NTNU
Norwegian University of
Science and Technology

Preserving projection properties when regular two-level designs are blocked

Yngvild Hamre

Master of Science in Physics and Mathematics
Submission date: June 2018
Supervisor: John Sølve Tyssedal, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This master thesis was written during the spring semester of 2018 as the completion of my Master of Science in Physics and Mathematics at the Norwegian University of Science and Technology (NTNU). The subject code of the thesis is TMA4905 Statistics, and it counts for 30 credits.

The objective of the thesis is to test different methods for dividing regular two-level designs into two or more blocks. The methods are based on using blocks known to be orthogonal on the main effects, but allow for partial confounding between the blocks and the interaction effects too see if higher projectivity can be achieved by sacrificing orthogonality.

I would like to thank my supervisor, John Sølve Tyssedal, whose enthusiasm, patience and support were crucial for my understanding of the topic and motivation for the writing. For that I am very grateful. I would also like thank my boyfriend Kristian Hole-Drabløs for always taking the time to read drafts and discuss thesis-related topics.

Abstract

Early on in an experimental investigation, it is often desirable to determine which factors are to be considered active, i.e. influence the measured response, and which factors are inert. When testing k different factors, there are in many cases only a small subset of factors of size $r < k$ which is active. Instead of thoroughly investigating the entire k-dimensional factor space, one may instead explore the active subspace of much lower dimensionality where the changes in measured response is caused. But this can only be done if the initial screening enables discovery of all possible active subset of size r.

An additional challenge in screening is introduced when experimental runs cannot be performed under the same conditions. Then the design should be divided into blocks with similar conditions. Performing experiments is often costly, thus limiting the number of runs needed to gather the desired information may be an important aspect of planning. In this thesis, the focus is to test different approaches for efficient blocking of several 16-, 32- and 64-run two-level fractional factorial designs, by utilising mirror image pairs and Hadamard matrices.

These methods were chosen as they yield candidate blocks which are orthogonal to the main effects columns, but sometimes partially confounded with the interactions. Preferable blocks are presented after evaluating the candidate blocks based on their projectivity properties and D_s-efficiencies for all combinations of a given number of active factors. Regular fractional factorial designs with good projection properties were chosen as they are popular in industry, but the usual blocking procedure based on confounding blocks with higher-order interactions often results in low projectivity.

The main finding is that by allowing blocks to be partially confounded with interactions, the projectivity was increased for all the designs which were tested, compared to the recommended blocking based on confounding of interactions. Using the suggested blocking thereby enables estimation of all effects for a higher number of active factors than when using blocking based on confounding of interactions. The drawback of allowing partial confounding between blocks and interactions is that it results in higher standard deviations for the estimates of the corresponding effects.

Samandrag

Når ein skal planleggje eit eksperiment ynskjer ein ofte å finne ut tidleg i prosessen kva faktorar som er aktive, dvs. påverkar responsen, og kva som er inaktive. Når ein testar k ulike faktorar vil det i mange høve berre vere ei lita undergruppa med $r < k$ faktorar som er aktive. I staden for å undersøke heile det k-dimensjonale faktorrommet kan ein då heller utforske det mykje mindre, aktive faktorrommet der endringane i responsen vert forårsaka. Det kan berre gjerast dersom innleiande screening er tilrettelagt for å kunne oppdage alle moglege aktive undergrupper med r faktorar.

Ei tilleggsutfordring ved screening oppstår når alle enkeltforsøka i eit eksperiment ikkje kan utførast under like tilhøve. Då må eksperimentet delast inn i mindre blokker med same tilhøve i kvar. Det er ofte dyrt å gjennomføre enkeltforsøk, så avgrensing av talet enkeltforsøk ein treng for å samle nok informasjon er ein viktig del av planlegginga. I denne masteroppgåva er fokuset på å teste ulike framgangsmåtar for å dele design med 16, 32 og 64 enkeltforsøk inn i blokker ved å nytte enkeltforsøkspegelbilete og Hadamardmatriser.

Desse metodane er valde fordi dei genererer blokker som er ortogonale på hovudeffektane, men somme tider delvis konfunderte med interaksjonar. Tilrådde blokker vert presenterte etter at alle blokkene har vorte evaluerte på bakgrunn av projektivitetseigenskapar og D_s-effisiens for alle moglege kombinasjonar av eit gitt tal aktive faktorar. Regulære design er valde fordi dei er mykje nytta i industrien, men den vanlege måten å dele dei i blokker basert på konfundering med høgare-ordens interaksjonar gjev dei ofte låg projektivitet.

Hovudfunnet i masteroppgåva er at høgare projektivitet enn ved konfundering av interaksjonar vart oppnådd for alle regulære design som vart delt inn i blokker med dei nye metodane. Dermed opnar dei tilrådde blokkene for estimering av alle effektar for fleire aktive faktorar. Ulempa med å tillate delvis konfundering mellom blokker og interaksjonar er at det fører til større standardavvik for estimata av tilhøyrande effektar.

Contents

Chapter 1

Introduction

Design of experiments is a branch of statistics concerned with the procedure of gathering data in a planned manner to accommodate the data analysis. The pioneering work was done by Ronald Fischer nearly hundred years ago, as a statistician at the Rothamsted Experimental Station, as described by Bodmer [1]. He was employed there in 1919, and in the following years developed ideas on randomisation and design of experiments to facilitate agricultural investigations, ultimately resulting the publishing of the book *The Design of Experiments* [2] in 1935.

During the Second World War, another key player entered the stage of statistics. Having joined the army as a chemistry student whose job was to perform experiments, George Box quickly figured out that he needed help from a statistician to analyse the data. As no statisticians were available, he had to learn the theory himself, even after having explained to his colonel that ”I once tried to read a book about it by someone called R. A. Fisher but I didn't understand it”, as he mentioned in the 50th anniversary speech at the Statistics Department of the University of Wisconsin, of which he was the founder [3]. Luckily, Box did not give up on his efforts to master statistics and design of experiments, and made numerous contributions to the field, starting what is often referred to as ”the second era” for statistical experiment design. This meant stronger focus on industrial experiments with more immediate responses and thereby possibilities for using

small, initial experiments to plan more extensive experiments. A contribution from Box particularly relevant for this thesis is the notion of projectivity, which he and Tyssedal introduced in [4].

The projectivity of a design is a measure of for how many factors all main effects and interactions can be estimated. Having good projectivity properties is important in screening situations where higher-order effects are assumed to be important. Traditional blocking of designs using confounding of interactions often destroys the projectivity properties of the designs, as interactions become confounded with block effects. Testing different blocks and comparing efficiencies used to be tedious work, but the last decades increased computing power has enabled testing to an extent which was previously unimaginable. Thus it might still be possible to find new blocking methods which accommodates estimation of more higher-order effects than blocking by confounding of interactions.

The focus of this thesis is blocking of regular fractional factorial designs, as they are widely used among experimenters. The emphasis is on testing if mirror image pairs can be used to block the designs in a manner which preserves the projectivity properties. Using mirror image pairs ensures that the block column is orthogonal to the main effect columns. Doubling of the designs for which mirror image pair-based blocking was successful is also tested, as a method to block designs with twice as many factors and runs. For two-level designs not consisting of mirror image pairs, another approach based on finding orthogonal columns known to be orthogonal to the main effects is tested. The candidate blocks are then found using Hadamard matrices.

Different blocks yielding the same projectivity are compared using D_s-efficiency, a measure of how efficient the block is in terms of minimising the generalised variance of the interesting parameter estimates. The theoretical background is introduced in chapter 2, while the blocking strategies are presented in chapter 3. Results including a preferred blocking for each design can be found in chapter 4, while chapter 5 is used to assess how much the standard deviations of the estimated effects are affected by using blocks with different D_s-efficiencies. Concluding remarks are given in chapter 6.

Chapter 2

Theory

The focus of this thesis is blocking of two-level fractional factorial designs, a subgroup of regular two-level designs. After a brief introduction to experimental design, two-level factorial designs will therefore be the first subject of the following theory. How to handle non-homogeneous experimental conditions by blocking is then discussed, along with means of evaluating different properties of the designs. Having knowledge about how the designs are created, why they are blocked and how the blocks may be evaluated is crucial in order to test new blocking strategies, which will be presented in chapter 3.

2.1 Experimental design

When conducting an experiment, the goal is to quantify the effect of one or more factors on a response by testing different factor levels in a controlled manner. The quality of the subsequent analysis is largely dependent on the experimental setup, and the design should therefore be carefully planned before the experiment. By doing so, valid and objective conclusions can be made and thereby new knowledge acquired. The methodology is useful in a variety of areas, ranging from process improvement in industry to marketing strategies for businesses.

Which experimental design to choose depends on the purpose of the experiment; should the estimated effects be very precise, or should the experiment merely be able to detect which factors seem to affect the response at all? Is there a large cost attached to the number of experimental runs? Is there a limited batch of raw material available, and does the experimental runs have to be performed in different days? To find a suitable design for a given situation, it is important to take all such questions into account and have a broad overview of different design strategies.

2.1.1 Two-level factorial designs

Factorial experiments are experiments in which all combinations of the levels of the factors are tested. A factor is an explanatory variable, for example the temperature of the chemicals, whether the operator is a man or a woman, and so on. The number of runs required depends on the number of levels for each factor. For a factorial experiment with two factors, A and B, with a and b levels respectively, the minimum number of runs is *ab*. Two-level factorial designs naturally require the smallest number of runs to investigate a given number of factors and are therefore commonly used in factor screening experiments. Testing all possible combinations of two levels for k factors require 2^k runs, thus giving them the name 2^k factorial designs. For each additional factor investigated, the number of runs doubles. To illustrate the concept, a 2^3 factorial design with the corresponding response y is shown in table 2.1. The columns A, B and C denote the main factors.

Tyssedal [5] defines the main effect of a factor as *"the expected average response when the factor is on the high level - expected average response when the factor is at the low level"*. The estimate of the main effect C can be found as $\bar{y}_{hC} - \bar{y}_{lC}$, where $\bar{y}_{hC}$ and $\bar{y}_{lC}$ are the responses for which C was at a high and a low level, respectively. The estimate of the main effect of C for the design in table 2.1 thus becomes $\hat{C} = \frac{21+23+17+19}{4} - \frac{13+14+18+17}{4} = 4.5$.

Table 2.1: A 2^3 factorial design and corresponding response.

A	B	C	y
-1	-1	-1	13
1	-1	-1	14
-1	1	-1	18
1	1	-1	17
-1	-1	1	21
1	-1	1	23
-1	1	1	17
1	1	1	19

However, the response for a given level of factor C might differ depending on the level of factor B. Then an interaction effect BC is present in the model. In [5], Tyssedal defines the interaction between two factors as *"Half the main effect of a factor when the other is on the high level - half the main effect of a factor when the other factor is at its low level"*. The interaction between B and C in the design in table 2.1 can be estimated as $\hat{BC} = \frac{1}{2}(\frac{(17+19)-(18+17)}{2} - \frac{(21+23)-(13+14)}{2}) = -4$.

In general, a design with k main factors has $\binom{k}{k} + \binom{k}{k-1} + ... + \binom{k}{1} = 2^k - 1$ possible effects and interactions in the resulting model. In many cases, three-factor interactions and higher are assumed to be negligible, but it is not the case in general. They are for example important in many chemical processes. Significant interactions may mask the significance of one or more of the main effects involved, but this does not mean that the factor should not be included.

If the factors are quantitative, the preferred strategy is often to fit a first order polynomial model of the form $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, where $\mathbf{Y}$ is a $n \times 1$ vector with response variables, and $\mathbf{X}$ is a $m \times n$ design matrix with one column for each factor and interaction, and an identity column; a column with ones for the intercept β_0. Thus $m = k + 1$. β is a $m \times 1$ vector of coefficients, one for the intercept, each factor and interaction, and ε is an error term with expectation zero and variance

σ. A model including the factors A and B and the interaction between them can for example be written as $\mathbf{Y} = \mathbf{X}^T\boldsymbol{\beta} = \beta_0 + \beta_A x_A + \beta_2 B_B + \beta_{AB} x_A x_B$. Having such a model allows for prediction of responses for factor values not included in the experiment. The design matrix for the 2^3 factorial design in table 2.1 including an identity column I and all interaction effects is shown in table 2.2. Note that the column AB is given by A$\odot$B, where $\odot$ denotes the Hadamard product. As it is common in the literature to omit the $\odot$ for simpler notation, that convention will be used throughout the thesis.

Table 2.2: Design matrix for the 2^3 design including all interaction effecs.

I	A	B	C	AB	AC	BC	ABC
1	-1	-1	-1	1	1	1	-1
1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	1	-1	1
1	1	1	-1	1	-1	-1	-1
1	-1	-1	1	1	-1	-1	1
1	1	-1	1	-1	1	-1	-1
1	-1	1	1	-1	-1	1	-1
1	1	1	1	1	1	1	1

A compact way to find the parameter estimates is to use the estimator $\hat{\boldsymbol{\beta}}$, which is given by $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. The matrix $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is called "the hat matrix", H. The covariance of the estimator $\hat{\boldsymbol{\beta}}$ is given by the covariance matrix $\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$, where the diagonal elements are the variances, and the off-diagonal elements are the covariances. For instance, $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ii}$ is the variance of parameter estimate number i, and $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ij}$ and $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ji}$ are the covariances between parameter estimate number i and j, as the matrix is symmetric. In experimental designs, $\mathbf{X}$ is often chosen to have orthogonal columns, as it minimises the joint confidence region containing the model regression coefficients, according to Montgomory in the book *Design and Analysis of Experiments* [6]. Then all the off-diagonal elements of $(\mathbf{X}^T\mathbf{X})^{-1}$, the covariances, are zero. To achieve orthogonality, it is common to re-code the two factor levels to -1 and 1,

where -1 is the low level and 1 is the high level. Then the parameter estimate $\hat{\beta}_a$ is half the main effect of factor A, as a factor change from the high to the low level is equal to a change of two in x_A.

As mentioned above, the experimenter chooses $(\mathbf{X}^T\mathbf{X})^{-1}$, but σ has to be estimated. When a 2^k factorial experiment is replicated $r-1$ times, yielding r values for each level combination, this can easily be done. The estimated variance of level combination i, $\hat{s}_i^2$, is given by

$$\hat{s}_i^2 = \frac{1}{r-1}\sum_{j=1}^{r}(y_{ij}-\bar{y}_i)^2, i = 1,2,...,2^k \tag{2.1}$$

where y_{ij} is observation number j of the response using combination i. The overall variance, σ^2, is then estimated with $(r-1)2^k$ degrees of freedom as

$$s^2 = \frac{1}{2^k}\sum_{i=1}^{2^k}\hat{s}_i^2. \tag{2.2}$$

There also exists methods for estimating σ^2 when the experiment is not replicated. One common approach is to assume that higher-order interaction effects are zero, and then use these to estimate the variance, as discussed by Tyssedal [5].

2.1.2 Fractional factorial designs

In some situations, it may not be possible to complete all 2^k runs of the experiment. Then a fraction of the runs can be performed, at the cost of not being able to estimate all effects. A half fraction of a 2^k design is called a $\frac{2^k}{2} = 2^{k-1}$ design, a quarter fraction is called a $\frac{2^k}{2^2} = 2^{k-2}$ design and so on. A 2^{k-p} design is created by constructing a usual experiment with $k-p$ factors, and letting the p last factors be defined by p interaction effects. This is equivalent to writing out the entire 2^k design and choosing the rows in which the entries of the interaction effects between the p last factors and the interactions used to define them are equal to 1. If for example a half-faction of a 2^3 design is to be constructed, one may either

write out a 2^2 design and add a column C=AB or write out the entire 2^3 design and choose the rows where ABC=1. In either case, I=ABC is called the defining relation, and the relation C=AB is called the generator for the design.

Using an interaction column to define a new factor makes the interaction effect impossible to separate from the main effect of the new factor. This is a phenomenon called aliasing. Aliasing between two effects means that their design columns are identical, thus making it impossible to differentiate between their effects. Hence using higher-order interaction effects which are assumed to be negligible to define the new factors is recommended.

To clarify the procedure when $p > 1$, consider a case where one wants to investigate five factors in eight runs. Running a complete 2^5 experiment requires 32 runs, so a quarter-fraction has to be chosen, making it a 2^{5-2} design. Many textbooks include recommended choices of design generators. Montgomery [6] recommends using AB=D and AC=E to define the last factors. Then the interaction effects AB and AC are not differentiable from the main effects D and E, respectively. But it is not only the generators that are no longer indifferentiable from other effects. As DB=ABB=A, BD and A are for instance aliased as well. Note that BB is equal to I, a column vector of ones.

The alias structure, which effects that are indifferentiable, can easily be found using the defining relation of the fractional design. The defining relation is defined as *"the set of all columns that are equal to the identity column"* [6]. These may also be referred to as words. If the design generators are D=AB and E=AC, the defining relation is given as I=ABD=ACE=BCDE. Multiplying an effect with the defining relation gives the alias structure for the effect. For example, A ABD =BD, thus A and BD are aliased when D=AB and E=AC are the design generators. The entire alias structure for A is A=BD=CE=ABCDE.

As higher-order effects are often assumed to be negligible, it is preferable to have a design with an alias structure that does not involve aliasing between any lower-order interactions. The term resolution is often used to describe this aspect of the alias structure. According to Montgomery [6], *"a design is of resolution R if no*

p-factor affect is aliased with another effect containing less than $R - p$ factors". The usual notation for resolution is to use a Roman numeral. A resolution V design is for example a design in which no two-factor effect is aliased with another effect containing less than three factors, thus no two-factor effect is aliased with another two-factor effect, and no main effect is aliased with an effect containing less than four factors. In general, the resolution is equal to the length of the shortest word in the defining relation. In the example in the above paragraph, ABD and ACE are the shortest words, making it a resolution III design. Then a two-factor interaction effect is not aliased with any other effect containing less than 3-2 factors, i.e. two-factor effects are aliased with main effects, as shown.

Regular designs
Wu and Hamada [7] defines regular designs as designs for which "*any two factorial effects either can be estimated independently of each other or are fully aliased*". This corresponds to all 2^k factorial and 2^{k-p} fractional factorial designs, so all designs considered in the results section belong to this class. Such designs require a number of runs that is always equal to a power of two, making the run-size quite inflexible. Non-regular designs such as Plackett-Burman designs are therefore also popular. They allow for effects to be correlated, and hence often need a smaller number of runs to estimate effects. Regular designs are still widespread due to being relatively easy to analyse and thoroughly studied in the literature.

2.1.3 Blocking

Blocking is a design technique used to reduce the effect of nuisance factors on the estimated effects of the different factor combinations. A nuisance factor is a factor whose effect on the response is not interesting, but still likely to be present [6]. A nuisance factor may be controlled or uncontrolled, known or unknown. Randomisation is a measure against uncontrollable and unknown nuisance factors. If the nuisance factor is known, but uncontrollable, it may be analysed statistically using analysis of covariance. An unknown nuisance factor may hardly be controllable, but the case of known and controllable can be dealt with using blocking.

Introducing a block factor is recommended whenever a known and controllable nuisance factor is likely to systematically affect the result. This might for example be different operators performing the trials or different batches of raw material being used. If for instance two units are used when conducting the experiment, and all high levels are tested on the first unit, whereas all low levels are tested on the second unit, it will not be possible to separate the effect of the levels from the variability between the units. By instead testing all combinations on each of the units, the variability between units, i.e. the block effect, may be estimated. This approach is called ”Randomised complete block design”, where ”complete” refers to the fact that all combinations are tested in each unit/block [6], and ”randomised” to the combinations being randomised within each block. In general, blocking represents a restriction on randomisation.

If the blocks are not large enough to contain all possible factor combinations in one replicate of the experiment, the design technique ”confounding” may be used to ensure that all main effects and lower-order interaction effects can be estimated. Higher-order interaction effects then become confounded with blocks, meaning that the higher-order interaction effects cannot be separated from the block effects. This is the exact same phenomena as aliasing, and both terms are common in the literature. The confounding technique is based on dividing the runs into blocks based on the sign of one or more interaction effects. If the factors are A, B and C, the interaction effect ABC can for example be used to define two blocks by the runs for which ABC=-1 and ABC=1, respectively. The interaction effect ABC is then called a block generator.

Table 2.3 shows a toy example with the 2^3 design mentioned above to illustrate the concept. Block 1 is chosen naively by letting the first four rows belong to one block, and the last four to the other, denoting them -1 and 1 respectively. The blocking column is then equal to the column C, so the main effect becomes indifferentiable from the block effect. A better choice of block is therefore Block 2, which is equal to the ABC-column. Then ABC is used as the block generator. The effect of ABC can no longer be estimated, but as higher-order effects most often are less likely to be active than lower-order effects, that is the best choice of block generator. The design matrix in this case will consist of the columns I, A,

Table 2.3: Design matrix for the 2^3 design including all interaction effects, and two candidate blocks.

