the following sections.

5.3.1 Concurrent Subspace Optimization

**Concurrent subspace optimization (CSSO)** is a non-hierarchic method, originally developed by Sobieszczanski-Sobieski (1988) at NASA Langley Research Center. The original formulation is inspired by the idea to optimize one subspace with corresponding design variables at a time, holding the other variables constant. The method has diverged into different variants, which makes it impossible to present a unified approach.

An overview of the original CSSO method can be found in Figure 5.10. Each step will briefly be described below. First, a system analysis is carried out to find a multidisciplinary consistent design for the design variables $x^k$ obtained from the previous iteration. A system sensitivity analysis is then performed in order to find the derivatives of the coupling variables with respect to the design variables, $dy^k/dx^k$. These derivatives are obtained by the solution of the so called global sensitivity equations (GSE) described in detail by Sobieszczanski-Sobieski (1990). Next, the subproblems are decoupled so that they can be optimized concurrently. Each shared variable is distributed to the subproblem for which it has the most influence on the objective and constraint functions, determined using the computed sensitivities. Every subspace optimization problem will then be solved with respect to its local variables and a subset of its shared variables, while all other variables are held constant. The constraints in each subspace are represented by one so called cumulative constraint. The formulation of each subspace optimization problem includes minimization of the objective function subject to the local cumulative constraint and to approximations of the cumulative constraints of the other subproblems. The responsibility for satisfying the local cumulative constraint of a certain subproblem is thus shared between all the subspace optimizers. How this responsibility is distributed is governed by the system coordinator. The cumulative constraints of the neighbouring subspaces and parts of the objective function that are only influenced by the subproblem indirectly, are calculated using the sensitivities computed in the previous step. The new design point $x^{k+1}$ is simply the combination of optimized variables from the different subspaces. This point is not necessarily feasible, as shown by Pan and Diaz (1989). Finally, the system coordinator redistributes
the responsibility for the different cumulative constraints to further reduce the objective function in the next iteration. The process will continue iteratively until convergence is reached.

Figure 5.10 Schematic picture of the iterative process in the original CSSO method.

Bloebaum et al. (1992) successfully implement the CSSO method, but incorporate some modifications to the system coordinator to achieve convergence. Pan and Diaz (1989) illustrate how a sequential solution strategy of the subspace optimizations and the combination of optimized variables can result in pseudo optimal points, i.e. points that are optimal in each subspace but that are not optimal for the full problem. They suggest a strategy to move away from pseudo-optimal points when solving the subspace optimization problems in sequence. Shankar et al. (1993) show that the original formulation fails solving simple quadratic problems. They propose a modified algorithm that is used to successfully solve large quadratic problems with weak coupling, but that does not behave well on large quadratic problems with strong coupling.

A variant of CSSO is presented by Renaud and Gabriele (1991). They introduce a totally different coordination procedure where optimization of a global approximation of the problem is performed. The approach is summarized below and is also depicted in Figure 5.11. The first steps in this formulation are in principle the same as in the original CSSO method, including the system analysis, the system sensitivity analysis, and the concurrently performed subspace optimizations. This results in a combination of optimized variables from the different subspaces, \( x^{k+1,sub} \). A design database is introduced, where information about the objective function, constraints, and associated gradients in the design points evaluated by the system and subspace analyzers are stored. The design database is used to formulate an approximation of the global problem around \( x^{k+1,sub} \). Optimization of the approximated global problem is then performed and the obtained optimum, \( x^{k+1} \), is the design vector input to the next iteration. Renaud and Gabriele (1993) develop the formulation by making the approximation of the global problem more accurate. In a later publication, Renaud and Gabriele (1994) replace the cumulative constraints by the individual constraints. Both measures yield improved convergence.
Starting from the modifications proposed by Renaud and Gabriele, Wujek et al. (1995) suggest further development of the CSSO method by introducing variable sharing between the subspaces and second order polynomials to approximate the global problem. Variable sharing allows variables to be allocated to more than one subproblem, making the approximation of the global problem more accurate. Sellar et al. (1996) also proceed from the aforementioned modifications, but use neural networks as a global approximation of the problem. The neural networks are first used by the subspace optimizers instead of the computed sensitivities to estimate how their design decisions affect other subproblems, and then by the system optimizer to find a new approximate optimal point.

Distributing the responsibility of the design variables to the different subspaces, which is done in the original CSSO method, is an attractive idea. However, this formulation has several shortcomings as has been discussed above. Renaud and Gabriele (1991) introduce an approach that is very different from the original method. The variants that are based on their work suffer from the drawback that all variables are dealt with at the system level. This restricts the autonomy of the groups responsible for each subspace, which was the main motivation for using a multi-level method.

5.3.2 Bilevel Integrated System Synthesis

Bilevel integrated system synthesis (BLISS) was first introduced by Sobieszczanski-Sobieski et al. (1998) at NASA Langley Research Center. The original implementation concerns four coupled subspaces of a supersonic business jet: structures, aerodynamics, propulsion, and aircraft range. Few other applications can be found in the literature. The method is iterative and optimizes the design in two main steps. First, subspace optimizations with respect to the local variables are performed in parallel. Next, the system optimizer finds the best design with respect to the shared variables.

A flowchart of the original BLISS method can be seen in Figure 5.12. The first two steps are identical to the first two steps in the CSSO method described in Section 5.3.1, but are explained more in detail.
here. A system analysis is first performed on the design variables obtained from the previous iteration in order to find a multidisciplinary consistent design, i.e. $y^k$ corresponding to the local and shared variables $x_{\text{local}}^k$ and $x_{\text{shared}}^k$ are found for iteration $k+1$. This typically includes performing subspace analyses in an iterative fashion in order to find the values of the coupling variables, see the MDF method in Section 5.2.1.

![Figure 5.12](image-url)

**Figure 5.12** Schematic picture of the iterative process in the original BLISS method.

In the second step, a sensitivity analysis is performed in order to find the derivatives of the coupling variables with respect to the local variables, $dy^k/dx_{\text{local}}^k$. Subspace sensitivity analyses are first computed in order to find the partial derivatives of the coupling variables output from every subspace with respect to the coupling and local variables input to that subspace. When this is done for every subspace, a linear equation system can be solved for each local variable in order to find the total derivatives of the coupling variables with respect to that variable. These equations are called the global sensitivity equations, see Sobieszczanski-Sobieski (1990) for more details. The third step is to perform subspace optimizations in parallel. In order to do so, objective functions for each subspace need to be formulated. The global objective function is treated as the $m$th component of the vector of coupling variables input to the first subspace and is denoted by $y_{\text{1},m}$. A linear approximation of the global objective function, keeping the shared variables constant, can be constructed using the computed sensitivities according to:

$$f = y_{1,m} = (y_{1,m})_0 + \sum_j \left( \frac{dy_{1,m}}{dx_{ij}} \right)^T \Delta x_{ij} \tag{5.3}$$

The objective function for each subspace is set to the part of Equation (5.3) that estimates the influence of that specific subspace on the global objective function. The subspace optimization
Multidisciplinary Design Optimization Methods

problem $j$ can then be formulated as the minimization of the subspace objective function with respect to the local variables and subject to local constraints, keeping the shared variables constant:

$$\min_{\Delta x_{ij}} f_j = \left( \frac{dy_{1*,m}}{dx_{ij}} \right)^T \Delta x_{ij}$$

subject to

$$g_j \leq 0$$

$$\Delta x_{ij}^{lower} \leq \Delta x_{ij} \leq \Delta x_{ij}^{upper}$$

(5.4)

The subspace optimization problems are solved in parallel using the subspace analyzers, resulting in updated local variables, $x_{ij}^{k+1}$. When the design has been improved by changing the local variables, a system optimization with respect to the shared variables will be performed. A linear approximation of the global objective function, keeping the local variables constant, is used as the system objective function. The total derivatives of the global objective function with respect to the shared variables, $df/dx_s^0$, are therefore computed as the fourth step of the algorithm. In the original reference to the BLISS method, two alternative approaches for obtaining these derivatives are presented: BLISS/A and BLISS/B. Details are left out in this description. The final step is the solution of the following system optimization problem that is unconstrained except for limits on the shared variables:

$$\min_{\Delta x_s} f = y_{1*,m} + \left( \frac{dy_{1*,m}}{dx_s} \right)^T \Delta x_s$$

subject to

$$\Delta x_s^{lower} \leq \Delta x_s \leq \Delta x_s^{upper}$$

(5.5)

If constraints in the subspace optimizations depend more strongly on the shared and coupling variables than on the local variables, they might need to be considered in the system optimization, turning the system optimization problem into a constrained one.

