Design of Experiments: The D-Optimal Approach and Its Implementation As a Computer Algorithm

Fabian Triefenbach

15th January 2008

Bachelor's Thesis in Information and Communication Technology
15 ECTS

Supervisor at UmU: Jürgen Börstler
Supervisor at FH-SWF: Thomas Stehling
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

South Westphalia University of Applied Sciences
Department of Engineering and Business Sciences
D-59872 MESCHEDE
GERMANY

Abstract

Motivation: The use of experiments in an effective way to analyze and optimize a given process is a problem statement which should be paid more attention to in the engineering community. Frequently, experiments are first performed and the measured data is analyzed afterwards. In contrast to this, statistical methods, like the concept of Design of Experiments, should be used to plan experimental designs and perform sets of well selected experiments to get the most informative combination out of the given factors. **Problem Statement:** The Swedish company Umetrics AB is developing special software tools which deal with this field of statistical experimental design. Inside the software MODDE a feature for the creation of a special design type, the D-optimal design, is given. The aim of the present thesis is the reimplementation of this program unit with improvements in the quality of the generated designs and the computational effort. **Approach:** D-optimal designs are based on a computer-aided exchange procedure which creates the optimal set of experiments. An intensive literature study brings forth six different approaches to handle this exchange with a computer algorithm. In detail a comparison and evaluation of the Fedorov procedure, the modified Fedorov procedure, the DETMAX algorithm, the k-exchange algorithm, and the kl-exchange algorithm is performed. Based on this, a new algorithm, the modified kl-exchange algorithm, is developed and used for the implementation in C++. **Results:** The outcome of the present thesis is an implementation of the four most advisable algorithms. The Fedorov and normal Fedorov algorithm generate the best designs considering the D-optimality of the designs but are very time-consuming for big data sets. As an alternative the k-exchange algorithm and a modified version of the kl-exchange algorithm are implemented to create comparable designs with a much higher computational efficiency. **Conclusion:** The new implementation of the D-optimal process creates designs which are at least comparable to the old approach considering the quality or D-optimality of the designs. In addition to this, the computational efficiency, especially regarding big data sets with 10 and more factors, is higher. The selection of the adequate algorithm for the given problem formulation has to be performed by the user. As a future work, this process should be carried out to an automatic procedure based on a complexity analysis of the four implemented alternatives.

keywords: *D-Optimal Designs, Design of Experiments, Exchange Algorithms, (Modified) Fedorov Algorithm, k-Exchange Algorithm, (Modified) kl-Exchange Algorithm, Mitchell's DETMAX Algorithm, Bayesian Modification, Inclusions, Mixture Designs, Sequential Start Design, Qualitative Factors.*

Acknowledgement

Overall, I would like to thank my German institute, the South Westphalia University of Applied Science in Meschede, and my host university in Sweden, the Umeå University, for making this thesis possible. Especially my both supervisors, Prof. Börstler and Prof. Stehling, helped to arrange an agreement and gave me the chance to write my thesis abroad. With reference to the received assistance during my thesis, I appreciate the encouragement of both supervisors. In addition to this, I would like to thank my examiner Per Lindström.

The second group of people I would like to give thanks to is the whole staff of the company Umetrics AB. My advisors, Åke Nordahl and Joakim Sundström, have always been great contact persons and were obliging in any situation. Also the other staff members helped me with advices and at least improved my Swedish skills. Therefore, I would like to thank Jonas, Conny, Erik, Lennart, Anders, Jerker and all the rest of the personnel at the Umeå headquarter. Special thanks are targeted at Ing-Marie Olsson for her previous work and the backup during the development process. *Tack ska ni ha.*

Dedication

I dedicate this thesis to my family and my girlfriend Christina, who gave me such a strong support during the whole development process. Thank you.

Contents

List of Figures

List of Tables

List of Algorithms

List of C++ Listings

Chapter 1

Introduction

1.1 Thesis Outline

The present thesis depicts the development process of a C++ computer algorithm to generate D-optimal designs in the field of Design of Experiments (DoE).

The report starts with the description of the basic principles of DoE and explains the elementary methods and definitions used in this field. In general, DoE is a concept that uses a desired set of experiments to optimize or investigate a studied object. After this brief overview about the concept, a special design type, the D-optimal design is examined in the following chapter. D-Optimality characterizes the selection of a special set of experiments which fulfills a given criterion. From the ground up, the whole concept of D-optimal designs and their application is shown. Starting with the algebraic and geometric approach, more statistical and methodical details are illustrated afterwards.

The generation of the D-optimal design is accomplished by the use of an exchange algorithm. Chapter 3 of this report contains a review and comparison of existing algorithms which are presented in diverse publications. The comparison considers Mitchell's (1974) DETMAX algorithm, the Fedorov (1972) procedure, the modified Fedorov procedure by Cook & Nachtsheim (1980), the *k*-exchange by Johnson & Nachtsheim (1983) and the *kl*-exchange algorithm (Atkinson & Donev 1989). In addition to this, a modified version of the *kl*-exchange is developed and included in the investigation. The algorithms are explained in detail and evaluated depending on the computational efficiency and the quality of the generated designs. In order to make the created designs independent on an assumed model, a Bayesian modification of the process is introduced and exemplified in chapter 2.

Chapter 4 gives detailed information about the implementation of the D-optimal selection process. The work described in this chapter is done at the Umetrics AB office in Umeå, Sweden, and will be implemented in a future release of the MODDE software. Starting with an explicitly definition of the requirements, the program flow and integration of the new source code is explained. After this, a description of the four implemented algorithm is given and clarified with extracts from the C++ source code. The chapter ends with a description of the main features of the implementation and explains the realization in more detail.

The thesis ends with a summary of the work and analyses the achievements made by the final implementation with a conclusion. After this discusion of the result, a prospect of the future work based on the present thesis is given.

1.2 Design of Experiments

1.2.1 Preface

Experiments are used in nearly every area to solve problems in an effective way. These experimentations can be found both in daily life and in advanced scientific areas. A presentable example for an ordinary experiment is the baking of a cake, where the selection and the amount of the ingredients influence the taste. In this basic example, the ingredients are flour, butter, eggs, sugar and baking powder. The aim of the experimenter is to create the cake with the best taste. A possible outcome of the first baking process is a cake that is too dry. As a result, the experimenter reduces the amount of flour or enhances the amount of butter. In order to bake the best cake, the recipe should be changed for every new cake. As a result of this process, we get a set of different recipes which lead to different tasting cakes.

Similar observations are used in the scientific area to find significant information about a studied object. An object can be, e.g., a chemical mixture, a mechanical component or every other product. A good example in this field is the development of a special oil which has to be observed under different conditions, like high pressure or low temperature. Finding the optimal mixture which has the best performance in different situations, is one of the most important stages during such a development process.

1.2.2 Experiments

In general, an experiment is an observation which leads to characteristic information about a studied object. The classical purpose for such an observation is a hypothesis that has to be veryfied or falsified with an investigation. In this classical approach, the experimental setup is chosen for the specified problem statement and the experimenter tests if the hypothesis is true or false (Trochim 2006).

In the present thesis, we deal with another use of experiments. With the concept of Design of Experiments (DoE) we use a set of well selected experiments which has to be performed by the experimenter. The aim of this so-called *design* is to optimize a process or system by performing each experiment and to draw conclusions about the significant behavior of the studied object from the results of the experiments. The examples in the preface follow this basic idea and show two useful situations where DoE is applicable. Considering the costs for a single experiment, minimizing the amount of performed experiments is always an aim. With DoE, this number is kept as low as possible and the most informative combination of the factors is chosen (Eriksson et al. 2000). Hence, DoE is an effective and economical solution.

The term *experiment* is used in DoE even if we do not have a hypothesis to test. To avoid misunderstandings, we strictly have to differentiate between the notation of DoE and normal experimental design. The following section of this introduction deals with the general approach of DoE and gives definitions used in the field.

1.2.3 Factors & Responses

We always have two types of variables when we perform experiments in the field of DoE, responses and factors. The response gives us information about the investigated system, and the factors are used to manipulate it. Normal factors can be set to two or more values and have a defined range. The factors can be divided into three groups, depending on different criteria.

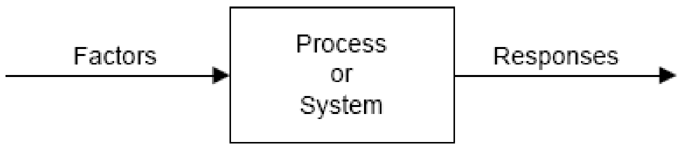


Figure 1.1: Factors & Responses: *A process or a system can be manipulated by one or more different input factors. These modifications takes effect on the responses and can be measured.*

Controllable & Uncontrollable Factors

One way to differentiate factors is to divide them into controllable and uncontrollable factors. The controllable factors are normal process factors that are easy to monitor and investigate. The experimenter can observe them and has the opportunity to change them. By contrast, an uncontrolled factor is hard to regulate because it is mostly a disturbance value or an external influence. Uncontrolled factors can have a high impact on the response and therefore should always be considered during the experiments (Eriksson et al. 2000, Wu & Hamada 2000). A good example of a controlled factor is the temperature inside a closed test arrangement. In contrast to this, the outside temperature is an example of an uncontrolled factor which can not be influenced.

Quantitative & Qualitative Factors

Another method to categorize factors in the broader sense is to distinguish between quantitative and qualitative factors. The values of a quantitative factor have a given range and a continuous scale, whereas qualitative factors have only distinct values (Wu & Hamada 2000, Montgomery 1991). To use the earlier example, the temperature can be seen as a quantitative value. An example of a qualitative factor is the differentiation between variant types of oil, developed by different suppliers.

Process & Mixture Factors

The third group consists of process and mixture factors. Process factors can be changed independently and do not influence each other. They are normally expressed by an amount or level and can be seen as regular factors. Mixture factors stand for the amounts of ingredients in a mixture. They all are part of a formulation and add up to a value of 100%. A mixture factor cannot be changed independently, so special designs are needed to deal with this type of factors (Eriksson et al. 2000, Montgomery 1991).

Responses

As described above, a response is the general condition of a studied system during the change of the factors. It is possible and often useful to measure multiple responses that react differently on the factor adjustments. A response can either be a continuous or a discrete value (Wu & Hamada 2000). A discrete scale, e.g., can be an ordinal measurement with three values (good, OK, bad). These values are hard to process, so it is always recommended to use a continuous scale if possible.

1.2.4 Basic Principles of DoE

To understand the basic idea of DoE, we should first examine how experimental design was done traditionally. As Eriksson et al. (2000, p. 2) describe, the intuitive approach is to 'chang[e] the value of one separate factor at a time until no further improvement is accomplished'. Figure 1.2 illustrates how difficult finding the optimum can be with this co-called COST approach (*Change Only* one *Separate* factor at a *Time*). The experimenter does not know at which value the changing of the factor x_1 should be stopped because a further improvement can not be observed. But the exact value is very important considering it in combination with the later adjusted factor x_2.

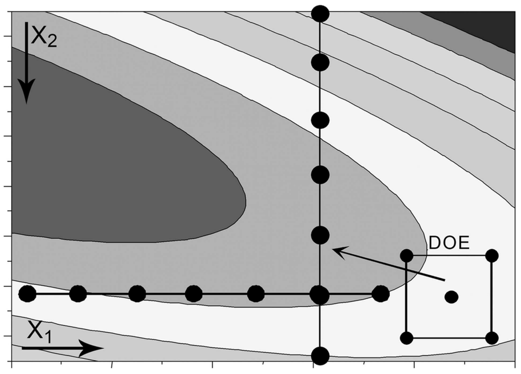


Figure 1.2: COST Approach & DoE, based on Eriksson et al. (2000, p. 3, fig. 0.1): *Changing all factors simultaneously, as shown in the lower right corner, gives better information about the optimum than the COST approach where all factors are changed successively.*

The given situation from figure 1.2 can also be investigated with the use of DoE. In this case, we create a special set of experiments around a so-called center-point. As shown in the lower right corner of the diagram, we use a uniform set of experiments which allows obtaining a direction for a better result. These basic examples clearly show the advantages of changing all relevant factors at the same time and consequently show the importance of DoE (Wu & Hamada 2000, Eriksson et al. 2000).

As shown above, the basic concept of DoE is to arrange a symmetrical distribution of experiments around a center-point. With a given range for all input factors, the calculation of this point is easy. Figure 1.3 shows a simple example with the factors x_1 (200 to 400), x_2 (50 to 100) and x_3 (50 to 100). The calculated center-point in the middle of the cubic pattern has the coordinates 300/75/75.

1.2.5 Model Concept

The base for DoE is an approximation of reality with the help of a mathematical model. The important aspects of the investigated system are represented by the use of fac-

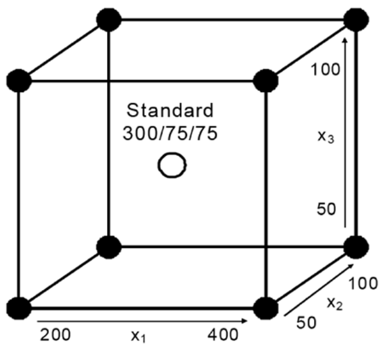


Figure 1.3: Symmetrical Distribution of Experiments, based on Eriksson et al. (2000, p. 8, fig. 1.1): *The three factors x_1, x_2 and x_3 are, depending on their range, ordered in a cubic pattern around the center-point experiment (standard).*

tors and responses. A model is never 100% right, but simply helps to transport the complexity of the reality into an equation which is easy to handle (Eriksson et al. 2000).

The simplest one is a linear model where the g factors $x_1, x_2, \ldots, x_g$ influence the response y in the following way:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_g + \epsilon. \tag{1.1}$$

In this case $\beta_1, \beta_2, \ldots, \beta_g$ represents the regression coefficients and 'ϵ is the random part of the model which is assumed to be normally disturbed with mean 0 and variance σ^2' (Wu & Hamada 2000, p. 12). We can extend this equation to one with N multiple responses and arrive at:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_g x_{ig} + \epsilon_i, \qquad i = 1, \ldots, N, \tag{1.2}$$

where y_i stands for the ith response with the factors $x_{i1}, x_{i2}, \ldots, x_{ig}$. All of the N equations can be written in matrix notation as:

$$Y = X\beta + \epsilon, \tag{1.3}$$

where the $N \times (g+1)$ model matrix X contains all factors for the responses and Y and ϵ are $N \times 1$ vectors. The regression coefficients β are the unknown parameter in the model (Wu & Hamada 2000).

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1g} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Ng} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_N \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_N \end{bmatrix} \tag{1.4}$$

The selection of the model is an important step during an investigation. Therefore, the experimenter should pay a lot of attention to this point. Aside from the linear model, there are other mathematical models as interactions or quadratic ones. They are described in detail in section 2.5.

1.2.6 Experimental Objectives

The term *experimental objective* is generally understood as the purpose for the creation of a design and can be divided into three significant types of designs.

Screening

A screening design is normally performed in the beginning of an investigation when the experimenter wants to characterize a process. In this case, characterizing means to determine the main factors and investigate the changes of the response by varying each factor (Montgomery 1991). Due to its characteristic of identifying 'significant main effects, rather than interaction effects' (NIST/SEMATECH 2007, ch. 5.3.3.4.6), screening designs are often used to analyze designs with a large number of input factors. This identification of the critical process factors can be very useful for later optimization processes because only a subset of the factors has to be considered.

Optimization

After the described screening, it is common to do an optimization. This design gives the experimenter detailed information about the influence of the factors and it determines the combination of factors that leads to the best response. Or in simple terms, the design helps to find the optimal experimental point by predicting response values for all possible factor combinations (Montgomery 1991). Response Surface Modeling (RSM) 'allow[s] us to estimate interaction and even quadratic effects, and therefore gives us an idea of the [. . .] shape of the response surface we are investigating' (NIST/SEMATECH 2007, ch. 5.3.3). Hence, RSM is normally used to find these improved or optimized process settings and helps to determine the settings of the input factors that are needed to obtain a desired output.

Robustness Test

Normally a robustness test is the last design that is created before a process is brought to completion. Its aim is to figure out how the factors have to be adjusted to guarantee robustness. In this case, robustness means that small fluctuations of the factors do not affect the response in a distinct way. If a process does not accomplish this test, the bounds have to be altered to claim robustness (Eriksson et al. 2000).

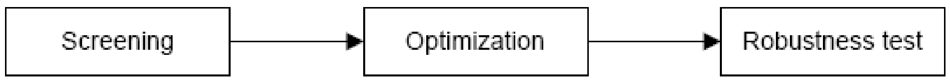


Figure 1.4: Experimental Objectives: *In DoE it is common to perform the three experimental objectives in the here displayed order. A screening design gives information about the important factors. After that, an optimization is performed where these important factors are used to get the best response. The last step is a robustness test.*

1.3 Experimental Designs

1.3.1 Statistical Designs

Aside from the experimental objectives, we have to decide which type of statistical design we want to use. There are three basic types for a regular experimental region which have different features and application areas.

Full Factorial Designs

The first design shown in figure 1.5 is a full factorial design. We call it *full* because the whole cube, including all its corners, is investigated. In addition to this, some replicated center-point experiments are made. Figure 1.5 shows them as a snowflake in the middle of the cube. The factors have two levels of investigation, so for k factors we need 2^k design runs. This type of design is normally used for screening in combination with a linear or an interaction model (Eriksson et al. 2000).

Fractional Factorial Designs

A fractional design does not consider all possible corners and reduces the number of design runs by choosing only a fraction of the 2^k runs of the full factorial design. Figure 1.5 shows an example, where only four of the possible eight points are investigated. Fractional factorial designs are normally used for screening and robustness tests (Eriksson et al. 2000).

Composite Designs

The last design type for a regular experimental area is the composite design. It combines the investigations done by a factorial design, the corners and replicated center-points, with the use of axial experiments. The right-hand cube in figure 1.5 shows an example of three factors where axial experiments are placed on the six squares. The factors normally have three or five levels of investigation and due to this, quadratic models are used (Eriksson et al. 2000).