I	A	B	C	AB	AC	BC	ABC	Block 1	Block 2
1	-1	-1	-1	1	1	1	-1	-1	-1
1	1	-1	-1	-1	-1	1	1	-1	1
1	-1	1	-1	-1	1	-1	1	-1	1
1	1	1	-1	1	-1	-1	-1	-1	-1
1	-1	-1	1	1	-1	-1	1	1	1
1	1	-1	1	-1	1	-1	-1	1	-1
1	-1	1	1	-1	-1	1	-1	1	-1
1	1	1	1	1	1	1	1	1	1

B, C, AB, AC, BC and Block 2 from table 2.3, and the resulting linear model is $Y = \beta_0 + \beta_A \cdot A + \beta_B \cdot B + \beta_C \cdot C + \beta_{AB} \cdot AB + \beta_{AC} \cdot AC + \beta_{BC} \cdot BC + \beta_{Block2} \cdot \text{Block2}$. It is usually assumed that the interaction effects between the blocks and the main and interaction effects are negligible. If the assumption is wrong, the error term will include these interactions.

If the design is a fractional factorial, the blocking procedure is further complicated. Now both the design generator and the block generator may introduce confounding. One way to proceed is then to look at the total number of clear effects, as suggested by Wu and Hamada [7]. A two-factor interaction is for example defined as clear if it is not confounded with any main effects, two-factor interactions or blocks. They do however note that simply considering the total number of clear effects may be too naive, as one for instance often will prefer having clear main effects rather than clear two-factor interactions. The book includes tables of the most common factorial designs and suggestions on how to block them using the total number of clear effects-criterion. This will from now on be referred to as the recommended blocking by confounding of interactions.

It also possible to use several blocks in an experiment. When for instance dividing the design into four blocks, they can be defined by using two columns

with the row combinations (-1,-1), (1,-1), (-1,1) and (1,1) to define the four different blocks. The interaction between them should also be included in the design matrix, which in total yields three columns resulting in rows (-1,-1,1), (1,-1,-1), (-1,1,-1) and (1,1,1). It is important to be sure that the block interaction is not confounded with any main or interaction effects. If for example both ABC and AC are used for blocking, ACBAC=B, making the blocks undesirable. Suggestions on how to divide fractional factorial designs in more than two blocks can be found in Wu and Hamada [7] as well.

2.1.4 Evaluating blocks

When looking for an optimal blocking, one must reflect upon with respect to which criteria the blocking should be optimal. Several different criteria have been proposed, such as the total number of clear effects, as mentioned above. In this thesis, the focus is on being able to estimate the maximum number of effects as efficiently as possible for a given number of active factors. Then the notions of projectivity and D_s-efficiency are useful.

Projectivity
When designing a screening experiment, it is important to consider whether higher-order interactions are believed to be active. To be able to effectively communicate whether the higher-order interactions can be estimated, Box and Tyssedal [4] defined the projectivity of a two-level design as *"A $n \times k$ design with n runs and k factors each at two levels is said to be of projectivity P if the design contains a complete 2^p factorial in every possible subset of P out of the k factors, possibly with some points replicated. The resulting design will then be called a (n, k, P) screen"*.

Investigating projectivity properties is particularly useful for screening designs, as it guarantees the possibility to get unbiased estimates of all effects up to P-order interactions when P or fewer factors are active. Knowing the projectivity properties makes it easier to find the smallest possible screening design for which the active factors are detectable and the corresponding effects estimable. In addition, for the set of active factors, replicated runs will have the same expected value, and

thus allow for model-independent estimation of the error variance. Another useful property to be aware of when choosing design is that a regular fractional factorial design of resolution R is of projectivity $P = R - 1$ [4].

The original definition of projectivity is rather strict, as it implies that the P-factor interaction and lower must be estimable. If there are many active factors, one might not be interested in the highest-order interactions, and rather prefer being able to find all active factors. Then the generalised projectivity is a useful measure. It was introduced by Evangelaras and Koukouvinos in [8] as *"a $n \times k$ design with n runs and k factors each at two levels is said to be of generalized projectivity P_α, if for any selection of P columns of the design all factorial effects including up to α-factor interactions are estimable"*.

The projectivity of regular designs is well known, but unfortunately often not preserved when the designs are blocked. In [9], Hussain and Tyssedal defines a blocked design to be of projectivity P or P_α *"if, in addition to the intercept, all factorial effects up to and including P-factor interaction or α-factor interactions are estimable respectively"*. In this thesis, the notation (n,k,P_α,b) screen and (n,k,P,b) screen will be used, where as before n is the number of runs and k is the number of factors in the design. P_α is the number of columns of the design for which all factorial effects including up to α-factor interactions are estimable no matter which columns are chosen, and b is the number of blocks.

In some cases, the notation $(n,k,P_{\alpha+a},b)$ screen will be used, where a denotes the number of $(\alpha+1)$-interactions that were estimable. This is particularly relevant for some of the designs where $\alpha < P$. If for example all three-factor interactions and two four-factor interactions are estimable when there are five active factors, $P = 5, \alpha = 3$ and $a = 2$. Note that a $(n,k,P_{\alpha+a},b)$ screen by definition always is a (n,k,P_α,b) screen as well. Note also that the a $(\alpha+1)$-interactions can be freely chosen, i.e. one may choose any a $(\alpha+1)$-interactions and estimate them. This definition makes $a = 0$ for regular designs blocked by confounding of interactions, as the $(\alpha+1)$-interactions cannot be freely chosen for these. If for instance ABC is the block generator, the interaction effect ABC can never be estimated, limiting the projectivity to $P = 2$.

Projectivity in relation to fold-over and doubling

According to Tyssedal in [10], the foldover of a $n \times k$ design matrix $\mathbf{X}$ without a column of ones for the intercept is given by $\tilde{\mathbf{X}}$= $\begin{pmatrix} \mathbf{X} & 1 \\ -\mathbf{X} & -1 \end{pmatrix}$,

where 1 is a $n \times 1$ vector of ones. The resulting design $\tilde{\mathbf{X}}$ has dimensions $n \times (k+1)$ and can thereby accommodate $k+1$ factors. This is a useful technique as it guarantees the projectivity of the resulting design. Folding over a regular design with an even resolution does not change the resolution and projectivity, but if the resolution is odd, the foldover increases the resolution and thereby also the projectivity by one. If for example the design $\mathbf{X}$ has resolution III and projectivity $P = 2$, the foldover design has resolution IV, and thereby projectivity $P = 3$. This happens because all words of length three are no longer present in $\tilde{\mathbf{X}}$. Utilising this property may for example be done if a small design of projectivity two has been used to identify three active factors, and one wishes to add runs to make it a projectivity $P = 3$ design in order to estimate the two-factor interaction effects without confounding with the three-factor interaction effect.

Another design technique used to maintain the projectivity when increasing the number of factors and runs in the design is doubling, a technique used to generate a new design $D(\mathbf{X})$ with twice the number of rows and more than twice the number of factors than the original design $\mathbf{X}$. This is done by using the following pattern:

$D(\mathbf{X})$= $\begin{pmatrix} \mathbf{X} & \mathbf{X} & \mathbf{1} \\ \mathbf{X} & -\mathbf{X} & \mathbf{-1} \end{pmatrix}$,

where $D(\mathbf{X})$ is a $n \times (2k+1)$ matrix, and $\mathbf{X}$ is a $n \times k$ matrix, as before. The useful property regarding projectivity is that if $\mathbf{X}$ is a $(n,k,3)$ screen, the doubling $D(\mathbf{X})$ is a $(2n,2k,3)$ screen when the rightmost column with n 1' and n -1's is removed. Samset and Tyssedal notes in [11] that a defining relation of four factors always exists for doubled designs. Thus the projectivity of doubled designs cannot exceed $P = 3$, as four active factors would possibly result in a main effect being confounded with a three-factor interaction.

D-optimality and D_s-efficiency

The columns of a 2^p factorial design are always orthogonal. Thus blocks based on confounding of interactions are orthogonal on all effects. It is however also possible to estimate effects when they are partially confounded with the block. Partial confounding between two design columns means that the inner product between them is non-zero, but smaller than the length of the columns. If the inner product is zero, they are orthogonal, and if it is equal to the length of the columns, they are confounded. Thus the further from zero, the stronger the partial confounding. The inner product can be found by inspecting the $\mathbf{X}^T\mathbf{X}$ matrix, where an off-diagonal element $(\mathbf{X}^T\mathbf{X})_{ij}$ with a non-zero value shows the partial confounding between the effect i and the effect j. This does in turn yield higher values of the diagonal elements $(\mathbf{X}^T\mathbf{X})^{-1}_{ii}$ and $(\mathbf{X}^T\mathbf{X})^{-1}_{jj}$, and thereby a higher estimate of the variances of effect i and effect j than if there had been no partial confounding.

How should the preferred block be chosen in the case of partial confounding? One of the most widely used criteria is D-optimality, as described in the book *Optimum Experimental Designs, with SAS* by Atkinson, Donev and Tobias [12]. They define a D-optimal design $\mathbf{X}_D$ as the design which minimises the generalised variance of the parameter estimates. The generalised variance is defined as the determinant of the covariance matrix, according to Gupta [13]. The covariance matrix of $\hat{\beta}$ is $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$, so minimising its determinant equals maximising the value of $|\mathbf{X}^T\mathbf{X}|$.

Evaluating the efficiency of a design $\mathbf{X}$ compared to the the optimal design $\mathbf{X}_D$ can be done using the D-efficiency, which is given by $(\frac{|(\mathbf{X}^T\mathbf{X})|}{|(\mathbf{X}_D^T\mathbf{X}_D)|})^{\frac{1}{p}}$, where p is the number of parameters in the model. The D-efficiency is always between 0 and 1, where 1 implies that the design $\mathbf{X}$ is D-optimal. The D-efficiency can thereby be used to rank different candidate designs, and the one with the highest D-efficiency is preferred.

The notion of D-optimality has its counterpart in D_s-optimality for designs for which only some parameters are important to estimate precisely, for example if block effects are included, but their estimated effects are not interesting. In [12],

they motivate the definition by writing the corresponding model as

$$E(Y) = f^T(x)\beta = f_1^T(x)\beta_1 + f_2^T(x)\beta_2,$$

where β_1 denotes the s parameters of interest, and the β_2 are the remaining $p-s$ parameters. The corresponding information matrix can be written as

$$\mathbf{X}^T\mathbf{X} = \begin{pmatrix} \mathbf{X}_1^T\mathbf{X}_1 & \mathbf{X}_1^T\mathbf{X}_2 \\ \mathbf{X}_2^T\mathbf{X}_1 & \mathbf{X}_2^T\mathbf{X}_2 \end{pmatrix}.$$

Let $\mathbf{X}^1$ denote the covariance matrix for the least square estimate of β_1, which is given by the upper left submatrix of $(\mathbf{X}^T\mathbf{X})^{-1}$ with dimension $s \times s$. The D_s optimal design $\mathbf{X}_{D_s}$ is the one for which the determinant of $(\mathbf{X}^1)^{-1}$ is maximised. An expression for this is found using linear algebra. According to Bibby, Kent and Mardia in the book *Multivariate Analysis* [14], the inverse of a matrix $\mathbf{A}$ may be partitioned as

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{A}^{11} & \mathbf{A}^{12} \\ \mathbf{A}^{21} & \mathbf{A}^{22} \end{pmatrix},$$

and if all the inverses exists, $\mathbf{A}^{11} = (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}\mathbf{A}_{21})^{-1}$. Another useful property of $\mathbf{A}$ is that

$$|\mathbf{A}| = \det\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = |\mathbf{A}_{22}||\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}|.$$

Setting $\mathbf{A} = \mathbf{X}^T\mathbf{X}$, $\mathbf{A}_{11} = \mathbf{X}_1^T\mathbf{X}_1, \mathbf{A}_{12} = \mathbf{X}_1^T\mathbf{X}_2$, $\mathbf{A}_{21} = \mathbf{X}_2^T\mathbf{X}_1, \mathbf{A}_{22} = \mathbf{X}_2^T\mathbf{X}_2$ and $\mathbf{A}^{11} = \mathbf{X}^1$ yields

$$\mathbf{X}^1 = ((\mathbf{X}_1^T\mathbf{X}_1) - (\mathbf{X}_1^T\mathbf{X}_2)(\mathbf{X}_2^T\mathbf{X}_2)(\mathbf{X}_2^T\mathbf{X}_1))^{-1}$$

and thereby

$$|(\mathbf{X}^1)^{-1}| = |((\mathbf{X}_1^T\mathbf{X}_1) - (\mathbf{X}_1^T\mathbf{X}_2)(\mathbf{X}_2^T\mathbf{X}_2)(\mathbf{X}_2^T\mathbf{X}_1))|.$$

As $|\mathbf{X}^T\mathbf{X}| = |\mathbf{X}_2^T\mathbf{X}_2||(\mathbf{X}_1^T\mathbf{X}_1) - (\mathbf{X}_1^T\mathbf{X}_2)(\mathbf{X}_2^T\mathbf{X}_2)^{-1}(\mathbf{X}_2^T\mathbf{X}_1)|$, $|(\mathbf{X}^1)^{-1}| = \frac{|\mathbf{X}^T\mathbf{X}|}{|\mathbf{X}_2^T\mathbf{X}_2|}$.

Thus a D_s-optimal design maximises $|(\mathbf{X}^1)^{-1}| = \frac{|\mathbf{X}^T\mathbf{X}|}{|\mathbf{X}_2^T\mathbf{X}_2|}$. This expression is then used to define the corresponding D_s-efficiency as

$$D_s = \frac{\left[\frac{|\mathbf{X}^T\mathbf{X}|}{|\mathbf{X}_2^T\mathbf{X}_2|}\right]^{\frac{1}{s}}}{n} \tag{2.3}$$

where s is the number of interesting effects, equal to the number of columns in $\mathbf{X}_1$, and n is the number of runs [9]. A design maximising the D_s-efficiency is said to be D_s-optimal. A useful property of the D_s-efficiency is that when all levels are coded -1 and 1, the design has projectivity $P = h$ if the D_s-efficiency is above 0 for every possible projection onto h dimensions [9]. If the D_s-efficiency is 1, all columns in the design are orthogonally blocked.

2.1.5 Combinatorial explosion

Having established a criterion which can be used to evaluate blocks, it seems straightforward to apply the criterion to all possible blocks to determine which blocking is preferable. The problem is however that this becomes unfeasible when the number of runs increases, as the number of possible blocks increases far more. This phenomenon is known as a ”combinatorial explosion”, a rapid growth of a problem due to complexity which increases with input.

When for instance blocking a $n = 2t$-run design into two blocks, there are $\frac{\binom{2t}{t}}{2!}$ distinct possible blockings, as pairs of blocking arrangements are identical when simply testing all combinations of t -1's and t +1's. It is for example indifferent whether runs 1 to t are placed in the first block and runs $(t+1)$ to $2t$ in the second block, or runs 1 to t in the second block and runs $(t+1)$ to $2t$ in the first block. Thus, when there are two runs in the design, there is only $\frac{\binom{2}{1}}{2!} = 1$ distinct blocking. When there are 32 runs, there are $\frac{\binom{32}{16}}{2!}$=300540195 distinct blockings.

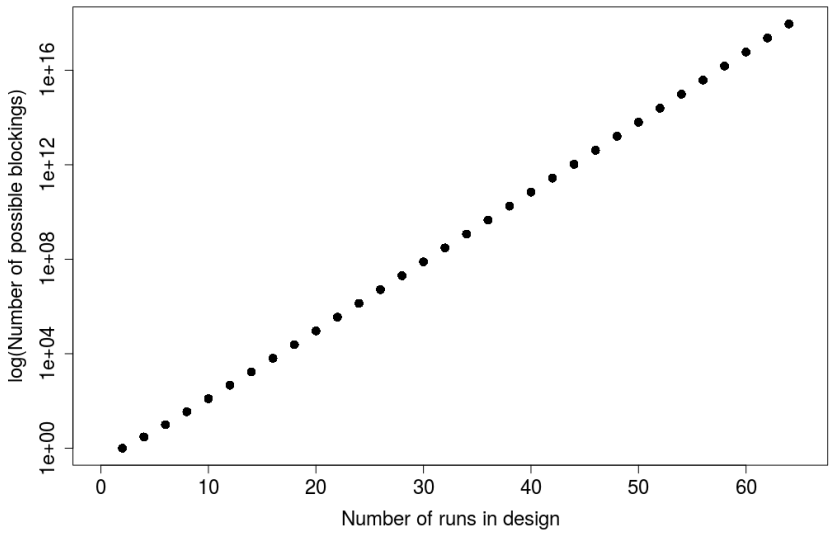

Figure 2.1: Illustration of the growth in number of possible blocks as the run size increases. The number of runs is on the x-axis, and the logarithm of the number of possible blocks on the y-axis.

Figure 2.1 shows a plot of the increase in possible combinations, where the number of runs is on the x-axis, and the natural logarithm of the number of combinations is on the y-axis. As the number of possible blocks quickly becomes several millions, testing all possible blocks in all cases is not feasible. Therefore, a central part of this thesis is to test alternative approaches for generating candidate blocks for large designs. Finding alternative approaches to the traditional blocking by confounding of interactions is motivated by the potential to achieve better projectivity properties. The methods for generating the blocks will be introduced in the next section.

Chapter 3

Blocking strategies

As mentioned in section 2.1.3, blocking a regular design by confounding of interactions leads to effects being aliased, and often lowers the resolution and thereby also the projectivity of the design. Finding blocks which keep the main effects and lower-order interactions not fully aliased and also maintain the projectivity of the design is therefore important when higher-order interactions are assumed to be active.

As an example to illustrate the loss of projectivity, consider the division of the 2_V^{5-1} design in table 3.1 into two blocks using the recommended block generator AB. The design is originally of projectivity $P = 4$, as all possible interactions can be estimated if there are four active factors. When the two-factor interaction AB is chosen as the block generator, the effects of interactions AB and CDE can no longer be estimated separately from the block effect b. The projectivity is thus reduced to $P = 1$, as only main effects are guaranteed to be estimable if two or more factors are active. In section 4.1.2.1, it is found that a 2_V^{5-1} design can be divided into two blocks resulting in a $(16,5,3,2)$ screen if the block is allowed to be partially confounded with the interactions.

Testing different blocks which are orthogonal to the main effects, but may be partially confounded with the interaction effects, is the main focus of the results

Table 3.1: A 2^{5-1} factorial design and the recommended block.

A	B	C	D	E=ABCD	Block
-1	-1	-1	-1	1	1
1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1
1	1	-1	-1	1	1
-1	-1	1	-1	-1	1
1	-1	1	-1	1	-1
-1	1	1	-1	1	-1
1	1	1	-1	-1	1
-1	-1	-1	1	-1	1
1	-1	-1	1	1	-1
-1	1	-1	1	1	-1
1	1	-1	1	-1	1
-1	-1	1	1	1	1
1	-1	1	1	-1	-1
-1	1	1	1	-1	-1
1	1	1	1	1	1

section of this thesis. The challenging part is to find the candidate blocks. Three different methods will therefore be introduced: Utilising mirror image pairs, doubling designs for which the mirror image approach worked, and finally testing other blocks known to be orthogonal to main effects by rearranging Hadamard matrices.

3.1 Using mirror image pairs

A mirror image pair consists of two rows whose signs are opposite. An example of a mirror image pair is the rows [-1,-1,1] and [1,1,-1]. The set of designs in which all rows belong to mirror image pairs does for example include all factorial designs which are not fractional, and all designs constructed by a full fold-over

Table 3.2: Design matrix for the 2^3 design including the three-factor interaction.

Row	A	B	C	ABC
1	-1	-1	-1	-1
2	1	-1	-1	1
3	-1	1	-1	1
4	1	1	-1	-1
5	-1	-1	1	1
6	1	-1	1	-1
7	-1	1	1	-1
8	1	1	1	1

of a smaller design. New factor columns may be added to the design without ruining the mirror image pairs as long as the factor columns are defined by odd-factor interactions. This is easily seen by considering the 2^3 factorial design and its interactions, as shown in table 2.2. The two-factor interaction columns are symmetric about the middle, so the signs are not opposite. This is because the factors A-C are mirror image, and thus products of an even number of factors yield columns with the same sign for both rows in a mirror image pair, destroying the mirror image property. If only the columns A, B, C and ABC had been included in the design matrix, it would have consisted of mirror image pairs. The resulting design matrix can be found in table 3.2.