The BLISS procedure separates the optimization with respect to the local and shared variables. If the problem contains non-convex constraints, a gradient-based optimization algorithm can terminate in a different solution, e.g. in a local optimum, than if all variables were optimized simultaneously. Kodiyalam and Sobieszczanski-Sobieski (2000) describe this problem and suggest solving it by adding copies of the shared variables to the subspace optimization problems, and by introducing compatibility constraints in the system optimization problem to ensure a consistent design at optimum. A variant of BLISS with second order polynomial metamodels was developed by Kodiyalam and Sobieszczanski-Sobieski (2000). The system optimizer uses metamodels of the objective and possible constraint functions that are constructed as functions of the shared variables, eliminating the need to find the derivatives of the global objective function with respect to the shared variables. Two different algorithms are proposed. The first constructs metamodels based on data from the system analyzer while the second constructs metamodels based on data from the subspace optimizers where the coupling variables are linearly extrapolated using the sensitivity information.

Sobieszczanski-Sobieski et al. (2003) present an extension of the BLISS method referred to in the literature as BLISS 2000 or simply BLISS. The key concept in this modified method is that the objective function in each subspace optimization is a sum of the coupling variables output from that specific subspace multiplied with weighting coefficients. By controlling the weighting coefficients, the system optimizer can instruct the subspaces on what emphasis should be put on each output in order to minimize the global objective. The weighting coefficients can be positive implying minimization, or negative implicating maximization, of the corresponding output. Another salient
The feature of BLISS 2000 is that surrogate models of the subspaces are used as the link between the subspace and the system optimizers, replacing the sensitivity analyses in the original formulation. The surrogate models represent a large set of feasible subspace designs available to the system optimizer. Polynomial surrogate models are used in the original version of BLISS 2000. Kim et al. (2004) demonstrate the use of Kriging surrogate models. Each subspace could in principle be given the freedom to choose their own surrogate model. An illustration of BLISS 2000 can be found in Figure 5.13 and each step is described below.

**Figure 5.13** The iterative process of BLISS 2000.

The first step in BLISS 2000 is to initialize the local, shared, and coupling variables as well as the weighting coefficients. A system analysis can be performed in order to have a consistent starting design, but is not required. The iterative process then starts with performing a DOE for each subspace, which means that a number of different input settings to that subspace are set up. The input to a subspace consists of the shared variables $x_{sj}$, the coupling variables $y_{j*}$, and the weighting coefficients $w_j$. $y_{j*}$ denotes the coupling variables input to subspace $j$ and the addition of the superscript plus sign indicates that the coupling variable is output from the system optimizer. $w_j$ denotes the weighting coefficients corresponding to the coupling variables output from subspace $j$, while $w$ denotes the collection of all weighting coefficients. The subspace optimization problem is formulated as

$$\min_{x_{ij}} f_j = \sum_{m} w_{jm} y_{j*,m}$$

subject to

$$g_j \leq 0$$

$$x_{ij}^{lower} \leq x_{ij} \leq x_{ij}^{upper}$$

(5.6)

where the subspace objective function is a sum of the coupling variables output from that specific subspace multiplied with weighting coefficients. The result of the subspace optimization is the values of the local variables $x_{ij}$ and the coupling variables $y_{j*}$ output from that subspace. The subspace
Multidisciplinary Design Optimization Methods

The optimization problem is solved for each point in the DOE. The next step is to fit surrogate models to represent approximations of how each element of $y_j$ depends on $x_{sj}$, $y_{j^*}$, and $w_j$. These surrogate models will constitute a database accessible to the system optimizer. Surrogate models of the local variables could also be generated, but is avoided as the local variables are not used by the system optimizer. In the final step of each iteration, the system optimizer finds values of $x_s$, $y^*$, and $w$ that minimize the global objective $f$ subject to the constraint that the design has to be consistent. The system optimization problem is formulated as

$$\min_{x_s, y^*, w} f$$

subject to

$$y^* - y = 0$$

$$(x_{lower}^j, y_{lower}^j, w_{lower}) \leq (x^*_s, y^*, w) \leq (x_{upper}^j, y_{upper}^j, w_{upper})$$

The iterative process will continue until convergence after which the optimal values of the local variables will be retrieved.

Sobieszczanski-Sobieski et al. (2003) prove that using the BLISS 2000 algorithm on a convex problem yields the same result as when solving the non-decomposed problem. Kim et al. (2004) describe how the BLISS 2000 algorithm will fail if the subspaces cannot find feasible solutions for certain combinations of variables input from the system optimizer. They solve these problems by introducing approximation models for constraint violation that are added to the system optimization problem.

BLISS and BLISS 2000 perform best for problems with a small number of shared variables and a large number of local variables. In most of the references referred to in this section, the method has been applied to examples in the aerospace industry with large coupling strength. According to Tedford and Martins (2006), BLISS 2000 may be inefficient for problems with large coupling breadth as the number of variables at the system level increases with a factor of two for every coupling variable, and the creation of surrogate models with many variables can become expensive. Further on, BLISS 2000 is not meaningful for problems lacking coupling variables.

### 5.3.3 Collaborative Optimization

Collaborative optimization (CO) is a bilevel hierarchical method that was developed at Stanford University. An early description of CO was published by Kroo et al. (1994). Braun (1996a) wrote his Ph.D. thesis on the subject.

In CO, the system optimizer is in charge of target values of the shared and coupling variables. The subspaces are given local copies of these variables that they have the freedom to change during the optimization process. The local copies converge towards the target values at optimum, i.e. a consistent design is obtained. An overview of the method can be found in Figure 5.14. The system optimizer minimizes the global objective function subject to constraints that ensure a consistent design. The subspace optimizers minimize the deviation from consistency subject to local constraints.

To describe the CO method more in detail, the target values of the shared and coupling variables are introduced. These are governed by the system optimizer. The collection of target values is called $z^+$ and the target values corresponding to subspace $j$ is denoted $z^+_j$. The local copies of the shared variables in subspace $j$ and of the coupling variables output from subspace $j$ are denoted $z_j$, i.e. $z_j = (x_{sj}, y_j)$. The local copies are controlled by the subspace optimizers. The $z_j$ in the different subspaces are obviously not mutually disjoint.
Figure 5.14 Overview of the collaborative optimization method for three subspaces.

The system optimization problem is formulated as:

$$\begin{align*}
\min_{z^+} & \quad f(z^+) \\
\text{subject to} & \quad J_j = \|z_j^+ - z_j\|_2^2 \leq 0, \quad j = 1, ..., n
\end{align*}$$

(5.8)

where $n$ is the number of subspaces. The system optimizer minimizes the global objective function $f$, with respect to the target values $z^+$ and subject to the constraints that the local copies in the subspaces $z_j$ are to match the target values $z_j^+$. Some CO formulations state the system level constraints as equalities and some as inequalities. Stating them as inequalities when using a solution algorithm that linearizes constraints can improve convergence, see Braun et al. (1996b) for more details.

The $j^{th}$ subspace optimization problem is formulated as:

$$\begin{align*}
\min_{x_{ij},x_{sj}} & \quad J_j = \|z_j^+ - z_j\|_2^2 \\
\text{subject to} & \quad g_j(x_{ij},x_{sj}) \leq 0
\end{align*}$$

(5.9)

The subspace optimizers minimize the deviations between the local copies $z_j$ and the corresponding target values $z_j^+$. The optimization is performed with respect to local and shared variables. There are also local constraints that need to be fulfilled.

There are a number of numerical problems associated with the CO method when used in combination with gradient-based algorithms, which is described by DeMiguel and Murray (2000) and also by Alexandrov and Lewis (2002). DeMiguel and Murray list five properties of CO:

1. The system level constraints $J_j$ are in general non-smooth functions of the target values $z^+$ and therefore not differentiable.
2. The jacobian for the system level constraints, $\nabla (J_1(z_1^+), J_2(z_2^+), ... , J_n(z_n^+))$, is singular at optimum.
3. Several local minima might exist in a subspace for each set of target values $z^*$.
4. The Lagrange multipliers in the subspace problems are zero or converge to zero at optimum.
5. The system level problem has no information about which constraints are actively constraining the solution.

These features hinder convergence proofs and have an adverse effect on the convergence rate, which make the system level problem difficult to solve for conventional non-linear programming algorithms. A number of attempts to modify the CO method to overcome these difficulties are documented in the literature. Three of these modifications are presented below: collaborative optimization using surrogate models, modified collaborative optimization, and enhanced collaborative optimization.