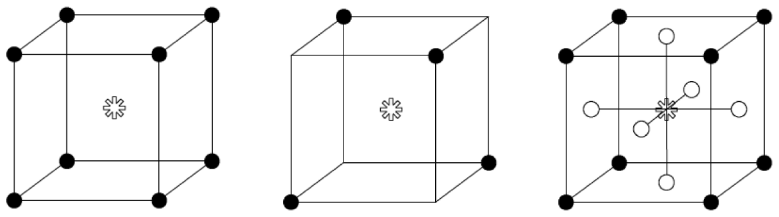


Figure 1.5: Comparison of Statistical Designs, based on Eriksson et al. (2000, p. 15, fig. 1.13): *These pictures shows three different designs for an investigation with three factors and a center-point. The first one shows an example of a full factorial design where all possible corners are investigated. The fractional factorial design in the middle considers only a fraction of all design points and needs just half of the runs. The last cube shows a composite designs which has six additional axial experiments.*

Experimental Region

The regular experimental region gives these three designs a uniform geometrical form and makes them easy to handle. In chapter 2 we see that other designs exists, e.g., when the experimental region becomes irregular. In this case, the above described designs are not applicable and the use of other solutions is indispensable (Eriksson et al. 2000).

1.3.2 General Notation

In figure 1.5 we use a graphical way of illustrating the single experiments. Besides this, it is common to present the data in table form. Table 1.1 shows the 2^2 full factorial example from above in three different notations. Because of the two levels of investigation, we only use the factors minimum and maximum to represent the unscaled data. Other design families, like the composite one, have values between these bounds.

To simplify the values and make them easier to handle, normally an orthogonal scaling is done. After that, all factors have a range of 2 and values between -1 and 1. The standard notation to present the orthogonal scaled factors is done with the algebraic signs $+$ and $-$. The center-points are denoted by 0. The problem with this notation is that values between the three steps ($-$,0,$+$) cannot be indicated. For this reason the use of an extended notation is necessary. Table 1.1 shows both notations for a full factorial design with seven runs (Eriksson et al. 2000).

	unscaled factors		standard notation		extended notation	
No.	x_1	x_2	x_1	x_2	x_1	x_2
1	100	30	$+$	$+$	1	1
2	100	10	$+$	$-$	1	-1
3	50	30	$-$	$+$	-1	1
4	50	10	$-$	$-$	-1	-1
5	75	20	0	0	0	0
6	75	20	0	0	0	0
7	75	20	0	0	0	0

Table 1.1: 2^2 Full Factorial Design: *Example of a full factorial design with two factors and three replicated center-points. The first two columns show the unscaled values in their original units. They are followed by the orthogonal scaled data that is displayed in two different ways, the standard and the extended notation.*

1.4 MODDE 8 and Umetrics AB

1.4.1 Umetrics AB, Umeå

The company Umetrics AB is a software developer, working with DoE and multivariate data analysis. Founded in 1987 by a group from the Department of Organic Chemistry at Umeå University, the company has become an international acting concern which is a member of MKS Instruments Inc. since 2006. Umetrics is located in Umeå, Sweden, and has additional offices in the UK, USA and Malmö, Sweden. Besides this, Umetrics has sales offices and provides training in over 25 locations worldwide. Umetrics offers different types of software solution for analyzing data. The strong combination of graphical functionality and analytic methods is used to deal with multivariate data analysis, DoE, batch processes and on-line data analyzing.

1.4.2 MODDE Version 8.0

The software package MODDE uses the principles of DoE to analyze processes or products and helps the user to get valuable information from the raw data. With different analyzing and visualization tools, the user gets help to understand complex processes and has the opportunity to improve the processes by performing suggested experimental designs. With a user-friendly design wizard and easy visualization methods, the evaluation of raw data and the corresponding decision making is simple.

The implementation of the D-optimal process, described in chapter 4 of the present thesis, is done at the Umetrics office in Umeå, Sweden, and will be implemented in a future release of the MODDE software.

Chapter 2

D-Optimal Designs

2.1 The Need for D-Optimal Designs

Chapter 1 introduces the basic principles of DoE and uses the three standard designs for regular experimental areas to clarify the concept. Besides these design families, we have other design alternatives that are useful in certain situations. In the present thesis, we deal with the computer generated D-optimal designs which are required when:

- the experimental region is irregular,
- already performed experiments have to be included,
- qualitative factors have more than two levels,
- the number of design runs has to be reduced,
- special regression models must be fitted,
- or process and mixture factors are used in the same design.

2.1.1 Irregular Experimental Regions

The experimental region is defined by the investigated factors. The single type of each factor, their space and the total amount of factors influence the shape of the area. To illustrate the region, the use of a simple plot is the most effective tool. In the prior chapter, we describe the use of quadratic or cubic designs, but besides these, other experimental regions like hypercubes et cetera, can be found (Eriksson et al. 2000).

All of these areas have no restriction in the problem formulation. A restriction means that, e.g., one corner of the region is not accessible for experimentation. The middle column of figure 2.1 shows an example of a quadratic design with a constraint in the upper right corner. This corner is not investigated and therefore a normal design is not applicable. Reasons for this can be that the experimenter wants to prevent special factor combinations or that this corner cannot be investigated due to outside influences (Eriksson et al. 2000).

There are two ways to handle the irregular experimental area of figure 2.1. The easiest one is to shrink down the area until it has a quadratic form again, but this would distort the whole investigation and is not recommended. A more effective solution is the creation of a computer aided D-optimal design. As we can see in the bottom right of the four squares in figure 2.1, the D-optimal algorithm chooses two points on the border of the constraint instead of the excluded corner. This increases the number of design runs but is essential to deal with the complexity of the constricted experimental region. In addition to this, the center-point is manipulated (Eriksson et al. 2000).

Mixture design can also have an irregular experimental region. The triangle in figure 2.1 shows a design with the three factors A, B and C. Normally, all corners of the triangle are reachable, but in this example the factors are constrained. For mixture factors a constraint means that the lower and upper bounds differ from 0 and 1 (Eriksson et al. 2000).

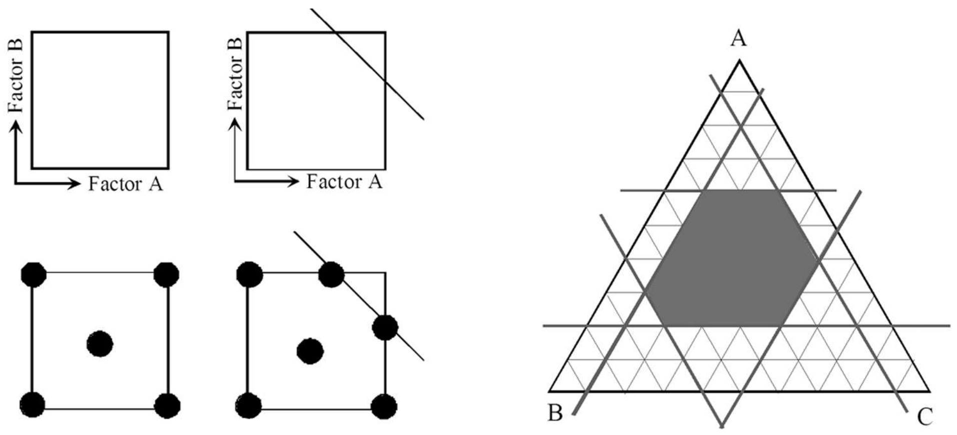


Figure 2.1: Examples for Irregular Experimental Regions, based on Eriksson et al. (2000, p. 214, fig. 18.1-3): *The four squares on the left of this picture represent an experiment with two process factors. The first column shows a regular experimental area with a full factorial design. In column two the area has a constraint in the upper right corner, and therefore a D-optimal design is used. The following triangle shows a mixture design with lower and upper bounds, different from 0 and 1. The experimental region is irregular, too.*

2.1.2 Inclusion of Already Performed Experiments

In some cases the experimenter has already performed a certain number of experiments and wants to include these findings into his investigation. For classical designs it is not possible to add the experiments to the design. With the application of D-optimal designs, we have the possibility to include these additional runs as a part of the design and consider them during the creation (Eriksson et al. 2000).

2.1.3 The Use of Qualitative Factors

In Section 1.2.3 we show that a qualitative factor has only discrete values and no continuous scale. If the number of these discrete steps becomes higher than two, the number of runs for a normal design increases drastically. Figure 2.2 shows an investigation with two qualitative and one quantitative factor. With three and four discrete values for the both qualitative factors, a full factorial design would need $4 * 3 * 2 = 24$ design runs to solve this problem.

The D-optimal approach reduces the number of design runs to only 12 experiments. These experiments, shown in figure 2.2 as filled circles, are chosen to guarantee a balanced design that is spread over the whole experimental region. 'A balanced design has the same number of runs for each level of a qualitative factor' (Eriksson et al. 2000, p. 245).

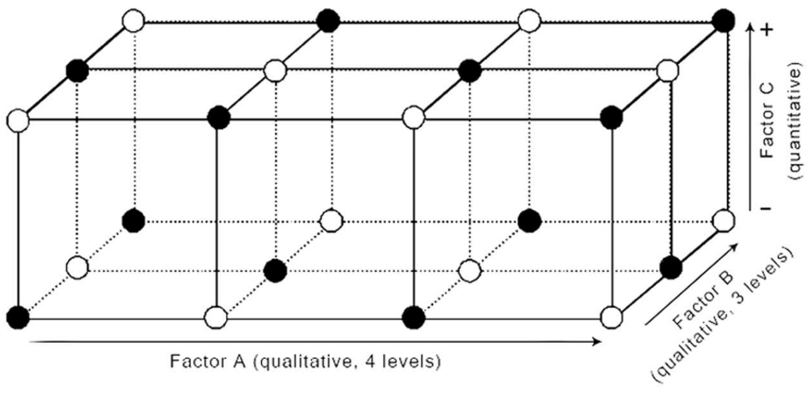

Factor A (qualitative, 4 levels)
Factor B (qualitative, 3 levels)
Factor C (quantitative)
+
-

Figure 2.2: Design With Multi-Level Qualitative Factors, based on Eriksson et al. (2000, p. 214, fig. 18.4): *This picture shows the experimental region for an screening investigation with 3 factors. Factor A and B are qualitative and have three or four discrete levels. Factor C is a normal quantitative factor with values between -1 and +1. The filled circles represent the point chosen by an D-optimal algorithm. A full* $4 * 3 * 2$ *factorial design would select all 24 points.*

2.1.4 Reducing the Number of Experiments

The classical designs from section 1.2.3 are very inefficient if the number of factors increases. Table 2.1 shows examples for the minimum number of runs for a full factorial, a fractional factorial and a D-optimal design. The needed runs for a D-optimal design are always lower and do not increases as fast as the classical design with a growing number of factors (Umetrics 2006).

Factors	Full Factorial	Fractional Factorial	D-Optimal
5	32	16	16
6	64	32	28
7	128	64	35
8	256	64	43
9	512	128	52

Table 2.1: Minimum Number of Design Runs for Screening Designs.

2.1.5 Fitting of Special Regression Models

D-Optimal designs give the opportunity to modify the underlying model in different ways. As equation 2.1 shows, it is possible to delete selected terms if the experimenter knows that they are unimportant for the response. This allows reducing the number of runs without having big influence on the investigation.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{12} x_{12} + \cancel{\beta_{13} x_{13}} + \beta_{23} x_{23} + \epsilon \tag{2.1}$$

The second possible model modification is the addition of single higher order terms. With classical designs it is only possible to change the whole model, e.g., from an interaction to a quadratic model. In contrast to this, D-optimal designs allow the addition of independent model terms. The following equation gives an example of a linear model with an additional interaction term.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{23} x_{23} + \epsilon \tag{2.2}$$

2.2 The D-Optimal Approach

A D-optimal design is a computer aided design which contains the best subset of all possible experiments. Depending on a selected criterion and a given number of design runs, the *best* design is created by a selection process.

2.2.1 Candidate Set

The candidate set is a matrix that contains all theoretically and practically possible experiments, where 'each row represents an experiment and each columns a variable' (de Aguiar et al. 1995, p. 201). This so-called matrix of candidate points has N rows and is represented by ξ_N.

For a simple investigation with two factors, x_1 and x_2, the candidate set has two columns and four rows. We get four rows because we only consider the 2^k experiments which have a minimum or maximum value for the factors. The selection of other experiments can be useful in special cases but is not considered in this basic example. Equation 2.3 shows the candidate set in the extended notation.

$$\xi_4 = \begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \tag{2.3}$$

2.2.2 Design Matrix

The design matrix X is a $n \times p$ matrix that depends on a model with p coefficients. The number of rows n can be chosen by the experimenter and represents the number of experiments in the design. With a given model and a candidate matrix, the construction of the design matrix is easy. Each column contains a combination of the factors from the candidate set, depending on the terms in the model. The matrix can also be called model matrix, but in most cases the model matrix means a $N \times p$ matrix which contains the model-dependent rows for all candidates (de Aguiar et al. 1995).

We use the earlier candidate set ξ_4 and the model from equation 2.4 for a simple example with $n = 4$ design runs.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon \tag{2.4}$$

As a result, we have a model matrix with four rows and four columns, where all candidates from ξ_4 are used in the design. Normally, the candidate set contains much more experiments and the model matrix is only a small subset.

$$X = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.5}$$

The first column of X represents the constant term β_0, so it only contains ones. Column two and three are the model terms for the investigated factors, x_1 and x_2, taken from the candidate set ξ_4. The last column of X represents an interaction between the both factors. Hence, we have to multiply the two columns from the candidate set.

With a bigger candidate set, the number of possible subsets of ξ_N increases and the selection of the design matrix has to be done depending on a special criterion. 'The best combination of these points is called optimal and the corresponding design matrix is called optimal design matrix' $X*$ (de Aguiar et al. 1995, p. 202). We deal with the different criteria in section 2.3.

2.2.3 Information and Dispersion Matrix

To use the later described criteria for the selection of the best design, we need to define two other types of matrices. The first one is the so-called information matrix $(X'X)$. This matrix is the multiplication of the transpose of the design matrix X' and X itself. The dispersion matrix $(X'X)^{-1}$ is the inverse matrix of this calculation (de Aguiar et al. 1995). The background of these equations can be found in the least-squares estimate $\hat{\beta}$ for an assumed model. A model with the matrix notation

$$y = X\beta + \epsilon \tag{2.6}$$

has the best set of coefficients according to least squares given by

$$\hat{\beta} = (X'X)^{-1}X'y. \tag{2.7}$$

Further information on the least square estimator can be found in Box et al. (1978) or Wu & Hamada (2000).

2.3 Criteria for the Best D-Optimal Design

In the following section, we discuss the different criteria for a D-optimal design. All of them belong to the group of information-based criteria because they try to maximize the information matrix $(X'X)$.

D-Optimality (Determinant)

The D-Optimality is the most common criterion 'which seeks to maximize $|X'X|$, the determinant of the information matrix $(X'X)$ of the design' (NIST/SEMATECH 2007, ch. 5.5.2). This means that the optimal design matrix X^* contains the n experiments which maximizes the determinant of $(X'X)$. Or in other words, the n runs 'span the largest volume possible in the experimental region' (Eriksson et al. 2000, p. 216). Equation 2.8 shows the selection of X^* out of all possible design matrices chosen from ξ_N. This connection between the design matrix and the determinant also explains the use of the "D" in the term D-optimal designs.

$$\left|X^{*\prime}X^*\right| = \max_{\xi_n \Xi_N} \left(|X'X|\right) \tag{2.8}$$

Maximizing the determinant of the information matrix $(X'X)$ is equivalent to minimizing the determinant of the dispersion matrix $(X'X)^{-1}$. This correlation is very useful to keep the later calculations as short as possible.(de Aguiar et al. 1995).

$$|X'X| = \frac{1}{|(X'X)^{-1}|} \tag{2.9}$$

A-Optimality (Trace)

Another criterion for an optimal design is called the A-criterion. The design matrix is considered as A-optimal when the trace of the dispersion matrix $(X'X)^{-1}$ is minimum. In this case, the trace of the square matrix is the sum of the elements on the main diagonal. Minimizing the trace of the matrix is similar to minimizing the average variance of the estimated coefficients.

$$\mathrm{trace}(X^{*\prime}X^*)^{-1} = \min_{\xi_n \Xi_N} \left(\mathrm{trace}(X'X)^{-1}\right) \tag{2.10}$$

$$\mathrm{trace}(X'X)^{-1} = \sum_{i=1}^{p} c_{ii} \tag{2.11}$$

A-optimal designs are rarely used because it is more computationally difficult to update them during the selection process.

V-optimality (Average Prediction Variance)

As de Aguiar et al. (1995, p. 203) describe, 'the variance function or leverage is a measurement of the uncertainty in the predicted response'. This variance of predication for a single candidate χ_i can be calculated with the equation

$$\mathrm{d}(\chi_i) = \chi_i' * (X'X)^{-1} * \chi_i, \tag{2.12}$$

where χ_i equals a vector that describes a single experiment and χ_i' represents the transpose of this vector. With the selection of a V-optimal design, the choose candidates have the lowest average variance of prediction, as shown in equation 2.13.

$$\frac{1}{n}\sum_{i=1}^{n} \chi_i' * (X^{*\prime}X^*)^{-1} * \chi_i = \min_{\xi_n \Xi_N} \left(\frac{1}{n}\sum_{i=1}^{n} \chi_i' * (X'X)^{-1} * \chi_i\right) \tag{2.13}$$

G-Optimality (Maximum Prediction Variance)

The last criterion is called G-optimal and deals also with the variance of prediction of the candidate points. The selected optimal design matrix is chosen to minimize the highest variance of prediction in the design.

$$\max\left(\chi_i' * (X^{*\prime} X^*)^{-1} * \chi_i\right) = \min_{\xi_n \Xi_N}\left(\max\left(\chi_i' * (X'X)^{-1} * \chi_i\right)\right) \tag{2.14}$$

G-Efficiency

In most cases, the G-criterion is not used to find the best design during the selection process but is applied to choose between several similar designs which were created with another criterion, like D-optimality. For this comparison the so-called G-efficiency is used. It is defined by

$$G_{eff} = 100\% * \left(\frac{p}{n * \mathrm{d}_{max}(\chi)}\right), \tag{2.15}$$

where p is the number of model terms or coefficients, n is the number of design runs and $\mathrm{d}_{max}(\chi)$ is the largest variance of prediction in the model matrix X. The G-efficiency can be seen as a comparison of a D-optimal design and a fractional factorial design. Consequently, the efficiency is given in a percentage scale.