Jacroux introduced the idea of utilising mirror image pairs in [15]. The idea is to allocate the rows belonging to a mirror image pair to the same block. For the design in table 3.2, this means that row 1 and 8 has to be in the same block, likewise row 2 and 7, 3 and 6 and 4 and 5. Possible unique blockings are then (b1=(1,8,2,7), b2=(3,6,4,5)), (b1=(1,8,3,6),b2=(2,7,4,5)) and (b1=(1,8,4,5), b2=(2,7,3,6)). Using this approach ensures that the block factor is orthogonal to the main effects, so they can be estimated without any partial confounding. Jacroux got promising results for resolution IV regular and non-regular designs. The idea was further tested by Hussain and Tyssedal [9], who used mirror image pairs to block MinResIV designs.

The idea of utilising mirror image pairs to find candidate blockings is applicable to any design consisting of such pairs. In general, a $n = 2t$-run design with t mirror image pairs can be divided into two blocks in $\frac{\binom{t}{t/2}}{2!}$ distinct ways, as two blockings arrangements are identical. Randomly dividing the $2t$ runs into two blocks yields $\frac{\binom{2t}{t}}{2!}$ possible blockings, $\frac{(2t)!((t/2)!)^2}{(t!)^3}$ as many as when using mirror image pairs. The fraction of the possible blockings that consists of mirror image pair-blockings is shown in figure 3.1, where the log of the fraction is plotted as a function of the run size. The fraction decreases substantially for each additional run, showing that the mirror-image pair approach may be very useful for large run sizes if the resulting blocks have high D_s-efficiencies. The idea is of course applicable when a larger number of blocks than two is desired as well.

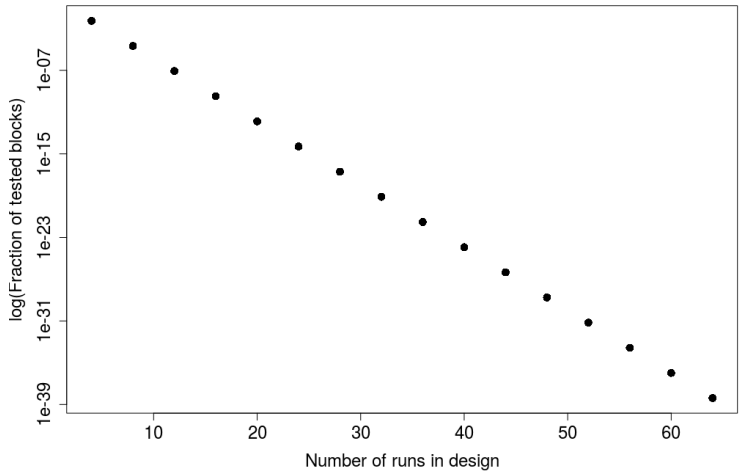

Figure 3.1: Illustration of the fraction of mirror image pair blocks among all possible blocks. The x-axis shows the number of runs in the design, and the y-axis the logarithm of the fraction of mirror image pair-blocks among all possible blocks.

As the method has given promising results for other designs and substantially reduces the number of blockings to be tested, it will be used for designs of size 2^{8-4}_{IV}, 2^{16-11}_{IV} and 2^{6-1}_{V}. These designs were chosen as they consist of mirror image pairs and have a high projectivity compared to other designs with the same number of runs. The results can be found in sections 4.1.1, 4.2.2, 4.2.3, 4.2.4.1, 4.2.4.2, 4.2.4.4,4.2.4.5 4.2.5.1 and 4.2.5.2.

3.1.1 Using the division into 2^i blocks for division into 2^{i+j} blocks

An idea which will be briefly tested in section 4.2.5.4 is to utilise the preferred division of a 32-run design into two blocks based on mirror image pairs to divide the same design into four blocks. This is done by dividing each block consisting of eight mirror image pairs into two blocks of four mirror image pairs each. Each block can be divided into $\frac{\binom{8}{4}}{2!} = 35$ different combinations of two blocks of four mirror image pairs. This yields $35 \cdot 35 = 1225$ ways to make four blocks based on each of the blocks used to divide the design into two blocks.

To clarify the procedure, a toy example is shown in table 3.3 below. The original block defined by -1 is divided into two blocks defined by ($b1$,$b2$)= (-1,-1) and (-1,1), while the original block defined by 1 is divided into two blocks defined by ($b1$,$b2$)=(1,-1) and (1,1). The rows corresponding to mirror image pair 1 are placed in one block, the rows of mirror image pair 2 in another, and so on. As each of the four resulting blocks contains one mirror image pair, each block could be divided into $\frac{\binom{2}{1}}{2!} = 1$ combination of four blocks. It was thereby only $1 \cdot 1 = 1$ way to make four blocks based on the original blocking.

The general idea is to use a $n = 2t$-run design with t mirror image pairs divided into 2^i blocks to divide the design into 2^{i+j} blocks, where $i, j > 0$, and $2^{i+j} \leq t$. This also requires that $\frac{t}{2^{i+j}}$ is an integer. But in how many ways may this be done? In [16], Kllogjeri and Kllogjeri found that if $n = k \cdot r$, a set with n elements can be

Table 3.3: Toy example of using the original division into two blocks, shown in column "Original block", to divide mirror image pairs into four blocks, as defined by columns "$b1$" and "$b2$". "$b1b2$" is the interaction between the new blocks.

Mirror image pair	Original block	$b1$	$b2$	$b1b2$
1	-1	-1	-1	1
2	1	1	-1	-1
3	-1	-1	1	-1
4	1	1	1	1
4	1	1	1	1
3	-1	-1	1	-1
2	1	1	-1	-1
1	-1	-1	-1	1

divided into k groups with r elements each in

$$\frac{(k \cdot r)!}{k!(r!)^k}$$

ways. This formula can be used to find the number of ways each of the 2^i blocks with $\frac{t}{2^i}$ mirror pairs each can be divided into 2^j blocks with $\frac{t}{2^{i+j}}$ mirror image pairs in each. Inserting $n = \frac{t}{2^i}$, $k = 2^j$ and $r = \frac{t}{2^{i+j}}$ into the formula yields the expression

$$\left(\frac{(\frac{t}{2^i})!}{(2^j)!(\frac{t}{2^{i+j}}!)^{2^j}} \right).$$

As this is the number of ways one block can be divided 2^j blocks, the expression has to be multiplied with itself 2^i times to yield the number of ways to divide the entire design into 2^{i+j} blocks based on the division into 2^i blocks. This yields the expression

$$\left(\frac{(\frac{t}{2^i})!}{(2^j)!(\frac{t}{2^{i+j}}!)^{2^j}} \right)^{2^i}.$$

3.2 Blocking based on doubling

As mentioned in section 2.1.4, doubling is a design technique in which one may utilise a small design with projectivity $P = 3$ to guarantee a design twice the size the same projectivity. Thus it seems reasonable to test if blocks which yield high D_s-efficiencies and high projectivity for a small design $\mathbf{X}$ may be used to find blocks which have similar properties for the doubled design D($\mathbf{X}$). To do this, a closer look at the structure of the doubled matrix D($\mathbf{X}$) is required.

3.2.1 Division into two blocks

Let $\mathbf{X}$ denote a regular design of projectivity $P = 3$, with k orthogonal columns and n runs. As all the columns of $\mathbf{X}$ are orthogonal, the doubled matrix D($\mathbf{X}$) has orthogonal columns as well. Let **B1** denote half the rows of $\mathbf{X}$, and **B2** the other half, where the rows of **B1** and **B2** can be chosen in any order. The approach chosen here is to let **B1** include the rows of $\mathbf{X}$ for which the block entries are 1, and **B2** the rows of X for which the block entries are -1. Removing the rightmost column with n 1's and n -1's, D($\mathbf{X}$) can be written as

$$\text{D}(\mathbf{X})= \begin{pmatrix} \mathbf{B1} & \mathbf{B1} \\ \mathbf{B2} & \mathbf{B2} \\ \mathbf{B1} & -\mathbf{B1} \\ \mathbf{B2} & -\mathbf{B2} \end{pmatrix}$$

This is a matrix with $2k$ orthogonal columns and $2n$ runs. As the maximum projectivity of a doubled design is $P = 3$, this design cannot screen for more than three active factors if all effects should be estimable. It is interesting to see if defining **B1** and **B2** by the blocks which were suitable for $\mathbf{X}$ can be used to make the division of D($\mathbf{X}$) into two blocks have $D_s > 0$ for all combinations of three active factors.

As interchanging the rows does not affect the orthogonality properties of D($\mathbf{X}$), the important matter is in how many ways the blocks can be defined. For division of four submatrices into two blocks, there are $\frac{\binom{4}{2}}{2!} = 3$ different ways to define

the blocks. Here, the blocks will be defined by letting the two first submatrices of D(**X**), (**B1**,**B1**) and (**B2**, belong to one block, and the two last submatrices, (**B1**,-**B1**) and (**B2**, -**B2**), to the other. This blocking will be called D_1, and can be written as

$$D_1 = \begin{pmatrix} \mathbf{B1} & \mathbf{B1} & -1 \\ \mathbf{B2} & \mathbf{B2} & -1 \\ \mathbf{B1} & -\mathbf{B1} & 1 \\ \mathbf{B2} & -\mathbf{B2} & 1 \end{pmatrix}$$

The block column with -1's and 1's is included to show that the first block is defined by -1's and the second block by 1's. For the rest of the thesis, division of doubled matrices will in general be tested by reordering the rows, and letting the first n rows belong to the block with -1's, and the last n rows to the block with 1's. In addition to D_1,

$$D_2 = \begin{pmatrix} \mathbf{B1} & \mathbf{B1} & -1 \\ \mathbf{B1} & -\mathbf{B1} & -1 \\ \mathbf{B2} & \mathbf{B2} & 1 \\ \mathbf{B2} & -\mathbf{B2} & 1 \end{pmatrix}$$

and

$$D_3 = \begin{pmatrix} \mathbf{B1} & \mathbf{B1} & -1 \\ \mathbf{B2} & -\mathbf{B2} & -1 \\ \mathbf{B1} & -\mathbf{B1} & 1 \\ \mathbf{B2} & \mathbf{B2} & 1 \end{pmatrix}$$

also represents valid blockings of the $2n$-run design matrix. This approach is tested on a 2^{8-4}_{IV} design and a 2^{16-11}_{IV} design, in sections 4.2.1 and 4.3.1 respectively.

A useful observation before testing the different patterns D_1, D_2 and D_3 is that D_1 makes two-factor interactions confounded with the block column. This is be-

cause the product of a column in **B1** or **B2** with itself yields a column of +1's, while a product of a column in **B1** or **B2** with the same column in -**B1** or -**B2** yields a column of -1's. Thus all two-factor interactions based on columns in D_1 are confounded with the block column, making the D_s-efficiency equal to zero when estimating more than main effects.

3.2.2 Division into four blocks

The same idea can be used to divide designs into four blocks. Similarly as the division of a $2n$-run, $2k$-column design D($\mathbf{X}$) into two blocks based on the blocks which were preferable for dividing a n-run, k-column design $\mathbf{X}$ into two blocks, the doubled design D($\mathbf{X}$) may be divided into four blocks using the blocks which were preferable for dividing $\mathbf{X}$ into four blocks. Let **B1**, **B2**, **B3** and **B4** denote the rows of the $\mathbf{X}$ belonging to each of the four blocks. Removing the rightmost column with n 1's and n -1's, the doubling of $\mathbf{X}$ may be written as

$$\mathrm{D}(\mathbf{X})= \begin{pmatrix} \mathbf{B1} & \mathbf{B1} \\ \mathbf{B2} & \mathbf{B2} \\ \mathbf{B3} & \mathbf{B3} \\ \mathbf{B4} & \mathbf{B4} \\ \mathbf{B1} & -\mathbf{B1} \\ \mathbf{B2} & -\mathbf{B2} \\ \mathbf{B3} & -\mathbf{B3} \\ \mathbf{B4} & -\mathbf{B4} \end{pmatrix}$$

$D(\mathbf{X})$ is now defined by eight submatrices, (**B1 B1**), (**B2 B2**) and so on, as indicated by the above pattern. Each has $\frac{n}{4}$ rows and $2k$ columns. To utilise the original blocking of $\mathbf{X}$, the division of $D(\mathbf{X})$ into four blocks is done by pairing the submatrices. Thus there are $\frac{\binom{8}{2}\binom{6}{2}\binom{4}{2}}{4!} = 105$ ways to arrange the eight submatrices of $D(\mathbf{X})$ into four blocks. The blockings are defined by rearranging the order of the submatrices, and then letting the first quarter of the rows belong to one block, the second quarter of the rows belong to the second block, and so on. This is illustrated below, where two columns are used to define the blocks. The

combination (1,1) does for example correspond to the first block. Note that the rows of $D(\mathbf{X})$ have been shuffled.

$$\begin{pmatrix} \mathbf{B1} & \mathbf{B1} & 1 & 1 \\ \mathbf{B4} & -\mathbf{B4} & 1 & 1 \\ \mathbf{B3} & -\mathbf{B3} & 1 & -1 \\ \mathbf{B2} & \mathbf{B2} & 1 & -1 \\ \mathbf{B3} & \mathbf{B3} & -1 & 1 \\ \mathbf{B4} & \mathbf{B4} & -1 & 1 \\ \mathbf{B2} & -\mathbf{B2} & -1 & -1 \\ \mathbf{B1} & -\mathbf{B1} & -1 & -1 \end{pmatrix}.$$

Recall that there are only three ways of reordering the submatrices when this method was tested for division into two blocks, but for division into four blocks, there are 105. Clearly, this method suffers from a combinatorial explosion when the number of blocks increases. It may therefore not be feasible when dividing large designs into many blocks. But if the candidate blocks have similar properties, it might still be more time-efficient than testing all possible blocks. If for example 400 blocks are suitable for blocking $\mathbf{X}$, and each of these yields the same D_s-efficiences when using one of the 105 possible ways to divide $D(\mathbf{X})$ into four blocks, it is sufficient to test one block per arrangement. This largely decreases the complexity. The method is tested for a 2_{IV}^{16-11} design in section 4.3.2.

3.3 Blocking using Hadamard matrices

If the design is not made up of mirror image pairs, another way of generating columns which are orthogonal to the main effects must be used to find candidate blocks. The orthogonality ensures that the block columns are not partially confounded with any main effects. Such columns can be obtained by using columns found from a Hadamard matrix.

In general, Hadamard matrices are square matrices of order n with all entries being either -1 or 1, and mutually orthogonal rows and columns. In other words, matrices with interesting properties regarding design of experiments. Hadamard matrices apparently exist for $n = 1, 2$ and $n \equiv 0 \pmod 4$, according to Peter Cameron in [17]. The smallest order of which no Hadamard matrix has yet been found is 668. Two Hadamard matrices are equivalent if one can be obtained by rearranging the rows or the columns of the other, or by using negations. Up to order 12, there is one Hadamard matrix up to equivalence for each order. The number of Hadamard matrices up to equivalence of order n then increases, and a combinatorial explosion happens for $n = 32$, according to Kharaghani and Tayfeh-Rezaie in [18].

A Hadamard matrix of size $2n$ can be made by arranging the Hadamard matrix of size n, $\mathbf{H}$, in the following pattern: $\begin{smallmatrix} \mathbf{H} & \mathbf{H} \\ \mathbf{H} & -\mathbf{H} \end{smallmatrix}$. Note that this is almost the same pattern that was used for doubling of designs in section 2.1.4. In general, all designs generated using that pattern are not Hadamard matrices, but if $\mathbf{X}$ is a Hadamard matrix, D($\mathbf{X}$) minus the rightmost column is a Hadamard matrix of size $2n$. An example of a Hadamard matrix of size 8 from the web page [19] can be found in table 3.4.

Table 3.4: A Hadamard matrix of order eight.

1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	1
1	1	-1	-1	-1	-1	1	1
1	-1	-1	1	-1	1	1	-1

As interchanging the rows of a Hadamard matrix does not change the orthogonality properties, one may try to rearrange the rows to obtain some columns that

match a regular design. The remaining columns can then be used as candidate blocks. This guarantees candidate blocks that are orthogonal on the main effects. If a projectivity $P = 2$ is desired, all candidate blocks equal to two-factor interactions must be removed, and likewise for higher projectivities. This approach will be tested for a 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} design in section 4.2.6, and a 2_{V}^{8-2} design in section 4.3.3. Blocking of a 2_{IV}^{5-1} design using a particularly good Hadamard matrix of order 16 is tested in section 4.1.2.

Chapter 4

Results

As shown in the beginning of chapter 3, blocking by confounding of interactions may lead to the blocked design having much lower projectivity than the original design. Testing whether there exist strategies for blocking designs that maintain the projectivity properties is therefore interesting. In this section, results from applying the methods introduced in section 3 to regular two-level designs of different sizes are presented.

For the approach using mirror image pairs (MIP), a 2_{IV}^{8-4}, a 2_{IV}^{16-11} and a 2_{VI}^{6-1} design were tested, as well as the doublings of the 2_{IV}^{8-4} and the 2_{IV}^{16-11} design. The reasons for choosing these designs will be more thoroughly explained in the introduction of each section, but in short, they combine the desired properties of high projectivity and ability to screen several factors. This makes the designs popular choices for screening in situations where it is important to be able to estimate all interactions between the active factors.

For comparison with the designs blocked using the mirror image pair approach, some designs not made up by mirror image pairs were blocked by using columns from rearranged Hadamard matrices (HM), as they were orthogonal on the main effects. A 2_{V}^{5-1} design was blocked to test a 16-run design with higher projectivity than the 2_{IV}^{8-4} design. For comparison with the 32-run mirror image pair designs,

blocking of a 2_{IV}^{7-2}, a 2_{IV}^{8-3} and a 2_{IV}^{9-4} design were investigated. The 64-run design considered was a 2_{V}^{8-2} design.

In all cases, the interactions between the block effects and the main and interaction effects were assumed to be negligible. As the projectivity of a design cannot be increased by blocking, the original projectivity yielded an upper limit for the number of active factors worth trying to screen for in each case.

For many of the blockings tested, there were partial confounding between block effects and some interactions. The D_s-efficiency was therefore used to assess how much information was preserved using the new blocks, to enable the experimenter to choose the most efficient blocking. For each design, an example blocking and frequencies of the D_s-efficiencies when testing all combinations of active factors are presented. The example blocking was chosen as the blocking having the highest minimum D_s-efficiency among the blocks having high average D_s-efficiencies.

For each blocking, one of the combinations of factors yielding the highest D_s-efficiency is also explicitly stated. The purpose of this is to ensure that the experimenter may allocate factors suspected to be active to the presented factor columns. If it turns out that these are not the active factors, a high D_s-efficiency will still be ensured by the high minimum D_s-efficiency. This makes the blocking robust and well suited for initial screening. Note that when presenting D_s-efficiencies, three decimals are used as long as more are not needed to differentiate between results. As a rule of thumb, D_s-efficiencies above 0.9 will be considered high, but when the best blocks resulted in D_s-efficiencies between 0.8 and 0.9, these are also presented.

The number of runs available for the experimenter is often limited, due to lack of time or resources. A common question is therefore how much information it is possible to extract from a given number of runs. To facilitate answering this question, the results are ordered by the run size of the investigated designs, starting with the 16-run designs 2_{IV}^{8-4} and 2_{V}^{5-1}, proceeding with the 32-run designs 2_{IV}^{16-11}, 2_{VI}^{6-1}, 2_{IV}^{7-2}, a 2_{IV}^{8-3} and 2_{IV}^{9-4}, and ending with the 64-run designs D(2_{IV}^{16-11})=2_{IV}^{32-26} and 2_{V}^{8-2}.

In section 4.4, a summary of the results for all the designs are presented in tables 4.48 and 4.49. Designs with equal run-sizes which enable estimation of all effects for the same number of active factors are also compared in terms of minimum-, maximum- and average D_s-efficiencies. How much the partial confounding between interaction effects and blocks affects the parameter estimates and their corresponding standard deviations is assessed in chapter 5.

Implementing the blocking strategies using R
The different blocking strategies were tested by implementing them in R, a programming language popular for statistical computing [20]. It is an open-source software with many useful embedded functions, but without any warranties. As most functions used to produce the results in this thesis were either implemented from scratch or easily verifiable, it is not believed to be worrying in this case, but it is still important to be aware of.