Sobieski and Kroo (2000) introduce the use of polynomial surrogate models to represent the subspace objective functions, which are also the system level constraints, for all subspaces. Note that the surrogate models are not used as approximations of the subspace analyses, which would otherwise be a natural application area of surrogate models within MDO. The surrogate models represent the optimum value of $J_j$ as a function of $z_j^+$ for every subspace $j$. This is achieved by solving the subspace optimization problem for a set of target values $z_j^+$ and creating second order polynomials to represent $J_j^\text{optimum}(z_j^+)$. Second order polynomials are unlikely to accurately represent the whole region of interest, and will therefore be regenerated when necessary during the system level optimization process. This approach solves issues 1 and 3 above, but not issues 2, 4, and 5 according to Roth (2008). However, slow convergence is a smaller problem as the surrogate models are cheap to evaluate.

**Modified collaborative optimization (MCO),** presented by DeMiguel and Murray (2000), overcomes some of the difficulties associated with the original formulation. Firstly, the $L_1$-norm is used instead of the $L_2$-norm in the subspace objective functions. Secondly, the system level problem becomes unconstrained as penalty terms are added to the objective function, replacing the constraints in the original formulation. These modifications solve problems 2 and 4 above. Problem 1 is dealt with by solving a sequence of so called perturbed MCO problems that unfortunately become ill-conditioned during the solution process, see DeMiguel and Murray (2000) for more details. Further on, problem 3 still exists. The MCO method is called exact penalty decomposition in the Ph.D. thesis written by DeMiguel (2001).

**Enhanced collaborative optimization (ECO)** was introduced by Rooth (2008). The method is a development of CO, but is also influenced by MCO and analytical target cascading which will be described in Section 5.3.4. An overview of the method can be found in Figure 5.15. In ECO, the goal of the system optimizer is to find a consistent design. There are no constraints on the system level, which makes the system optimization problem trivial to solve. The objective functions of the subspaces contain the global objective in addition to terms for the deviation from consistency. It is intuitively more appealing for the subspaces to work towards minimizing a global objective, rather than to just minimize a deviation from consistency as is done in the original CO formulation. The subspaces are subject to local constraints as well as to linearized versions of the constraints in the other subspaces. The inclusion of the latter constraints provides a direct understanding of the preferences of the other subspaces, as compared to CO where this knowledge is only obtained
indirectly from the system optimizer. The benefits of the ECO method compared to the CO method lie in the resolution of the five problematic issues described previously. However, the complexity of the ECO method is a major drawback.

![Diagram of enhanced collaborative optimization method](image)

**Figure 5.15** Overview of the enhanced collaborative optimization method.

### 5.3.4 Analytical Target Cascading

**Analytical target cascading (ATC)** is a multi-level hierarchical method that was developed at the University of Michigan in cooperation with the automotive industry. The method is discussed in the Ph.D. thesis by Kim (2001). It was originally intended as a product development tool for propagating targets, i.e. convert targets on the overall system to targets on smaller parts of the system, but can also be used for optimization. Traditionally, MDO refers to simultaneously considering different disciplines, or aspects, during an optimization process. As was discussed in Section 5.1.2, a problem can either be decomposed by aspect or by object. In contrast to the previously described multi-level optimization methods, ATC was designed for decomposition by object. However, the different disciplines, or aspects, can be studied for each object.

Analytical target cascading in the product development process can be described by four steps, see Figure 5.16. The method can be used for an arbitrary number of levels that are hierarchically interrelated, but is here assumed to be used for three levels: a system level as well as subsystem and component levels. The first step consists of specifying the system targets. Next, the actual ATC process takes place in which targets are propagated to the subsystem and component levels, a process that is further described below. This step is typically performed early in the product development process using coarse models of the subspaces. The system, subsystems, and components are thereafter designed in parallel and autonomously to meet the specified targets. They are modelled in detail and no interaction between them is needed in this step. However, if subsystems and/or components fail to meet the specified targets, the ATC process in the previous
step must be performed once again. Verification of the system targets is finally performed, and if not successful, the whole process must be repeated.

Figure 5.16 Analytical target cascading in the product development process.

The categorization of local, shared, and coupling variables defined in Section 5.1.1 and used in the previously described MDO methods is not applicable to the ATC formulation. The types of variables used by Kim (2001) are presented here. **Local variables** in a system, subsystem, or component subspace refer to variables controlled by that specific subspace. These variables may not necessarily be original design variables. **Linking variables** are variables that are common to more than one subspace on the same level sharing the same parent subspace. These variables may or may not be original design variables. **Responses** are variables that are generated by a subspace and sent to its parent subspace. **Targets** are to be matched by a subspace and set by its parent subspace. Allison et al. (2005b) describe how linking variables and responses relate to shared and coupling variables defined in Section 5.1.1. Linking variables are equivalent to shared variables, and responses can, but must not, be coupling variables.

The original problem can be defined as minimizing the differences between the targets and the responses obtained from models of the system, with respect to the design variables, while satisfying a number of constraints.

\[
\min_x \| T - R(x) \|_2^2 \\
\text{subject to } g(x) \leq 0
\]  
(5.10)
In the target cascading process, the problem is divided into system, subsystem, and component levels, called levels a, b, and c, respectively. Targets for all subspaces are to be found while meeting the original targets and fulfilling the local subspace constraints. An overview of the data flow between the subspaces can be found in Figure 5.17, and the nomenclature is explained below.

**Figure 5.17** Overview of the data flow between subspaces in the ATC process.

The optimization problems at the system, subsystem, and component levels will now be formulated according to Kim et al. (2003) with some of the modifications proposed by Michalek and Papalambros (2005). An optimizer solves the optimization problem in each subspace using an analyzer to evaluate the responses in different design points. The nomenclature used is given in Table 5.2.

**Table 5.2** List of symbols used for analytical target cascading.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_a )</td>
<td>Number of subsystems</td>
</tr>
<tr>
<td>( n_{bi} )</td>
<td>Number of components belonging to subsystem ( i )</td>
</tr>
<tr>
<td>( x_a )</td>
<td>Local system variables</td>
</tr>
<tr>
<td>( x_{bi} )</td>
<td>Local variables in subsystem ( i )</td>
</tr>
<tr>
<td>( x_{cij} )</td>
<td>Local variables in component ( j ) belonging to subsystem ( i )</td>
</tr>
<tr>
<td>( y_b )</td>
<td>Linking variables in subsystems, ( y_b = y_{b1} \cup y_{b2} \cup ... \cup y_{bn_a} )</td>
</tr>
<tr>
<td>( y_{bi} )</td>
<td>Linking variables in subsystem ( i )</td>
</tr>
<tr>
<td>( y_{bi}^{\text{UL}} )</td>
<td>Targets for linking variables in subsystem optimizer ( i ), passed from system optimizer</td>
</tr>
<tr>
<td>( y_{ci}^{\text{L}} )</td>
<td>Linking variables in subsystem optimizer ( i ), passed to system optimizer</td>
</tr>
<tr>
<td>( y_{ci} )</td>
<td>Linking variables in components belonging to subsystem ( i ), ( y_{ci} = y_{c1i} \cup y_{c2i} \cup ... \cup y_{cin_{bi}} )</td>
</tr>
<tr>
<td>( y_{cij} )</td>
<td>Linking variables in component ( j ) belonging to subsystem ( i )</td>
</tr>
<tr>
<td>( y_{cij}^{\text{UL}} )</td>
<td>Targets for linking variables in component optimizer ( j ), passed from subsystem optimizer ( i )</td>
</tr>
<tr>
<td>( y_{cij}^{\text{L}} )</td>
<td>Linking variables in component optimizer ( j ), passed to subsystem optimizer ( i )</td>
</tr>
</tbody>
</table>
The system problem controls its local variables, the linking variables in the subsystem problems, the responses from the subsystem problems, and certain consistency tolerances. It is formulated as follows.

\[
\begin{align*}
\min_{x_a, y_b, R_b, \varepsilon_R, \varepsilon_y} & \quad \| R_a \left( R_b, x_a \right) - T_a \|_2^2 + \varepsilon_R + \varepsilon_y \\
\text{subject to} & \quad \sum_{i=1}^{n_a} \| R_{bi} - R_{bi}^L \|_2^2 \leq \varepsilon_R \\
 & \quad \sum_{i=1}^{n_a} \| y_{bi} - y_{bi}^L \|_2^2 \leq \varepsilon_y \\
 & \quad g_a \left( R_b, x_a \right) \leq 0
\end{align*}
\]  

(5.11)

The objective is to minimize the differences between the system responses and targets. There are local constraints, \( g_a \), as well as consistency constraints that coordinate the subsystem responses and linking variables. The consistency tolerances \( \varepsilon_R \) and \( \varepsilon_y \) that are included in the system objective and in the consistency constraints approach zero at convergence. Michalek and Papalambros (2005) include weighting coefficients for linking variables and responses, but these are left out in this description.