Condition Number

The condition number (CN) is an evaluation criteria like the G-efficiency and is used to rate an already created D-optimal design. It evaluates the sphericity and the symmetry of the D-optimal design by calculating 'the ratio between the largest and smallest singular values of' X (Eriksson et al. 2000, p. 220). A design with a condition number of 1 would be orthogonal, while an increasing condition number indicates a less orthogonal design.

2.4 A Basic Example

This example is taken from Eriksson et al. (2000) and uses an investigation with two factors, x_1 and x_2, to demonstrate the D-optimal approach in a geometrical way. The two factors are investigated in three levels, -1, 0 and 1. The corresponding candidate set is listed in table 2.2.

Nr.	1	2	3	4	5	6	7	8	9
x_1	-1	-1	-1	0	0	0	1	1	1
x_2	-1	0	1	-1	0	1	-1	0	1

Table 2.2: Candidate Set for Two Factors with Three Levels.

If we would display the candidate set with the matrix notation, the matrix ξ_9 would contain two columns, one for each factor, and $N = 9$ experiments. Figure 2.3 shows the nine candidates from ξ_9 as a plot over the experimental area.

Considering a design with only three design runs, we have $9!/(3! * 6!) = 84$ possible subsets out of this candidate set. In this example, we evaluate four possible design

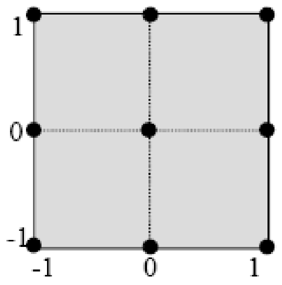


Figure 2.3: Distribution of the Candidate Set, based on Eriksson et al. (2000, p. 218, fig. 18.22): *This picture shows the distribution of the candidates over the experimental region. The investigation considers two factors with three levels and therefore nine candidate points.*

matrices and compare them depending on the D-criteria. Equation 2.16 shows the four selected subset in the matrix notation and figure 2.4 displays the experimental region with the corresponding candidates.

$$\xi_{3A} = \begin{bmatrix} -1 & -1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}, \ \xi_{3B} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \\ 0 & 1 \end{bmatrix}, \ \xi_{3C} = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \ \xi_{3D} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.16}$$

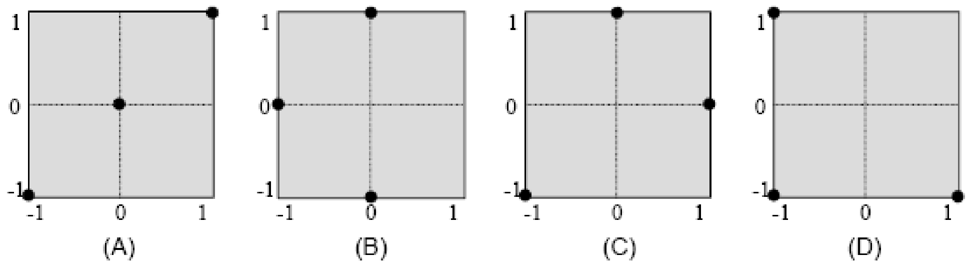


Figure 2.4: Distribution of the Design Matrix, based on Eriksson et al. (2000, p. 218, fig. 18.23-27): *For four different design matrices, the distribution of the selected candidates is shown over the experimental region. The designs are taken from the candidate set ξ_9 from figure 2.3.*

To compute the optimality of these designs, we need to select a model first. In order to keep the example as simple as possible, we choose the following linear model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon. \tag{2.17}$$

Comparing the designs depending on the D-criterion is only possible if we calculate the determinants of the information matrix $(X'X)$ for each of the four designs. In this example, we only show the calculation for the subset ξ_{3B}, but the principle will be the same for all. First, we have to create the model depending design matrix X and its transpose X'. With the linear model, we have a simple design matrix that contains the constant β_0 in the first column and the two factors in the following columns. The calculation of the information matrix is simple the multiplication of these both matrices.

$$(X'_B X_B) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{2.18}$$

The next step is the calculation of the determinant of this information matrix. For small matrices the rule of Sarrus should be used. Figure 2.5 shows that we have to extend the matrix, so that we can calculate the successive diagonals from upper left to bottom right by multiplying the values. The results are summed and the same calculation is done with the diagonals from upper right to bottom left. The difference between the two sums is the determinant of the matrix. Equation 2.19 shows this arithmetic for the information matrix above. The other determinants are shown in table 2.3.

$$\begin{aligned} |X'_B X_B| = &((3*1*2) + ((-1)*0*0) + (0*(-1)*0)) \\ &- ((0*1*0) + (0*0*3) + (2*(-1)*(-1))) = 4 \end{aligned} \tag{2.19}$$

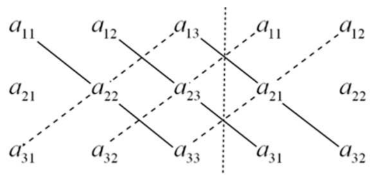


Figure 2.5: Rule of Sarrus for the Calculation of a Determinant: *The rule of Sarrus is a method to calculate the determinant of a square matrix. The matrix is extended by copying the first two rows at the end. After this, the products of the solid lines are added and the the products of the dashed lines is subtracted.*

If we compare the outcome of this investigation, it is obvious that design ξ_{3D} has the highest determinant and therefore is the best D-optimal design. The selected candidates of ξ_{3D} are all located on the corners of experimental region. All designs which investigate three out of the four possible corners have a determinant of 16 and are as good as this selected design. Figure 2.4 shows that these designs also span the biggest area over the experimental region, as described in section 2.3.

Design	Determinant
ξ_{3A}	0
ξ_{3B}	4
ξ_{3C}	9
ξ_{3D}	16

Table 2.3: Determinants of Different Designs.

2.5 Fitting the Model

The described above D-optimal approach is always model-dependent. If we, e.g., change the used model in the basic example of section 2.4, we have another design matrix X

and consequently another information matrix $(X'X)$. Hence, the selection of the model is an important step of the problem formulation. In section 1.2.5, we explain the basic model concept with an example of a linear model. In the following paragraphs, this model is used, and in some case also interaction or quadratic terms are mentioned. To handle more complex investigations, we examine these models in more detail.

Linear Models

Linear models are used for screening designs or robustness tests. Inside the model each factor only appears as a linear term. In this case, a linear term means a combination of a coefficient β_i and a factor x_i. A linear model with three factors has the following equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon. \tag{2.20}$$

Interaction Models

Interaction models are more complex than linear ones but are used for the same experimental objectives as their linear counterpart. The interaction model contains the same terms like the linear model but has additional interaction terms. An interaction term is the combination of two factors x_i and x_j with a conjoint coefficient β_{ij}. Equation 2.21 gives an example of an interaction model with three factors.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \epsilon \tag{2.21}$$

Quadratic Models

The third commonly used model is the quadratic model. It extends the interaction model with additional quadratic terms for each factor. A quadratic term is the square of a factor x_i with its coefficient β_{ii}. Quadratic models are the most complex of the three basic model types and are used for optimization processes. In the same way as the former models, a quadratic model with three factors has the following equation:

$$\begin{aligned} y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{33} x_3^2 \\ + \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \epsilon. \end{aligned} \tag{2.22}$$

2.6 Scaling

Scaling means that the raw data, which has different upper and lower bounds for each factor, is reduced to a set of data where all factors have the same range. We use a kind of scaling in section 1.3.2 when we explain the general and extended notation to display the candidate set. Scaling can be done in three different ways. For D-optimal designs the orthogonal scaling is the most common, but the other ones are applicable, too.

Orthogonal Scaling

When an orthogonal scaling is performed, the factors are scaled and centered depending on the high and low values from the factor definition and the midrange of the values. An original factor x_i is scaled as follows:

$$z_i = \frac{x_i - \bar{M}}{\bar{R}}, \tag{2.23}$$

where $\bar{M}$ is the midrange of all values in the candidate set and $\bar{R}$ represents the range between the high and low value from the factor definition.

$$\bar{M} = \frac{\max x + \min x}{2}, \qquad \bar{R} = \frac{\max x_{def} - \min x_{def}}{2} \tag{2.24}$$

Midrange Scaling

A midrange scaling only centers the values depending on the midrange of the highest and lowest factor value in the candidate set. The values are not scaled. Equation 2.25 gives the arithmetic for this calculation.

$$z_i = x_i - \bar{M} \tag{2.25}$$

Unit Variance Scaling

The last scaling is called unit variance scaling and also uses the midrange of the highest and lowest factor value to center the data. In addition to this, a scaling depending on the standard deviation σ of the values is performed. The following equation shows the calculation for an unscaled factor x_i.

$$z_i = \frac{x_i - \bar{M}}{\sigma} \tag{2.26}$$

2.7 Number of Design Runs

In order to fit a optimality criterion from section 2.3, we have to define the number of experiments we want to have in the design. The selection of this factor n is very important because changing the number of the design runs alters the model matrix and another optimal design is chosen. There are no rules to define this number, but the minimum is model-dependent. A model with p coefficients can only be investigated with a D-optimal design which has at least p runs. In most cases, it is useful to create different designs that differ in the number of runs and compare the efficiency of the designs. A design with a few more or less design runs than the desired one can have a higher determinant and hence is the best design to perform.

2.8 Bayesian Modification

D-optimal designs use different criteria to choose the best design from a pool of all possible factor combinations. For all these criteria, the selection process is strongly model-dependent and therefore the experimenter should pay attention to the choice of the model. Changing one or more coefficients creates a different model matrix which leads to another optimal design.

2.8.1 The Addition of Potential Terms

This dependence on the assumed model can be a problem if the model is not chosen carefully. A Bayesian modification reduces this influence of the model by adding additional potential terms. Considering a model with p primary terms, we add q potential terms and the model matrix X has the following form:

$$X = (X_{pri}|X_{pot}). \tag{2.27}$$

The additional terms are normally not included in the model and have a higher degree than the primary terms. The selection of the potential terms is problem-dependent and is explained in detail in chapter 4 (DuMouchel & Jones 1994).

2.8.2 Scaling the Factors

In order to profit from the Bayesian modification, the primary and potential terms have to be scaled in different ways. Each non constant primary term has a range of 2 and varies from -1 to 1. Normally, these terms are scaled and centered with the orthogonal scaling.

$$\max(X_{pri}) = 1, \quad \min(X_{pri}) = -1 \tag{2.28}$$

For the Bayesian modification, DuMouchel & Jones (1994, p. 39) defined that the potential terms have the following condition:

$$\max(X_{pot}) - \min(X_{pot}) = 1, \tag{2.29}$$

where the maximum and minimum values 'are taken over the set of candidate points for the design'. The scaling of the potential terms is 'achieved by performing a regression of the potential terms on the primary terms, using the candidate points [...] to compute α and Z' (DuMouchel & Jones 1994, p. 40).

$$\alpha = (X'_{pri} X_{pri})^{-1} * X'_{pri} X_{pot} \tag{2.30}$$

$$R = X_{pot} - X_{pri} * \alpha \tag{2.31}$$

$$Z = \frac{R}{\max(R) - min(R)} \tag{2.32}$$

In this case, α is the least square coefficient of X_{pot} on X_{pri} and R represents the residual of this regression. The used minimum and maximum values are taken column-wise over the whole candidate set. The outcoming matrix Z is the scaled and centered version of the potential terms and is used in the model matrix instead of X_{pot}.

$$X = (X_{pri}|Z) \tag{2.33}$$

To use the Bayesian modified model matrix with normal D-optimal criteria, we have to update the dispersion matrix $(X'X)^{-1}$ by adding a further matrix. This so-called Q-matrix is a $(p+q) \times (p+q)$ diagonal matrix with zeros in the first p diagonal elements and ones in the last q diagonal elements. Depending on a chosen τ-value, we modify the dispersion matrix as follows:

$$\left(X'X + \frac{Q}{\tau^2} \right)^{-1}. \tag{2.34}$$

DuMouchel & Jones (1994, p. 40) defined that '[t]he value $\tau = 1$ is a suitable default choice, implying that the effect of any potential term is not expected to be much larger than the residual standard error'.

2.8.3 Bayesian Example

Using the basic example from section 2.4 and increasing the number of design runs to $n = 5$, a normal selection process with D-optimality as estimation criterion creates an optimal design X^*_{norm} that chooses all four corners of the experimental region. In addition to this, the fifth point is a replicate of one of the already selected corners. The left area in figure 2.6 shows a possible combination of the five selected points, and equation 2.35 gives the matrix notation. The calculated determinant for this optimal and model-dependent design is 512.

$$X^*_{norm} = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{2.35}$$

A Bayesian modification changes the selection process by modifying the model matrix with additional potential terms. The selected experiments are displayed in figure 2.6 on the right side. In our example, we use the simple interaction model from equation 2.17 and add the pure quadratic terms $\beta_{11}x_1^2$ and $\beta_{22}x_2^2$. With this modification, the primary and potential terms of the optimal design matrix X^*_{bay} have the following form:

$$X^*_{bay,prim} = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad X^*_{bay,pot} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.36}$$

The primary term matrix contains four model-dependent columns, where the first one represents the constant, column two and three are the normal factors x_1 and x_2, and the last column contains the interaction term $x_1 * x_2$. The two columns of the potential term matrix are simply the squares of the factors x_1 and x_2.

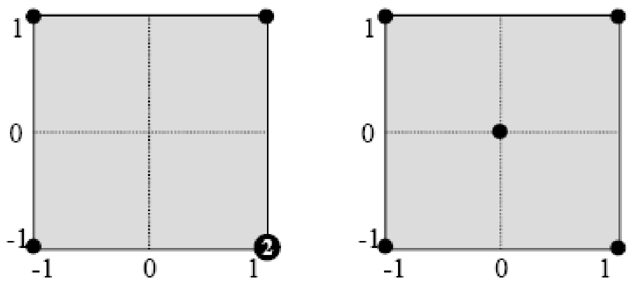


Figure 2.6: Distribution of Normal and Bayesian Design Matrix, based on DuMouchel & Jones (1994, p. 42, fig. 1): *This picture shows the distribution of the selected candidates by a normal D-optimal selection process on the left and the Bayesian modification on the right. The candidates are shown over the experimental region and taken from the candidate set ξ_9 from section 2.4.*

After the Bayesian scaling of the potential terms, we combine the primary term matrix with the modified potential matrix Z. Equation 2.37 shows the new design matrix X^*_{bay} with the scaled potential terms in the last two columns. The lower values of the q additional terms shows that the terms in Z are less important than the p primary terms in the original design matrix.

$$X^*_{bay} = \begin{bmatrix} 1 & -1 & -1 & 1 & 0.2 & 0.2 \\ 1 & -1 & 1 & -1 & 0.2 & 0.2 \\ 1 & 1 & -1 & -1 & 0.2 & 0.2 \\ 1 & 1 & 1 & 1 & 0.2 & 0.2 \\ 1 & 0 & 0 & 0 & -0.8 & -0.8 \end{bmatrix} \tag{2.37}$$

To complete this example, we have to add the Q-Matrix to the dispersion matrix $(X'X)^{-1}$ before we can select the optimal design matrix X^*_{bay} with the highest determinant. For our investigation with $p = 4$ primary and $q = 2$ potential terms, Q is a 6×6 diagonal matrix with the following values:

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.38}$$

A comparison of the two designs with the D-optimal criterion shows that the determinant, taken over the primary terms, of the matrix X^*_{norm} is higher than the determinant of X^*_{bay}. However, an observation of figure 2.6 indicates that a Bayesian modified version of the D-optimal approach selects a better design, without duplicates. In addition to this, the selection is minor model-dependent.

$$\left|X^{*\prime}_{norm} X^*_{norm}\right| = 516, \quad \left|X^{*\prime}_{bay,prim} X^*_{bay,prim}\right| = 320 \tag{2.39}$$

Chapter 3

D-Optimal Designs as a Computer Algorithm

3.1 Exchange Algorithms

Chapter 2 describes the basic principles and criteria for the construction of D-optimal designs but does not explain the selection process itself. Based on the complexity of the D-optimal designs and the huge number of possible combinations of experiments, computer algorithms are used for the selection process. The present thesis only deals with the so-called exchange algorithms. Figure 3.1 summarizes the basic steps which have to be performed before such an exchange algorithm is applicable.

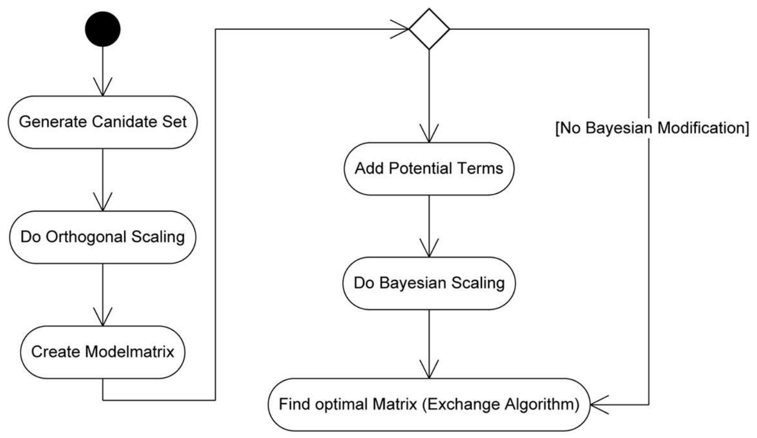


Figure 3.1: Flow Chart of the D-Optimal Process.

An exchange algorithm selects the optimal design matrix X^* by exchanging one or more points from a generated start design and repeats this exchanges until the best matrix seems to be found. The algorithms can be differentiated in two groups where a rank-1 algorithm adds and deletes the points sequentially, and a rank-2 algorithm realizes the exchange by a simultaneous adding and deleting process (Meyer & Nachtsheim 1995). We now explain and evaluate six different algorithms which are universally applicable but differ in the used computing time and the quality or efficiency of the generated designs.