The main focus when implementing the strategies was to ensure that all combinations of factors and all blockings were tested. The scripts used to generate each result presented here in the results section can be found in Appendix B: R code. Several functions are equal for all scripts, but they are nevertheless included everywhere to make each code section possible to run independently.

4.1 16-run designs

The 16-run designs tested were a 2^{8-4}_{IV} design and a 2^{5-1}_{V} design.
The 2^{8-4}_{IV} design was chosen as it is the 16-run design of projectivity $P = 3$ which accommodates the highest number of screening factors. It consists of mirror image pairs and was divided into two blocks by utilising these. The results for three active factors can be found in section 4.1.1. Division into four blocks was also tested, but as none of the candidate blocks yielded $D_s > 0$ for all combinations of three active factors, the results are not included.

The 2_V^{5-1} design was chosen as it is the 16-run design of projectivity $P = 4$ which accommodates the highest number of screening factors. As the design does not consist of mirror image pairs, it was divided into two blocks by reordering a Hadamard matrix and by combining two-factor interactions. The results for three and four active factors are presented in sections 4.1.2.1 and 4.1.2.2 respectively.

4.1.1 Blocking a 2_{IV}^{8-4} design using MIP

In this section, a 2_{IV}^{8-4} design of projectivity $P = 3$ is divided into two blocks using the mirror image pair approach. The best blocks found using this approach are used to divide the doubling, a 2_{IV}^{16-11} design, into two blocks in section 4.2.1. The 2_{IV}^{8-4} design is given in table 4.1. The columns A, B, C and D will be referred to as the original factors, as they are not defined by three-factor interactions, unlike E, F, G and H.

Traditional blocking of the design using confounding leads to a loss of projectivity. In Wu and Hamada [7], the recommended way to block this design in two blocks of size four is to use AB as the block generator. The block can be seen in table 4.1. The block factor is not aliased with any main effects, but aliased with the two-factor effects GH, CE and DF, as GH=ACDBCD=AB, CE=CABC=AB and DF=DABD=AB. Thus traditional blocking guarantees only main effects to be estimable without confounding, resulting in a $(16, 8, 1, 2)$ screen.

Clearly, finding a blocking which keeps the lower-order interactions unaliased is desirable. This can be done by exploiting that the rows of the design make up $t = \frac{n}{2}$ mirror-image pairs. The pairs with row number $1 + r$, $n - r$, $r \in [0, t - 1]$ are mirror images of each other. An approach to dividing the design into two blocks is to let each block contain $\frac{t}{2}$ mirror image pairs, as explained in section 3.1. As the blocking arrangements with opposite signs are equivalent, there are $\frac{\binom{8}{4}}{2!} = 35$ possible blocking arrangements separating the mirror image pairs into two blocks for this design. The goal being to enable estimation of all effects when any three out of the eight factors are active, blocks that were identical with two-factor interaction columns had to be removed.

Table 4.1: The 2_{IV}^{8-4} design, the recommended block generator (AB) and the preferred block found in this section.

Row	A	B	C	D	E=ABC	F=ABD	G=ACD	H=BCD	Recommended block generator	Preferred block
1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
2	1	-1	-1	-1	1	1	1	-1	-1	1
3	-1	1	-1	-1	1	1	-1	1	-1	1
4	1	1	-1	-1	-1	-1	1	1	1	-1
5	-1	-1	1	-1	1	-1	1	1	1	1
6	1	-1	1	-1	-1	1	-1	1	-1	-1
7	-1	1	1	-1	-1	1	1	-1	-1	-1
8	1	1	1	-1	1	-1	-1	-1	1	-1
9	-1	-1	-1	1	-1	1	1	1	1	-1
10	1	-1	-1	1	1	-1	-1	1	-1	-1
11	-1	1	-1	1	1	-1	1	-1	-1	-1
12	1	1	-1	1	-1	1	-1	-1	1	1
13	-1	-1	1	1	1	1	-1	-1	1	-1
14	1	-1	1	1	-1	-1	1	-1	-1	1
15	-1	1	1	1	-1	-1	-1	1	-1	1
16	1	1	1	1	1	1	1	1	1	1

All two-factor interactions are aliased with either one of the six two-factor interactions between the original factors, or the four-factor interaction ABCD including all original factors. For instance AB=GH=CE=FD, and BG=DE=CF=AH. As the factors have opposite signs in the upper and bottom half of the design, the signs of interactions between an even number of factors are symmetrical about the middle of the design. Thus they must have an equal number of 1 and -1 entries in each half of the design, just as the candidate blocking columns. Therefore, each even interaction was aliased with one of the 35 candidate blocks. Using one of those blocks would lead to a two-factor effect being aliased with the block effect. Thus, in total $\binom{4}{2} + \binom{4}{4} = 7$ of the candidate blocks were not suitable.

The remaining 28 blocks can be found in tables 6.1 and 6.2 in Appendix A. To test if any of these blocks were better suited than the others for screening for three active factors, the D_s-efficiency was calculated for all $\binom{8}{3} = 56$ possible combinations of three active factors. All blocks were found to yield a D_s-efficiency of 0.917 for 48 combinations, and a D_s-efficiency of 1 for the remaining eight combinations, yielding a mean D_s-efficiency of 0.929. Thus a high D_s-efficiency is expected regardless of which three factors are active, making the blocks suitable for screening. As none of the blocks yielded a higher average D_s-efficiency than the others, the experimenter may choose any of the candidate blocks among the 28. As even the three-factor interaction is possible to estimate, using any of the 28 blocks makes the design a (16,3,3,2) screen.

In case of suspicion about which factors might be active, it is useful to know if there are some factor combinations for which $D_s = 1$ for all blocks. To test this, the results using all blocks was stored for each combination. For each of the 56 combinations, there were 24 blocks yielding D_s=0.917, as two of the two-factor interactions were partially aliased with the block effect. The remaining four blocks yielded D_s=1. As there was no combination for which $D_s = 1$ for all blocks, considering which block to use is important if there is any suspicion about which factors may be active. Then a block that yields D_s=1 for that combination should be chosen. The preferred block shown in table 4.1 yields D_s=1 when the active factors are A, B and C. Thus if there are three factors suspected to be active, they may be allocated to the design columns A, B and C in table 4.1.

The recommended block by confounding of interactions is also shown in table 4.1. Note that both blocks are symmetric about the middle of the design. The recommended block by confounding is symmetric because it is equal to a two-factor interaction, and those are always symmetric in designs consisting of mirror-image pairs, as explained in section 3.1. The recommended block is symmetric as the rows belonging to one mirror image pair have to be in the same block, and the rows of a mirror image pair are on opposite sides of the middle.

4.1.2 Dividing a 2_V^{5-1} design into two blocks using HM

In section 4.1.1, a 2_{IV}^{8-4} design was divided into two blocks, resulting in a $(16,8,3,2)$ screen. Another design with 16 runs is the 2_V^{5-1} design with projectivity $P = 4$ and resolution V. Thus the 2_V^{5-1} design is able to screen for one more active factor than the 2_{IV}^{8-4} design if all interactions are to be estimated, but the number of screening factors is lower. Which 16-run design to use thus depends on the number of factors the experimenter would like to investigate.

But can the projectivity of the 2_V^{5-1} design be maintained when the runs are divided into two blocks? As the unblocked projectivity is $P = 4$, it is interesting to test if any blocks may allow for estimation of higher-order interaction effects when four factors are active. As the design does not consist of mirror image pairs, blocks which are orthogonal to the main effects must be found otherwise.

The 2_V^{5-1} design can be found in table 4.2. The columns of the design use the same notation as the design in table 4.1. Thus P=AB is equal to the product of column A and B in table 4.1. Usually, the columns of a 2_V^{5-1} design would have been chosen as column A-D in table 4.1, with the last column being defined as E=ABCD. The design in table 4.2 can be rewritten to that form by rearranging the rows. This can be seen in table 4.3. The representation in table 4.2 is used in the rest of this section, as the columns A, C, D, H and P=P_2 were found in the design H_2 presented by Box and Tyssedal in [21]. The H_2 design was made by interchanging the rows of a Hadamard matrix, and thereby has orthogonal columns. The first eight columns of H_2 can be rearranged to a 2_{IV}^{8-4} design, while the four columns J_2, K_2, L_2 and M_2 can be added to obtain a (16,12,3,1) screen. Thus choosing the columns A, C, D, H and P_2=P to make up the 2_V^{5-1} design leaves the columns J_2, K_2, L_2 and M_2 as potential candidate blocks.

In addition to testing the columns from the article known to be orthogonal to the main effects, all possible blocks were created. The blocks which were orthogonal to all main effects, and only partially confounded with two-factor interactions, were chosen for further investigation. These were the best possible blocks, as none were orthogonal on all two-factor interactions. Of all $\frac{\binom{16}{8}}{2!} = 6435$ possible

Table 4.2: The 2_V^{5-1} design, the recommended block generator and the preferred block when estimating three active factors.

	A	C	D	H=BCD	P=AB	Recommended block generator	Preferred block
1	-1	-1	-1	-1	1	1	1
2	1	-1	-1	-1	-1	-1	1
3	-1	-1	-1	1	-1	1	1
4	1	-1	-1	1	1	-1	-1
5	-1	1	-1	1	1	-1	1
6	1	1	-1	1	-1	1	-1
7	-1	1	-1	-1	-1	-1	-1
8	1	1	-1	-1	1	1	-1
9	-1	-1	1	1	1	1	-1
10	1	-1	1	1	-1	-1	-1
11	-1	-1	1	-1	-1	1	-1
12	1	-1	1	-1	1	-1	1
13	-1	1	1	-1	1	-1	-1
14	1	1	1	-1	-1	1	1
15	-1	1	1	1	-1	-1	1
16	1	1	1	1	1	1	1

ways to block the design, only 60 had these properties. All of them were partially confounded with four of the $\binom{5}{2} = 10$ possible two-factor interactions.

At first, results from estimating all effects for three active factors will be presented in section 4.1.2.1. Then results from trying to estimate as many two-factor interactions as possible for four active factors are presented in section 4.1.2.2. That section also introduces a method to generate the preferred blocks when four specific factors are suspected to be active. Section 4.1.2.3 presents the results from estimating three two-factor interactions when four factors are active.

Table 4.3: The 2_V^{5-1} design, the recommended block generator and the preferred block when estimating three active factors. Here the columns have been reordered to the usual representation. The column names from the original order in table 4.2 are in parentheses.

	A	B(=C)	C(=D)	D(=H)	E(=P)	Recommended block generator	Preferred block
1	-1	-1	-1	-1	1	1	1
2	1	-1	-1	-1	-1	-1	1
7	-1	1	-1	-1	-1	-1	-1
8	1	1	-1	-1	1	1	-1
11	-1	-1	1	-1	-1	1	-1
12	1	-1	1	-1	1	-1	1
13	-1	1	1	-1	1	-1	-1
14	1	1	1	-1	-1	1	1
3	-1	-1	-1	1	-1	1	1
4	1	-1	-1	1	1	-1	-1
5	-1	1	-1	1	1	-1	1
6	1	1	-1	1	-1	1	-1
9	-1	-1	1	1	1	1	-1
10	1	-1	1	1	-1	-1	-1
15	-1	1	1	1	-1	-1	1
16	1	1	1	1	1	1	1

4.1.2.1 Three active factors

Among the 60 preferred blocks, all blocks were equally capable of estimating all effects when three factors were active. There are $\binom{5}{3} = 10$ different combinations of three factors. For each block, two of the combinations gave a D_s-efficiency of 1, while the remaining eight gave a D_s-efficiency of 0.917, yielding an average of 0.934. For each combination of three factors, there were 12 blocks yielding D_s=1, and 48 blocks yielding D_s=0.917.

Thus a high D_s-efficiency can be expected no matter which of the blocks is used and which three factors are active, making the blocks suitable for screening. Using any of the 60 blocks resulted in a (16,5,3,2) screen. If there is any suspicion about which three factors might be active, the preferred block shown in table 4.2 can be used. It yields D_s=1 when A, C and P are the active factors, so the suspected active ones should be assigned to those design columns. The block is presented next to the recommended block when using blocking by confounding for comparison. They differ quite a bit, but are both symmetric across the middle of the design.

4.1.2.2 Four active factors

When four active factors are present, there are $\binom{4}{2} = 6$ two-factor interactions. Each of the 60 blocks enabled estimation of all main effects and two-factor interaction effects for four out of five possible combinations of four active factors, with a D_s-efficiency of 0.939. For the combination of four active factors for which all two-factor interaction effects were not estimable, there are six possible ways to choose five out av six two-factor interactions. The block then enabled estimation of all five two-factor interactions for four of the six combinations, with a D_s-efficiency of 0.871. Two of the combinations could not be estimated as each block can be written as a linear combination of four two-factor interactions. If all four of these are included in the five two-factor interactions to be estimated, the determinant of the design matrix becomes 0. Ensuring that these four factors are not all included is sufficient to enable estimation of five two-factor interactions. The section ”Generating the blocks” below introduces a method for generating suitable blocks if there is a suspicion about which factors may be active.

Generating the blocks
All 60 blocks were made up of combinations of four two-factor interactions on the form $\frac{1}{2}(f_A + f_B + f_C - f_D)$, where f_A, f_B, f_C and f_D denotes different two-factor interactions. They have the property that $f_A f_B = \pm(f_C f_D)$ is equal to a two-factor interaction f_E. The two-factor interaction f_E might consist of any combination of two factors among the columns A, C, D, P and H in table 4.2 and B in table 4.1 except AB, AP and BP, as using the interaction $AB = P$, $AP = AAB = B$ or $AAB = B$ makes the two-factor interaction effect f_E aliased with a main effect. This leaves 12 possible choices for f_E. Having found the four factors that make up f_E, they can be combined into four different blocks b_1, b_2, b_3 and b_4 using the formulas $b_1 = \frac{1}{2}(f_A + f_B + f_C - f_D)$, $b_2 = \frac{1}{2}(f_A + f_B - f_C + f_D)$, $b_3 = \frac{1}{2}(f_A - f_B + f_C + f_D)$ and $b_4 = \frac{1}{2}(-f_A + f_B + f_C + f_D)$.

Worked example
To clarify the procedure, assume for example that the experimenter suspects the factors A, C, D and H to be active. To ensure that she is able to estimate all two-factor interactions, she should choose a block which is not made up merely by two-factor interactions combining A, C, D and H. Thus she chooses A, C, D and P instead. Then the possible choices of f_E are $AC = ADCD = APCP$, $AD = ACCD = APDP$, $CD = ACAD = CPDP$, $CP = ACAP = CDDP$ and $DP = ADAP = CDCP$. As CP and AD, and DP and AC, respectively, are made up of the same four two-factor interactions, these choices yield the same blocks. Choosing $f_E = AC$, $f_A = AD$, $f_B = CD$, $f_C = AP$ and $f_D = CP$, it is sufficient to generate one of the blocks $b_1 = \frac{1}{2}(AD + CD + AP - CP)$, $b_2 = \frac{1}{2}(AD + CD - AP + CP)$, $b_3 = \frac{1}{2}(AD - CD + AP + CP)$ or $b_4 = \frac{1}{2}(-AD + CD + AP + CP)$. If the designer is unlucky, and the active factors turn out to be A, C, D and P, she will not be able to estimate all the two-factor interactions AD, CD, AP and CP at the same time, but must choose one to leave out. The two last two-factor interactions, AC and DP, are not used to define the blocks, and can always be estimated. Note that if any other combination of factors than A, C, D, P is active, she will be able to estimate all two-factor interactions.

The block resulting from choosing the formula $\frac{1}{2}(AD+CD+AP-CP)$ is
-1, 1, 1, -1, -1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1
^{T}. When estimating all effects for three active factors, it yields a D_s-efficiency of 1 for the combinations ACH and PDH, and 0.917 for the rest. If four factors are active, as in the example, the D_s-efficiencies become more spread depending on the active factors. The results when estimating five out of six two-factor interactions are shown in table 4.4. "Rem. f1f2" means that all main effects and two-factor interactions have been estimated except f1f2.

Table 4.4: D_s-efficiencies obtained for different combinations of four active factors for the example block. All main effects and five out out six two-factor interactions are estimated in each case. "Rem f1f2" means that the two-factor interaction f1f2 was not estimated.

f1	f2	f3	f4	Rem. f1f2	Rem. f1f3	Rem. f1f4	Rem. f2f3	Rem. f2f4	Rem. f3f4
A	P	C	D	0.871	0	0.871	0.871	0	0.871
A	P	C	H	0.972	0.9332	0.933	0.972	0.933	0.933
A	P	D	H	0.972	0.972	0.933	0.933	0.933	0.933
A	C	D	H	0.933	0.972	0.933	0.972	0.933	0.933
P	C	D	H	0.972	0.933	0.933	0.972	0.933	0.933

As expected, the block performs badly when A, P, C and D are the active factors. When AC or DP is removed, all the two-factor interactions that make up the block remains, and the D_s-efficiency is 0. Another interesting observation is that when any of the other combinations of four factors is active, two combinations of five two-factor interactions yield a D_s-efficiency of 0.972, while the remaining four yield a D_s-efficiency of 0.933. When for example A, P, C and H are active, the D_s-efficiency is 0.972 when AP or CP is not included. These are the only two-factor interactions for A,P,C and H which are used in the formula for the block. The same pattern is repeated for the other combinations. It seems reasonable that the D_s-efficiency is high when interactions that are partially confounded with the block effect are not included. Note also that all six two-factor interactions could have been estimated when APCH, APDH, ACDH or PCDH are the active factors, with a D_s-efficiency of 0.933.

4.1.2.3 Estimating three two-factor interactions

As shown above, five out six two-factor combinations could not be estimated for all combinations of four active factors. If any a two-factor interactions should be estimable regardless of which four factors are active, $a = 3$ is the highest number for which no combination yielded $D_s = 0$. Thus the blocked design is a (16,5,4_{1+3},2) screen. No matter which combination of four active factors is active, all main effects and three optional two-factor interactions were estimable. All 60 blocks yielded the same frequencies of D_s-efficiencies, which can be found in table 4.5. It shows the frequencies of different values for all 20 possible combinations of three two-factor interactions for the five combinations of four active factors.

All blocks did for example have $D_s = 1$ for four combinations of two-factor interactions for four out of five combinations of four active factors, thus the total in the table is 16 for $D_s = 1$. The average D_s-efficiency was 0.952 for all blocks. The block [1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1]^T, which is also used in the "Worked example" section above, yielded a D_s-efficiency of 1 among others when the active factors were APCH, and the estimated two-factor interactions were AH, PH and CH.

Table 4.5: Frequencies of the different D_s-values for all combinations of four active factors when blocking the 2_V^{5-1} design, estimating all possible combinations of three out of six two-factor interactions.

D_s-efficiency	Occurences
1	16
0.9646786	52
0.9170040	28
0.8408964	4

Evaluation of results
Although the design is a (16,5,4_{1+3},2) screen, this blocking approach might still be preferable to the blocking by confounding of interactions. If the confounding approach is applied, there will be one two-factor interaction which cannot be estimated in any case. Thus five out of six two-factor interactions can be estimated, but there is no way to choose which these are. Using one of the blocks suggested by the procedure presented in section 3.3 gives a $\frac{4}{5}$ chance that all two-factor interactions can be estimated, and if all are not estimable, four given combinations of five two-factor interactions can be estimated, or three freely chosen. Hence there is much greater flexibility, which is important when screening for active factors.

One important aspect which might be relevant for larger designs is that all four candidate blocks from the H_2 design in the article were among the 60 preferred blocks. When using larger designs, testing all blocks and looking for patterns among the preferred blocks is not feasible. Then rearranging a Hadamard matrix to form the design and choosing the remaining columns as blocking candidates is a less work-intensive approach. Note however that the number of Hadamard-matrices largely increases when the design size increases.

4.2 32-run designs

Blocking of five 32-run designs was tested: A 2_{IV}^{16-11} design , a 2_{VI}^{6-1} design, a 2_{IV}^{7-2} design, a 2_{IV}^{8-3} design and a 2_{IV}^{9-4} design.

The 2_{IV}^{16-11} design was tested as it is the design of projectivity $P = 3$ with 32 runs which can accommodate the highest numbers of factors, as all odd-numbered interactions are used as design generators. The 2_{IV}^{16-11} design was blocked both by utilising mirror image pairs and the blocking of the 2_{IV}^{8-4} design. Results from using mirror image pairs to divide the design into two blocks can be found in section 4.2.2, while results from using mirror image pairs to divide the designs into four blocks are presented in section 4.2.3. Results from utilising the blocking of the 2_{IV}^{8-4} design can be found in section 4.2.1. In all cases, three active factors were considered.