There are \( n_a \) number of subsystem problems, and the \( i^{\text{th}} \) problem is formulated below.

\[
\begin{align*}
\min_{x_{bi}, y_{bi}, y_{ci}, R_{ci}, \varepsilon_R, \varepsilon_y} & \quad \| R_{bi} \left( R_{ci}, x_{bi}, y_{bi} \right) - R_{bi}^U \|_2^2 + \| y_{bi} - y_{bi}^U \|_2^2 + \varepsilon_R + \varepsilon_y \\
\text{subject to} & \quad \sum_{j=1}^{n_{bi}} \| R_{cij} - R_{cij}^L \|_2^2 \leq \varepsilon_R \\
 & \quad \sum_{j=1}^{n_{bi}} \| y_{cij} - y_{cij}^L \|_2^2 \leq \varepsilon_y \\
 & \quad g_{bi} \left( R_{ci}, x_{bi}, y_{bi} \right) \leq 0
\end{align*}
\]  

(5.12)
Here, the objective is to minimize the differences between the subsystem responses and the corresponding targets passed from the system level, as well as the differences between the subsystem linking variables and the corresponding targets. The targets for subspace responses and linking variables are determined from the solution of the system problem, see Equation (5.11). In analogy to the system level problem, there are local constraints and consistency constraints that coordinate the component responses and linking variables. Moreover, the consistency tolerances $\varepsilon_R$ and $\varepsilon_y$ approach zero at convergence. The subsystem problem is the most general one, as it is in the middle of the hierarchical structure.

For each subsystem $i$, there are $n_{bi}$ number of component problems, and the $j^{th}$ component problem is stated below.

$$\min_{x_{cij}, y_{cij}} \left\| R_{cij}(x_{cij}, y_{cij}) - y_{cj}^U \right\|^2 + \left\| y_{cij} - y_{cj}^U \right\|^2$$
subject to $g_{cij}(x_{cij}, y_{cij}) \leq 0$ (5.13)

The objective in the component problem is to minimize the differences between the component responses and the corresponding targets from the subsystem level, and between the component linking variables and the corresponding targets, subject to local constraints. As the component problem is on the bottom of the hierarchy, there are no lower level problems to coordinate. Therefore, there are no consistency constraints or tolerances in the objective function.

Analytical target cascading can be used for MDO. However, ATC was not developed as an optimization tool and differs from the previously described methods in several ways. Traditionally, MDO is used to simultaneously optimize several disciplines, and it is therefore natural to use aspect-based decomposition resulting in a bilevel structure. Analytical target cascading requires a hierarchical model structure and can handle an arbitrary number of levels, which is typically appropriate when using object-based decomposition. The question is then how the ATC framework can incorporate the classical MDO problems. Kokkolaras et al. (2004) show how optimization can be performed by setting the targets to zero or infinity in a minimization or maximization problem, respectively. Further on, Allison et al. (2005b) describe how the original formulation can be extended to include responses calculated by parent subspaces to be input to child subspaces, which is needed in a general MDO framework. This formulation is used by Tosserams et al. (2008) when studying an MDO problem from the aerospace industry involving four disciplines. The example is the supersonic business jet problem used in the development of the BLISS algorithm. The authors use a bilevel structure with one discipline at the top level and the other three disciplines at the lower level. They also extend the ATC formulation to include non-hierarchical target-response coupling, i.e. communication directly between child subspaces, which results in a lower computational cost than when solving the same problem without this modification. Allison et al. (2005b) describe how ATC can be used to decompose a problem by object, and how another MDO method can be employed to study different disciplines within each object.
6 Multidisciplinary Design Optimization for Automotive Applications

The roots of MDO lie in structural optimization and many methods have been developed in collaboration with the aerospace industry. To be able to evaluate the currently available MDO methods for automotive applications, there is a need for some basic knowledge of the product development process and the simulations involved in the automotive development. This information is given in the first part of the chapter, followed by a general comparison between the automotive and the aerospace industries. A brief summary of one common application of MDO within the aerospace industry is then presented as a short background before the applications and experiences from the automotive industry are described.

6.1 Simulations in the Automotive Industry

The development of a new car is a complicated task and many experts with different skills and responsibilities are needed. Development has gone from being solely done based on trial and error in a hardware environment, to become a process where almost every aspect of the development is done with help of CAE tools, and hardware is only available as the final product and seldom as prototypes. Today's development therefore depends heavily on detailed simulations of every aspect of all parts of the automotive structure.

<table>
<thead>
<tr>
<th>Design area simulations</th>
<th>Performance area simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Body</strong></td>
<td>Aerodynamics and thermal</td>
</tr>
<tr>
<td></td>
<td><strong>Noise, vibration and harshness (NVH)</strong></td>
</tr>
<tr>
<td><strong>Chassis</strong></td>
<td>Safety</td>
</tr>
<tr>
<td><strong>Interior</strong></td>
<td><strong>Vehicle dynamics</strong></td>
</tr>
</tbody>
</table>

*Figure 6.1* Schematic illustration of simulation areas within the automotive industry, example from Saab Automobile.

As reflected by the former Saab Automobile organisation, simulations can roughly be divided into two different categories. The first one supports certain design areas, e.g. body, chassis, or interior design. The other one evaluates disciplinary performance, such as safety or aerodynamics, which depends on more than one design area, see Figure 6.1. The former is consequently evaluating many
different aspects, e.g. stiffness, strength, and durability, for a certain area of the vehicle, while the latter focuses on one performance area which often depends on the complete vehicle. In conjunction with the different simulation areas there is, in most cases, also a corresponding test organisation performing the hardware validation at the end of the project. The division of the simulation work into design area simulations (division by object) and performance area simulations (division by aspect) reflects the different types of decompositions proposed for MDO problems, see Section 5.1.2.

Many parts and subsystems in a car are developed by suppliers and consequently simulations on these parts and subsystems are first done by the suppliers. The integration of these systems into the vehicle is then checked by the car manufacturer. One large such system, where extensive detailed simulations normally are done separately, is the powertrain system, i.e. engine and gearbox.

Many different loadcases are evaluated within each simulation area. Regarding safety, e.g. front, side, and rear end crashes are studied and for each of these crash directions various impact speeds, barriers, and occupants are considered. Optimization can be used to guide the design within one discipline, maybe find the balance between contradicting loadcases, or be multidisciplinary and consider loadcases from more than one discipline or simulation area, see Figure 6.2.

![Figure 6.2 Example of breakdown of one discipline into loadcases. Optimization could be done at different levels and be multidisciplinary (important loadcases from several disciplines) or multiloadcase (important loadcases from one discipline).](image)

6.2 Product Development Process in the Automotive Industry

The product development process (PDP) can differ between companies, but the main idea is the same, i.e. to describe what should be done at different stages during the development. The PDP starts with initial concepts that are gradually refined with the aim of eventually fulfilling all predefined targets. At certain stages during the development, the complete design is evaluated and if found satisfactory, the design process is allowed to progress to the next stage. During the last two decades, numerical simulations through finite element methods (FEM) have been well integrated into the PDP. Today the development is more or less driven by numerical simulations, as noted by Duddeck (2008). Consequently, both the development of the computer aided engineering (CAE) models, as well as the development of the resulting hardware, is integrated within the PDP of today,
One result of the increased focus on simulations is that the number of prototypes needed to test and improve different concepts has been reduced, although the number of aspects to be considered during development has increased considerably. Hence, the increased use of simulations has resulted both in shortened development times and in reduced development costs.

Structural optimization can be used within the different stages of the PDP, see Figure 6.4. In the early phases, optimization can be used to find promising concepts and in the later phases, when the design is more fixed, optimization can be used to fine-tune the design. Even if optimization has shown to give better designs in many cases, the software, hardware, and knowhow needed to implement optimization within the PDP have delayed the utilization of its full potential. This is certainly the case for MDO and it is important to find methods that can fit into a modern PDP without jeopardizing the strict time limits.