3.1.1 General Exchange Procedure

In order to begin the selection process, we have to create a start design with n experiments in the design matrix X_n. The aim of an exchange algorithm is to remove or add points to this design matrix and determine the effect of the modification. With reference to section 2.2, we call the information matrix $(X'_n X_n)$ and use the following equation for the variance of prediction of a single candidate χ_x:

$$\mathrm{d}(\chi_x) = \chi'_x * (X'_n X_n)^{-1} * \chi_x. \tag{3.1}$$

The addition of a new experiment χ_j to the design matrix X_n, creates a new matrix X_{n+1}, and as de Aguiar et al. (1995) showed, the relation between these two matrices can be expressed by the use of the information matrices. Equation 3.2 shows this interconnection with the use of the added single candidate χ_j and its transpose.

$$(X'_{n+1} X_{n+1}) = (X'_n X_n) + (\chi_j * \chi'_j) \tag{3.2}$$

Furthermore, the influence of the exchange can be used to update the determinant of the new matrix without calculating it in the ordinary way. In this case, the determinant increases proportional to the variance of prediction $\mathrm{d}(\chi_j)$ of the added point χ_j.

$$\left|X'_{n+1} X_{n+1}\right| = |X'_n X_n| * (1 + \mathrm{d}(\chi_j)) \tag{3.3}$$

Removing an experiment χ_i from the design matrix relies on the same fundamentals and is given with the following equations:

$$(X'_{n-1} X_{n-1}) = (X'_n X_n) - (\chi_i * \chi'_i), \tag{3.4}$$

$$\left|X'_{n-1} X_{n-1}\right| = |X'_n X_n| * (1 - \mathrm{d}(\chi_i)). \tag{3.5}$$

The selection of the added and removed points among the candidate set is different in each algorithm and is discussed in the following sections (de Aguiar et al. 1995).

3.1.2 DETMAX Algorithm

The DETMAX algorithm was published by Mitchell (1974) and is a typical rank-1 algorithm. With a random start design of n runs, the algorithm tries to improve the determinant of the information matrix by either adding or deleting a point. The added experiment χ_j is the one with the highest variance of prediction $\mathrm{d}(\chi_j)$. As equation 3.3 shows, this experiment leads to the maximum possible increase of the determinant. The deleted point χ_i is the one with the lowest variance of prediction because an exchange of this point decreases the determinant the least. If a point is added or deleted first, is chosen randomly. The result of such an exchange process is a determinant that is higher or equal than the previous one.

So far this algorithm would be the Wynn-Mitchell algorithm from 1972, but Mitchell (1974) modified this approach to gain more flexibility and allowed excursions in the design. In this case, an excursion means that a $n+1$-point design must not be reduced immediately to a n-point design but can become a design with $n+2$ points. Therefore, the replacement of more than one point of the original design can be possible during one iteration. The limit of excursions is defined by Mitchell (1974) to be $k = 6$. Considering a design with the currently best n points, the algorithm adds or deletes up to k points until the excursion limit is reached. The size of the created designs varies from $n - k$ to $n + k$. If no improvement in the determinant is found during this excursion, all the created design are saved in a list F that contains failure designs. This set F is used for the next excursion where we consider two different rules defined by Mitchell (1974, p. 204):

> Letting D be the current design at any point during the course of an excursion, the rules for continuing are as follows:
>
> (i) If the number of points in D exceed n, subtract a point if D is not in F and add a point otherwise.
>
> (ii) If the number of points in D is less than n, add a point if D is not in F and subtract a point otherwise.

The following listing 3.1 shows the use of these excursion rules and visualizes the flow of the algorithm with a simple and abstract programming notation.

```
create random design with desired number of points;
while excursion limit is not reached do
    if number of candidates equals number of desired runs then
        randomize between adding or deleting a point;
    else if number of candidates is bigger than number of desired runs then
        if new designs is not inside set of failure design then
            delete candidate with lowest variance of prediction;
        else
            add candidate with highest variance of prediction;
    else if number of candidates is smaller than number of desired runs then
        if new designs is not inside set of failure design then
            add candidate with highest variance of prediction;
        else
            delete candidate with lowest variance of prediction;
    if no improvement of the determinant is found then
        save design to list of failure designs;
    else
        clear list of failure designs;
```

Algorithm 3.1: DETMAX algorithm by Mitchell (1974).

With an empty set of failure designs F, the algorithm simply adds and subtracts points, as described in the first paragraph of this section. Considering the failure designs of previous iterations, the algorithm always 'proceeds in the direction of a n-point design, unless it comes to a design in F, which has already lead to failure. Then it reverses direction [...]' (Mitchell 1974, p. 204). If an excursion improves the design, the failure designs in F are deleted and a new start is performed (de Aguiar et al. 1995, Mitchell 1974).

3.1.3 Fedorov Algorithm

Fedorov's (1972) algorithm is a simultaneous exchange method which always keeps the size n of the desired design without any excursions. After the generation of a random start design, the algorithm selects a point χ_i from the design that should be removed by a point χ_j from the candidate set. The procedure of adding and removing a point is done in one step and can be denoted as a real exchange. The effect of such an exchange can be shown by the use of the information matrix. With reference to equation 3.2 and 3.4, an exchange is a simultaneous addition and subtraction of a point.

$$(X'_{new}X_{new}) = (X'_{old}X_{old}) - (\chi_i * \chi'_i) + (\chi_j * \chi'_j) \tag{3.6}$$

In contrast to the general exchange from section 3.1.1, Fedorov (1972) considered the interaction between the variance functions of the two candidates for the calculation of the new determinant. Instead of adding and deleting the variance of prediction of the two points according to equation 3.3 and 3.5, he defined a so-called "delta"-function which changes the determinant of the the matrix in the following way:

$$|X'_{new}X_{new}| = |X'_{old}X_{old}| * (1 + \Delta(\chi_i, \chi_j))\,. \tag{3.7}$$

The calculation of the Δ-value for a couple of χ_i and χ_j uses the variance of prediction of both points and a combined variance function called $\mathrm{d}(\chi_i, \chi_j)$.

$$\Delta(\chi_i, \chi_j) = \mathrm{d}(\chi_j) - \left[\mathrm{d}(\chi_i)\,\mathrm{d}(\chi_j) - (\mathrm{d}(\chi_i, \chi_j))^2\right] - \mathrm{d}(\chi_i) \tag{3.8}$$

$$\mathrm{d}(\chi_i, \chi_j) = \chi'_i * (X'_nX_n)^{-1} * \chi_j = \chi'_j * (X'_nX_n)^{-1} * \chi_i \tag{3.9}$$

The basic idea of the Fedorov algorithm is to calculate the Δ-value for all possible couples (χ_i,χ_j) and select the one with the highest value. The point χ_i is taken from the currently selected design, and χ_j can either be taken from the remaining points or from the whole candidate set. Considering only the remaining points is called an exhausting search which avoids duplicated points in the design. With a non-exhausting search, the selection of an experiment which has to be performed twice is possible. As equation 3.7 shows, the points with the highest Δ-value increase the determinant most. If more than one couple with the same Δ-value is found, the algorithm chooses randomly among them. While couples with a positive Δ-value are found, the algorithm exchanges the points and updates the information and dispersion matrix. Sometimes the algorithm finds couples that increase the determinant so little that no significant difference is achieved. To avoid the algorithm dealing with the exchange of these couples, Fedorov (1972) defined a threshold value ϵ_{fed} and breaks the algorithm if the maximum Δ-value is smaller than ϵ_{fed}. 10^{-6} is a common value for ϵ_{fed} (de Aguiar et al. 1995, Fedorov 1972). The general outline of the algorithm is given in listing 3.2.

```
create random design with desired number of points;
while couples with positive delta are found do
    for design point $\chi_1$ to design point $\chi_n$ do
        calculate variance of prediction $\mathrm{d}(\chi_i)$ for this design point;
        for support point $\chi_1$ to support point $\chi_N$ do
            calculate variance of prediction $\mathrm{d}(\chi_j)$ for this support point;
            calculate variance function $\mathrm{d}(\chi_i, \chi_j)$ for this couple;
            calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
            check if maximum delta and save couple;
    if maximum delta is positive then
        if more then one couple with same maximum delta then
            select couple randomly;
        exchange selected point $\chi_i$ with $\chi_j$;
        update information and dispersion matrix;
        reset maximum delta;
```

Algorithm 3.2: Fedorov Algorithm (1972).

3.1.4 Modified Fedorov Algorithm

Cook & Nachtsheim (1980) made a comparison of different algorithms for constructing exact D-optimal designs and invented an own algorithm depending on the basic Fedorov algorithm from 1972. The normal algorithm by Fedorov calculates the Δ-values for all possible exchange couples during one iteration but only uses one of the values to perform an exchange. This calculation is expensive to use. With the modified version by Cook & Nachtsheim (1980, p. 317), each iteration of the normal algorithm 'is broken down into [n] stages, one for each support point in the design at the start of the iteration.' With a randomly ordered design matrix, the algorithm starts with the first support point χ_i and calculates the Δ-values for all possible couples with this fixed support point. After finding the best exchange for this point, the design is updated and the next support point is suggested for an exchange. In other words, one iteration of the normal Fedorov algorithm is modified to exchange up to n design points if the determinant would increase (Cook & Nachtsheim 1980, Atkinson & Donev 1989). This behavior is visualized in listing 3.3.

The difference between the two approaches can be made clear with a simple example. Considering a desired design with $n = 5$ runs and a candidate set with $N = 20$ experiments, $n * N = 100$ possible couples can be suggested for an exchange. Each iteration of the Fedorov algorithm calculates these 100 Δ-values for all possible couples and uses only one of them for an exchange. By contrast, the modified version of the algorithm starts with the first design point and only calculates the 20 Δ-values for the possible couples including this point. After this, an exchange is performed and the dispersion matrix has to be updated. The algorithm goes on with the next design point and calculates again 20 Δ-values. Overall, both algorithm calculate the 100 values during one iteration, but the modified version exchanges up to 5 points during this procedure.

A study by Cook & Nachtsheim (1980) and the data presented in the following chapter 3.3 show that the modified approach can be twice as fast as the normal Fedorov

```
create random design with desired number of points;
while couples with positive delta are found do
    for design point $\chi_1$ to design point $\chi_n$ do
        calculate variance of prediction $d(\chi_i)$ for this design point;
        for support point $\chi_1$ to support point $\chi_N$ do
            calculate variance of prediction $d(\chi_j)$ for this support point;
            calculate variance function $d(\chi_i, \chi_j)$ for this couple;
            calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
            select couple with maximum delta;
            if maximum delta is positive then
                if more then one couple with same maximum delta then
                    select couple randomly;
                exchange selected point $\chi_i$ with $\chi_j$;
                update information and dispersion matrix;
                reset maximum delta;
```

Algorithm 3.3: Modified Fedorov Algorithm by Cook & Nachtsheim (1980).

algorithm and creates design with a comparable efficiency. The additional time needed to update the dispersion matrix after each exchange is adjusted by the profit of the multiple exchanges.

3.1.5 *k*-Exchange Algorithm

During a comparison of the normal Fedorov (1972) algorithm and Mitchell's (1974) DETMAX, Johnson & Nachtsheim (1983, p. 274) figured out that the points selected by the Fedorov algorithm for deletion are normally not the ones with the lowest variance of prediction, but 'the frequency of ranks of points deleted could be characterized as skewed towards the lower variance ranks'. Simply said, instead of either considering all candidates or only the one with the lowest variance of prediction, a set of k points with the lowest variance should be selected. Similar to the modified Fedorov algorithm, an iteration is now broken down into k steps. Inside each of this k steps the Δ-values are calculated and the corresponding couple is exchanged if the determinant would be increased (Johnson & Nachtsheim 1983).

The similarity to the previous described algorithms can be recognized if we define the k-value. With $k = 1$ the algorithm is similar to the Wynn-Mitchell algorithm from section 3.1.2, and with $k = n$ it becomes the modified Fedorov algorithm (Cook & Nachtsheim 1980). The selection of the k-value is difficult and in most cases problem-dependent. A common value, which is also suggested by Johnson & Nachtsheim (1983), is $k = 3$ or $k = 4$. Meyer & Nachtsheim (1995) later advised to select the value with the following condition:

$$k \leq \frac{n}{4}. \tag{3.10}$$

Listing 3.4 shows the general structure of the k-exchange algorithm which reduces the list of design points by selecting only the k points with the lowest variance of prediction.

```
create random design with desired number of points;
while couples with positive delta are found do
    calculate variance of prediction d($\chi_i$) for all design points;
    select $k$ design points with lowest variance of prediction;
    for design point $\chi_1$ to design point $\chi_k$ do
        calculate variance of prediction d($\chi_i$) for this design point;
        for support point $\chi_1$ to support point $\chi_N$ do
            calculate variance of prediction d($\chi_j$) for this support point;
            calculate variance function d($\chi_i, \chi_j$) for this couple;
            calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
            select couple with maximum delta;
            if maximum delta is positive then
                if more then one couple with same maximum delta then
                    select couple randomly;
                exchange selected point $\chi_i$ with $\chi_j$;
                update information and dispersion matrix;
                reset maximum delta;
```

Algorithm 3.4: *k*-Exchange Algorithm by Johnson & Nachtsheim (1983).

Continuing the example with $n = 5$ and $N = 20$, the k-exchange speeds up the selection process by reducing the complete number of calculated Δ-values. Similar to the modified Fedorov procedure, each iteration of the basic Fedorov algorithm is broken down into Δ-calculations for each design point. But depending on the defined k-value, the algorithm only considers some of the design points. With $k = 3$ the k-exchange algorithm calculates three times the 20 Δ-values and performs up to three exchanges. This approach has a higher computational efficiency compared to the modified Fedorov procedure. Especially for huge data set with a high value of n, the definition of the k-value has an impact. The quality of the design is not always as good as the one from the normal Fedorov algorithm, but with respect to the needed computing time, the algorithm gives good results. A detailed analysis of the efficiency and quality of the designs follows in section 3.3.

3.1.6 *kl*-Exchange Algorithm

The *kl*-exchange is also a modification of the basic Fedorov algorithm from 1972 and therefore a typical rank-2 algorithm. Developed by Atkinson & Donev (1989), this algorithm reduces the list of support and exchange points before calculating the Δ-values for all possible couples. The use of the k points with the lowest variance of prediction is similar to the k-exchange procedure. In addition to this, we only consider the l candidates with the highest variance of prediction among the support points. By defining $k = n$ and $l = N$, this algorithm is the normal Fedorov procedure. A normal initialization of the algorithm with $k < n$ and $l < N$ reduces the amount of calculated Δ-values and speed up the algorithm. Like the normal Fedorov algorithm, the *kl*-exchange selects the points with the highest Δ-value and performs a single exchange. Even if the name of the *kl*-exchange sound like a modification of the k-exchange by Cook &

Nachtsheim (1980), we have to observe that the *kl*-procedure relies on the basic Fedorov algorithm and does not perform multiple exchanges during one iteration. Only the concept of picking the *k* point with the lowest variance of prediction is similar to the *k*-exchange algorithm.

Atkinson & Donev (1989) defined two modifications of this algorithm. The first one is similar to the basic idea of the modified Fedorov algorithm by Cook & Nachtsheim (1980) and performs more than one exchange during each iteration. With the selection of $l = 1$, this changed *kl*-exchange becomes either the modified Fedorov algorithm with $k = n$ or the *k*-exchange with $k < n$. The second modification changes the selection criteria for the *k* and *l* points. Instead of a variance dependent choice, the algorithm selects randomly among the design and support points (Atkinson & Donev 1989). In listing 3.5 the basic version is shown without any modification.

```
create random design with desired number of points;
while couples with positive delta are found do
    calculate variance of prediction d($\chi_i$) for all design points;
    select $k$ design points with lowest variance of prediction;
    calculate variance of prediction d($\chi_j$) for all support points;
    select $l$ support points with highest variance of prediction;
    for design point $\chi_1$ to design point $\chi_k$ do
        calculate variance of prediction d($\chi_i$) for this design point;
        for support point $\chi_1$ to support point $\chi_l$ do
            calculate variance of prediction d($\chi_j$) for this support point;
            calculate variance function d($\chi_i, \chi_j$) for this couple;
            calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
            check if maximum delta and save couple;
    if maximum delta is positive then
        if more then one couple with same maximum delta then
            select couple randomly;
        exchange selected point $\chi_i$ with $\chi_j$;
        update information and dispersion matrix;
        reset maximum delta;
```

Algorithm 3.5: *kl*-Exchange Algorithm by Atkinson & Donev (1989).

Compared with the normal Fedorov algorithm, the *kl*-exchange reduces the amount of calculated Δ-values between each iteration. Our example with $n = 5$, $N = 20$ and $k = l = 3$ leads to the calculation of only $k * l = 3 * 3 = 9$ Δ-values. Compared with the 100 couples of the normal Fedorov algorithm, this *kl*-approach speeds up the calculation in an effective way. Admittedly, we have to observe that the amount of performed iterations of the *kl*-exchange can be higher because not all couples are considered for an exchange, and therefore the algorithm has to perform more loops. But in general, the *kl*-exchange algorithm creates designs very fast and in most cases gives acceptable results considering the D-optimality.

3.1.7 Modified *kl*-Exchange Algorithm

Based on the previous experiences of the company Umetrics AB and the knowledge gained during the implementation process of different algorithms, a modified version of the *kl*-exchange algorithm is developed in the present thesis. The first simple change in the algorithm is to extend the k-value if the k selected points have the same variance of prediction and a similar $(k+1)$th value exists. Using a random selection if more than k candidates have the same variance can also be a solution but is not as foolproof as this one.

The second modification tries to prevent the *kl*-exchange algorithm from exchanging some couples which are not the best choice by reducing the list of support points. Starting with $l = N$ points in the pool of possible support points, the algorithm removes points depending on different criteria. Generally, an exchange should only be performed if the variance of prediction $\mathrm{d}(\chi_j)$ of the support point χ_j is higher than the variance of prediction $\mathrm{d}(\chi_i)$ of the removed design point χ_i. A second criterion to reduce the list of support point is developed dependent on the average variance of prediction from the last iteration. Only the support points with a higher variance of prediction than the average variance are considered. In this case, the average variance of prediction is calculated with the use of all possible candidates. If the algorithm does not find a maximum Δ-value among these possible best support points, the remaining support points are checked. The complete algorithm is shown in listing 3.6.

This algorithm is difficult to compare to the previous approaches because the amount of design and support point considered for an exchange is problem-dependent. We do not have fixed values for l and k and the criteria for the selection of the points can change during the process. But in general, the modified *kl*-exchange is comparable to the normal *kl*-exchange, considering the computational efficiency. The calculations need some more time because we sometimes change the variance criteria and calculate the Δ-values again to find the best exchange. But overall, the algorithm is comparable to the normal *kl*-exchange. But considering the quality of the generated designs, this algorithm leads to better results and in most cases also beats the k-exchange procedure. Of course, the results do not have the same quality as the designs created with the Fedorov algorithm. But with an admissible computing time, we create good D-optimal designs.