The 2_{VI}^{6-1} design was tested as it is the 32-run design of projectivity $P = 4$ which can accommodate the highest numbers of factors. The design was divided into two and four blocks by utilising mirror image pairs. Results for dividing the design in two blocks and considering three, four and five active factors can be found in sections 4.2.4.1, 4.2.4.2, 4.2.4.4 and 4.2.4.5. Results from dividing the design into four blocks and considering three, four and five active factors are presented in sections 4.2.5.1, 4.2.5.2 and 4.2.5.3.

A 2_{IV}^{7-2}, a 2_{IV}^{8-3} and a 2_{IV}^{9-4} design were also blocked, to see if they yielded better results than using a subset of the columns of the 2_{IV}^{16-11} design in the cases of 7, 8 and 9 active factors. As these designs do not consist of mirror image pairs, they were blocked by rearranging Hadamard matrices. The results from dividing the 2_{IV}^{7-2} design into two and four blocks can be found in sections 4.2.6.1 and 4.2.6.2, while the results from dividing the 2_{IV}^{8-3} design into two and four blocks can be found in sections 4.2.6.3 and 4.2.6.4. Finally, the results from dividing the 2_{IV}^{9-4} design into two and four blocks can be found in sections 4.2.6.5 and 4.2.6.6. In all cases, three active factors were considered.

4.2.1 Dividing a 2_{IV}^{16-11} design into two blocks using the blocking of the 2_{IV}^{8-4} design

Having found the best blocks using the mirror image pair strategy for the 2_{IV}^{8-4} design in section 4.1.1, they might be possible to exploit when blocking the doubled design using the method explained in section 3.2.1. The doubling, a 2_{IV}^{16-11} design, has projectivity $P = 3$. It is equal to the design which will later be blocked using mirror image pairs in section 4.2.2. As 28 blocks yielded good results for the 2_{IV}^{8-4} design, and there were three different patterns which might be used to define the new blocks, there were 84 candidate blocks for the design. As mentioned in section 3.2.1, the D_1 pattern cannot be used when estimating more than main effects. The blocks originating from this pattern were therefore not interesting, leaving 56 candidate blocks.

The remaining 56 blocks were tested for all $\binom{16}{3} = 560$ ways to choose three active factors among the 16 screening factors. The blocks based on D_2 and D_3 all yielded $D_s > 0$ for all combinations of three active factors, making the blocked design a $(32, 16, 3, 2)$ screen. Furthermore, they were equally good in terms of frequencies of D_s-efficiencies. The average D_s-efficiency for all blocks was 0.970. The frequencies of the different D_s-efficiencies can be found in table 4.6. An example of the design with corresponding preferred block b is presented in table 4.7. The design was found using the pattern D_2 together with block $b28$ from table 6.2. The resulting block yields D_s=1 for among others the three factors A, B and D.

Table 4.6: Frequencies of the different D_s-values when testing the preferred block for the doubling of the 2^{8-4}_{IV} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurrences
1	208
0.965	256
0.917	96

As will be shown in the next section, a 32-run design with 16 factors can be blocked with a slightly higher average D_s-efficiency and the same minimum D_s-efficiency using the mirror image pairs strategy. Thus the doubling approach presented here is not best method for finding blocks, but may be useful for larger designs because of the scalability of the method. For instance, testing all combinations of mirror image pairs cannot be done effectively for a 64-run design, and then the doubling approach might be used to find candidate blocks instead. The method can also be used when dividing into four or more blocks. That will be tested in section 4.3.2.

Table 4.7: The 2_{IV}^{16-11} design arranged in one of the preferred blockings in section 4.2.1.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	*b*
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1	-1
-1	1	-1	-1	1	1	-1	1	-1	1	-1	-1	1	1	-1	1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1
-1	-1	1	-1	1	-1	1	1	-1	-1	1	-1	1	-1	1	1	-1
1	1	-1	1	-1	1	-1	-1	1	1	-1	1	-1	1	-1	-1	-1
-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1
1	-1	1	1	-1	-1	1	-1	1	-1	1	1	-1	-1	1	-1	-1
-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1	-1
1	-1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	-1	1	-1
-1	1	-1	-1	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	1	-1	1	-1	1	1	1	1	-1	1	-1	1	-1	-1	-1
1	1	-1	1	-1	1	-1	-1	-1	-1	1	-1	1	-1	1	1	-1
-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1
1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	-1
-1	1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1
-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1
1	1	1	-1	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1
-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	1
1	-1	-1	1	1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	1
-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1
-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	1
1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	1	1	1	1
-1	-1	-1	1	-1	1	1	1	1	1	1	-1	1	-1	-1	-1	1
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1
-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1

4.2.2 Dividing a 2_{IV}^{16-11} design into two blocks using MIP

The idea of using mirror image pairs to find candidate blocks which are orthogonal to the main effects may be exploited for designs of any size, as long as they are made up of mirror image pairs. The approach will here be tested for the 32-run 2_{IV}^{16-11} design in table 4.10, which is also shown with interchanged rows in table 4.2.2. In addition to being the doubling of the 2_{IV}^{8-4} design, this is the 32-run regular design of projectivity $P = 3$ which can be used to screen the highest number of factors. This is because all odd-numbered interactions are used as design generators. The best blocks found in this section will be used to divide the doubling, a 2_{IV}^{32-26} design, into two blocks in section 4.3.1.

There is no suggested block generator for this design in Wu and Hamada [7]. However, as all odd interactions are used as design generators, using a three-factor interaction as block generator leads to confounding with a main effect, as does using the five-factor interaction or a four-factor interaction. This can easily be seen by multiplying the potential block generator with the defining relation of the 2_{IV}^{16-11} design, I=ABCF=ABDG=ABEH=...=ABCDEP. Using a two-factor interaction as block generator is therefore a likely choice when using confounding of interactions, as it keeps the main effects unaliased when there is assumed to be no interactions between the block effects and the main effects. This choice nevertheless yields projectivity $P = 1$, as a two-factor interaction is then confounded with the block. As the projectivity of the original design is $P = 3$, it is interesting to see if estimating all effects is possible if three out of 16 factors are active when using the mirror-image blocking approach.

The design consists of 16 mirror image pairs, yielding $\frac{\binom{16}{8}}{2!} = 6435$ possible blocks with eight mirror image pairs in each block. As for the 2_{IV}^{8-4} design, blocks which are identical with interaction factors had to be removed to yield a D_s-efficiency above 0. There are in total $\binom{5}{2} + \binom{5}{4} = 15$ symmetrical two- and four-factor interaction effects. This left 6420 candidate blocks, all yielding $D_s > 0$ for all combination of three active factors, making the blocked design a $(32, 16, 3, 2)$ screen. There are in total $\binom{16}{3} = 560$ possible combinations of three active factors, so 3595200 combinations of blocks and active factors were tested.

Considering the different blocks, the average D_s-efficiencies when testing all combinations of three active factors were in the range 0.967 to 0.971. The blocks that had the lowest average D_s-efficiency corresponded to the occurrences of D_s-efficiencies of 0.88. Removing these blocks removed the risk of getting the lowest efficiency. Inspecting the dot product of the block columns with all the $\binom{16}{2} = 120$ possible two-factor interactions revealed that these block effects were all strongly partially confounded with eight of the two-factor interactions each. Strongly partially confounded is here defined as having a dot product equal to 24, whereas a completely confounded block effect has a product of 32.

The average D_s-efficiencies after removing these blocks were between 0.970 and 0.971. Again considering the confounding with two-factor interactions, it was discovered that the blocks having a dot product of 16 with 32 two-factor interactions yielded the lowest average. This was the case for 420 blocks. The remaining 5040 blocks had a dot product of 16 with only 16 two-factor interactions each. Removing the 420 worst blocks lead to all the remaining blocks having an average D_s-efficiency of 0.971. All blocks had the same frequencies of different D_s-values, which can be found in table 4.8. Considering each combination separately, they had an equal number of blocks yielding each D_s-efficiency. The frequencies can be found in table 4.9. Column b in table 4.10 shows one of the blocks, b. It has $D_s = 1$ for among others the active factors A, B and I, so any factors suspected to be active should be assigned to those columns when performing an experiment using that block.

Table 4.8: Frequencies of the different D_s-values when testing the preferred block for the 2_{IV}^{16-11} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurences
1	32
0.983	320
0.965	64
0.943	128
0.917	16

Table 4.9: Frequencies of the different D_s-values when testing all blocks for the 2_{IV}^{16-11} design for each combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurences
1	288
0.983	2880
0.965	576
0.943	1152
0.917	144

Comparison with the blocking based on doubling

In section 4.2.1, the 2_{IV}^{16-11} design was blocked by creating the design and corresponding blocks based on doubling of the 2_{IV}^{8-4} design and its preferred blocks. The highest average D_s-efficiency obtained using that approach was 0.967, only slightly lower than the highest average of 0.971 obtained when testing all blocks based on mirror image pairs in this section. The minimum- and maximum D_s-values were equal in both cases. The number of blocks tested for the doubling approach was 84, while the number of mirror image pair blocks tested was 6435. Thus the doubling approach was much more efficient in terms of trials to find the preferred block, and might be a useful method for larger designs despite not necessarily finding a block yielding the highest average D_s-efficiency.

Table 4.10: The 2_{IV}^{16-11} design and the preferred block in section 4.2.2.

					ABC	ABD	ABE	ACD	ACE	ADE	BCD	BCE	BDE	CDE	ABCDE	b
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1
1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1	1
-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1	1
1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	1
-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1	1
1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1	1
-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1	-1
1	1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	1	1	-1
-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1	1
1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	-1
-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1
1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1	-1
-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1	-1
1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1	-1
-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1	-1
1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1	-1
1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1	-1
-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1	-1
1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1	-1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1
-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	-1
1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1
-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1	-1
1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	-1
-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1	1
1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1	1
-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1
1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1	1
-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4.2.3 Dividing a 2_{IV}^{16-11} design into four blocks using MIP

Having found in section 4.2.2 that the 2_{IV}^{16-11} design in table 4.10 could be divided into two blocks yielding a $(32, 16, 3, 2)$ screen and high D_s-efficiencies, an interesting question is whether the design can also be divided into four blocks and maintain a high projectivity and high D_s-efficiencies. The four blocks are represented by two columns, where the row combinations (-1,-1), (-1,1), (1,-1) and (1,1) denotes the four different blocks. When calculating the D_s-efficiency for a design with four blocks, the interaction effect between the blocks is also included in the design matrix. The best blocks found in this section will be used to block the doubling, a 2_{IV}^{32-26} design, into four blocks in section 4.3.2.

There are $\frac{\binom{16}{4}\binom{12}{4}\binom{8}{4}}{4!} = 2627625$ possible ways to block the mirror image pairs into four blocks when each block must contain both runs of a pair. Testing all these blocks, 2098336 of them yielded an average D_s-efficiency above 0 for all the $\binom{16}{3} = 560$ possible combinations of three active factors, making the blocked design a $(32, 16, 3, 4)$ screen. Among these, 715680 obtained the highest possible minimum D_s-efficiency of 0.834. All of these had an average D_s-efficiency between 0.905 and 0.909. 0.909 was also the highest average D_s-efficiency among all blocks. As the averages were very equal, using any of these blocks ensures good results. If one has a suspicion about which factors might be active, the 50400 of the blocks having the highest possible minimum D_s-efficiency which also had a max D_s-efficiency of 1 are the most suited.

One example is shown in table 4.11. The order of the rows in the corresponding design matrix should be the same as for the 2_{IV}^{16-11} design in table 4.10. This blocking yields D_s=1 when the factors A, B and C are active. Table 4.12 shows the frequencies of the different D_s-efficiencies obtained when using this design. The average D_s-efficiency using these blocks was 0.908, which is the highest among the blocks having both the highest minimum and maximum D_s-efficiency, and only marginally lower than the highest obtained average of 0.909.

Table 4.11: One of the preferred ways to divide the 2_{IV}^{16-11} design into four blocks.

$b1$	$b2$	$b1b2$
-1	-1	1
-1	-1	1
-1	-1	1
-1	1	-1
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1
1	-1	-1
1	1	1
1	1	1
1	1	1
1	-1	-1
1	-1	-1
-1	1	-1
-1	1	-1
-1	1	-1
-1	1	-1
1	-1	-1
1	-1	-1
1	1	1
1	1	1
1	1	1
1	-1	-1
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1
-1	1	-1
-1	-1	1
-1	-1	1
-1	-1	1

Table 4.12: Frequencies of the different D_s-values when testing the preferred division into four blocks for the 2_{IV}^{16-11} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurrences
1	16
0.965	32
0.931	64
0.927	192
0.917	32
0.885	32
0.883	128
0.834	64

4.2.4 Dividing a 2_{VI}^{6-1} design into two blocks using MIP

The 2_{IV}^{16-11} design discussed in section 4.2.2 may be reduced to a regular 2_{VI}^{6-1} design of projectivity $P = 5$ and resolution VI by removing all the factors defined by three-factor design generators. The resulting design can be found in table 4.13. This design also consists of 16 mirror image pairs. The most important difference from the 2_{IV}^{16-11} design is that screening for more than three active factors may be possible, as no three-factor interactions are confounded with main effects in this case. At first, the results from testing blocks in the case of three factors are presented in section 4.2.4.1. Then the case of four active factors is presented in section 4.2.4.2, until finally the case of five active factors is presented in section 4.2.4.3. The best blocks in the cases of three and four active factors will be used to divide the design into four blocks in section 4.2.5.4.

Table 4.13: The 2_{VI}^{6-1} design, the recommended block generator (AB) and the preferred block in the case of three, four and five active factors.

A	B	C	D	E	F=ABCDE	Recommended block generator, ABC	Preferred block three active factors	Preferred block four active factors	Preferred block five active factors
-1	-1	-1	-1	-1	-1	-1	1	1	1
1	-1	-1	-1	-1	1	1	1	1	1
-1	1	-1	-1	-1	1	1	1	1	1
1	1	-1	-1	-1	-1	-1	1	1	1
-1	-1	1	-1	-1	1	1	1	1	1
1	-1	1	-1	-1	-1	-1	1	1	1
-1	1	1	-1	-1	-1	-1	-1	-1	-1
1	1	1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	1	1
1	-1	-1	1	-1	-1	1	1	-1	-1
-1	1	-1	1	-1	-1	1	-1	1	-1
1	1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	1	-1	-1	1	-1	-1	-1
1	-1	1	1	-1	1	-1	-1	-1	-1
-1	1	1	1	-1	1	-1	-1	-1	-1
1	1	1	1	-1	-1	1	1	-1	1
-1	-1	-1	-1	1	1	-1	1	-1	1
1	-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	1	-1	1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1
-1	-1	1	-1	1	-1	1	-1	-1	-1
1	-1	1	-1	1	1	-1	-1	1	-1
-1	1	1	-1	1	1	-1	1	-1	-1
1	1	1	-1	1	-1	1	-1	1	1
-1	-1	-1	1	1	-1	-1	-1	-1	-1
1	-1	-1	1	1	1	1	-1	-1	-1
-1	1	-1	1	1	1	1	1	1	1
1	1	-1	1	1	-1	-1	1	1	1
-1	-1	1	1	1	1	1	1	1	1
1	-1	1	1	1	-1	-1	1	1	1
-1	1	1	1	1	-1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1

4.2.4.1 Three active factors

Testing if the 2_{VI}^{6-1} design yielded higher D_s-efficiencies than the 2_{IV}^{16-11} design when divided into two blocks and used to screen for three active factors was done by testing the 5040 blocks which were preferable for the 2_{IV}^{16-11} design. These blocks were chosen as the rest were strongly partially confounded with two-factor interactions, and thereby would be problematic for the 2_{VI}^{6-1} design as well. Testing all 5040 blocks for all $\binom{6}{3} = 20$ possible combinations of three active factors resulted in average D_s-efficiencies of either 0.971 or 0.972. The minimum D_s-efficiencies were either 0.917 or 0.943, so all blocks yielded a (32,6,3,2) screen. 720 of the blocks with the highest minimum D_s-values also yielded the highest average value.

None of the blocks which obtained the highest minimum D_s-efficiency and highest average D_s-efficiency yielded $D_s = 1$ for any combination of factors. As the average D_s-efficiencies were almost equal for all blocks, the advantage of being able to allocate factors believed to be active to a combination with $D_s = 1$ without risking a D_s-efficiency of 0.917 was prioritised. Thus one of the 1440 blocks with the highest minimum D_s-efficiency which yielded a maximum D_s-efficiency of 1 was chosen as the preferred block. It can be seen along the recommended block by confounding of interactions in table 4.13. Table 4.14 shows the frequencies of different D_s-efficiencies when testing the block for all 20 combinations of three active factors, estimating all interactions. The average D_s-efficiency was 0.971, and a D_s-efficiency of 1 is obtained when the active factors are A, B and C.

Table 4.14: Frequencies of the different D_s-values when testing the preferred division into two blocks for the 2_{VI}^{6-1} design for all combinations of three active factors, estimating all interactions.

Ds-efficiency	Occurrences
1	2
0.983	10
0.965	2
0.943	6

4.2.4.2 Four active factors

The preferred blocks for four active factors were found by testing all $\binom{6}{4}$ =15 combinations of four active factors for the 5040 blocks also tested in section 4.2.4.1. All blocks resulted in the same frequencies of D_s-efficiencies, which can be found in table 4.15. The average D_s-efficiency was 0.959, and the minimum D_s-efficiency was 0.917. Thus a reasonably good D_s-efficiency may be expected for all combinations of four factors, as is desirable for screening. To see if all blocks obtained the same D_s-efficiency for each combination of four active factors, the results using all blocks were stored for each combination. The frequencies were equal, and can be found in table 4.16.

Table 4.15: Frequencies of the different D_s-values when testing the preferred block for all combinations of four active factors, estimating all interactions.

Ds-efficiency	Occurrences
0.982	4
0.958	9
0.917	2

Table 4.16: Frequencies of the different D_s-values when testing all preferred divisions into two blocks for each combination of four active factors, estimating all interactions.

Ds-efficiency	Occurrences
0.982	1344
0.958	3024
0.917	672

As the 5040 blocks were equally good, one is here chosen for consideration. It can be seen along the preferred block for three active factors in table 4.13. As the D_s-efficiencies are all above 0, the resulting design is a $(32,6,4,2)$ screen. The highest D_s-efficiency of 0.982 is obtained when among others A, B, C and D are the active factors. The D_s-efficiencies for all the 15 different possible factor combinations can be found in table 6.13 in Appendix A.

4.2.4.3 Five active factors

If there are five active factors among the six factors in the 2^{6-1}_{VI} design in table 4.13, it is not possible to estimate all effects when the design is blocked, as the number of effects then exceeds the number of rows. But removing the five-factor interaction, there are only 32 effects remaining. Testing all 5040 blocks used in the previous sections revealed that estimating all effects up to the four-factor interactions was not possible. For 720 blocks, it was however possible to estimate two out of five four-factor interactions for all six possible combinations of five active factors. In addition, each block enabled estimation of three out of five four-factor interactions for two of the six possible combinations of five active factors. The results when estimating all effects up to three-factor interactions for five active factors are presented in section 4.2.4.4. The results when two four-factor interactions are included can be found in section 4.2.4.5.

4.2.4.4 Estimating three-factor interactions

For all combinations of five active factors, all effects up to three-factor interactions were estimable using any of the 5040 candidate blocks, making the blocked design a (32,6,5_3,2) screen. 720 of these had both the highest minimum D_s-efficiency of 0.948 and the highest average D_s-efficiency of 0.958. The blocks did however not have the highest maximum D_s-efficiency; the highest possible was 0.982, whereas the highest for the 720 preferred blocks was 0.963. For each block, the D_s-efficiency was 0.948 for two combinations of five active factors, and 0.963 for the remaining four combinations of five active factors. For the block shown in table 4.13, which will also be used as the example in the next section, the D_s-efficiency was 0.963 for the combinations ABCDE, ABCDF, ABCEF and BCDEF.

4.2.4.5 Estimating two four-factor interactions

For 720 of the 5040 candidate blocks, it was possible to estimate two out of five four-factor interactions for all six possible combinations of five active factors. The resulting blocked design is then a $(32, 6, 5_{3+2}, 2)$ screen. These 720 blocks could

estimate two out of five four-factor interactions with an average D_s-efficiency of 0.939. For all blocks, the lowest D_s-efficiency was 0.906, and the highest D_s-efficiency 0.966. One of the blocks for which these results were obtained is shown in table 4.13. Note that the same block was used when only considering three-factor interactions. The number of combinations for which different D_s-efficiencies were obtained can be found in table 4.17. The occurrences sum to 60 as there are six ways to choose five active factors, and for each of them, there are ten possible combinations of two out of five four-factor interactions. The highest D_s-efficiency for this block is obtained for the combinations shown in table 4.18.