**Figure 6.3** A generic development plan with emphasis on the simulation activities.

**Figure 6.4** Examples of different types of optimizations that could be realized during different phases of the product development. Topology optimization is used to find where material should be placed to be most effective, shape optimization is used to find the best possible shape of an existing part, and size optimization is used to find the suitable size of a variable, e.g. a thickness, see Section 3.2.
6.3 Comparison between the Aerospace and Automotive Industries

The different MDO methods were initially developed within the aerospace industry in cooperation with research organisations, but have now also gained interest within other industries, such as the automotive industry. However, there are some differences between the aerospace and automotive industries that might influence which methods that are suitable and to what extent they might be used.

The aerospace industry has long product and design cycles and produces few but very expensive products compared to the automotive industry. In addition, the aerospace industry usually has a military branch, which mainly is state funded, and where there may be more time and resources available to develop new processes and methods. The development in the aerospace industry is rigorously ruled by standards and regulations while passenger cars are designed to fulfil a number of market requirements and expectations in addition to the legislative requirements. The number of large automotive manufacturers is also greater than the number of large aerospace manufacturers. This might lead to stronger competition in the automotive industry, and the strive for better products as well as shorter and less expensive product development may therefore be more pronounced. Thus, it is logical that some methods and processes are developed within the aerospace industry, which might have the time and resources available, and that these methods subsequently are adopted, and maybe become even more used within the automotive industry, which constantly is seeking improvements due to the fierce competition.

The aerospace industry has to follow methods and processes during the product development that are approved by governmental safety agencies such as FAA (Federal Aviation Administration) in the USA and EASA (European Aviation Safety Agency) in Europe. The development process has therefore become rather conservative. The product development within the automotive industry, on the other hand, is not as rigorously controlled. The requirements are more related to the performance of the vehicle, like safety or CO\textsubscript{2}-emission, and specific methods are not prescribed for the product development. These facts might be additional reasons for the faster introduction of new processes and methods within the automotive industry compared to the aerospace industry. The question is then whether the MDO methods developed specifically for the aerospace industry, are suitable for automotive applications as is, or if some characteristics of the automotive applications requires the methods to be adjusted. Another possibility could be that the methods found insufficient for aerospace applications are better suited for automotive applications.

One of the differences between the development processes in the aerospace and the automotive industries is the development of the structural parts, i.e. the wings and fuselage of the aeroplane and the body of the car. The wings are for example typically dimensioned with respect to fatigue, and although there is considerable movement of the wings during flight, the stresses are kept within the elastic region. The car body is, to a large extent, dimensioned by crashworthiness requirements, and the problem then becomes highly non-linear with large plastic deformations. However, during normal operation, the car body has small deformations compared to the aeroplane structure. Thus, when studying the aerodynamics of an aeroplane, it is essential to take the deformations induced by the aerodynamic forces into account, while this is not as important when studying the aerodynamics of a passenger car. The forces induced by the deformations are one of the major loads that the wing structure should carry, while the corresponding forces on a car are negligible compared to the forces applied during a crash event. The coupling between disciplines, e.g. aerodynamics and structural
performance, is in this example thus much stronger in the aeroplane case compared to the passenger car case.

As a consequence of the coupling between disciplines, an iterative approach is needed to find a consistent solution, i.e. a solution in balance. This might be done by first estimating the aerodynamic loads for the structural simulation. The deflections obtained are then applied in the aerodynamic simulation to find the aerodynamic forces. The iteration is continued until the forces and deflections match each other. There are examples of coupled disciplines in the automotive industry as well, e.g. vehicle dynamics and chassis structural performance, which both depend on the chassis stiffness, but they are not dominating the product development. Incorporating MDO into the automotive design process is therefore presumably simpler than in the aerospace industry since the disciplines are more loosely coupled, as stated by Agte et al. (2010). It could be said that automotive designs are created in a multi-attribute environment rather than in a truly multi-disciplinary environment, and aspects, such as NVH and crashworthiness, are only coupled by shared system level variables. The absence of strong coupling between disciplines makes it easier to incorporate metamodels in the optimization process and consequently also possible to include very computationally expensive simulations more conveniently.

It is often possible to use direct optimization methods for linear simulations, since the computational cost for every simulation is low and the studied responses do not include many local minima and maxima. Non-linear simulations are often computationally costly and the responses complex, and consequently more advanced optimization methods are required. These methods, however, demand more evaluations to find the optimum, and therefore the use of metamodels becomes interesting.

Another difference between the aerospace and automotive industries is how the development is done. In the aerospace industry, different parts of the aeroplane are developed by different companies in a joint project, and the different companies have fixed input with which they should fulfil certain requirements. Although there are system suppliers in the automotive industry with responsibility for the performance of their parts, the responsibility of the complete vehicle is still left to the vehicle manufacturer. Thus, there might be longer communication paths in aerospace development projects, which result in a stronger need for the different parties to work independently also when doing full scale MDO. Hence, the need for autonomy, as offered by multi-level optimization methods, is even more obvious.

Some of the results of the differences mentioned, e.g. coupling of variables, can be seen later in this chapter when the experiences from the automotive industry regarding MDO are presented and compared with the experiences from the aerospace industry.

### 6.4 Multidisciplinary Design Optimization Applications

Multidisciplinary design optimization is not yet implemented as a general tool within the automotive product development. However, some of the successful applications of MDO are presented here to give insight to what has been achieved so far. The presentation starts with introducing a typical application from the aerospace industry which is then followed by a typical automotive application. In this way the differences between the industries are highlighted before some more examples from the automotive industry are presented. It will be clear that the use of multi-level optimization methods has not advanced into the every-day use within the automotive industry, although some successful examples are recorded within the academic world.
6.4.1 Typical Aerospace Example

One of the most common applications of MDO within the aerospace industry has been simultaneous aerodynamic and structural optimization of aircraft wings or complete aircraft configurations as described by Sobieszczanski-Sobieski and Haftka (1997). The trade-off between aerodynamic and structural efficiency drives aircraft design and the appropriate balance needs to be found between slender shapes with less drag, resulting in lower operating cost due to lower fuel consumption, and more stubby shapes with less mass, giving lower manufacturing cost. Two aerodynamic-structural interactions affect the trade-off. First, the structural weight affects the required lift and, thus, drag. Second, structural deformations change the aerodynamic shape. The second effect can be compensated for by building the structure such that it will deform to the desired shape. This simplification means that the aerodynamic design affects all aspects of the structural design while the structural design affects the aerodynamic design only through the structural weight. This asymmetry allows a two-level optimization with the aerodynamic design at the upper level and the structural design at the lower level. Each aerodynamic analysis hence requires a structural optimization, see Figure 6.5. This approach makes sense since the structural analysis usually is much cheaper than the aerodynamic analysis. This sequential technique works when structural deformations are approximately constant throughout the main part of the flight time, as is the case for most conventional transport aircrafts. However, it does seldom lead to the optimal design of the global system, as noted by Kroo (1997). In addition, when aerodynamic performance is important for multiple design conditions with different structural deformations, a completely integrated structural and aerodynamic optimization may be necessary to obtain high-performance designs.

The wish for disciplinary autonomy and parallelisation of work have resulted in the development of different multi-level optimization methods that have been tested on academic examples, see Section 5.3. Although showing promising results, the regular use of these methods within the industry has not yet been realized and incorporation of MDO methodology in efficient multi-level strategies, rather than only single level approaches, is still missing according to Agte et al. (2010).

Figure 6.5 Schematic description of simultaneous optimization of aerodynamic and structural performance of aerospace structures. 

- **a) Coupling between disciplines**
  - Aerodynamic simulation
  - Structural simulation
  - Weight + deformation
  - Load distribution

- **b) Two-level optimization strategy**
  - Aerodynamic simulation
  - Structural optimization
  - Aircraft weight can be constrained during the upper level aerodynamic optimization to be lesser than or equal to its optimum value from the structural optimization.
6.4.2 Typical Automotive Example

The coupling between disciplines is weaker and implementing MDO is hence more straightforward in the automotive industry compared to the aerospace industry. A typical example of an MDO application within the automotive industry has been to minimize the mass of the body-in-white (BIW) under constraints of crashworthiness and NVH (noise, vibration, and harshness), but other examples exist.

Crashworthiness simulations are computationally expensive and it is only in recent years, with increased availability of affordable high performance computing (HPC) systems and the possibility of parallel computing, that it has been feasible to include full vehicle crashworthiness simulations in MDO studies. Although the computers have become much faster over the years, the level of detail of the models has also increased, leaving a crashworthiness simulation to still run for several hours. It is therefore important to be able to run several simulations in parallel in order for the MDO process to become useful during product development.