In general, the comparison of the quality of the designs is difficult because most of the algorithms have different strength and weaknesses but should be applicable for all problem formulations. This problem is examined in detail in section 3.3.

3.2 Generation of the Start Design

Another method to construct D-optimal designs was developed by Dykstra (1971) before the common exchange algorithms from section 3.1 were used. Instead of beginning with a design with the desired size n, Dykstra (1971) started with an empty design and added sequentially the design points with the highest variance of prediction to maximize the determinant. This solution gives poor results considering the efficiency of the design. But the approach can be used as a fast method for generating a starting design for other exchange algorithms. Compared with a randomized starting design, a sequentially selected one reduces the number of iterations of an algorithm because the points in the design are already a preselection.

```
create random design with desired number of points;
while couples with positive delta are found do
    calculate variance of prediction d($\chi_i$) for all design points;
    if more than k design points have same lowest variance of prediction then
        extend given $k$;
    select $k$ design points with lowest variance of prediction;
    calculate variance of prediction d($\chi_j$) for all support points;
    calculate average variance of prediction over whole candidate set;
    select $l$ support points with higher variance of prediction than the average
    value;
    for design point $\chi_1$ to design point $\chi_k$ do
        calculate variance of prediction d($\chi_i$) for this design point;
        for support point $\chi_1$ to support point $\chi_l$ do
            calculate variance of prediction d($\chi_j$) for this support point;
            if variance of the support point is higher than the variance of the
            design point then
                calculate variance function d($\chi_i, \chi_j$) for this couple;
                calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
                check if maximum delta and save couple;
    if maximum delta is negative then
        select $N - l$ support points that have not been checked;
        for design point $\chi_1$ to design point $\chi_k$ do
            calculate variance of prediction d($\chi_i$) for this design point;
            for support point $\chi_1$ to support point $\chi_{N-l}$ do
                calculate variance of prediction d($\chi_j$) for this support point;
                calculate variance function d($\chi_i, \chi_j$) for this couple;
                calculate delta function $\Delta(\chi_i, \chi_j)$ for this couple;
                check if maximum delta and save couple;
    if maximum delta is positive then
        if more then one couple with same maximum delta then
            select couple randomly;
        exchange selected point $\chi_i$ with $\chi_j$;
        update information and dispersion matrix;
        reset maximum delta;
```

Algorithm 3.6: Modified *kl*-Exchange Algorithm.

Galil & Kiefer (1980) used the idea of a sequential start design for Mitchell's (1974) DETMAX algorithm and changed the start design to contain some random points in the beginning. Thereafter, the design is sequentially filled with the best support points. The number of the t random points was defined by Galil & Kiefer (1980) to be 'considerable smaller than k for moderate k [...]'. Atkinson & Donev (1989) referred to this topic by using a sequential start design for the kl-exchange algorithm. They defined the number of random selected points t as a randomly chosen integer with the following condition:

$$0 \leq t \leq \min\left(\left[\frac{1}{2}p\right], N-1\right), \tag{3.11}$$

where $\left[\frac{1}{2}p\right]$ denotes the integer part of the fraction and p is the number of used factors.

3.3 Comparison of Exchange Algorithms

Generally, a comparison of the algorithms has to be done depending on two main criteria. The first one is the used computing time to create a D-optimal design, and the second one is the quality of the design. This quality or efficiency of the design can be evaluated by the criteria taken from section 2.3. In this analysis we use the D-Optimality or its analog, the determinant to evaluate the optimal designs. Comparing other criteria, like the condition number or the G-Efficiency, can lead to other results than the ones shown in this section. The base for the given data in this comparison is an analysis of the four algorithms implemented at Umetrics AB. The case study was performed with an interaction model with 5, 7 or 9 factors. The software MODDE is used to create the desired designs and calculates their efficiency, the logarithm of the determinant. During the calculation of 10 designs for each model and algorithm, the computing time is measured and the mean value is calculated. The collected data can be found in table 3.1.

	5 factors		7 factors		9 factors	
	time/ms	log(Det)	time/ms	log(Det)	time/ms	log(Det)
Fedorov	21.9	**20.47**	424.8	**43.74**	6887.2	75.33
mod. Fedorov	21.8	**20.47**	286.7	**43.74**	3535.4	**75.36**
k-exchange	**10.9**	**20.47**	**84.6**	43.71	**669.2**	74.83
mod. kl-exchange	20.3	**20.47**	126.6	43.72	931.7	74.99

Table 3.1: Comparison of Different Algorithms: *This comparison is based on the implemented algorithms from chapter 4. Depending on an interaction model with five, seven or nine factors, a set of ten designs has been created for each algorithm and model. The computing time is the average value of the ten calculations and the efficiency is taken form the best of the designs. The best values of each column are printed in bold.*

During the description of the algorithms in section 3.1, we show that some of the modifications are used to reduce the amount of calculations before an exchange is performed. The Fedorov algorithm, e.g., calculates the Δ-value for all possible couples and only performs one exchange depending on this data. With a large number of factors, this approach is very expensive to use. The modified version by Cook & Nachtsheim (1980) reduces the number of calculations by performing up to n exchanges during each iteration and speeds up the algorithm by using more than one of the values. Table 3.1 shows that the average computing time used for the modified Fedorov algorithm can be up to 50% faster than the normal algorithm from 1972. Especially with large input data, the difference between the two algorithms becomes obviously. Other algorithms, like the k-exchange by Johnson & Nachtsheim (1983) or the modified kl-exchange, are much faster in the creation of D-optimal designs and can be used for huge numbers of factors. The k-exchange, e.g., can handle the interaction model with nine factors from table 3.1 ten-times faster than the basic Fedorov procedure. The Mitchell algorithm is not implemented during the work of the present thesis but was studied and compared by Johnson & Nachtsheim (1983). The evaluated data, depending on different models and factor setting, showed that the DETMAX algorithm has a computing time which is comparable with the normal k-exchange.

Admittedly, a fast algorithm is not expedient if the created design has not the desired quality or D-optimality. Hence, we have to compare the algorithms depending on the time used and the quality of the created design. At this point we have to realize that the measured data from table 3.1 is difficult to evaluate. All of the algorithms can trap into a so-called local optimum and stop the exchange process even if the best design is not found yet. With an increasing number of factors or model terms, the amount of local optima increases, too. Therefore, most of the D-optimal designs are different even if they rely on the same algorithm. In addition to this, the randomized functions during the selection process influence the outcome. Table 3.1 only shows the efficiency of the best design out of ten generated possibilities. In general, it is recommended to create more than one design and to select the best one out of a pool of possible designs. The determinant of the information matrix $(X'X)$ for a small test, like the one in table 3.1 with five factors, has the same value for all algorithms. In this case, it does not matter which algorithm is used, they all create at least one design out of ten with the same efficiency. If the number of factors increases or the model gets more complex, the Fedorov and modified Fedorov normally create the best designs, but in general, the selection of the best algorithm is problem-dependent. The modified kl-exchange, e.g., creates better designs than the Fedorov or modified Fedorov in some cases or is at least comparable. Indeed, this algorithm is much faster than the Fedorov version and should be preferred.

The findings from the above paragraph bear on the data presented in table 3.1 and additional random samples. All of them are based on the implemented versions of the algorithms inside the software MODDE. Therefore, this data is only an indicator that leads to suppositions which algorithm should be used according to different aims. In addition to this, only the four implemented algorithms are considered. A comparison of all algorithms under the same test condition does not exist in literature. Therefore, the selection of the four algorithms is based on two different comparisons. The first one was performed by Cook & Nachtsheim (1980) and analyzes Mitchell's (1974) DETMAX, the Fedorov procedure, the modified Fedorov procedure, the Wynn-Mitchell algorithm, and some other negligible algorithms. The conclusion of the article is that the best designs are created with the Fedorov or modified Fedorov, but the Wynn-Mitchell algorithm should be used if a fast generation is more important. The second evaluation was performed by Johnson & Nachtsheim (1983) and compared the Fedorov algorithm, the modified Fedorov algorithm, the k-exchange procedure, DETMAX ,and the Wynn-Mitchell algorithm. The outcome of this paper is the advice of the authors to use the Wynn-Mitchell algorithm or its rough equivalent procedure, the k-exchange. Based on these findings and the single publications of the algorithms, the four algorithms from table 3.1 are implemented in MODDE. Our small case study approves the advises from Johnson & Nachtsheim (1983) and Cook & Nachtsheim (1980) but in order to have a reliable comparison, a separated complexity analysis has to be performed where all algorithms are compared under the same criteria with the use of different problem formulations. But such an investigation goes beyond the limits of the present thesis.

Overall, the finding of the optimal combination of computing time and efficiency of the generated design is a difficult topic and cannot be concluded with a strict advice which algorithm to use. To sum up briefly, the modified Fedorov algorithm is similar to the normal Fedorov procedure if we consider the quality of the design but needs less computing time. Actually, both algorithms need a lot of calculations if the input data becomes big. Hence, for small designs the modified Fedorov algorithm by Cook & Nachtsheim (1980) should be used. For bigger designs the use of the simple *k*-exchange or the modified *kl*-exchange is advisable. The created designs are a bit inferior in some cases, but with respect to the computing time this is the best choice. If the experimenter wants to create exact D-optimal designs and has the opportunity to evaluate a problem formulation with different algorithms and different parameters, this is the suggested way to find the optimal design.

Generally, we discourage creating exact D-optimal designs because the selection of the right model is difficult and should be handled with care. To refer back to section 2.8, we repeat the same case study from table 3.1 with the use of a Bayesian modification and try to confirm the findings from above. The addition of the potential terms makes the designs more model-independent, and consequently the efficiency of the designs is a bit below the ones created by the normal D-optimal process in table 3.1. A comparison of both studies shows that the outstanding values follow the same regularity and prove the suggestions for the right algorithm selection.

	5 factors		7 factors		9 factors	
	time/ms	log(Det)	time/ms	log(Det)	time/ms	log(Det)
Fedorov	51.6	**19.46**	798.8	**43.41**	11343.4	74.66
mod. Fedorov	35.8	**19.46**	295.6	**43.41**	4303.0	**74.77**
k-exchange	**21.9**	**19.46**	**129.5**	43.16	**915.5**	74.32
mod. *kl*-exchange	31.3	**19.46**	215.7	42.71	1394.4	74.43

Table 3.2: Comparison of Different Algorithms (Bayesian Modification): *This comparison belongs to the same case study as table 3.1, but the model matrix is extended with additional potential terms to make the selected designs more model-independent.*

Chapter 4

The Implementation in C++

4.1 Purpose & Goal

4.1.1 Current Implementation

The software MODDE version 8.0 already contains an implementation of a D-optimal algorithm, but the company Umetrics AB is unsatisfied with this solution and wants a complete reimplementation of this program feature. The algorithm itself works fine for small data sets with up to ten factors and creates reliable D-optimal designs. The only limitation is that in some cases the algorithm gets stuck during the calculation. Another more serious restriction is the highly increasing computing time if the number of factors reaches a level higher than ten. In addition to this, the code is a translation of an old APL code which has been ported to C++ and is not documented in an adequate way. Considering these problems, long-term maintenance becomes very difficult and time-consuming.

4.1.2 Task & Goal

My task at Umetrics AB in Umeå, Sweden, is to implement a complete new D-optimal algorithm in C++ and to integrate the code into the current software MODDE. The algorithm principle should be independent from the old approach to prohibit the creation of an algorithm with the same limitations and to develop a real alternative to the existing code. Another important point during the development process is to create reliable code which is well documented and easy to understand. This helps Umetrics to carry on with the generated code and allows easy changes in the future. The new algorithm should be able to handle the same parameters from the design wizard inside the MODDE software and use the same output as the old code. With the use of a different approach, the new D-optimal algorithm should be able to create designs with a comparable speed and similar efficiency. At the same time, the new approach must handle bigger inputs with at least 15 factors in an admissible time. Besides these general obligations, the code must be able to handle already performed experiments, the so-called inclusions, and integrate them in the design. Furthermore, different types of factors like mixture or qualitative factors must be considered during the generation of the design matrix. In general, the underlying model must be editable and the possibility of a Bayesian modification must be given for each design. In addition to this, constraints for single factors must be considered and the calculation of the center-points has to be done before the design is shown on the worksheet. If more than one design is created, the possibility to select the design depending on the different optimal criteria has to be given in the design wizard. To sum up briefly, the requirements are:

1. Handling of more than 15 factors,
2. a computational efficiency which is comparable with the old implementation,
3. creation of designs which have a comparable optimality,
4. handling of the parameters from the design wizard,
5. output of the designs on the evaluation page and the worksheet of MODDE,
6. the possibility of a Bayesian modification must be given,
7. inclusion must be considered during the design generation,
8. the model must be editable,
9. mixture and qualitative factors must be handled,
10. center-point have to be added to the design,
11. and the code must be well structured and documented.

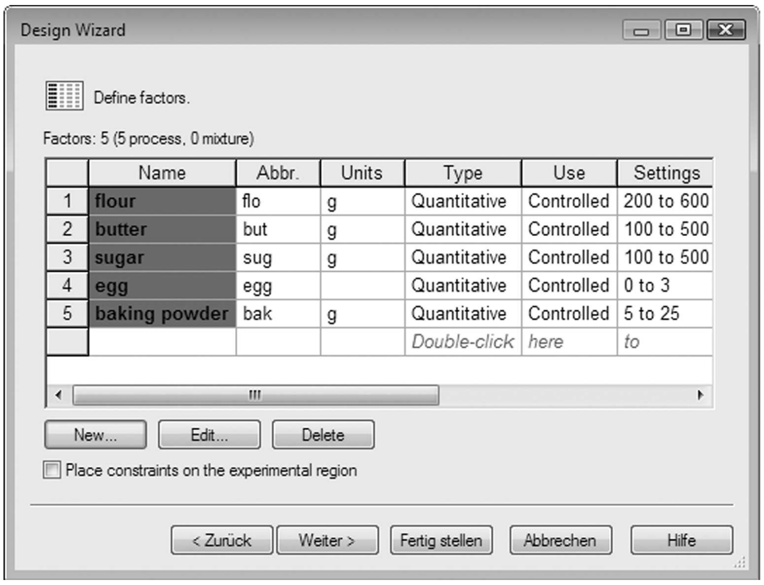
4.1.3 Preliminaries for a Reimplementation

Umetrics AB does not target the implementation of a certain algorithm. Therefore, an investigation of the different opportunities and a comparison of the approaches is necessary before the implementation is started. Depending on the publications used in chapter 3, the basic Fedorov algorithm is used for a preliminary investigation with MathWorks MATLAB. This analysis is used to get a basic overview about the general exchange procedure and the necessary matrix calculations. The basis for this investigation are some MATLAB source files written by Ing-Marie Olsson from Umetrics AB. These files are independently developed from the current implementation in MODDE. In order to improve the knowledge about the D-optimal topic, the original files are recoded, extended, and restructured to get acquainted with the whole D-optimal process. The modified files are used to test and investigate different D-optimal approaches and calculations. Besides the Fedorov algorithm, the use of a Bayesian modification, the orthogonal scaling, and the generation of the model-dependent design matrix are included in this investigation. The revised files can be found in Appendix A.

4.2 Final Implementation

4.2.1 Input & Output

In MODDE the user can utilize the design wizard to create D-optimal designs with a user-friendly interface. The first step is the definition of the factors. Figure 4.1 shows a screenshot of the wizard where the user has to define the factors, their types, and their settings. The possible options for the type and use of a factor are described in section 1.2.3 and can be used to create, e.g., mixture designs. The settings are needed to define the upper and lower bound for each factor of the design. In addition to this, the user has the possibility to select the scaling method and some further options. After defining the factors, the user has to do the same procedure for one or more responses.



Define factors.

Factors: 5 (5 process, 0 mixture)

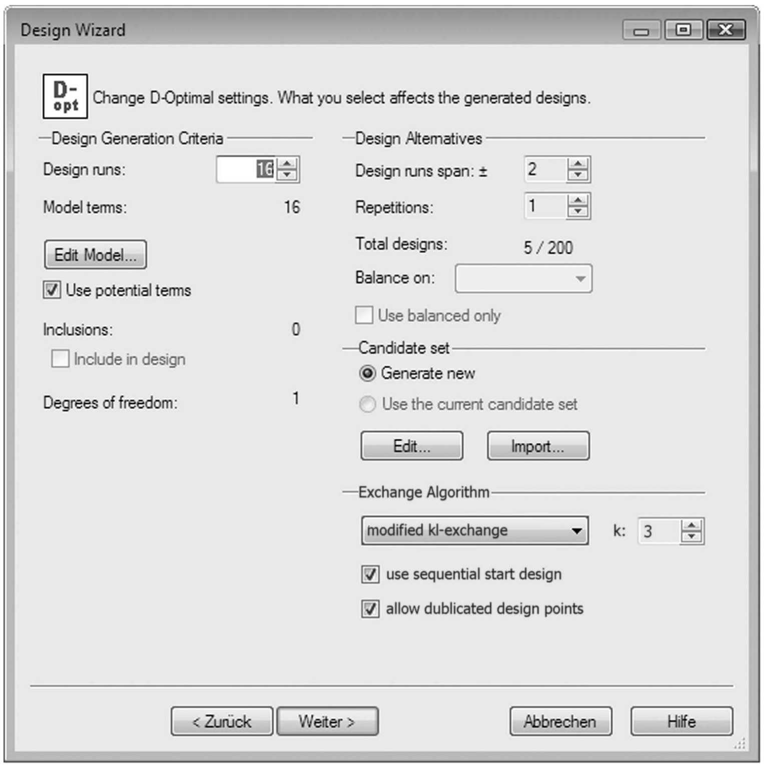
	Name	Abbr.	Units	Type	Use	Settings
1	flour	flo	g	Quantitative	Controlled	200 to 600
2	butter	but	g	Quantitative	Controlled	100 to 500
3	sugar	sug	g	Quantitative	Controlled	100 to 500
4	egg	egg		Quantitative	Controlled	0 to 3
5	baking powder	bak	g	Quantitative	Controlled	5 to 25
				Double-click	here	to

☐ Place constraints on the experimental region

Figure 4.1: Screenshot: MODDE Design Wizard — Factor Definition.

The next step in the wizard is the selection of the experimental objective described in section 1.2.6. Depending on this selection, MODDE suggests the possible design types and the advisable model. In our case we select a D-optimal design and come to the wizard page shown in figure 4.2.