Table 4.17: Frequencies of the different D_s-values when testing the preferred blocking of the 2^{6-1}_{VI} design for all combinations of five active factors, estimating two out of five four-factor interactions.

D_s-efficiency	Occurences
0.966	4
0.959	16
0.952	4
0.942	8
0.928	20
0.906	8

Table 4.18: The combinations yielding the highest D_s-efficiency for the preferred block when dividing the 2^{6-1}_{VI} design into two blocks and estimating all effects for the five active factors up to the two four-factor interactions "Four factor interaction 1" and "Four factor interaction 2".

Active factors	Four factor interaction 1	Four factor interaction 2
ABCDE	ABCD	ABCE
ABCDF	ABCD	BCDF
ABCEF	ABCE	BCEF
BCDEF	BCDF	BCEF

4.2.5 Dividing a 2_{VI}^{6-1} design into four blocks using MIP

As the 2_{VI}^{6-1} design discussed in section 4.2.4.2 could be divided into two blocks in 5040 ways which guaranteed a D_s-efficiency above 0.9 when screening for four active factors and estimating all their effects, investigating the possibility of division into even more blocks is interesting. The design was therefore divided into four blocks, using the same method as in section 4.2.3. Thus there were more than 2.6 million possible blocks also in this case.

At first, it was tested whether the 2_{VI}^{6-1} design could be used to screen for three active factors with higher D_s-efficiencies than the 2_{IV}^{16-11} design tested in section 4.2.3 when divided into four blocks. The results are presented in section 4.2.5.1. Then estimating all effects for four active factors is tested in section 4.2.5.2. As it worked when dividing the design into two blocks, estimating up to three-factor interactions for five active factors is tested in section 4.2.5.3. Since the resulting D_s-efficiencies were rather low, estimation of additional four-factor interactions was not tested.

4.2.5.1 Three active factors

Testing all the possible blocks, 2252640 were found to yield a minimum D_s-efficiency above 0 for all combinations of three active factors when estimating all effects. The highest minimum D_s-efficiency was 0.875, which was obtained by 2880 blocks. The highest average D_s-efficiency among these was 0.908, while the highest obtained average D_s-efficiency among all the 2252640 blocks was 0.911. Thus little is lost by choosing one of the blocks with the highest minimum D_s-efficiency to gain robustness. The maximum D_s-efficiency among all blocks was 1, while the maximum D_s among the 2880 preferred blocks was 0.949. Table 4.20 shows the frequencies of different D_s-efficiencies obtained for the block in table 4.19. The highest D_s-efficiencies was obtained among others for the combination ABD of active factors.

4.2.5.2 Four active factors

Testing all blocks revealed that for the 1491840 blocks which had a D_s-efficiency above 0 for all 15 possible combinations of four active factors, the average D_s-efficiencies were between 0.861 and 0.870. The maximum D_s-efficiency was 0.924 for all blocks, but the minimum D_s-efficiencies were either 0.805 or 0.826. Thus the 483840 blocks with a minimum D_s-efficiency of 0.826 were preferable. The average D_s-efficiencies for these blocks were 0.869 for 322560 of the blocks and 0.870 for the remaining 161280 blocks. Although the results were very similar for all blocks which could handle all combinations of four factors, 161280 were slightly better than the rest. One of these blocks are shown in table 4.21. The D_s-efficiencies for different combinations of factors are shown in table 6.14 in Appendix A, while the frequencies of different D_s-efficiencies can be found in table 4.22. The maximum is obtained when among others A, B, C and D are the active factors. As the D_s-efficiency is above 0 for all combinations, the design is a $(32,6,4,4)$ screen.

4.2.5.3 Five active factors

In the case of five active factors, it was tested whether up to three-factor interactions could be estimated, as it was possible when dividing the design into two blocks. Testing all possible blocks, 1988160 enabled estimation of all effects up to three-factor interactions, with average D_s-efficiencies between 0.82976 and 0.8660, making the blocked design a (32,6,5_3,4) screen.

The highest minimum D_s-efficiency was 0.866, obtained by 1920 blocks. These blocks obtained the same D_s-efficiency for all six combinations of five active factors. The maximum D_s-efficiency obtained among the 1988160 blocks with a minimum D_s above 0 was 0.954, but then a D_s-efficiency of 0.801 was obtained for three of the six possible combinations of active factors. The 1920 blocks with the highest minimum D_s-efficiency were therefore preferred. One of the blocks can be found in table 4.23. A frequency table is not included here, as the D_s-efficiency was 0.866 for all combinations.

Table 4.19: One of the preferred ways to divide the 2^{6-1}_{VI} design into four blocks when estimating all effects for three active factors.

$b1$	$b2$	$b1b2$
-1	-1	1
-1	-1	1
-1	-1	1
1	-1	-1
1	-1	-1
-1	-1	1
1	1	1
1	1	1
-1	1	-1
-1	1	-1
1	1	1
1	-1	-1
1	1	1
1	-1	-1
-1	1	-1
-1	1	-1
-1	1	-1
-1	1	-1
1	-1	-1
1	1	1
1	-1	-1
1	1	1
-1	1	-1
-1	1	-1
1	1	1
1	1	1
-1	-1	1
1	-1	-1
1	-1	-1
-1	-1	1
-1	-1	1
-1	-1	1

Table 4.20: Frequencies of the different D_s-values when testing the preferred division into four blocks for the 2^{6-1}_{VI} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurrences
0.949	8
0.885	6
0.875	6

Table 4.21: One of the preferred ways to block the 2^{6-1}_{VI} design into four blocks, estimating all interactions for four active factors.

$b1$	$b2$	$b1b2$
-1	-1	1
-1	-1	1
-1	-1	1
1	-1	-1
-1	1	-1
-1	-1	1
1	-1	-1
1	1	1
-1	1	-1
1	1	1
1	-1	-1
1	-1	-1
1	1	1
1	1	1
1	-1	-1
-1	1	-1
-1	1	-1
1	-1	-1
1	1	1
1	1	1
1	-1	-1
1	-1	-1
1	1	1
-1	1	-1
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1
-1	1	-1
-1	-1	1
-1	-1	1
-1	-1	1

Table 4.22: Frequencies of the D_s-efficiencies obtained for the blocking in table 4.21 when testing all combinations of four active factors, estimating all interactions.

D_s-efficiency	Occurences
0.924	2
0.890	4
0.871	1
0.862	3
0.853	2
0.826	3

Table 4.23: One of the preferred ways to block the 2^{6-1}_{VI} design into four blocks, estimating up to three-factor interactions for five active factors.

$b1$	$b2$	$b1b2$
-1	-1	1
-1	-1	1
-1	-1	1
1	-1	-1
-1	1	-1
-1	-1	1
1	1	1
1	1	1
-1	1	-1
1	-1	-1
-1	1	-1
1	-1	-1
1	1	1
1	1	1
1	-1	-1
-1	1	-1
-1	1	-1
1	-1	-1
1	1	1
1	1	1
1	-1	-1
-1	1	-1
1	-1	-1
-1	1	-1
1	1	1
1	1	1
-1	-1	1
-1	1	-1
1	-1	-1
-1	-1	1
-1	-1	1
-1	-1	1

4.2.5.4 Utilising the division into two blocks for division into four blocks

Another approach to finding blocks than creating all combinations based on mirror image pairs is to let the division into four blocks be based on the division into two blocks, as explained in section 3.1.1. That is, to divide each block of eight mirror image pairs into two blocks of four mirror image pairs each. Each block can be divided into $\frac{\binom{8}{4}}{2!} = 35$ different combinations of two blocks of four mirror image pairs. This yields $35 \cdot 35 = 1225$ ways to make four blocks based on each of the blocks which were preferred when dividing the design into two blocks. The approach was tested in the case of three and four active factors.

Testing three active factors
As the 5040 blocks that were tested for dividing the 2^{6-1}_{VI} design into two blocks yielded very similar results, they were all used to find divisions into four blocks. Thus there where 6174000 blocks created this way. Some of these were duplicates, as the blocks were still based on mirror image pairs, and only 2.6 million unique blocks existed using the mirror image approach in section 4.2.5. To find the best blocks using this method, all the 1225 combinations of four blocks were tested for all the 5040 blocks. The new block yielding the highest average D_s-efficiency and the new block yielding the highest minimum D_s-efficiency was stored for all 5040 blocks.

The highest average D_s-efficiencies were all in the range from 0.9095 to 0.9107. The maximum value was 0.949 for 1097 blocks, and 0.965 for the remaining 3943. The minimum value differed a lot more, ranging from 0.813 to 0.865. Thus it seemed like considering the blocks yielding the highest minimum values might be advantageous.

Among the blocks obtaining the highest minimum D_s-efficiency when testing all combinations of three active factors, the minimum value ranged from 0.841 to 0.875, which was obtained for 630 blocks. The maximum value was between 0.949 and 1, while the average was in the range from 0.904 to 0.910. None of the blocks with the highest minimum D_s-efficiency obtained neither the highest maximum D_s-efficiency nor the highest average D_s-efficiency. All the blocks with the

highest minimum D_s-efficiency obtained a maximum D_s-efficiency of 0.949. The highest average D_s-efficiency obtained among them was 0.908, obtained by 260 blocks. These were the same numbers as obtained when testing all divisions into four blocks using mirror image pairs in section 4.2.5.1. Thus the 260 preferred blockings here are a subset of all the 2880 preferred blockings. As one of these has already been presented in section 4.2.5.2, it will neither be repeated here, nor in the summary of the results in tables 4.48 and 4.49 in section 4.4.

Testing four active factors
When aiming to estimate all effects for four active factors, the blocks were based on the 5040 blocks which were preferred for division into two blocks in section 4.2.4.2. These were the same as when testing three active factors.

As when testing three active factors, the new block yielding the highest average was stored for all the 5040 blocks. Among these, 4479 blocks had an average of 0.87012, while the remaining 561 had an average D_s-efficiency of 0.87005. The blocks with the highest average D_s-efficiency had a minimum D_s-efficiency of 0.805 for one of the factor combinations. The blocks with the lowest average D_s-efficiency had a minimum D_s-efficiency of 0.826, and thus might be preferred for robustness.

When instead the new block yielding the highest minimum value of D_s-efficiency was stored for all 5040 blocks, the average D_s-efficiency was found to be 0.870 for all blocks, but the lowest D_s-efficiency obtained for any factor combination for each block was 0.826. Thus a higher minimum D_s-efficiency was not obtained using this method, so the 561 blocks with the second-highest average D_s-efficiency and the highest minimum D_s-efficiency were preferable. These 561 blocks had the exact same average and minimum D_s-results as the best blocks found when testing all possible blockings into four blocks, thus they are a subset of the best possible blocks and will therefore not be further investigated.

Evaluation of method
When testing the method based on utilising the preferred division into two blocks, an advantage was found: Each division into two blocks was able to create at least

one division into four blocks which yielded $D_s > 0$, both when testing three and four active factors. This means that at most 1225 combinations had to be tested to ensure finding a usable block, but yielding no guarantee of getting a high D_s-efficiency. However, as 57% of all possible blocks were found to be usable in section 4.2.5, it should not take too long when using the mirror image pairs approach either.

The disadvantage of utilising the preferred divisions into two blocks was having to test more than twice the number of possible blockings based on mirror image pairs in order to try all combinations, as some of the blocks based on the preferred divisions into two blocks were duplicates. For larger design this method may nevertheless be useful, as it might be impossible to test all possible blockings based on mirror image pairs.

4.2.6 Dividing a 2_{IV}^{7-2}, a 2_{IV}^{8-3} and a 2_{IV}^{9-4} design into two and four blocks using HM

In sections 4.2.2, 4.2.4, 4.2.3 and 4.2.5, 32-run matrices with 16 and six factors respectively where divided into two and four blocks using mirror image pairs. One option when having more than six but less than 16 screening factors is to use a subset of columns from the 2_{IV}^{16-11} design. The results will then become the same as in section 4.2.2. But is it possible to do better? The recommended design generators for the 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} designs all include one four-factor interaction and one, two and three three-factor interactions, respectively. Thus the designs resulting from choosing columns from the 2_{IV}^{16-11} design are not the recommended ones. The problem with using the recommended designs is that including a main effect defined by a four-factor interaction destroys the mirror-image property.

In this section, the 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} design will therefore be divided into two and four blocks using blocks obtained by rearranging Hadamard matrices, as explained in section 3.3. Recall that Hadamard matrices are square matrices whose columns are all orthogonal. The matrix found to yield good results for two blocks in this section is the order 32 Hadamard matrix had.32.t1 from the web page *A*

Library of Hadamard Matrices [19], while had.32.t2 yielded good results for four blocks. The matrices, from now on called **M** and **M2**, can be found in table 6.15, 6.16, 6.17 and 6.18 in Appendix A.

The approach tested for the designs was to interchange the rows of the Hadamard matrices to generate the 2^5 design which was used to make the 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} designs by adding main effects based on the recommended design generators. The remaining columns were then used as candidate blocks. When dividing into two blocks, one column represented the block, while when dividing into four block, two columns represented the blocking. Then their interaction was also included in the design matrix. The reordered Hadamard columns were believed to be good candidate blocks as they were all orthogonal to the columns of the 2^5-design, which can be found in tables 4.24 and 4.25. When implementing the method, the identity column of the Hadamard matrix was removed, as it could neither be used for blocking nor factors columns.

As one column was removed, there were $\frac{31!}{(31-5)!} = 20389320$ ways to choose the columns that should make up the factor columns A, B, C, D and E in the 2^5 design. Not all combinations could be sorted into the design. Hence implementing the method, design columns were chosen at random until a combination which could be sorted into the design was found. Some of the Hadamard matrices tested were not arranged into the desired design even when running the script for a long time, but it was not possible to know if they could have been had all combinations been tested. After finding one for which the sorting could be performed, blocking using the remaining columns were tested. This procedure was repeated until a sorting which yielded the desired projectivity was found. Note that as all column combinations which could be used to make up the factor columns were not tested, it possibly exists combinations which yields better results than the ones presented here.

Table 4.24: First 16 rows of the 2^5 design.

Row	A	B	C	D	E
1	-1	-1	-1	-1	-1
2	1	-1	-1	-1	-1
3	-1	1	-1	-1	-1
4	1	1	-1	-1	-1
5	-1	-1	1	-1	-1
6	1	-1	1	-1	-1
7	-1	1	1	-1	-1
8	1	1	1	-1	-1
9	-1	-1	-1	1	-1
10	1	-1	-1	1	-1
11	-1	1	-1	1	-1
12	1	1	-1	1	-1
13	-1	-1	1	1	-1
14	1	-1	1	1	-1
15	-1	1	1	1	-1
16	1	1	1	1	-1

Table 4.25: Last 16 rows of the 2^5 design.

Row	A	B	C	D	E
17	-1	-1	-1	-1	1
18	1	-1	-1	-1	1
19	-1	1	-1	-1	1
20	1	1	-1	-1	1
21	-1	-1	1	-1	1
22	1	-1	1	-1	1
23	-1	1	1	-1	1
24	1	1	1	-1	1
25	-1	-1	-1	1	1
26	1	-1	-1	1	1
27	-1	1	-1	1	1
28	1	1	-1	1	1
29	-1	-1	1	1	1
30	1	-1	1	1	1
31	-1	1	1	1	1
32	1	1	1	1	1

4.2.6.1 A 2_{IV}^{7-2} design divided into two blocks

In the case of two blocks, the matrix **M** was sorted to make the columns 11, 3, 4, 25 and 7 correspond to columns A, B, C, D and E in the 2^5-design in table 4.24. This was done be rearranging the rows in the order 32, 28, 9, 13, 16, 12, 25, 29, 23, 19, 2, 6, 7, 3, 18, 22, 24, 20, 1, 5, 8, 4, 17, 21, 31, 27, 10, 14, 15, 11, 26 and 30. Then the factor F was defined as ABC, and G as ABDE. The resulting design can be seen in table 4.27. The recommended blocking by confounding of interactions for this design is to use ACD as block generator, resulting in a (32,7,2,2) screen. Thus being able to estimate all effects for three active factors would be an improvement.

Eight among the 26 columns of **M** not used to define main effects were not fully aliased with any effects up to three-factor interactions, and four of these yielded $D_s > 0$ for all combinations of three active factors, making the blocked design a (32,7,3,2) screen. The four preferred blocks correspond to columns 26, 28, 30 and 32 in **M**, and can be found in columns $b1, b2, b3$ and $b4$ in table 4.27. All four blocks yielded the same frequencies of D_s-efficiencies when testing all $\binom{7}{3} = 35$ combinations of three active factors, yielding a mean value of 0.982. The frequencies can be found in table 4.26. Using block $b1$ yields $D_s = 1$ when the active factors are among others B, D and E.

Table 4.26: D_s-efficiencies when estimating all interactions for three active factor for the 2_{IV}^{7-2} design when using the preferred division into two blocks.

Ds-efficiency	Occurences
1	23
0.965	8
0.917	4

Table 4.27: The 2^{7-2}_{IV} design and the preferred blocks.

A	B	C	D	E	F	G	$b1$	$b2$	$b3$	$b4$
-1	-1	-1	-1	-1	-1	1	-1	1	-1	1
1	-1	-1	-1	-1	1	-1	-1	1	-1	1
-1	1	-1	-1	-1	1	-1	1	1	-1	-1
1	1	-1	-1	-1	-1	1	1	1	-1	-1
-1	-1	1	-1	-1	1	1	1	-1	1	-1
1	-1	1	-1	-1	-1	-1	1	-1	1	-1
-1	1	1	-1	-1	-1	-1	-1	-1	1	1
1	1	1	-1	-1	1	1	-1	-1	1	1
-1	-1	-1	1	-1	-1	-1	1	-1	1	-1
1	-1	-1	1	-1	1	1	1	-1	1	-1
-1	1	-1	1	-1	1	1	-1	-1	1	1
1	1	-1	1	-1	-1	-1	-1	-1	1	1
-1	-1	1	1	-1	1	-1	-1	1	-1	1
1	-1	1	1	-1	-1	1	-1	1	-1	1
-1	1	1	1	-1	-1	1	1	1	-1	-1
1	1	1	1	-1	1	-1	1	1	-1	-1
-1	-1	-1	-1	1	-1	-1	1	-1	-1	1
1	-1	-1	-1	1	1	1	1	-1	-1	1
-1	1	-1	-1	1	1	1	1	1	1	1
1	1	-1	-1	1	-1	-1	1	1	1	1
-1	-1	1	-1	1	1	-1	-1	1	1	-1
1	-1	1	-1	1	-1	1	-1	1	1	-1
-1	1	1	-1	1	-1	1	-1	-1	-1	-1
1	1	1	-1	1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	-1	1	-1	1	1	-1
1	-1	-1	1	1	1	-1	-1	1	1	-1
-1	1	-1	1	1	1	-1	-1	-1	-1	-1
1	1	-1	1	1	-1	1	-1	-1	-1	-1
-1	-1	1	1	1	1	1	1	-1	-1	1
1	-1	1	1	1	-1	-1	1	-1	-1	1
-1	1	1	1	1	-1	-1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

4.2.6.2 A 2^{7-2}_{IV} design divided into four blocks

In the case of four blocks, the 2^{7-2}_{IV} design was made by rearranging the matrix **M2** in the order 32, 5, 13, 24, 30, 23, 15, 6, 12, 17, 25, 4, 10, 3, 27, 18, 16, 21, 29, 8, 14, 7, 31, 22, 28, 1, 9, 20, 26, 19, 11 and 2. The columns 4, 8, 9, 2 and 31 were used to define A, B, C, D and E. As before, F was defined as ABC, and G as ABDE. The recommended division into four blocks by confounding of interaction is to use ACD and BCD as block generators, making the blocked design have projectivity $P = 1$. Finding blocks which enable estimation of all effects for three active factors was therefore desirable in this case as well.

The 20 columns of **M2** not used to define main effects and not confounded with two- or three-factor interactions were used as candidate blocks, by pairing them and letting the rows (-1,-1), (1,-1), (-1,1) and (1,1) define the blocks. All $\binom{20}{2} = 190$ possible ways of defining four blocks were tested, and 40 of them yielded $D_s > 0$ for all $\binom{7}{3} = 35$ possible combinations of three active factors. Thus using any of these thus results in a (32,7,3,4) screen. The minimum D_s-efficiencies obtained for the blocks ranged from 0.841 to 0.917, and the average D_s-efficiencies ranged from 0.919 to 0.939. All blockings obtained a maximum of 1. Four of the blockings obtained both the highest minimum and the highest average. These were given by the column combinations (11,25), (11,26), (12,25) and (12,26). The columns can be found in table 4.29. The four preferred blockings obtained the same D_s-frequencies, which can be found in table 4.28. Using the combination (11,25) to define the blocks yields D_s=1 when the active factors are among others D, E and G.