The early examples of NVH and crashworthiness MDO only used a limited number of design variables and loadcases together with polynomial metamodels and gradient-based optimization algorithms. The more recent examples include more design variables and loadcases and use more complex metamodels and advanced optimization algorithms, see Table 6.1 for a summary of references and Figure 6.6 for a typical application example. These methods are now used by the industry but have not yet been fully implemented within the product development process. According to Duddeck (2008), the difficulties so far have mainly been related to the overall computational time and the accuracy of the metamodels.

Table 6.1 Examples of NVH and crashworthiness MDO studies, ordered by the number of loadcases included in the optimization.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Load-cases</th>
<th>Number of variables</th>
<th>Modelsizes, elements x 1000</th>
<th>Metamodel</th>
<th>Optimization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craig et al. (2002)</td>
<td>1 NVH</td>
<td>7</td>
<td>~18 (~30)</td>
<td>Polynomial</td>
<td>Gradient-based</td>
</tr>
<tr>
<td></td>
<td>1 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sobieszczanski-Sobieski et al. (2001)</td>
<td>2 NVH</td>
<td>39</td>
<td>~68 (~120)</td>
<td>Polynomial</td>
<td>Gradient-based</td>
</tr>
<tr>
<td></td>
<td>1 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yang et al. (2001)</td>
<td>2 NVH</td>
<td>44</td>
<td>~68 (~100 – 120)</td>
<td>Polynomial</td>
<td>Gradient-based</td>
</tr>
<tr>
<td></td>
<td>3 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kodiyalam et al. (2004)</td>
<td>2 NVH</td>
<td>49</td>
<td>~68 (~100 – 120)</td>
<td>Kriging</td>
<td>Gradient-based</td>
</tr>
<tr>
<td></td>
<td>4 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hoppe et al. (2005)</td>
<td>2 NVH</td>
<td>96</td>
<td>~280 (~1100)</td>
<td>None</td>
<td>Evolutionary</td>
</tr>
<tr>
<td></td>
<td>5 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duddeck (2008)</td>
<td>2 NVH</td>
<td>136</td>
<td>~280 (~1100)</td>
<td>None</td>
<td>Evolutionary</td>
</tr>
<tr>
<td></td>
<td>5 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sheldon et al. (2011)</td>
<td>1 NVH</td>
<td>35</td>
<td>RBF net</td>
<td>Hybrid</td>
<td>simulated annealing</td>
</tr>
<tr>
<td></td>
<td>6 crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.4.3 Experiences from the Automotive Industry

The simplest form of MDO is size optimization, where the influence of different thicknesses on the responses is studied. It is thus mainly useful in later stages of the product development when the design is relatively fixed. The typical automotive MDO application example of weight optimization of BIW with constraints on NVH and crashworthiness performance is an example of such an optimization. Optimization in the early phases of the product development, where it in fact might have the largest potential, is more related to geometrical changes, i.e. shape optimization. These studies require parametric models and a pre-processing step before the analyses are performed. The geometry changes can either be done by simple morphing, i.e. altering the existing mesh, see Korbetis and Siskos (2009), or by modifying the geometry and then re-meshing the parts, see Xu (2007).
The simple single-objective deterministic optimization has been extended to multiple objectives by Su et al. (2011) and reliability-based design optimization by Yang et al. (2002). More evaluations are needed in these more complex optimization studies. The complexity of optimization problems is generally increased by an increased number of design variables and loadcases, as well as the inclusion of multiple objectives and robustness considerations. In real applications, both the time and available cpu resources are limited, and there will always be a balance between what can be done within the product development process and what is desired.

Since the industry is working in a competitive environment with strict schedules, there is often limited time for collecting information and presenting recorded achievements, e.g. in technical journals. At the same time, much of the work done within the industry is also regarded as confidential, and it is therefore not obvious for people outside a company to get information concerning the state-of-the-art methods and processes. For MDO methods to be widely used within a company on a daily basis, they need to be implemented in commercially available software. This means that the common practice within the industry is to some extent mirrored by the success-stories presented by the different software vendors or by users at these vendors' users' meetings. Two of the most well-known software applications for process integration and design optimization (PIDO) are iSIGHT by Simulia and modeFRONTIER by ESTECO. However, many more commercial MDO tools exist and the ones listed by Wikipedia (2011) are presented in Table 6.2. Browsing through the websites of MDO software, it is evident that true multidisciplinary design optimization within the automotive industry, other than studies similar to the NVH/crashworthiness example mentioned earlier, is rather rare. On the other hand, there are some examples presented of optimization with multiple loadcases within the same discipline, see e.g. Müllerschön et al. (2009) for a crashworthiness example and Burnham (2007) for a vehicle dynamics example.

<table>
<thead>
<tr>
<th>Software</th>
<th>Vendor</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boss quattro</td>
<td>Samtech</td>
<td><a href="http://www.samtech.com">www.samtech.com</a></td>
</tr>
<tr>
<td>FEMtools Optimization</td>
<td>Dynamic Design Solutions</td>
<td><a href="http://www.femtools.com">www.femtools.com</a></td>
</tr>
<tr>
<td>HEEDS</td>
<td>Red Cedar Technology</td>
<td><a href="http://www.redcedartech.com">www.redcedartech.com</a></td>
</tr>
<tr>
<td>HyperStudy</td>
<td>Altair Engineering</td>
<td><a href="http://www.altairhyperworks.com">www.altairhyperworks.com</a></td>
</tr>
<tr>
<td>IOSO</td>
<td>Sigma Technology</td>
<td><a href="http://www.iosotech.com">www.iosotech.com</a></td>
</tr>
<tr>
<td>iSIGHT</td>
<td>Dassault Systèmes Simulia</td>
<td><a href="http://www.simulia.com">www.simulia.com</a></td>
</tr>
<tr>
<td>LS-Opt Correlation</td>
<td>Livermore Software Technology</td>
<td><a href="http://www.lsoptsupport.com">www.lsoptsupport.com</a></td>
</tr>
<tr>
<td>modeFRONTIER</td>
<td>Esteco</td>
<td><a href="http://www.esteco.com">www.esteco.com</a></td>
</tr>
<tr>
<td>Nexus</td>
<td>iChrome</td>
<td><a href="http://www.ichrome.eu">www.ichrome.eu</a></td>
</tr>
<tr>
<td>Optimus</td>
<td>Noesis</td>
<td><a href="http://www.noesissolutions.com">www.noesissolutions.com</a></td>
</tr>
<tr>
<td>OptiY</td>
<td>OptiY e.K.</td>
<td><a href="http://www.optiy.eu">www.optiy.eu</a></td>
</tr>
<tr>
<td>PHX ModelCenter</td>
<td>Phoenix Integration</td>
<td><a href="http://www.phoenix-int.com">www.phoenix-int.com</a></td>
</tr>
<tr>
<td>SmartDO</td>
<td>FEA-Opt Technology</td>
<td><a href="http://www.fea-optimization.com">www.fea-optimization.com</a></td>
</tr>
<tr>
<td>VisualDOC</td>
<td>Vanderplats Research and Development</td>
<td><a href="http://www.vrand.com">www.vrand.com</a></td>
</tr>
</tbody>
</table>

Other sources of information regarding what is achieved within different companies in the automotive industry are papers presented within the Society of Automotive Engineers (SAE). A couple of interesting applications of MDO methodology used during the development of vehicle components can, for instance, be found. One example is related to optimization of engine mounts.
with respect to NVH, ride comfort, and driveability, see Olsson et al. (2011). Another example is the optimization of an engine maniverter (a combination of exhaust manifold and catalytic converter) with respect to engine and catalyst performance, first natural frequency, and cost, presented by Usan (2006). In both these examples, the presented MDO processes are found to reduce the resources and the development time required compared to more conventional approaches. It was also found that additional benefits, such as less required prototype material and increased product innovation, could be achieved.

### 6.4.4 Multi-Level Optimization Methods for Automotive Applications

All the examples of MDO applications presented so far have been executed using single-level optimization methods. Successful applications of multi-level optimization methods within the automotive industry are few. It has even been concluded by Song and Park (2006) that most methods are developed for strongly coupled systems, as wing design in the aerospace industry, and therefore are too complicated to be applied to real complex structures within the automotive industry when coupling between disciplines is present only through shared variables. For these cases, simpler optimization methods are instead claimed to be more appropriate. Such methods have, for example, successfully been applied to the weight optimization of an automotive door by Song and Park (2006). However, some applications of the more well-known multilevel optimization methods described in Section 5.3 have been presented also for automotive applications by researchers from different universities.