On this page the user can give detailed specifications for the generated D-optimal design. This data is the input for the implemented algorithms. In the upper left corner, the user can select the number of desired design runs. This number has to be at least as high as the number of model terms; otherwise the degrees of freedom would not be high enough to create a design. In general, the number of degrees of freedom should be higher than five. On the previous page, the user selected the model for the investigation. With the button “edit model”, this model can be modified in the way that the user can add or delete terms. In most cases the use of a Bayesian modification is recommended and can be activated with the check-box “use potential terms”. If the user has entered some additional experiments which should be included in the design, the check-box “include in design” is available. On the right side of this wizard page, the user has the opportunity to define the number of design alternatives. As described in section 2.7, the creation of designs with a higher or lower number of runs than the desired one can lead to much better results. In addition to this, more than one design for each investigation should be performed to prevent the algorithm from picking a local optimum. Hence, the design run span gives the range for the number of design runs and the value “repetitions” creates the duplicates. If the factor definition contains qualitative factors, the design wizard gives the contingency to balance the design on one of the qualitative factors. A balanced design means that the same number of experiments is performed for each level of the factor. The next step is the generation of the candidate



Change D-Optimal settings. What you select affects the generated designs.

Design Generation Criteria
Design runs: 16
Model terms: 16
Edit Model...
[✓] Use potential terms
Inclusions: 0
[ ] Include in design
Degrees of freedom: 1

Design Alternatives
Design runs span: ± 2
Repetitions: 1
Total designs: 5 / 200
Balance on:
[ ] Use balanced only

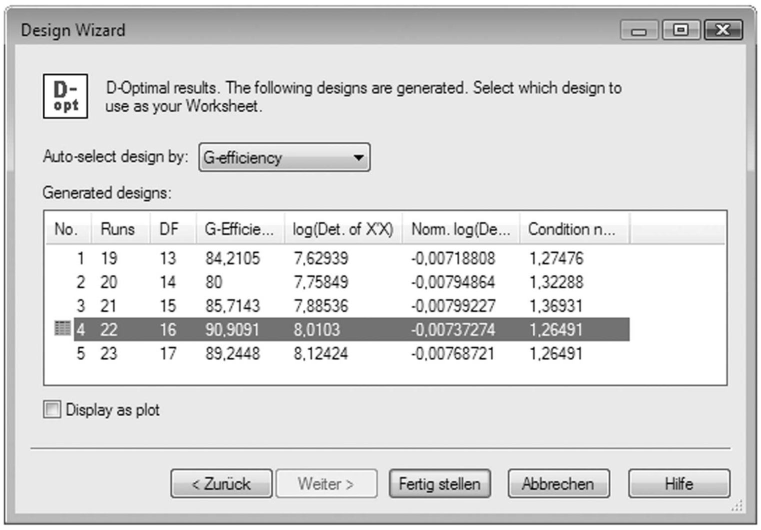
Candidate set
(●) Generate new
( ) Use the current candidate set
Edit... Import...

Exchange Algorithm
modified kl-exchange k: 3
[✓] use sequential start design
[✓] allow dublicated design points

Figure 4.2: Screenshot: MODDE Design Wizard — D-Optimal Specification.

set. In general, MODDE creates the candidate set depending on the factor definition and uses this data set for the calculation. Alternatively, the user can import an own candidate set or edit the suggested one. The last group of options, in the bottom right corner, is new for the reimplementation of the D-optimal algorithm and gives the user more options for the exchange procedure itself. The selection of the algorithm can be done problem-dependent and additional parameters, like the k-value for the modified kl-exchange or the normal k-exchange by Cook & Nachtsheim (1980), can be defined. Normally, a sequential start design is recommended, but in some cases, the user needs the opportunity to use the basic randomized start design to create other designs. The randomized function can be used by deactivating the check-box “use sequential start design”. This features is explained in detail in section 4.2.5. The last option in the design wizard is the question if the user wants to have duplicated design points in the design or not. If this point is deactivated, the algorithm only considers the remaining support points for an exchange. In comparison to this, the use of duplicated design points invokes a non-exhausting exchange procedure where all candidates are considered. In this case a design with replicated experiments is possible.

After all parameters are set, the algorithm is called and creates the desired number of designs. The designs are listed on the next page of the wizard, shown in figure 4.3. Depending on the four criteria: G-efficiency, the logarithm of the determinant, the normalized logarithm of the determinant, and the condition number, the user has to select the favored design.



D-Optimal results. The following designs are generated. Select which design to use as your Worksheet.

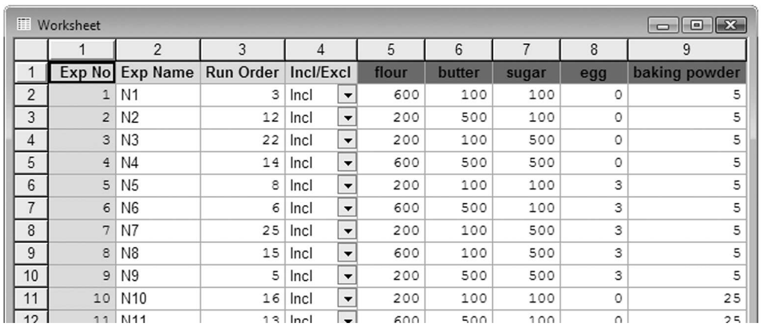
Auto-select design by: G-efficiency

Generated designs:

No.	Runs	DF	G-Efficie...	log(Det. of X'X)	Norm. log(De...	Condition n...
1	19	13	84,2105	7,62939	-0,00718808	1,27476
2	20	14	80	7,75849	-0,00794864	1,32288
3	21	15	85,7143	7,88536	-0,00799227	1,36931
4	22	16	90,9091	8,0103	-0,00737274	1,26491
5	23	17	89,2448	8,12424	-0,00768721	1,26491

Figure 4.3: Screenshot: MODDE Design Wizard — Evaluation Page.

The selected design is printed out on the worksheet. Figure 4.4 shows an example of a design with five factors. In general, the center-points and inclusions are part of this design and placed in the first and last rows. The factors are unscaled and presented in their normal range, as defined in the beginning of the design wizard.



	1	2	3	4	5	6	7	8	9
1	Exp No	Exp Name	Run Order	Incl/Excl	flour	butter	sugar	egg	baking powder
2	1	N1	3	Incl	600	100	100	0	5
3	2	N2	12	Incl	200	500	100	0	5
4	3	N3	22	Incl	200	100	500	0	5
5	4	N4	14	Incl	600	500	500	0	5
6	5	N5	8	Incl	200	100	100	3	5
7	6	N6	6	Incl	600	500	100	3	5
8	7	N7	25	Incl	200	100	500	3	5
9	8	N8	15	Incl	600	100	500	3	5
10	9	N9	5	Incl	200	500	500	3	5
11	10	N10	16	Incl	200	100	100	0	25
12	11	N11	13	Incl	600	500	100	0	25

Figure 4.4: Screenshot: MODDE Worksheet.

4.2.2 Integration & Program Flow

The core of the new D-optimal algorithm is placed in a class called *newDopt* with the source file *newDopt.cpp* and the header file *newDopt.h*. The class name is chosen in this way to implement the new code next to the existing version *Dopt.cpp*. The created files can be found in Appendix A.

In general, the design wizard starts the creation of the D-optimal designs by calling the public function *compute()* of the class *newDopt*. This class itself is inherited from the class *design* which is used inside MODDE for all design types. The parameters and settings from the wizard are sent to the D-optimal algorithm inside the two instances *pModelSpec* and *pDOptSpec*. The model specification contains detailed information about the factor constellation inside the model, the types of factors, and further information. The D-optimal specification contains the settings for the D-optimal algorithm, like the number of runs, the chosen algorithm, and so on. The general outline of the program flow can be seen in figure 4.5.

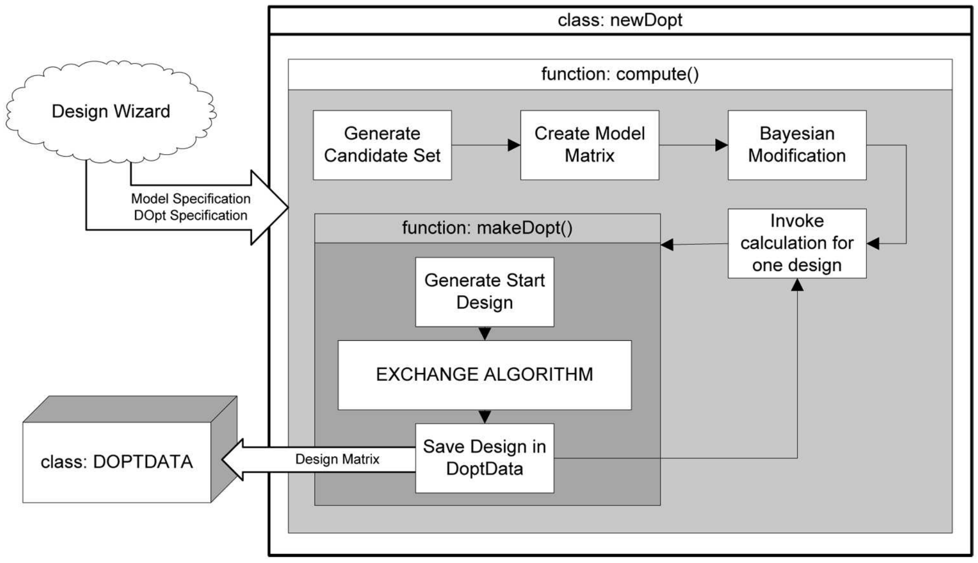


Figure 4.5: Program Flow I: *This figure shows the general program flow of the new implemented source code. With the specifications from the design wizard, the function compute() of the class newDopt is called. This function generates the extended model matrix and calls makeDopt() for each design. MakeDopt() selects the best design matrix with an exchange algorithm and saves the design in the class DOptData.*

The present thesis only deals with the algorithms to create D-optimal designs depending on an existing candidate set. Therefore, the generation of a new candidate set is implemented with a call of the old function *generateCandidateSet()*. This generated candidate set and the specifications from the design wizard are the basis for the following D-optimal algorithms. Depending on the model specification and the factor values in the candidate set, the first step is the generation of the model matrix for all candidates. As described in section 2.2.2, the model matrix is a $N \times p$ matrix for all N candidates, where each of the p model terms is represented in one column. In general, this calculation can be easily done with the data provided in the model specification and the simple candidate set. Only some special cases like mixtures or qualitative factors should be handled cautiously. These special cases are explained in detail in section 4.2.5. If the use of a Bayesian modification is selected in the design wizard, the model matrix is extended with the additional scaled potential terms. Which potential terms have to be added is model-dependent and is described in the following section. After the modification the model matrix contains all primary and potential model terms which can be used for the following calculations. Now, the function *makeDopt()* is called for each of the desired designs. Depending on the given design run span and the number of repetitions, the calculation is called until all designs are created. For each design, a new starting design is generated. The design can either be sequentially or randomly created depending on the selection in the design wizard. The start design is the basis for the following exchange algorithm and should be different for each generation. An explanation of the implemented exchange algorithms follows later in this section. Once the generation of a D-optimal design is finished, the data is saved in the class *DOptData*, and the calculation for the next design is invoked. Inside *DOptData* the design matrix and the efficiency depending on different criteria are saved. The class *DOptData* is already implemented in MODDE and therefore kept from the old implementation. The class is only modified to fit the requirements of the new approach. After all designs are created, the evaluation page of the design wizard pops up and lists the saved designs. After the selection by the user, the design is converted back into the unscaled data and printed out on the worksheet.

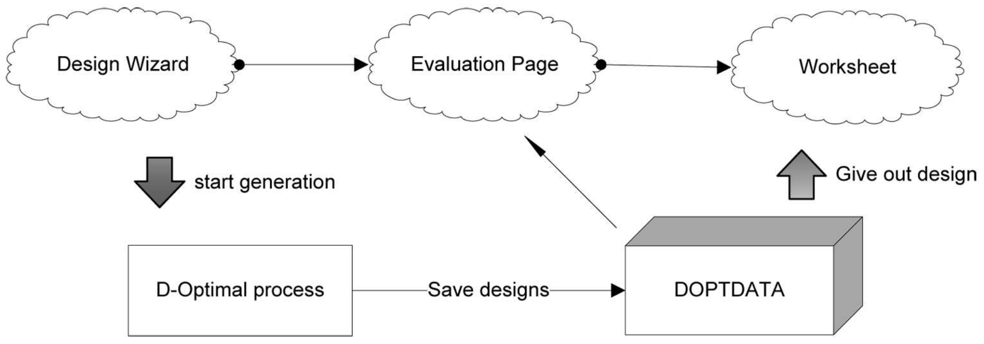


Figure 4.6: Program Flow II: *This picture shows the interaction between the normal user activities inside the design wizard and the D-optimal generation process. The wizard starts the generation, saves the designs in DoptData, and lists them in the evaluation page. The selected design is printed out on the worksheet.*

4.2.3 Variables

In order to find the optimal design matrix, different matrix calculations and exchange procedures are required. In this section we show the important member variables used for the calculation and define their types. Table 4.1 shows a list of these variables.

The class *UmPointer* is developed by Umetrics AB and represents a self- deallocating pointer which can be used for different types of variables. The types *FMatrix*, *FVector*, and *I2Vector* belong to a math library used in MODDE, where *F* stands for float and *I2* for integer. The keyword *matrix* is used for a multidimensional array with special functionality for matrix multiplications or modifications. The *vector* class is similar to this but has only one dimension.

As described in the first paragraph of this section, the specifications from the design wizard are saved in two instances, one for the model information and one for the D-optimal settings. They have the names *mpModelSpec* and *mpDOptSpec* and are accessible members in the whole *newDopt* class. An instance of the class *DOptData* is also implemented as a member variable to save the created designs and make them available for the worksheet.

The generated candidate set is saved in two different ways. First, we have the unscaled data set, called *mpUCS*, taken as a float matrix from the existing code. Here, each factor represents one column and each candidate one row of the matrix. This is the

variable	**type**	**description**
mpDOptData	DOptData*	used to save the created designs
mpModelSpec	ModelSpec*	model specification
mpDOptSpec	DOptSpec*	D-optimal specification
mpUCS	UmPointer<FMatrix>	unscaled candidate set
mpCS	UmPointer<FMatrix>	model-dependent candidate set
mpXpri	UmPointer<FMatrix>	matrix for primary terms
mpXpot	UmPointer<FMatrix>	matrix for potential terms
mpDispersion	UmPointer<FMatrix>	dispersion matrix $(X'X)^{-1}$
mpDelta	UmPointer<FMatrix>	delta matrix for all couples
mpDxixj	UmPointer<FMatrix>	variance matrix for all couples
mpDxi	UmPointer<FVector>	variance for the design points
mpDxj	UmPointer<FVector>	variance for the support points
mpvX	UmPointer<I2Vector>	indices of the design points
mpvY	UmPointer<I2Vector>	indices of the support points

Table 4.1: List of Variables.

raw data which is used to create the worksheet at the end of the process. The second matrix is the normal candidate set *mpCS*. Depending on the settings made during the factor definition, this matrix is scaled with one of the methods from section 2.6. This simple candidate set is not used in the D-optimal process any more, but instead of this the extended model matrix, which contains the primary and potential terms for all candidates, is used. Therefore, the matrix *mpCS* is expanded with the model-dependent terms and the outcome of the Bayesian modification. From this point on, *mpCS* is the complete model matrix which is used for the D-optimal selection procedure. The matrix has the following form:

const	primary terms	potential terms
1	x x x x x x	z z z z z z
1	x x x x x x	z z z z z z
1	x x x x x x	z z z z z z
1	x x x x x x	z z z z z z

Figure 4.7: Layout of the Extended Model Matrix: *The first column of this matrix is the constant model coefficient β_0 with the value 1. The following columns contain the scaled coefficients of the model, depending on the setting in mpModelSpec. At the end of the matrix, the potential terms from the Bayesian modification are added.*

To keep the data and memory usage as low as possible, we use the candidate set *mpCS* as the basis for most calculations. If a subset of this matrix is needed, the data is accessed over the index of each candidate. For example, the current design matrix is represented by the integer vector *mpvX* which contains the indices of the affected candidates. Instead of exchanging the whole candidates or changing the order of a matrix, a simple modification of the integer vector can be used. In contrast to the design vector *mpvX*, the vector *mpvY* represents the indices of the support points. An exchange can now be realized with a simple swap of two integer values.

In addition to these essential variables, a lot of parameters and auxiliary variables are used in order to solve independent problems. These variables are explained when they are used to keep this section as compact as possible.

4.2.4 Exchange Algorithms

Depending on the findings from chapter 3, the following four exchange algorithms are suggested for an implementation:

- Fedorov Algorithm,
- Modified Fedorov Algorithm,
- *k*-Exchange Algorithm,
- Modified *kl*-Exchange Algorithm.

The Fedorov algorithm from 1972 and its modified version by Cook & Nachtsheim (1980) showed the best results over all tests and should be used for small data sets. With an increasing number of factors, the *k*-exchange algorithm and the modified *kl*-exchange algorithm created comparable designs with a much higher computational efficiency. Hence, these two algorithms should be used for huge data sets.

The DETMAX algorithm by Mitchell (1974) and Johnson & Nachtsheim's (1983) the basic *kl*-exchange algorithm are not considered for an implementation. The reason for this is the similarity in the created designs compared with the *k*-exchange algorithm and the modified *kl*-exchange algorithm. In addition to this, the DETMAX algorithm has a complete different approach and is not familiar with the other solutions. The four selected algorithms can be implemented next to each other while using the same fundamentals like the calculation of the Δ-value or the exchange procedure.