Table 4.28: D_s-efficiencies when estimating all interactions for three active factors for the 2^{7-2}_{IV} design when using the preferred division into four blocks.

Ds-efficiency	Occurences
1	7
0.931	14
0.917	14

Table 4.29: The 2_{IV}^{7-2} design and the columns used to define the preferred divisions into four blocks.

A	B	C	D	E	F	G	11	12	25	26
-1	-1	-1	-1	-1	-1	1	-1	-1	1	1
1	-1	-1	-1	-1	1	-1	1	-1	1	-1
-1	1	-1	-1	-1	1	-1	1	-1	-1	1
1	1	-1	-1	-1	-1	1	1	1	-1	-1
-1	-1	1	-1	-1	1	1	1	1	1	1
1	-1	1	-1	-1	-1	-1	-1	1	1	-1
-1	1	1	-1	-1	-1	-1	-1	1	-1	1
1	1	1	-1	-1	1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	1	1	1
1	-1	-1	1	-1	1	1	-1	1	-1	1
-1	1	-1	1	-1	1	1	-1	1	1	-1
1	1	-1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	1	1
1	-1	1	1	-1	-1	1	1	-1	-1	1
-1	1	1	1	-1	-1	1	1	-1	1	-1
1	1	1	1	-1	1	-1	1	1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	1	1	1	1	-1	-1	1
-1	1	-1	-1	1	1	1	1	-1	1	-1
1	1	-1	-1	1	-1	-1	1	1	1	1
-1	-1	1	-1	1	1	-1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	1
-1	1	1	-1	1	-1	1	-1	1	1	-1
1	1	1	-1	1	1	-1	-1	-1	1	1
-1	-1	-1	1	1	-1	1	1	1	-1	-1
1	-1	-1	1	1	1	-1	-1	1	1	-1
-1	1	-1	1	1	1	-1	-1	1	-1	1
1	1	-1	1	1	-1	1	-1	-1	1	1
-1	-1	1	1	1	1	1	-1	-1	-1	-1
1	-1	1	1	1	-1	-1	1	-1	1	-1
-1	1	1	1	1	-1	-1	1	-1	-1	1
1	1	1	1	1	1	1	1	1	1	1

4.2.6.3 A 2^{8-3}_{IV} design divided into two blocks

The 2^{8-3}_{IV} design was obtained by rearranging **M** in the order 32, 8, 30, 6, 21, 13, 15, 23, 3, 27, 1, 25, 10, 18, 20, 12, 11, 19, 9, 17, 2, 26, 28, 4, 24, 16, 22, 14, 29, 5, 7 and 31. The columns 29, 7, 25, 14 and 26 were used to define factors A, B, C, D and E, respectively. F was defined by ABC, G by ABD and H by ACDE. The resulting design matrix can be found in table 4.31. The recommended blocking by confounding of interactions for this design is to use ABE as block generator, yielding a (32,8,2,2) screen. Thus being able to estimate all effects for three active factors would be an improvement here as well.

When testing the 26 columns of **M** not used in the design, eight were found not to be fully aliased with any effects up to three-factor interactions, and therefore chosen as candidate blocks. Four of these yielded $D_s > 0$ for all $\binom{8}{3} = 56$ combinations of three active factors when estimating all effects, thus resulting in a (32,8,3,2) screen. The blocks can be found in columns $b1, b2, b3$ and $b4$ alongside the design in table 4.31. They are based on columns 2, 4, 6 and 8 of **M**. All four blocks obtained the same frequencies of D_s-efficiencies when testing all possible combinations of three active factors, and thus the same average D_s-efficiency of 0.981. The frequencies are presented in table 4.30. Using block $b1$ yields $D_s = 1$ among others when E, D and F are the active factors.

Table 4.30: D_s-efficiencies when estimating all interactions for three active factors for the 2^{8-3}_{IV} design when using the preferred division into two blocks.

Ds-efficiency	Occurences
1	36
0.965	13
0.917	7

Table 4.31: The 2^{8-3}_{IV} design and the preferred blocks.

A	B	C	D	E	F	G	H	$b1$	$b2$	$b3$	$b4$
-1	-1	-1	-1	-1	-1	-1	1	1	-1	1	-1
1	-1	-1	-1	-1	1	1	-1	1	1	-1	-1
-1	1	-1	-1	-1	1	1	1	1	1	1	1
1	1	-1	-1	-1	-1	-1	-1	-1	1	1	-1
-1	-1	1	-1	-1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	-1	1	1	-1	1	-1	1
-1	1	1	-1	-1	-1	1	-1	1	-1	-1	1
1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	-1	1	-1	1	-1	1	-1
1	-1	-1	1	-1	1	-1	1	1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	1	1
1	1	-1	1	-1	-1	1	1	-1	1	1	-1
-1	-1	1	1	-1	1	1	1	-1	-1	1	1
1	-1	1	1	-1	-1	-1	-1	-1	1	-1	1
-1	1	1	1	-1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	1
1	-1	-1	-1	1	1	1	1	-1	-1	1	1
-1	1	-1	-1	1	1	1	-1	-1	-1	-1	-1
1	1	-1	-1	1	-1	-1	1	1	-1	-1	1
-1	-1	1	-1	1	1	-1	1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
-1	1	1	-1	1	-1	1	1	-1	1	1	-1
1	1	1	-1	1	1	-1	-1	1	1	1	1
-1	-1	-1	1	1	-1	1	1	-1	1	-1	1
1	-1	-1	1	1	1	-1	-1	-1	-1	1	1
-1	1	-1	1	1	1	-1	1	-1	-1	-1	-1
1	1	-1	1	1	-1	1	-1	1	-1	-1	1
-1	-1	1	1	1	1	1	-1	1	1	-1	-1
1	-1	1	1	1	-1	-1	1	1	-1	1	-1
-1	1	1	1	1	-1	-1	-1	-1	1	1	-1
1	1	1	1	1	1	1	1	1	1	1	1

4.2.6.4 A 2_{IV}^{8-3} design divided into four blocks

Considering four blocks instead, the 2_{IV}^{8-3} design was made by rearranging **M2** in the order 32, 4, 5, 25, 30, 18, 3, 15, 12, 24, 17, 13, 10, 6, 23, 27, 28, 8, 1, 29, 26, 22, 7, 11, 16, 20, 21, 9, 14, 2, 19 and 31. The columns 3, 7, 29, 4 and 15 were used to define factors A, B, C, D and E, respectively. As before, F was defined by ABC, G by ABD and H by ACDE. The resulting design matrix is the same as shown in table 4.31. The recommended block generators for division into four blocks are AC and AD. They make $P = 2$ and $P = 3$ impossible, thus being able to estimate all effects for three active factors was the goal also in this section.

The 20 columns not confounded with main or interaction effects were used to define the candidate blockings, just as in section 4.2.6.2. Among the resulting 190 possible ways to define four blocks, 32 yielded $D_s > 0$ for all $\binom{8}{3} = 56$ combinations of three active factors. Using these blockings thereby yield a (32,8,3,4) screen. The minimum D_s-efficiencies were in the range from 0.841 to 0.853, and the average D_s-efficiencies in the range from 0.915 to 0.929. All maximum D_s-efficiencies were 1. Among the candidate blocks, 16 obtained both the highest minimum and the highest average D_s-efficiency. They were given by the column combinations (9,19), (9,23), (10,19), (10,23), (11,31), (11,32), (12,31), (12,32), (13,25), (13,26), (14,25), (14,26), (18,25), (18,26), (22,25) and (22,26). The columns are shown in table 4.33. All combinations resulted in the same frequencies of D_s-efficiencies, which can be found in table 4.32. Using the columns 9 and 19 to define the blocks yields D_s=1 among others when the active factors are A, B and D.

Table 4.32: D_s-efficiencies when estimating all interactions for three active factors for the 2_{IV}^{8-3} design when using the preferred division into four blocks.

Ds-efficiency	Occurences
1	10
0.931	16
0.917	24
0.853	6

Table 4.33: The columns used to define the preferred divisions into four blocks for the 2^{8-3}_{IV} design shown in table 4.31.

9	10	11	12	13	14	18	19	22	23	25	26	31	32
1	1	-1	-1	1	1	-1	1	-1	1	-1	-1	1	1
-1	1	1	-1	1	-1	1	-1	-1	1	1	-1	1	-1
1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1
1	1	-1	-1	1	1	1	-1	1	-1	1	1	-1	-1
-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1
1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
1	1	1	1	-1	-1	1	-1	1	-1	-1	-1	1	1
-1	-1	1	1	1	1	1	1	1	1	1	1	1	1
-1	1	-1	1	-1	1	1	1	-1	-1	-1	1	-1	1
1	-1	-1	1	1	-1	1	1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	1	-1	1	-1
1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
-1	-1	1	1	-1	-1	1	-1	1	-1	-1	-1	1	1
1	-1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1
-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1
-1	-1	-1	-1	1	1	1	-1	1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1
1	-1	-1	1	-1	1	1	-1	-1	1	-1	1	-1	1
-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
-1	-1	-1	-1	1	1	-1	1	-1	1	-1	-1	1	1
1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
1	-1	1	-1	1	-1	-1	-1	1	1	-1	1	-1	1
-1	1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1
1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
1	-1	1	-1	1	-1	1	1	-1	-1	1	-1	1	-1
-1	1	1	-1	-1	1	1	1	-1	-1	1	-1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

4.2.6.5 A 2_{IV}^{9-4} design divided into two blocks

The 2_{IV}^{9-4} design was generated by arranging the rows of **M** in the order 32, 22, 28, 18, 20, 26, 24, 30, 16, 6, 12, 2, 4, 10, 8, 14, 9, 3, 13, 7, 5, 15, 1, 11, 25, 19, 29, 23, 21, 31, 17 and 27, and letting columns 17, 18, 23, 6 and 25 correspond to the design columns A, B, C, D and E. The factors F, G, H and I were defined by the design generators ABC, ABD, ACD and BCDE. The recommended blocking by confounding of interactions for this design is to use the block generator AB, which results in a (32,9,1,2) screen. Thus being able to estimate all effects for all combinations of two or three active factors would be an improvement.

Among the 26 remaining columns of **M**, eight were not fully aliased with any effects up to three-factor interactions. These were number 10, 12, 14, 16, 26, 28, 30 and 32. They can be found in columns $b1, b2, b3$, $b4$, $b5$, $b6$, $b7$ and $b8$ in table 4.35. All eight blocks yielded $D_s > 0$ for all $\binom{9}{3} = 84$ possible combinations of three active factors, making the blocked design a (32,9,3,2) screen. The average D_s-efficiency for all blocks was 0.982, and they all obtained the same frequencies of D_s-values, which can be found in table 4.34. The block $b1$ yields $D_s = 1$ when used for among others the active factors D, E and G.

Table 4.34: D_s-efficiencies when estimating all interactions for three active factors for the 2_{IV}^{9-4} design when using the preferred block.

Ds-efficiency	Occurences
1	52
0.965	24
0.917	8

Table 4.35: The 2^{9-4}_{IV} design and the preferred blocks.

A	B	C	D	E	F	G	H	I	$b1$	$b2$	$b3$	$b4$	$b5$	$b6$	$b7$	$b8$
-1	-1	-1	-1	-1	-1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
1	-1	-1	-1	-1	1	1	1	1	1	-1	1	-1	1	-1	1	-1
-1	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1
1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	1
-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	-1	1	-1	-1	1	1	1	1	1	1	1	1
-1	1	1	-1	-1	-1	1	1	1	-1	1	-1	1	1	-1	1	-1
1	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
-1	-1	-1	1	-1	-1	1	1	-1	1	1	1	1	-1	-1	-1	-1
1	-1	-1	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1
-1	1	-1	1	-1	1	-1	1	1	1	-1	-1	1	-1	1	1	-1
1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
-1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1	1	-1	1
1	-1	1	1	-1	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1
-1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1
1	1	1	1	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1
1	-1	-1	-1	1	1	1	1	-1	-1	1	-1	1	-1	1	-1	1
-1	1	-1	-1	1	1	1	-1	1	1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1
-1	-1	1	-1	1	1	-1	1	1	-1	-1	1	1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	1	-1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	1
1	1	1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1
-1	-1	-1	1	1	-1	1	1	1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	1	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	1
-1	1	-1	1	1	1	-1	1	-1	-1	1	1	-1	1	-1	-1	1
1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1
-1	-1	1	1	1	1	1	-1	-1	-1	1	-1	1	1	-1	1	-1
1	-1	1	1	1	-1	-1	1	-1	1	-1	-1	1	1	-1	-1	1
-1	1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	-1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4.2.6.6 A 2_{IV}^{9-4} design divided into four blocks

When considering four blocks, the 2_{IV}^{9-4} design was made by rearranging the rows of **M2** in the order 32, 17, 1, 16, 22, 21, 11, 12, 18, 31, 15, 2, 28, 27, 5, 6, 26, 23, 7, 10, 20, 19, 13, 14, 24, 25, 9, 8, 30, 29, 3, and 4. The columns 4, 32, 31, 16 and 3 were used to define factors A, B, C, D and E. As in section 4.2.6.5, the factors F, G, H and I were defined by the design generators ABC, ABD, ACD and BCDE. The recommended way to divide the design into four blocks using confounding of interactions is to use the design generators AB and AC. The resulting blocked design has $P = 1$. Testing whether all effects could be estimated for three active factors when using other blocks was therefore interesting.

The 12 columns that were not confounded with any main or two- or three-factor interaction effects were used to define the candidate blockings. Testing all $\binom{12}{2} = 66$ possible candidate blockings, 12 of them were found to yield $D_s > 0$ for all $\binom{9}{3} = 84$ combinations of three active factors. Using any of these 12 blockings thus results in a (32,9,3,4) screen. The minimum D_s-efficiencies were in the range from 0.841 to 0.853, while the average D_s-efficiencies were in the range from 0.916 to 0.925. All the maximum D_s-efficiencies were 1. The four blockings yielding both the highest minimum value and the highest maximum value were found using the column combinations (11,25), (11,26), (12,25) and (12,26). The columns can be found along with the design in table 4.37. Using any of these combinations to block the design yielded the same D_s-frequencies, which can be found in table 4.36. Using the combination (11,25) yields D_s=1 among others when D, E and I are the active factors.

Table 4.36: D_s-efficiencies when estimating all interactions for three active factors for the 2_{IV}^{9-4} design when using the preferred divisions into four blocks.

Ds-efficiency	Occurences
1	8
0.931	40
0.917	28
0.853	8

Table 4.37: The 2_{IV}^{9-4} design and the columns used to define the preferred divisions into four blocks.

A	B	C	D	E	F	G	H	I	11	12	25	26
-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	1	1
1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
-1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1
1	1	-1	-1	-1	-1	-1	1	-1	1	-1	1	-1
-1	-1	1	-1	-1	1	-1	1	-1	-1	1	-1	1
1	-1	1	-1	-1	-1	1	-1	-1	1	-1	-1	1
-1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	-1
1	1	1	-1	-1	1	-1	-1	1	1	1	1	1
-1	-1	-1	1	-1	-1	1	1	-1	1	1	1	1
1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	1	-1	1	-1	1	1	1	-1	1	-1
1	1	-1	1	-1	-1	1	-1	1	-1	1	1	-1
-1	-1	1	1	-1	1	1	-1	1	1	-1	-1	1
1	-1	1	1	-1	-1	-1	1	1	-1	1	-1	1
-1	1	1	1	-1	-1	-1	-1	-1	1	1	-1	-1
1	1	1	1	-1	1	1	1	-1	-1	-1	1	1
-1	-1	-1	-1	1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1
-1	1	-1	-1	1	1	1	-1	1	-1	1	-1	1
1	1	-1	-1	1	-1	-1	1	1	1	-1	-1	1
-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1
1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
-1	1	1	-1	1	-1	1	1	-1	1	1	-1	-1
1	1	1	-1	1	1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	1	1	-1	1	1	1	-1	-1	1	1
1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1
-1	1	-1	1	1	1	-1	1	-1	1	-1	-1	1
1	1	-1	1	1	-1	1	-1	-1	-1	1	-1	1
-1	-1	1	1	1	1	1	-1	-1	1	-1	1	-1
1	-1	1	1	1	-1	-1	1	-1	-1	1	1	-1
-1	1	1	1	1	-1	-1	-1	1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1

4.3 64-run designs

The 64-run designs which were blocked were a 2_{IV}^{32-26} design and a 2_{V}^{8-2} design. The 2_{IV}^{32-26} design was chosen as it is the doubling of the 2_{IV}^{16-11} design. The results for dividing the design into two blocks are presented in section 4.3.1, while the results for division into four blocks are presented in section 4.3.2. In both cases, three active factors were considered.

The 2_{V}^{8-2} design was chosen as it has projectivity $P = 4$ and can be generated using the columns of the doubling of the Hadamard matrices which yielded good results for the 32-run 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} designs. Division into two, four and eight blocks was tested, each for both three and four active factors. The results for division into two blocks can be found in sections 4.3.3.1 and 4.3.3.2. The results for division into four blocks are presented in sections 4.3.3.3 and 4.3.3.4, and the results for division into eight blocks can be found in sections 4.3.3.5 and 4.3.3.6.

4.3.1 Dividing a 2_{IV}^{32-26} design into two blocks using the blocking of the 2_{IV}^{16-11} design

Using the idea introduced in section 3.2.2, a 64-run design was created by doubling the 32-run 2_{IV}^{16-11} design found in table 4.10 in section 4.2.2. This resulted in a 2_{IV}^{32-26} design with projectivity $P = 3$ and resolution IV. No recommended blocking by confounding of interactions was found in the literature for this design, so the goal was to obtain a blocking with projectivity $P = 3$.

The blocking was done in the same manner as in section 4.2.1, testing all 5040 blocks that worked for the 2_{IV}^{16-11} design. The blocks generated using patterns D_2 and D_3 yielded a D_s-efficiency above 0 when estimating all effects for all $\binom{32}{3}$ =4960 possible combinations of three active factors, making the resulting blocked design a $(64, 32, 3, 2)$ screen. The first pattern, D_2, yielded an average D_s-efficiency of 0.98651. For the second, D_3, the average was slightly higher, at 0.98653. Both patterns yielded a minimum D_s-efficiency of 0.917 and a maximum D_s-efficiency of 1 when testing all blocks for all combinations of three active factors. Having the highest average, D_3 was chosen for further consideration. All

blocks found using that pattern yielded the same frequencies of D_s-efficiencies, which can be found in table 4.38.

An example of blocking can be found by creating factors based on **B1** and **B2** in table 6.3 and 6.4 in Appendix A, using pattern D3, and letting the first 32 rows belong to one block, and the last 32 to the other. **B1** and **B2** were found using the example block from section 4.2.2, $[1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,$ $-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,1,1,1]^T$. **B1** was chosen as the rows of the matrix in table 4.10 for which the corresponding block entry was 1, and **B2** as the rows for which the corresponding block entry was -1. The resulting design matrix can be found in table 6.5, 6.6, 6.7 and 6.8 in Appendix A. The preferred block yields D_s=1 when the three first factors, A, B and C, are active.

Table 4.38: Frequencies of the different D_s-values when testing the preferred block for the doubling of the 2_{IV}^{16-11} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurences
1	1600
0.992	1536
0.983	896
0.965	384
0.954	512
0.917	32

4.3.2 Dividing a 2_{IV}^{32-26} design into four blocks using the blocking of the 2_{IV}^{16-11} design

Similarly as the division of the 2_{IV}^{32-26} design into two blocks based on the preferable blocking of the 2_{IV}^{16-11} design into two blocks, the 2_{IV}^{32-26} design was divided into four blocks using the preferable division of the 2_{IV}^{16-11} design into four blocks. The idea is described in section 3.2.2, and will be repeated shortly here for easier

referencing to the best block. As before, **B1**, **B2**, **B3** and **B4** denote the submatrices of the 32-run design matrix belonging to each of the four blocks. A 64-run matrix may be constructed as any matrix including each row of

$$\mathbf{D}= \begin{pmatrix} a & \mathbf{B1} & \mathbf{B1} \\ b & \mathbf{B2} & \mathbf{B2} \\ c & \mathbf{B3} & \mathbf{B3} \\ d & \mathbf{B4} & \mathbf{B4} \\ e & \mathbf{B1} & -\mathbf{B1} \\ f & \mathbf{B2} & -\mathbf{B2} \\ g & \mathbf{B3} & -\mathbf{B3} \\ h & \mathbf{B4} & -\mathbf{B4} \end{pmatrix}$$

The letters in the first column are the names that will be used for the submatrices (**B1 B1**), (**B2 B2**) and so on. The rows in submatrix a must always be in the same block, likewise with the rows in submatrix b, and so on. The blocks are defined by reordering the submatrices a,b,c,d,e,f,g and h, and letting the first 16 rows belong to one block, the next 16 rows belong to the second block and so on. As in section 4.2.3, the blocks are represented by two columns, where the combinations (1,1), (1,-1), (-1,1) and (-1,-1) denotes the four different blocks.