Analytical Target Cascading (ATC) was developed in cooperation with the automotive industry as a systematic way of propagating the desired top-level system design targets to appropriate specifications for subsystems and components in a consistent and efficient manner, see Section 5.3.4. The ATC method can also be used for optimization studies in which the system design target is set to zero when the aim is to minimize the objective. The method has successfully been applied to different automotive applications by researchers at University of Michigan, where the method was first developed. It has for example been applied to the optimal design of the powertrain and suspension of a heavy truck which included novel technologies, as described by Kokkolaras et al. (2004). A number of different concepts were considered and the objective was to improve fuel efficiency and performance. This extensive study demonstrated that ATC is useful in determining system design specifications that result in overall system optimality and consistency. A similar study has also been applied to the redesign of an existing truck by Kim et al. (2002). It was concluded that the main benefit of ATC can be the reduction in vehicle design cycle time and the increased likelihood of physical prototype matching, which would avoid costly design iterations late in the development process. The method of ATC has also been extended to applications of product family design optimization by Kokkolaras et al. (2002) and probabilistic-based design optimization by Kokkolaras et al. (2006) and Liu et al. (2006).

One reason why multi-level optimization methods rarely are used in automotive development is the labour cost associated with creating a suitable partition of the system and the knowledge required to select and implement a proper coordination strategy. Another reason is the apparent additional computational cost incurred by coordination between subspaces. However, it has been shown by Guarneri et al. (2011) that for special cases, ATC and a modified SQP algorithm can address the latter issue. In an optimization of comfort and road holding for an automotive suspension system, they found that the computational cost was only slightly higher compared to the same optimization done
with a single-level method. In this study, the trade-off between the objectives was evaluated by finding the Pareto optimal set.

The use of the MDO methods developed at research departments in collaboration with the aerospace industry, such as CSSO, BLISSL, and CO, seems to be almost non-existent within the automotive industry. Instead, these methods are studied for aerospace applications. Researches in Canada have compared MDO methods for a conceptual design of a supersonic business jet involving four different disciplines/subsystems, see Chen et al. (2002) and Perez and Behdinan (2004). The idea was to maximize the flight range subject to individual disciplinary constraints from the coupled disciplinary systems representing structures, aerodynamic, propulsion, and performance. The problem involved 10 design variables and 9 coupling variables and the subsystem evaluations were done with empirical analytical expressions representative for an aircraft conceptual design. It was concluded that CO is suitable for systems with loosely coupled disciplines while BLISSL is better for highly coupled systems and that CSSO is efficient only for systems with few disciplines. It was also found that the multi-level optimization methods, although more computationally expensive, gave better results than the single-level methods MDF and IDF. The difficulty in finding general-purpose methods was demonstrated by the fact that even for this particular example, the two groups advocated different methods. Chen et al. (2002) favoured BLISSL while Perez and Behdinan (2004) favoured CO.
7 Conclusions

When implementing multidisciplinary design optimization in the automotive industry, there are several questions related to the subjects studied in this report that need to be answered. Simulations associated to structural applications within the automotive industry are computationally expensive, which motivate the use of metamodel-based design optimization. Section 7.1 concludes which types of design of experiments, metamodels, and optimization methods that could be appropriate for automotive applications. Further on, an MDO method must be chosen. It must be determined whether a single- or multi-level method should be used. Using a multi-level method increases the complexity of the optimization process considerably compared to using a single-level method. Therefore, in order to motivate the use of a multi-level method, the benefits must be greater than the cost. In Section 7.2, a choice of MDO method for automotive applications is proposed and motivated.

7.1 Metamodel-Based Design Optimization for Automotive Applications

Many of the disciplines within the automotive industry that are relevant to include in an MDO study rely on detailed simulation models that are computationally costly to evaluate. To relieve some of the computational burden, metamodels can be used for optimization studies. This is called metamodel-based design optimization and is described in Chapter 4.

Historically, simple polynomial metamodels have been very popular. However, some of the disciplines commonly included in automotive MDO studies have complex responses, and simple polynomial metamodels are then only valid in a small portion of the design space. The trend has therefore gone towards the use of more advanced metamodels that are better suited to capture complex responses in the complete design space. For deterministic simulations, it might seem natural to use interpolating metamodels. However, for non-linear dynamic simulations with complex responses and other situations where numerical noise might be present, it may be advantageous to use approximating metamodels instead. It should be noted that no metamodel type is the best for all problem categories. Instead, the most well suited metamodel type depends on the nature and complexity of the problem. Radial basis function neural networks are often a good choice due to their accuracy, especially for small fitting sets, and since they are relatively fast to build. However, the accuracy of advanced metamodels strongly depends on the settings, and a well-tuned model is not always easy to obtain. There is also a constant development in the field of metamodels, and the numerous software available complicate matters further.

For advanced metamodels that can capture a complex response in many design variables over a large design space, it is generally recommended to use some kind of space-filling design of experiment to obtain the database of variable settings and corresponding responses needed to build the metamodel. There exist many different space-filling DOEs, but an improved Latin hypercube sampling or a DOE based on low-discrepancy sequences is often a good choice. In general, the more samples that could be afforded in the DOE, the better it is. At least three to four times the number of design variables is recommended initially.

In order to reduce the number of simulations needed to build the metamodels, it is important to be careful in selecting only design variables that are important for the studied problem. The simplest way to find these variables is to use a one-factor-at-a-time plan. By changing one variable at a time while keeping the other variables constant, it is easy to identify the ones that contribute the most to
Conclusions

changes in the responses. This is an inexpensive method, but with the drawback that no interaction effects between variables can be estimated. If numerical noise is present, the results might not be as easy to interpret as desired, so engineering judgement and knowledge about the system is vital. It is most effective to select the design variables solely based on previous knowledge about the system, i.e. without performing any screening simulations. However, care needs to be taken so that important variables are not omitted. The screening process can exhaust a considerable part of the available simulation budget if many variables need to be considered. It might therefore be efficient to select most of the design variables based on knowledge, and then also include some additional variables that have an uncertain effect on the system. If more simulations can be afforded in the screening process, methods such as analysis of variance for polynomial metamodels and global sensitivity analysis for arbitrary metamodels are good tools to identify the most important design variables. If these methods are used in conjunction with linear metamodels fitted to the smallest possible dataset, they are as inexpensive as the one-factor-at-a-time approach described above.

If the available computing time is significantly restricted, it is probably best to use a single stage strategy, i.e. to use a large part of the available simulation budget to establish the database needed to build the metamodels. The obtained metamodels are then hopefully accurate enough. However, the accuracy of the metamodels needs to be checked carefully before use, and the metamodels should be refined if they are found to be inadequate. In case of a larger simulation budget, it could be beneficial to start with a small DOE and use a sequential strategy in which the metamodels are iteratively refined. When many trade-off solutions in the form of a Pareto front are desired, the sequential strategy is not recommended to be combined with domain reduction in order to avoid varying accuracy of the Pareto front.

It is a complicated task to check the accuracy of a metamodel. Standard error measures for a metamodel only give information on how well the model describes the fitting data. Interpolating metamodels will therefore report no errors and overfitted models will also give deceivingly low error measures. More important is how well the model can predict results in unknown points, which can be checked using a separate validation set. The results obtained from the metamodels are then compared with the results from the validation set. In practical situations, the data is often limited and all sampling points are needed to fit the metamodels. Cross validation, where the same set of data is used for fitting and validating the model, is therefore often a convenient approach. In principle, a small portion of the available data is filtered out and the metamodel is fitted to the remaining set. The omitted points are then used as a validation set. The process is repeated and requires the metamodel to be fitted several times, which can be time-consuming. One version of CV is the so called leave-one-out CV in which only one data point is omitted each time. Despite the computational expense and the fact that CV might be misleading for metamodels fitted to small datasets, it could be the only practical way to validate a metamodel. For some metamodels, such as polynomial and RBF models, leave-one out CV errors can inexpensively be computed from a single metamodel fitted to all points. Consequently, this is an efficient method to estimate the prediction errors for these metamodels.

The risk of overfitting is increased if the metamodel is overly complex. There are some error measures, such as generalized cross validation, that take both the residuals and the model complexity into account. These types of error measures can be useful when trying to find the best possible version of a complex metamodel.
Global optimization algorithms are more likely to find the global optimum compared to local optimization methods and should therefore be preferred. These algorithms often need many evaluations to converge, but this is normally not an issue since evaluations using metamodels are very fast compared to evaluations using detailed simulation models. Simulated annealing and evolutionary algorithms such as genetic algorithms are examples of global optimization methods that can be good alternatives. In cases of multi-objective optimization, it is often hard to get information about the relative importance of the different objectives in advance. It is then desirable to find several different trade-off solutions that can be compared so that the decision maker can select the solution that fits his or her preferences the best. Evolutionary algorithms are population-based and can therefore obtain many trade-off solutions in one optimization run. Algorithms recognised to work well are those based on the non-dominated sorting genetic algorithm, NSGA-II.