Fedorov Algorithm

Depending on the calculated dispersion matrix $(X'X)^{-1}$, the function *calculateDeltaMatrix()* is called. Inside this function the necessary variance of prediction for the design and support points is calculated and the complete matrix of Δ-values is filled for all possible couples. In addition to this, the indices of the suggested design and support points with the maximum Δ-value are saved in the member variables *mixi* and *mixj*. This data is used to start an exchange between the two integer vectors *mpvX* and *mpvY*. A swap of the values creates a new design matrix, and corresponding to this, the dispersion matrix has to be updated. This simple flow of actions is repeated until the function *calculateDeltaMatrix()* does not find a Δ-value bigger than *mfepsilon*. In this implementation the threshold value ϵ_{fed} is set to 10^{-6}. The following listing shows the C++ routines for the Fedorov algorithm.

```
if(meAlgorithm == Fedorov)
{
    //calculate dispersion matrix (Xd'*Xd)^-1 for the first time
  calculateDispersionMatrix();
    //calculate the delta values for all couples and find biggest
  calculateDeltaMatrix();
    //loop until no delta bigger e is found
  while((mfmaxdelta >= fepsilon) && (icount < imaxloops)){
      //exchange the from calculateDeltaMatrix() found couple
    exchangePoints(mixi, mixj, mballowdublicates);
      //update the dispersion matrix with the new point
```

```cpp
      updateDispersionMatrix(mixi);
        //calculate all delta values with the new dispersion matrix
      calculateDeltaMatrix();
    }
}
```

C++ Listing 4.1: Fedorov Algorithm.

Modified Fedorov Algorithm

Instead of implementing the modified Fedorov algorithm as an autonomous routine, the knowledge from section 3.1.5 is used. As Johnson & Nachtsheim (1983) noted, the modified Fedorov algorithm can be seen as a special case of the k-exchange procedure. Instead of considering k of the design points, the modified Fedorov algorithm checks all n points for a possible exchange. Based on this, the implementation can be done with a simple call of the normal k-exchange algorithm with $k = n$. Again, the procedure is shown in a C++ listing.

```cpp
if(meAlgorithm == ModifiedFedorov)
{
  mik = miruns;
  meAlgorithm = KExchange;
}
```

C++ Listing 4.2: Modified Fedorov Algorithm.

k-Exchange Algorithm

The k-exchange is similar to the Fedorov procedure but only considers k of the design points for an exchange and performs up to k exchanges during each iteration. The algorithm also starts with the calculation of the dispersion matrix, but instead of calculating the Δ-values for all couples, the variance of prediction for the design points is calculated. Depending on this float vector *mpDxi*, the indices of the k design points with the lowest variance of prediction are saved. The calculation of the variance vector and the evaluation is done inside the function *calculateDxiVector(mik)*, where *mik* represents the k-value. The selected indices are saved in the integer vector *mpvkListLow.* Inside the following loop shown in line 12 of listing 4.3, the Δ-values for the k points are calculated. In this case the function *calculateDeltaRow(m)* calculates the row of the Δ-matrix for the corresponding candidate m, where m is one of the k considered design points. If a suggested exchange is found inside *calculateDeltaRow(m)*, the corresponding couple is exchanged and the dispersion matrix has to be updated. If none of the k design points can be exchanged with a better point, the algorithm breaks. Otherwise, the selection of k design points is started again.

```cpp
1  if(meAlgorithm == KExchange)
2  {
3      //calculate dispersion matrix (Xd'*Xd)^-1 for the first time
4    calculateDispersionMatrix();
5      //bool value to check if one of the deltas was positive (>fepsilon)
6    bool bposdelta = false;
7    while(true){
      //calculate d(xi) for all points in the design and find k values
      //with highest variance of prediction
    calculateDxiVector(mik);
      //step through the k points
    for(int i=1;i<=mik;i++){
        //get candidate set index of the point i
      int m = (*mpvkListLow)[i];
        //calculate delta values for design points and all points
        //outside the design
      calculateDeltaRow(m);
        //if positive delta found
      if((mfmaxdelta >= fepsilon)){
          //exchange the from calculateDeltaRow(m) found couple
        exchangePoints(mixi, mixj, mballowdublicates);
          //update the dispersion matrix for next run
        updateDispersionMatrix(mixi);
        bposdelta = true;
      }
    }
      //if none of the k points has a positive delta value break
    if(!bposdelta){
      break;
    }else{
      bposdelta = false;
    }
  }
}
```

C++ Listing 4.3: *k*-Exchange Algorithm.

Modified *kl*-Exchange Algorithm

The modified *kl*-exchange algorithm is more complex than the previously discussed but is based on the same fundamentals as the Fedorov exchange. Between the *k* design points with the lowest variance of prediction, the algorithm finds the best exchange by checking a reduced candidate list of *l* support points. The detailed flow can be seen in listing 4.4. Again, the algorithm starts with the calculation of the dispersion matrix and calls the function *calculatekDeltaMatrix(mik,true)* to find the best exchange. With the given *k*-value as a parameter, the function selects the *k* design points with the lowest variance of prediction and calculates the rows of the Δ-matrix for the suggested points. If necessary, the *k*-value is extended as described in section 3.1.6. The support points for this calculation are taken from the full list *mpvY* without any constraints.

In some cases the algorithm does not find a positive Δ-value between the suggested design points. If this happens before the exchange procedure is started, we have to check all possible couples like the normal Fedorov procedure would do. Afterwards, the algorithm exchanges the best couple and updates the dispersion matrix. From now on, the algorithm uses a reduced candidate list to prevent inadvisable exchanges. Therefore, the variance of prediction for all design and support points is calculated and used by the function *calculateAverageVaraiance()* to get the average value. Now, the existing list of support points *mpvY* is saved in a backup vector and reduced by the function *reduceCanidateList(true)*. In this case the boolean parameter decides if the support points with a higher variance than the average one or the support points with a lower variance than the average one are kept in the list. Then, the Δ-calculation is performed with the reduced list of support points. The true boolean value in the function call *calculatekDeltaMatrix(mik,true)* achieves a second variance check during the Δ-calculation. Only couples whose support point has a higher variance of prediction than the design point are considered for an exchange.

If no positive Δ-value is found during this check, the algorithm switches the list *mpvY* by restoring the old list and calling the function *reduceCanidateList(false)* which selects only the points with a variance of prediction less or equal the average one. The function *calculatekDeltaMatrix(mik,false)* is called without a variance check and finds the best possible exchange. Afterwards, the list of support points is restored and the next iteration begins. If the whole process does not find a positive Δ, the algorithm breaks.

```
if(meAlgorithm == ModifiedKLExchange)
{
    //average variance of prediction over the whole candidate set
  mfavVar=0.0;
    //value and id of the last exchanged point to prevent a
    //reverse exchange
  milastexval = 0;
  milastexid = 0;
    //calculate the dispersion matrix (Xd' * Xd)^-1 for the first time
  calculateDispersionMatrix();
    //calculate the delta values for the k design points (variance of
    //the candidate > variance of design point) and find biggest delta
  calculatekDeltaMatrix(mik,true);
    //if no max delta is found during the k design points check again
    //for all points (k=miruns, no variance criteria)
  if(mfmaxdelta<fepsilon){
    calculatekDeltaMatrix(miruns,false);
  }
    //loop while positive deltas are found
  while((mfmaxdelta >= fepsilon) && (icount < imaxloops)){
      //exchange the from calculatekDeltaMatrix() found couple
    exchangePoints(mixi, mixj, mballowdublicates);
      //update the dispersion matrix (Xd'*Xd)^-1
    updateDispersionMatrix(mixi);
      //calculate the vector with the d(xj) and d(xi) function
      //(variance of prediction) and get the average variance
      //of these vectors
    calculateDxjVector();
    calculateDxiVector();
    calculateAverageVaraiance();
      //save the list of candidates (mpvY) and create a reduced
      //the list to speed up the algorithm. the reduced list
      //contains only the point that have a variance of prediction
      //> the average one from the last iteration
    saveCanidateList();
    reduceCanidateList(true);
      //for the next run: calculate the delta values for the k design
      //points (variance of the candidate > variance of design point)
      //and find biggest delta
    calculatekDeltaMatrix(mik,true);
      //restore the before saved candidate set (mpvY)
    restoreCanidateList();
      //if no max delta was found in the reduced candidate set, take
      //the candidates that haven't been considered in the
      //investigation, calculate the deltas and restore the candidate
      //set (no variance criteria)
    if(mfmaxdelta<fepsilon){
      reduceCanidateList(false);
      calculatekDeltaMatrix(mik, false);
      restoreCanidateList();
    }
  }
}
```

C++ Listing 4.4: Modified *kl*-Exchange Algorithm.

Delta-Calculation

All of the four algorithm call a slightly different function to calculate the needed Δ-values. Listing 4.5 shows, e.g., the function *calculateDeltaMatrix()* which is used by the Fedorov algorithm. The function calculates the variance of prediction of the design and the support points with the functions *calculateDxiVector()* and *calculateDxjVector()* and uses this data to calculate the Δ-values for all possible couples. The values are saved in the matrix *mpDelta* and the coordinates of the couple with the biggest Δ-value are saved in *mixi* and *mixj*.

```
void newDopt::calculateDeltaMatrix()
  //calculate all delta values for all points in the design
{
    //reset ids of possible exchange points, reset maxdelta
  mixi = 0;
  mixj = 0;
  mfmaxdelta = -99.9;
    //get d(xi) vector for design points
  calculateDxiVector();
    //get d(xj) vector for support points
  calculateDxjVector();
    //step through all couples of xi and xj and calculate their deltas
  for(int i = 1+miIncl; i<=miruns;i++){
    for(int j = 1; j<=mpvY->NVal();j++){
        //get d(xi,xj) for this couple
      (*mpDxixj)[i][j]= calculateDxixj(i,j);
        //get delta(xi,xj) for this couple
      (*mpDelta)[i][j]= calculateDelta(i,j);
        //check if this delta is maximum and save coordinates
      if((*mpDelta)[i][j]>mfmaxdelta){
        mfmaxdelta = (*mpDelta)[i][j];
        mixi = i;
        mixj = j;
      }
    }
  }
}
```

C++ Listing 4.5: Function: *calculateDeltaMatrix()*.

The other implemented algorithms have similar solutions but the number of considered couples is smaller. The data is also saved inside *mpDelta* even if not all storage positions of the array are used. The connection between the candidate set *mpCS*, the design vector *mpvX*, the support vector *mpvY*, the variance vectors *mpDxi* and *mpDxj*, and the Δ-matrix *mpDelta* is shown in figure 4.8. The current design is realized by the use of the indices of the candidates. Each of these candidates has a variance of prediction which is saved in *mpDxi* or *mpDxj*. These float values are also connected by the indices of the candidates. The Δ-matrix *mpDelta* combines the design and support points to couples and uses the corresponding variances for the calculation.

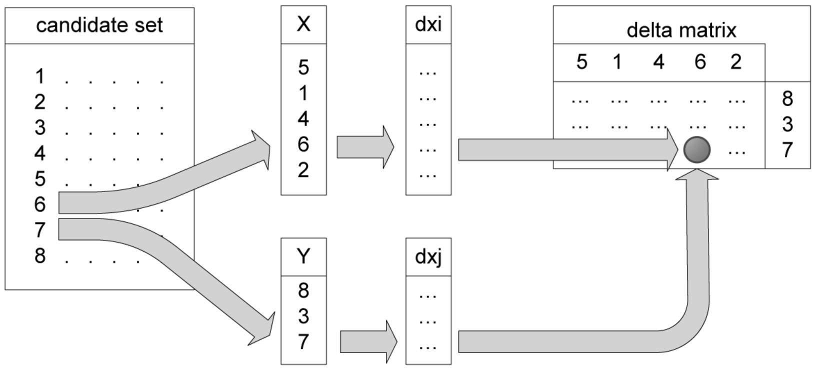


Figure 4.8: Use of Indices to Calculate the Delta-Matrix: *This picture shows the connection between the candidate set, the design vectors, and the Δ-matrix. The vectors X and Y contain the indices of the candidate points which are currently in the design X or in the list of support points Y. The arrow show an example how the Δ-value for the couple (6,7) is calculated.*

4.2.5 Features

Besides the normal exchange procedure, the implementation of the D-optimal process integrates some additional features. Those features also have been implemented in the old version but are completely new implemented in the present thesis. In this section some of them are explained in detail, and in some cases the used C++ source code is provided and outlined to clarify the approach.

Inclusions

An inclusion is an already performed experiment which should be integrated into the optimal design matrix. Theses experiments are entered by the user and have to be present in the created worksheet. Therefore, theses candidates have to be considered during the generation of the start design and during the exchange procedure. Letting o be the number of included candidates, the easiest way is to place the inclusions on top of the candidate set and handle the following calculations with an offset of o points. This means, when the starting design is generated the first o points are taken from the candidate set followed by the randomly or sequentially included points. The exchange procedure works with the same concept. For the first o candidates no Δ is calculated and therefore, the points are not considered for an exchange. Attention should be paid to the calculation of the dispersion matrix and efficiency. Here, the inclusions have to be considered because they are part of the evaluated design. Table 4.9 shows a simple example of a candidate set with inclusions and center-points. In this implementation the integer variable *miIncl* contains the offset value.

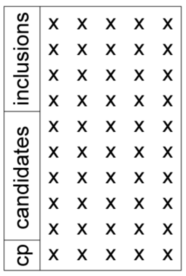


Figure 4.9: Layout of the Candidate Set: *The candidate set contains the inclusions in the first rows. These candidates will not be exchanged and are present in the final worksheet. All following candidates are normal support points. The last row of the matrix is the center-point.*

Center-Points

A center-point is a candidate which has the mean value of the upper and lower bounds of each factor definition as a value. Above all, we have to differentiate between two different applications of center-points. On the one hand, we have the additional center-points which are normally used to test the robustness of an already created D-optimal design. Therefore, the additional center-points, normally three, are not considered during the normal exchange procedure and they are added after the generation. On the other hand, one center-point is included in the candidate set as a possible candidate. This candidate is handled like a normal support point and is considered during the start design and the exchange process. In order to keep the candidate set clear, the center-point is normally added in the last row. This makes the selection of the additional center-points easy and the candidate can be used as a normal support point. The placement of the center-point can also be seen in figure 4.9. To add the robustness testing center-point to the final design, the function *addCentroids()* is called. Here, the design matrix is expanded with the desired number of center-points taken from the last row of the unscaled candidate set *mpUCS*.

Bayesian Modification

The Bayesian modification, described in section 2.8, is realized by a call of the function *addPotentialTerms()*. Inside this function the matrix with the primary terms X_pri is analyzed, and depending on the model and the factor definitions, the potential terms from table 4.2 are added to the matrix X_{pot}.

After the creation of the matrix, the Bayesian scaling is applied. Listing 4.6 shows the C++ code of the scaling procedure implemented in the function *doBayesianScaling()*. The first step is to calculate the α-matrix which contains the least square coefficients of X_{pot} on X_{pri}. Depending on this calculation, the residual of the regression is saved in the matrix R. The matrix calculations are solved with the math library of MODDE. The needed column-wise maximum and minimum values of R can easily be extracted as a float vector by the use of the class *FMatrix*. These two float vectors, *Rmax* and *Rmin*, are used for the final calculation of the Z-matrix *mpZ* which is defined in equation 2.32 of section 2.8.2. In order to finish the Bayesian modification, the new Z-matrix is attached to the candidate set with the function *updateCS()*. In general, the τ-value in this implementation is set to one. With this default the Q-matrix is a simple diag-

model	**process factors**		**mixture factors**	
	count	potential terms	count	potential terms
linear		all interactions	up to 20	all interactions
		all squares^{a,b}	up to 20	all squares
interation		missing interactions		
		all squares^{a,b}		
quadratic		missing squares		
		all interactions		
	up to 10	all cubes^{b}	up to 12	all cubes
	up to 6	all three-way inter.	up to 6	all three-way inter.
special cubic			up to 6	all three-way inter.
cubic			up to 6	all three-way inter.

^{a} only included if no interaction terms are added or the model contains a square value

^{b} factor must be quantitative or multilevel

Table 4.2: List of Potential Terms: *This table shows the selection of the potential terms based on the assumed model and the type of the used factors. We differentiate between process and mixture factors in general. Depending on the amount of factors, different potential terms are added to the matrix.*

onal matrix with ones in the diagonal if the column belongs to a potential term. The simple construction of this matrix is shown in listing 4.7. During the calculation of the dispersion matrix, this matrix is considered, as shown in equation 2.34.

```
void newDopt::doBayesianScaling()
  //alpha = (Xpri' * Xpri)^-1 * Xpri' * Xpot
  //R = Xpot - (Xpri * alpha)
  //Z = R / (max(R) -min(R))
{
    //calculate the needed transpose of Xpri
  for(int j = 1; j<=mpXpri->NRow();j++){
    for(int k = 1; k<=mpXpri->NCol();k++){
      (*mpXpriT)[k][j] = (*mpXpri)[j][k];
    }
  }
    //calculate alpha
  mpAlpha.reset(new FMatrix((*mpXpriT)*(*mpXpri)));
  mpAlpha->MInv();
  (*mpAlpha) = ((*mpAlpha)*(*mpXpriT))*(*mpXpot);
    //calculate R
  mpR.reset(new FMatrix((*mpXpot) - ((*mpXpri)*(*mpAlpha))));
    //get max of each column of R
  FVector Rmax = mpR->MaxCol();
    //get min of each column of R
  FVector Rmin = mpR->MinCol();
    //calculate Z
  mpZ.reset(new FMatrix(mpR->NRow(),mpR->NCol(),0.0,1));
  for(int i=1; i<=mpR->NRow();i++){
    for(int j=1; j<=mpR->NCol();j++){
      (*mpZ)[i][j] = (*mpR)[i][j] / (Rmax[j]-Rmin[j]);
    }
  }
}
```

C++ Listing 4.6: Function: *doBayesianScaling()*.

```
void newDopt::createQ()
  //creates a matrix with ones in the diagonal if the column
  //belongs to a potential term
{
    //square matrix
  mpQ.reset(new FMatrix(micols,micols,0.0,1));
    //for potential terms add ones in the diagonal,
    //all others are zeros
  for(int i=miprimterms+1;i<=(miprimterms+mipotterms);i++){
    (*mpQ)[i][i]=1;
  }
}
```

C++ Listing 4.7: Function: *createQ()*.