It should be noted that the use of metamodels introduces an additional source of error. After finding the optimum solution, or other interesting solutions, it is therefore always necessary to check the obtained results with results from the detailed models. Large discrepancies between results obtained using metamodels and results from the detailed models indicate metamodels with insufficient accuracy. Adding extra points to the DOE and rebuilding the metamodels might solve this issue. Small but non-acceptable deviations can often be taken care of by manually changing one or a few of the design variables based on sensitivity information from the metamodels.

7.2 Multidisciplinary Design Optimization Methods for Automotive Applications

Large-scale MDO problems need to be decomposed, which was motivated by the need for concurrency and autonomy in Section 5.1. Concurrency concerns the possibility to parallelize work, both when it comes to human and to computational aspects. Autonomy refers to giving the individual groups freedom to make their own design decisions as well as to govern methods and tools. Single-level methods with distributed analyses will parallelize work and give the groups autonomy in the sense of governing methods and tools. Using these methods, however, the groups will not make design decisions during the optimization process. To solve this issue, multi-level methods were introduced, where the groups are given the possibility to be involved in design decisions on the local and possibly also on the global level.

When using MDO within the automotive industry, the possibility of using metamodels in some, or all, of the disciplines is essential. This is motivated by the high computational cost of the detailed simulations in many of the disciplines. Both single- and multi-level methods offer the possibility to include metamodels. When using metamodel-based optimization instead of direct optimization, the groups can automatically work in parallel. The metamodels and the required simulations can be created concurrently before the optimization process starts. However, some of the groups might have simulations with low computational cost. Direct optimization can be preferred in these groups since it can be unnecessarily complicated to make metamodels of the inexpensive simulation models.

The existence of coupling variables complicates the MDO methodology considerably. When comparing the automotive industry with the aerospace industry in Section 6.3, it was made clear that coupling variables are much more important for aerospace than for automotive applications. To neglect the coupling variables for aerospace applications is a very crude approximation, while doing the same thing for automotive applications might be reasonable. In the discussion that follows
concerning the choice between different methods, it will be assumed that coupling variables do not exist.

The most straightforward and simple approach to solve an MDO problem in the automotive industry is to use a single-level method in combination with metamodels. Each design group can work autonomously, using its preferred methods and tools, when performing the simulations and building the metamodels. The groups will consequently be able to work in parallel. The metamodels are used by a central optimizer that performs the optimization. A drawback is that all design decisions will be taken on a central level, and the groups are therefore not autonomous in this sense. However, this drawback could be relieved by involving the different groups in the setup of the optimization problem and the assessment of the results. Another disadvantage of using a single-level optimization method is that individual groups cannot govern methods and tools in the optimization process. Groups that already have a defined procedure for performing optimization must abandon it and let the central optimizer control the whole optimization process. When it comes to the choice of single-level method, the multidisciplinary feasible and individual discipline feasible methods presented in Section 5.2 coincide when there are no coupling variables.

The main motivation for using multi-level methods is to gain autonomy of design decisions in the different groups. These methods also enable the groups to govern their own optimization procedure, including the choice of optimization algorithm and software. The groups can then work fully autonomously and in parallel. A drawback of multi-level methods is their complexity. The methods are often very complicated to implement and can also be complex to use. The multi-level problem can become less transparent than the corresponding single-level problem. An example is when the local objective functions do not mirror the global objective function, which makes it difficult for the individual groups to grasp the global goal in the optimization process. Another drawback is the increased cost associated with multi-level methods. Multi-level methods often require more computational resources and involve more people than single-level methods.

Different aspects need to be considered when determining if a multi-level method is suited for an automotive application. The method in question needs to be simple to grasp for the people working with it, meaning that the formulation should be sufficiently transparent. This can for example mean that the global objective is mirrored by the subspace optimization formulations to create an awareness of the global goal in the groups. The method should preferably also be relatively simple to implement. It should be stable, meaning that it always finds a solution if a solution exists. Moreover, its computational cost should be acceptable, implying moderate communication requirements between system and subsystem levels and reasonable convergence speed. Finally, the method should be efficient for non-coupled problems.

None of the studied multi-level methods fulfil all these requirements. Concurrent subspace optimization has diverged into many different variants. The original version has several shortcomings, as shown in Section 5.3.1. In the versions of CSSO that are based on the work by Renaud and Gabriele (1991), all variables are dealt with at the system level. The different groups will therefore not be autonomous when it comes to design decisions, which was the main objective of using a multi-level method. Bilevel integrated system synthesis was specifically developed for the aerospace industry where coupled variables need to be considered. The BLISS 2000 formulation depends on the existence of coupling variables, and without these the method is no longer
meaningful. This discussion consequently excludes the CSSO and BLISS methods for automotive applications.

Collaborative optimization is an attractive formulation, but it is associated with a lack of transparency for the groups. The purpose of the subspace optimization problems is to minimize the deviation from a consistent design, and it is only implicitly through the system optimizer that the global objective is minimized. Further on, it suffers from a number of numerical problems when used in combination with gradient-based optimization algorithms, as described in Section 5.3.3. However, CO could be interesting if used together with other types of optimization algorithms. A number of improvements to the CO method are presented in the literature. Collaborative optimization using surrogate models overcomes some of the numerical difficulties associated with the original formulation. However, instead of approaching the source of the numerical problems in the formulation itself, an approximation of the problem makes it numerically easier to solve. This method is therefore not considered to be of interest. Modified collaborative optimization approaches some of the issues in the original method, but requires a sequence of ill-conditioned problems to be solved. Enhanced collaborative optimization resolves the numerical problems associated with CO, but is complicated to implement and use. In summary, collaborative optimization in combination with a non-gradient-based optimization algorithm, modified collaborative optimization, and enhanced collaborative optimization are all possible choices of multi-level methods for automotive applications in general. However, none of these methods fulfil all requirements that are wished for in a multi-level method.

Analytical target cascading was developed to propagate targets during automotive development, but can be used as an MDO tool. The original method requires a hierarchical model structure, and is primarily interesting to use for optimization within the automotive industry if this structure already exists. It is not an effective alternative for a company that does not employ hierarchical model structures. A summary of benefits, drawbacks, and possible MDO methods for automotive applications is summarized in Table 7.1.

Table 7.1 Benefits, drawbacks, and possible MDO methods for automotive applications using metamodels.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Single-level Methods</th>
<th>Multi-Level Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomy in methods and tools for simulation</td>
<td>Autonomy in methods and tools for simulation and optimization</td>
<td></td>
</tr>
<tr>
<td>Parallelization</td>
<td>Autonomy in design decisions</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>Parallelization</td>
<td></td>
</tr>
<tr>
<td>Drawbacks</td>
<td>No autonomy in methods and tools for optimization</td>
<td>Complex</td>
</tr>
<tr>
<td>No autonomy in design decisions</td>
<td>Expensive</td>
<td></td>
</tr>
<tr>
<td>Possible Methods</td>
<td>MDF (equivalent to IDF)</td>
<td>CO, MCO, or ECO</td>
</tr>
<tr>
<td></td>
<td>ATC</td>
<td></td>
</tr>
</tbody>
</table>

The purpose of the authors’ research is to develop an efficient MDO methodology to be used for automotive applications. In this report, a number of multi-level methods have been studied, but it is concluded that the cost of implementing and using these methods is greater than the benefits. For multi-level methods to have greater potential than a single-level method when used in combination with metamodels, research is needed to find new methods or develop the existing ones. The authors must therefore choose whether to focus their research on questions related to implementing single-
level methods and/or focus on improving multi-level methods. It is of interest also to consider other questions, such as multi-objective and probabilistic optimization, and the implemented method(s) should therefore be able to address these issues. Using single-level methods, it is straightforward to include the state-of-the-art within these areas in the optimization process.

The next step in the authors’ research is decided to involve implementation of a single-level optimization method. As metamodels are used, different groups will be able to work autonomously when it comes to methods and tools for simulation and also be able to work in parallel. The groups can be involved in the setup of the optimization problem and in the assessment of the results, which makes it possible for them to indirectly participate in design decisions even though a single-level method is employed.
References

http://en.wikipedia.org/wiki/Multidisciplinary_design_optimization


References