Duplicated Design Points

The previous chapters explains the difference between an exhausting and a non-exhausting search method for the selection of the support points. Because of different aims during the generation of a design, the implementation gives the user the possibility to select whether duplicated points should be avoided or not. The default option is not to use duplicated experiments. The two main differences between the approaches can be seen during the creation of the start design and the exchange procedure. If no duplicated experiments are allowed, the start design is generated by picking the points out of a list of the remaining support points. With allowing repeated experiments, the support points are selected from the full set of candidate points. During the exchange procedure the vector of support points *mpvY* contains either the remaining candidates which are not considered in the current design or the full candidate set. If an exchange is performed, normally both points from the design vector *mpvX* and the support vector *mpvY* are exchanged. But if duplicated points are allowed in the design, the support vectors must not be changed. Only the design is updated with a new point. Listing 4.8 shows the function which is used to perform an exchange.

```
void newDopt::exchangePoints(int xi, int xj, bool bdublicates)
//exchanges two points (design point from X with support point
//from Y)
{
    //save the exchange to prevent reverse exchange
  milastexval = (*mpvX)[xi];
  milastexid = xi;
    //exchange the point xi from X with the point from Y
  if(!bdublicates){
    int temp = (*mpvY)[xj];
    (*mpvY)[xj]=(*mpvX)[xi];
    (*mpvX)[xi]=temp;
    //if duplicates are allowed, just copy the candidate
    //into the design without changing Y
  }else{
    (*mpvX)[xi]=(*mpvY)[xj];
    }
}
```

C++ Listing 4.8: Function: *exchangePoints(*int *xi,* int *xj,* bool *bdublicates).*

If the option “allow duplicated points” is activated in the design wizard, we have to consider two checks to avoid problems during the design creation. First, we have to check if the entered inclusions are already parts of the candidate set. In this case we have to delete one of the duplicated experiments. The second check is performed to examine if the selected number of design runs is higher than the amount of candidates. This means that the number of support points must be high enough to fill the starting design with the desired number of runs. If this is not possible, the user must consider duplicated experiments in the design or add further candidates.

Start Design

The randomized start design is based on a repeated exchange process between the design vector *mpvX* and the vector of support points *mpvY*. Instead of just shuffling a list of points, the use of the exchange procedure can handle the duplicated option, mentioned above. First, the list of design points is filled continuously with candidates until the number of desired runs is reached. Then, the list of support points is initialized with the remaining points or with all candidates depending on the option in the wizard. The exchange procedure now randomly generates indices for both lists and takes care of the exchanges. During this process the fixed inclusions in the first rows of the start design are considered.

If we use the sequential start design, as described in section 3.2, the randomized design can be used as a basis for the generation process. The design consists of three types of support points which are added consecutively. The first group are the o inclusions which have to be in each design. These points are followed by q randomized points, where the integer q itself is a randomized number. The missing $n - o - q$ design points are chosen from the list of support points to increase the determinant most. Figure 4.10 shows the layout of the sequential start design.

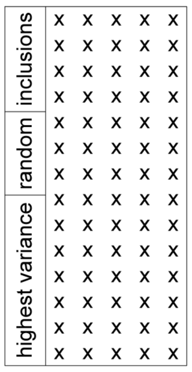


Figure 4.10: Layout of the Sequential Start Design: *This table shows the general structure of a sequential start design. The inclusions and random points are taken from a randomized design. The missing support points are added depending on their variance of prediction.*

With an already generated randomized design with the o inclusions in the first row, the first $o+q$ design point can be used for the selection process of the first additional point. The dispersion matrix of the design with these points has to be calculated, and corresponding to this, the point with the highest variance of prediction can be found and added to the design. This process is repeated until the desired number of design runs is reached. The source code for the creation of both start designs can be found in appendix A.

Mixture Factors

In order to handle mixture designs, the candidate set is modified by removing one of the factor definitions. Considering, e.g., a design with three mixture factors, we only need to perform the D-optimal process with two of the definitions because they sum up to 100% and therefore the missing factors can be calculated after the exchange procedure. If the factors have the values a=0.3 and b=0.5, consequently the factor c must be equal to 0.2. This modification speeds up the algorithm, but the excluded factor has to be considered during the creation of the primary and potential terms. To handle this problem, the index of the factor is saved in the variable *miexclfactor*. Each term is checked if it contains the excluded factor and if needed the indices are fixed inside the functions *addPrimaryTerms()* and *addPotentialTerms()*.

Qualitative and Quantitative Factors

Qualitative factors have at least two discrete values and can also be used in D-optimal designs. Inside the candidate set these factors are represented in distinct columns depending on the number of levels. This means the data is displayed in a binary coding where x columns can represent 2^x discrete values. A qualitative factor with, e.g., three levels A, B and C uses two columns to represent the data in a valuable way. Table 4.3 shows the number of needed columns for different factor levels. Building the model matrix depending on the normal model specification is difficult because each of the qualitative factors each has an expanded count. With the use of offset values this problem is also solved in the functions *addPrimaryTerms()* and *addPotentialTerms()*.

Levels	Columns
2	1
3-4	2
5-8	3
9-16	4

Table 4.3: Columns Needed to Handle Qualitative Factors.

Balanced Designs

Using the qualitative factors mentioned above, the user has the opportunity to use the D-optimal process only for the creation of balanced designs. These designs contain the same number of experiments for each level of a qualitative factor. Considering, e.g., a design with a qualitative factor which has three discrete values A, B, and C, a balanced design with $n = 12$ runs contains four experiments with value A, four with value B, and four with value C. The exchange procedure of the D-optimal algorithms does not automatically create balanced designs, but with the function *IsBalanced(PtrI2Vector pDesignN, std::map<int,int> *pBalancedQualSetting)* a generated design vector can be analyzed. If the option "use balanced only" is activated in the design wizard, the evaluation page only shows the created designs which fulfill the balancing criteria. In order to avoid the calculation of design which cannot be balanced due to their number of runs, a check before the call of the function *makeDopt()* is performed. A design with a desired number of runs which is not a multiple of the amount of discrete factor values cannot be balanced. In our example above, only designs with $n = z * 3$ design runs can be balanced. If more than one qualitative factor is used in the design, the user has to select the one to balance on in the wizard. The number of discrete values and the index of the factor is saved in the variable *std::map<int,int> *pBalancedQualSetting* which is used for the verification of the designs.

Irregular Experimental Region

An irregular experimental region can be handled with the D-optimal implementation like a normal symmetrical area. Instead of a candidate set which uses the upper and lower bounds of each factor, the defined constraints are used to change the candidate set. Depending on this changed candidates, the D-optimal algorithm performs a normal exchange until the optimal design matrix is found.

Chapter 5

Summary & Conclusions

5.1 Summary

The aim of the present thesis is the development and documentation of a D-optimal process to create designs which fulfill different criteria. In Chapter 1 and 2, the basic principles and the need for such D-optimal designs are shown and exemplified. With the use of the model concept and simple statistical design, the reader has the opportunity to get started with the concept of DoE. We clarify that normal designs are not applicable in situations with, e.g., an irregular experimental region or when already performed experiments have to be included in the design. To take up our example from the preface in chapter 1, the baking of a cake can now be analyzed with the use of DoE, e.g., with a full factorial design. To expand this problem formulation for a D-optimal design, we simply have to consider some recipes of already baked cakes which have to be included in the design.

The D-optimal process is used to select the optimal combination of experiments out of all possible designs. This selection process depends on the given criteria from section 2.3 and is solved by a computer algorithm. A goal before the implementation can be started is the selection of a suitable exchange algorithm. Chapter 3 deals with this topic and explains and analyses six different algorithms which differentiate in the computational efficiency and the quality of the created designs. The outcome of this comparison is the implementation of four different algorithms, the Fedorov algorithm, the modified Fedorov algorithm, the k-exchange algorithm, and the modified kl-exchange algorithm. All of these exchange approaches are problem-dependent, and therefore the choice of only one algorithm to fulfill all requirements is impossible. The implemented Fedorov and modified Fedorov algorithm create the best designs considering the D-optimality but are very time-consuming. Therefore, the k-exchange should be applied if the number of factors reaches a level of 10 or more. In this case the Fedorov algorithm is also adaptable, but the calculation needs much time. Considering the advice to create more than one design and select the best one, the k-exchange creates normally comparable result. In some cases the k-exchange algorithm shows weakness when, e.g., more than k design points have the same lowest variance of prediction. To avoid this, the modified kl-exchange algorithm developed during the present thesis should be applied. In general, this approach needs a longer computing time than the k-exchange algorithm but generates better results for huge data sets with, e.g., 15 factors. Overall, the advice which algorithm to use is difficult and one of the reasons why up to four of the algorithms are implemented during the present report.

In Chapter 4 the implementation of the D-optimal process is shown. With the use of flow diagrams and code examples, the single exchange algorithms and features are explained. A comparison of the final implementation and the previous requirements follows in the next section.

In general, we advice the use of a Bayesian modification as described in section 2.8. With this approach the dependency on an assumed model is reduced and the created designs gain more flexibility. This means that the effect of an incorrect model can be compensated and a suitable design will be created.

5.2 Conclusions

The requirements, given in section 4.1.2 of the present thesis, are to develop a D-optimal algorithm which handles a set of parameters from the design wizard of MODDE and creates designs with a high computational efficiency and a D-optimality which is comparable with the previous implementation. The old version of the D-optimal process has a limited number of factors, and therefore a comparison is only possible for data sets with up to 10 factors. Table 5.1 shows a comparison of the D-optimality of designs created by both implementations. The data is based on the case study from section 3.3 and is created with the new and old implementation in MODDE. Overall, the designs created with the new implementation are at least as good as the old ones. Admittedly, we have to observe that the selection of the best algorithm has to be performed by the user in the new implementation.

	5 factors	7 factors	9 factors
new implementation	20.47[a]	43.74[b]	75.36[c]
old implementation	20.47	43.74	75.32
new implementation (Bayesian)	19.46[a]	43.41[b]	74.77[c]
old implementation (Bayesian)	19.46	43.36	74.77

[a] Fedorov, modified Fedorov, k-exchange or modified kl-exchange

[b] Fedorov or modified Fedorov

[c] modified Fedorov

Table 5.1: Comparison of the Old and New Implementation: *This table shows the logarithm of the determinant of designs created by the new and old implementation of the D-optimal approach inside MODDE. Depending on an interaction model with 5, 7, or 9 factors, up to 10 designs are created. The table shows the best result of each approach. For the new implementation the selected algorithm is given in the footnote.*

In addition to this, the new implementation is able to handle problem formulations with 15 to 20 factors. In general, even higher numbers of factors can be considered with the use of the k-exchange or modified kl-exchange, but the calculation needs a long time. As described in the requirements in section 4.1.2, the new implementation has to handle the parameters from the design wizard of MODDE and create the design on the worksheet of the program. This behavior is realized in the final implementation and is also extended with additional parameters for the exchange procedure. The design wizard gives now the possibility to select the type of the start design and the desired algorithm and allows selecting if duplicated support points should be considered. During the selection process the application of a Bayesian modification is possible. The additional features demanded by Umetrics AB like handling of inclusions, mixture designs, quantitative factors et cetera are all implemented and explained in detail in section 4.2.5 of the present thesis.

5.3 Limitations

The developed D-optimal process has no real limitations considering the requirements from section 4.1.2. But during the testing of the algorithm one weakness of the modified *kl*-exchange was found. The algorithm is not able to handle designs which have a too low number of degrees of freedom. If the number is below five, the created designs have a weak D-optimality and should not be used. In this case, the use of another algorithm is advised. In general, the design wizard warns the user for all four algorithms if the number of degrees of freedom is below five and designs should not be created under these conditions, even if the other algorithms can handle the problem. To avoid a low number of degrees of freedom we have to increase the number of design runs or remove model terms.

A restriction of the D-optimal process in general is the risk that the algorithm can trap into a local optimum and stop the selection process before the optimal matrix is found. To avoid this problem, the creation of more than one design is advised in any case. Between the created designs the user can select the design with the best quality.

5.4 Future Work

The selection of the used algorithm has to be performed by the user inside the design wizard. This manually process could be automated based on a complexity analysis of the different algorithms implemented in MODDE. This analysis must contain a huge set of problem formulations and evaluate which algorithm should be used to find the D-optimality for the given situation. The present thesis gives the base for such an investigation which can easily be performed with the MODDE software. The outcome of such an analysis could be a suggestion of two algorithms for each problem-formulation. The first suggested algorithm should be selected in order to create the design with the best efficiency or D-optimality. In most cases this will be the Fedorov or modified Fedorov, but also other results are possible. The second suggestion should be the algorithm which can be used to create acceptable designs in an admissible time.

An extension of the new implementation can be done in the area of balanced designs. The user has the possibility to target the D-optimal process to create those balanced designs if the problem formulation contains qualitative factors. Currently, MODDE creates a wide range of designs and selects only the balanced ones out of this pool. In general, this approach can be modified by influencing the generation of the single designs with respect to a blocking option. Possible reference for such a modification are a current study of Goos & Donev (2005) or the publication made by Atkinson & Donev (1989).

References

Atkinson, A. C. & Donev, A. N. (1989), 'The Construction of Exact D-optimum Experimental Designs with Application to Blocking Response Surface Designs', *Biometrika* **76**(3), 515–526.

Box, G. E. P., Hunter, W. G. & Hunter, J. S. (1978), *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building*, John Wiley & Sons, INC., New York.

Cook, R. D. & Nachtsheim, C. J. (1980), 'A Comparison of Algorithms for Constructing Exact D-Optimal Designs', *Technometrics* **22**(3), 315–324.

de Aguiar, P. F., Bourguignon, B., Khots, M. S., Massart, D. L. & Phan-Than-Luu, R. (1995), 'D-optimal Designs', *Chemometrics and Intelligent Laboratory Systems* **30**(2), 199–210.

Donev, A. N. & Atkinson, A. C. (1988), 'An Adjustment for the Construction of Exact D-Optimum Experimental Designs', *Technometrics* **30**(4), 429–434.

DuMouchel, W. & Jones, B. (1994), 'A Simple Bayesian Modification of D-Optimal Designs to Reduce Dependence on an Assumed Model', *Technometrics* **36**(1), 37–47.

Dykstra, O. (1971), 'The Argumentation of Experimental Data to Maximize $|X'X|$', *Technometrics* **13**(3), 682–688.

Eriksson, L., Johansson, E., Kettaneh-Wold, N., Wikström, C. & Wold, S. (2000), *Design of Experiments: Principles and Applications*, Learnways AB, Umeå.

Fedorov, V. V. (1972), 'Theory of Optimal Experiments (Review)', *Biometrika* **59**(3), 697–698. Translated and edited by W. J. Studden and E. M. Klimko.

Galil, Z. & Kiefer, J. (1980), 'Time- and Space-Saving Computer Methods, Related to Mitchell's DETMAX, for Finding D-Optimum Designs', *Technometrics* **22**(3), 301–318.

Goos, P. & Donev, A. N. (2005), 'Blocking Response Surface Designs', *Computational Statistics and Data Analysis* **51**(2), 1075–1088.

Johnson, M. E. & Nachtsheim, C. J. (1983), 'Some Guidelines for Constructing Exact D-Optimal Designs on Convex Design Spaces', *Technometrics* **25**(3), 271–277.

Meyer, R. K. & Nachtsheim, C. J. (1995), 'The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs', *Technometrics* **37**(1), 60–69.

Mitchell, T. J. (1974), 'An Algorithm for the Construction of D-Optimal Experimental Designs', *Technometrics* **16**(2), 203–210.

Montgomery, D. C. (1991), *Design and Analysis of Experiments*, third edn, John Wiley & Sons, INC., USA.

Morris, M. D. (2000), 'Three Technometrics Experimental Design Classics', *Technometrics* **42**(1), 26–27.

NIST/SEMATECH (2007), *e-Handbook of Statistical Methods.* URL: `http://www.itl.nist.gov/div898/handbook/`, visited 2007-12-10.

Powell, M. J. D. (1964), 'An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives', *The Computer Journal* **7**, 155–162.

StatSoft (2007), *Electronic Statistics Textbook.* URL: `http://www.statsoft.com/textbook/stathome.html`, visited 2007-12-04.

Trochim, W. M. (2006), *Research Methods Knowledge Base.* URL: `http://www.socialresearchmethods.net/kb/design.php`, visited 2007-12-22.

Umetrics (2006), *MODDE 8 - User Guide & Tutorial*, 8.0 edn, Umeå.

Weiss, M. A. (1996), *Algorithms, Data Structures and Problem Solving with C++*, ADDISON-WESLEY, USA.

Wu, C. F. J. & Hamada, M. (2000), *Design of Experiments: Planing, Analysis and Parameter Design Optimization*, John Wiley & Sons, INC., USA.

Nomenclature

$(X'X)$	information matrix
$(X'X)^{-1}$	dispersion matrix
α	least square coefficient of X_{pri} on X_{pot}
$\bar{M}$	midrange of a factor
$\bar{R}$	range of a factor
β_i	coefficient of the model
χ_i	single candidate (experiment)
χ_i'	transpose of a single candidate (experiment)
ϵ	experimental error
ϵ_{fed}	treshold value (Fedorov)
$\hat{\beta}_i$	least squares estimate of β_i
$\mathrm{d}(\chi_i)$	variance of prediction for a single candidate
$\mathrm{d}_{max}(\chi)$	maximal variance of prediction of the model matrix
σ	standard deviation
τ	τ-value of the Bayesian modification
ξ_N	matrix of candidate points
ξ_n	matrix of chosen candidate points from ξ_N
g	number of factors of a model
G_{eff}	G-efficiency
N	number of candidate points
n	number of design runs
o	number of inclusions
p	number of primary terms (model terms)

Q	Q-matrix of the Bayesian modification
q	number of potential terms
R	residual of α
X	design matrix
X^*	optimal design matrix
X'	transpose of X
x_i	variable of the model
X_{pot}	matrix of the potential terms
X'_{pot}	transpose of X_{pot}
X_{pri}	matrix of the primary terms
X'_{pri}	transpose of X_{pri}
y	response of the model
Z	Bayesian model matrix
z_i	scaled candidate (experiment)
CN	Condition Number
COST	*Change Only* one *Separate* factor at a *Time*
DoE	Design of Experiments
RSM	Response Surface Modeling

Declaration of Authorship

I, Fabian Triefenbach, confirm that the present thesis and the work presented in it are my own achievement. It is expressed in my own words and every uses made within it of the works of any other author are referenced at their point of use.

Fabian Triefenbach

[illegible]

Umeå, 15th January 2008

Appendix A

Source Code

Preliminary Investigation with MATLAB

filename	description
ftDOptimal.m	main file
ftBayesian.m	Bayesian modification
ftAlgorithm.m	fedorov exchange algorithm
matlabcomplete.pdf	all files as printable pdf

Implementation in C++

filename	description
newDopt.cpp newDopt.h	class for the D-optimal process
DOPTDATA.cpp DOPTDATA.h	class to save created designs
cppcomplete.pdf	all files as printable pdf