As explained in section 3.2.2, there are $\frac{\binom{8}{2}\binom{6}{2}\binom{4}{2}}{4!} = 105$ ways to divide the eight submatrices of **D** into four blocks. For dividing the 2_{IV}^{16-11} design into four blocks, there were the 40320 equally good blocks, as they all had the highest average among the 50400 blocks having the highest minimum D_s-efficiency for all combinations of three active factors. This yields over 4 million potential candidate blocks.

When testing division into two blocks, all the best blocks for the 16-run design gave the same results when used to define two blocks for the 32-run design. The different combinations of the rows of the 32-run matrix, D_2 and D_3, did however yield different results. Thus it seemed reasonable to test all 105 possible ways to arrange **D** into four blocks for a subset of all the 40320 candidate blocks. 10 of

the preferred blockings for division into four blocks in section 4.2.3 was chosen for testing, trying to estimate all effects for all $\binom{32}{3}$ =4960 possible combinations of three active factors. 96 among the 105 blocking arrangements yielded $D_s > 0$ for all combinations, making the blocked design a $(64,32,3,4)$ screen when using any of them. The nine blocking arrangements not yielding $D_s > 0$ for all combinations were the combinations in which the submatrices *abcd* were placed in two blocks, and $efgh$ in the two other blocks. Then two-factor interactions became confounded with the block columns, just as when using pattern D_1 for division into two blocks, as explained in section 3.2.1.

The minimum D_s-efficiencies for the 96 ways of blocking yielding $D_s > 0$ for all combinations of three active factors ranged from 0.834 to 0.913, and the averages ranged from 0.958 to 0.960. All obtained a maximum D_s-efficiency of 1. The four blocking arrangements that obtained the highest minimum D_s-efficiency also obtained the highest average D_s-efficiency, and were therefore preferred. These were *adhcgfbe*, *adhegbcf*, *afhegdbc* and *ahgfedbc*, with the first two submatrices in the first block, the next two submatrices in the second block, and so on. These results were equal for all 10 divisions into four blocks from section 4.2.3 that were used to define the new blocks. The preferred blocking from section 4.2.3 was chosen for further investigation, using it to define **B1**, **B2**, **B3** and **B4**, as shown in table 4.41

An example of occurrences of D_s-frequencies is shown in table 4.39. The result is obtained using combination *adhcgfbe* to define the new blocks. Block one with *a* and *b* is defined by (1,1), block two with *h* and *c* by (1,-1), block three with *g* and *f* by (-1,1) and block four with *b* and *e* by (-1,-1). The matrix in table 4.40 illustrates the pattern, and shows how the first $\frac{1}{4}$ of the design matrix belongs to the block defined by (-1,-1) and so on. Note that the interaction effect between the block columns is also included when calculating the D_s-efficiency, despite not being included below. Tables 6.9, 6.10, 6.11 and 6.12 in Appendix A show the design matrix written out. Note that D_s=1 when A, B and C are the active factors.

Table 4.39: Frequencies of the different D_s-values when testing the preferred division into four blocks for the doubling of the 2_{IV}^{16-11} design for all combinations of three active factors, estimating all interactions.

D_s-efficiency	Occurences
1	256
0.992	128
0.984	256
0.983	256
0.976	64
0.975	192
0.975	192
0.974	128
0.967	128
0.967	128
0.967	256
0.965	448
0.959	64
0.958	384
0.957	64
0.956	128
0.954	64
0.949	512
0.947	128
0.946	256
0.940	128
0.938	192
0.930	128
0.928	128
0.927	128
0.917	32
0.913	192

Table 4.40: The pattern used to define the preferred division into four blocks.

$$\begin{pmatrix} a & \mathbf{B1} & \mathbf{B1} & -1 & -1 \\ d & \mathbf{B4} & \mathbf{B4} & -1 & -1 \\ h & \mathbf{B4} & -\mathbf{B4} & 1 & -1 \\ c & \mathbf{B3} & \mathbf{B3} & 1 & -1 \\ g & \mathbf{B3} & -\mathbf{B3} & -1 & 1 \\ f & \mathbf{B2} & -\mathbf{B2} & -1 & 1 \\ b & \mathbf{B2} & \mathbf{B2} & 1 & 1 \\ e & \mathbf{B1} & -\mathbf{B1} & 1 & 1 \end{pmatrix}$$

Table 4.41: One of the preferred ways to define **B1**, **B2**, **B3** and **B4**.

$b1$	$b2$	$b1b2$	Part
-1	-1	1	**B4**
-1	-1	1	**B4**
-1	-1	1	**B4**
-1	1	-1	**B3**
-1	-1	1	**B4**
-1	1	-1	**B3**
1	-1	-1	**B2**
1	1	1	**B1**
1	-1	-1	**B2**
1	1	1	**B1**
1	1	1	**B1**
1	1	1	**B1**
1	-1	-1	**B2**
1	-1	-1	**B2**
-1	1	-1	**B3**
-1	1	-1	**B3**
-1	1	-1	**B3**
-1	1	-1	**B3**
1	-1	-1	**B2**
1	-1	-1	**B2**
1	1	1	**B1**
1	1	1	**B1**
1	1	1	**B1**
1	-1	-1	**B2**
1	1	1	**B1**
1	-1	-1	**B2**
-1	1	-1	**B3**
-1	-1	1	**B4**
-1	1	-1	**B3**
-1	-1	1	**B4**
-1	-1	1	**B4**
-1	-1	1	**B4**

4.3.3 Dividing a 2_V^{8-2} design into two, four and eight blocks using HM

In this section, division of a 2_V^{8-2} design into two, four and eight blocks is tested. The design can be found in tables 6.21 and 6.22. It has projectivity $P = 4$ prior to blocking. Division into two blocks by the recommended confounding of interactions using ACE as block generator yields a (64,8,2,2) screen, just as division into four blocks using the recommended generators ACE and BDE. Division into eight blocks using the recommended block generators ADE, BDE and ACDEF yields a (64,8,1,8) screen. Is it possible to do better in terms of projectivity? To enable comparison with the 64-run 2_{IV}^{32-26} design in section 4.3.1, the divisions into two and four blocks will be tested for all combinations of both three and four active factors. As the 32-run designs could be divided into four blocks with eight rows each, testing whether the 64-run 2_V^{8-2} design could be divided into eight blocks with eight rows each was also interesting.

Since testing all possible ways to block the design was not feasible, finding reasonably good candidate blocks was crucial. As rearranging Hadamard matrices worked well for the 32-run designs in section 4.2.6, the doubling of these Hadamard matrices were utilised for the 2_V^{8-2} design. For division into two blocks, a 64×64 matrix **R** was created by doubling the 32-run matrix **M** from section 4.2.6 as described in section 2.1.4. For division into four blocks, a 64×64 matrix **R2** was created by doubling the 32-run matrix **M2**. For division into eight blocks, the doubling of the Hadamard matrix had.32.t3 from [19] was found to yield good results. The matrix matrix had.32.t3 will from now on be referred to as **M3**, and can be found in table 6.19 and 6.20 in Appendix A. Its doubling will be denoted **R3**. As **M**, **M2** and **M3** are Hadamard matrices, their doublings **R**, **R2** and **R3** are also Hadamard matrices, and thus consist of orthogonal columns. For both **R**, **R2** and **R3**, the column with ones was removed, as it could neither be reordered to represent a factor nor a block.

To generate the 2_V^{8-2} design, the rows of the Hadamard matrices were first interchanged to generate the standard 2^6 design with six factors A, B, C, D, E and F. The 2_V^{8-2} design with projectivity $P = 3$ and resolution V was obtained by setting G=ABCD and H=ABEF, as suggested by Wu and Hamada [7]. As the columns

which were not used in the design are orthogonal to the design columns, they are interesting candidate blocks. Using any of these columns to define blocks ensures that all main effects are not even partially confounded with them.

4.3.3.1 Two blocks, three active factors

Using the columns 10, 45, 37, 5, 53 and 32 of **R** to find the design columns A, B, C, D, E and F respectively, the rows of the matrix has to be in the order 64, 52, 59, 55, 61, 49, 62, 50, 26, 22, 29, 17, 27, 23, 28, 24, 37, 41, 34, 46, 40, 44, 39, 43, 3, 15, 8, 12, 2, 14, 1, 13, 11, 7, 16, 4, 10, 6, 9, 5, 45, 33, 42, 38, 48, 36, 47, 35, 18, 30, 21, 25, 19, 31, 20, 32, 56, 60, 51, 63, 53, 57, 54 and 58. Testing the remaining 57 columns as blocks, 32 of them yielded a D_s-efficiency above 0 when testing all effects for all $\binom{8}{3}$ =56 possible combinations of three active factors. The resulting design is a $(64, 8, 3, 2)$ screen.

Eight of the blocks obtained a minimum D_s-efficiency of 0.917, while the remaining 24 obtained a minimum of 0.965. The average value was in the range from 0.984 to 0.992 for all 32 blocks. The 12 blocks obtaining the highest minimum D_s-efficiency, highest average D_s-efficiency and a maximum D_s-efficiency of 1 were regarded as the best. These correspond to columns 9, 10, 11, 12, 13, 14, 15, 16, 50, 52, 54 and 56 in **R**. They all obtained the same frequencies of D_s-efficiencies when testing all combinations of three active factors, as shown in table 4.42. The blocks can be found along with the design matrix in columns $b1$, $b2$, $b3$, $b4$, $b5$, $b6$, $b7$, $b8$, $b9$, $b10$, $b11$ and $b12$ in the tables 6.21 and 6.22 in Appendix A. The block $b1$ yields D_s=1 among other when D, E and F are the active factors.

Table 4.42: D_s-efficiencies when estimating all interactions for three active factors for the 2_V^{8-2} design when using the preferred block.

Ds-efficiency	Occurences
1	44
0.965	12

4.3.3.2 Two blocks, four active factors

Using the same columns of **R** as in the above section to define the design columns, the remaining 57 columns were now tested as blocks for all $\binom{8}{4}$ =70 possible combinations of four active factors. 24 of them yielded $D_s > 0$ when estimating all effects for all combinations, thereby making the resulting design a $(64, 8, 4, 2)$ screen. They all had a maximum D_s-efficiency of 1 and a minimum of 0.958. 12 of them obtained the highest average D_s-efficiency of 0.986. These corresponded to the columns 9, 10, 11, 12, 13, 14, 15, 16, 50, 52, 54 and 56 of **R**, just as the best blocks in the case of three active factors. Using the block $b1$ yields D_s=1 among other when B, D, E and F are the active factors. When testing all combinations, the four preferred blocks obtained the same frequencies of D_s-efficiencies. The frequency table can be found in table 4.43.

Table 4.43: D_s-efficiencies when estimating all interactions for four active factors for the 2_V^{8-2} design when using the preferred block.

Ds-efficiency	Occurences
1	33
0.982	25
0.958	12

4.3.3.3 Four blocks, three active factors

As the matrix **M2** yielded good results for dividing 32-run designs in four blocks, the doubling **R2** was used to divide the 2_V^{8-2} design into four blocks. To generate the design, the rows of **R2** were sorted in the order 64, 52, 22, 26, 21, 59, 29, 51, 14, 2, 40, 44, 39, 9, 47, 1, 53, 57, 31, 19, 32, 50, 24, 58, 7, 11, 45, 33, 46, 4, 38, 12, 60, 56, 18, 30, 17, 63, 25, 55, 10, 6, 36, 48, 35, 13, 43, 5, 49, 61, 27, 23, 28, 54, 20, 62, 3, 15, 41, 37, 42, 8, 34 and 16. The columns 21, 28, 34, 24, 9 and 12 were used to define the factors A, B, C, D, E and F. The remaining 57 columns were used to find candidate blocks, by testing all $\binom{57}{2} = 1596$ combination of two columns, and using the combinations (-1,-1), (-1,1), (1,-1) and (1,1)

to define the four blocks. The interaction effect was also included in the design matrix. Among the 1596 candidate blockings, 432 yielded $D_s > 0$ for all $\binom{8}{3} = 56$ possible combinations of three active factors. Using any of these thus results in a (64,8,3,4) screen. The minimum D_s-efficiencies ranged from 0.834 to 0.931, while the average D_s-efficiencies ranged from 0.940 to 0.982. All blockings obtained a maximum D_s-efficiency of 1. 20 of the blockings obtained both the highest minimum value and the highest average. These were given by the column combinations (13,26), (13,40), (13,59), (13,60), (14,36), (14,40), (14,59), (14,60), (15,33), (15,37), (15,47), (15,48), (16,33), (16,37), (16,47), (16,48), (33,47), (33,48), (37,47), and (37,48). The columns can be found in tables 6.23 and 6.24 in Appendix A, and the corresponding design can be found in table 6.21 and 6.22 in Appendix A. All the combinations obtained the same frequencies of D_s-efficiencies when tested for all combinations. They can be found in table 4.44. When using columns 13 and 26 to define the blocks, D_s=1 among others when D, E and F are the active factors.

Table 4.44: D_s-efficiencies when estimating all interactions for three active factors for the 2_V^{8-2} design when using the preferred division into four blocks.

Ds-efficiency	Occurences
1	30
0.965	21
0.931	5

4.3.3.4 Four blocks, four active factors

The 1596 candidate blockings were tested also for the case of four active factors. 192 of them yielded $D_s > 0$ for all $\binom{8}{4} = 70$ possible combinations of four active factors. These were a subset of the 432 blocks which yielded good results for three active factors. The resulting design when using one of the blocking is a (64,8,4,4) screen. The minimum D_s-efficiencies were in the range from 0.862 to 0.917, while the average D_s-efficiencies were in the range from 0.947 to 0.966. The maximum was 1 for all blockings.

20 of the blockings obtained both the highest minimum and the highest average. These were given by the same column combinations as in section 4.3.3.3. These 20 preferred blockings yielded the same frequencies of D_s-efficiencies when tested for all combinations of four active factors. They can be found in table 4.45. When using columns 13 and 26 to define the blocks, D_s=1 was obtained among others when C, E, F and H were the active factors.

Table 4.45: D_s-efficiencies when estimating all interactions for four active factors for the 2_V^{8-2} design when using the preferred division into four blocks.

Ds-efficiency	Occurences
1	12
0.965	39
0.958	13
0.924	4
0.917	2

4.3.3.5 Eight blocks, three active factors

In the case of eight blocks, the matrices **R** and **R2** was not sorted into a combination yielding good blocks when running the script for quite a time. It might very well be possible that they could have yielded the desired results if the script had been run even longer, but as the doubling of the Hadamard matrix **M3**, **R3**, yielded the desired result immediately, it was chosen for further consideration.

The design was generated by sorting the rows of **R3** in the order 64, 45, 60, 41, 11, 26, 15, 30, 63, 44, 12, 31, 25, 14, 46, 57, 47, 62, 43, 58, 28, 9, 32, 13, 48, 59, 27, 16, 10, 29, 61, 42, 49, 36, 53, 40, 6, 23, 2, 19, 50, 37, 5, 18, 24, 3, 35, 56, 34, 51, 38, 55, 21, 8, 17, 4, 33, 54, 22, 1, 7, 20, 52 and 39. The columns 62, 34, 39, 33, 22 and 5 were used to define the main factors A, B, C, D, E and F.

The candidate blockings were found as all $\binom{57}{3} = 29260$ possible combinations of the 57 columns not used to define the main effects. The row combinations (-1,-1,-1), (-1,-1,1), (-1,1,-1), (1,-1,-1), (1,1,-1), (1,-1,1), (-1,1,1) and (1,1,1) defined the eight different blocks. The two- and three-factor interaction effects between them were also included in the design matrix. Thus there were seven columns representing the blocking. Testing all 29260 candidate blockings, 4752 of them obtained $D_s > 0$ for all the $\binom{8}{3}$=56 possible combinations of three active factors. Using any of these blockings thereby results in a (64,8,3,8) screen. The minimum D_s-efficiencies for the 4752 blockings ranged from 0.581 to 0.853, while the averages ranged from 0.847 to 0.921 and the maximums from 0.917 to 1.

The blockings with the highest minimum D_s-efficiency did not obtain the highest average, but the eight of them obtained and average of 0.918 and a maximum of 1, and were therefore preferred. The blockings were defined by the column combinations (18,20,51), (18,20,59), (18,28,51), (18,28,59), (20,26,51), (20,26,59), (26,28,51) and (26,28,59). The columns can be found in tables 6.25 and 6.26 in Appendix A. They all obtained the same D_s-frequencies when tested for all combinations of three active factors. The frequencies are shown in table 4.46. When the columns 18, 20 and 51 are used to define the blocking, D_s=1 when the active factors are A, D and E.

Table 4.46: D_s-efficiencies when estimating all interactions for three active factors for the 2^{8-2}_V design when using the preferred division into eight blocks.

Ds-efficiency	Occurences
1	1
0.931	37
0.917	10
0.866	2
0.853	6

4.3.3.6 Eight blocks, four active factors

The same 29260 blockings that were tested for three active factors in section 4.3.3.5 were also tested for all $\binom{8}{4} = 70$ possible combinations of four active factors. 352 of the blockings yielded $D_s > 0$ for all combinations, making the resulting design a (64,8,4,8)-screen. The minimum D_s-efficiencies ranged from 0.695 to 0.808, and the averages ranged from 0.872 to 0.889. All the maximums were 0.917. Among the blockings with the highest minimum D_s-efficiency, none obtained the highest average.

The best average obtained by the blockings with the highest minimum D_s-efficiency was 0.889, which was obtained by the same eight blocking that were preferred in section 4.3.3.5. Thus those were therefore preferred here as well. They all yielded the same frequencies of D_s-efficiences when tested for all possible combinations of four active factors. The frequencies can be found in table 4.47. When the columns 18, 20 and 51 defines the blocking, D_s=0.917 among others when the active factors are A, D E and F.

Table 4.47: D_s-efficiencies when estimating all interactions for four active factors for the 2_V^{8-2} design when using the preferred division into eight blocks.

Ds-efficiency	Occurences
0.917	7
0.891	30
0.885	20
0.881	7
0.878	4
0.862	1
0.808	1

4.4 Summary of results

The goal of testing different blocking methods was to find blocks which yielded higher projectivity than traditional blocking using confounding of interactions without compromising the D_s-efficiency substantially. To evaluate the results, a summary for all designs is presented in table 4.48 and 4.49. Table 4.48 shows the minimum, maximum and average D_s-efficiency obtained when estimating as many effects as indicated by the "Screen" column. Note that the notation $(n,k,P_\alpha*,b)$ indicates that the D_s-efficiencies were found when estimating up to α interactions for P factors, but the design can be used to screen for more factors ($(n,k,P_{\alpha+a},b)$ is not considered more than (n,k,P_α,b) in this case). This is typically relevant when fewer active factors than possible were tested for comparison with other designs.

An average D_s-efficiency above 0.9 was obtained for all blockings except three. An average below 0.9 was obtained for the 2^{6-1}_{VI} design divided into four blocks when estimating all effects for four factors, and up to three-factor interactions for five active factors, and for the 2^{8-2}_{VI} design divided into eight blocks when estimating all effects for four factors. The minimum D_s-efficiencies were above 0.8 in all cases, and above 0.9 in all except nine cases, eight of which where for division into four or more blocks. Hence all the suggested blockings are quite efficient. It is however important to remember that the motivation for increasing the projectivity is to be able to find active effects. The ability to do so does not only depend on the effect columns not being fully aliased, but also the standard deviations of the effect estimates being low enough to determine significance. How much the standard deviations of the estimates are affected by using blocks with different D_s-efficiencies will be investigated in chapter 5.

Table 4.48: Summary of the results for all the designs tested.

Design	Section	Strategy	Screen	Max D_s	Min D_s	Mean D_s
2^{8-4}_{IV}	4.1.1	Mirror image	(16,8,3,2)	1	0.917	0.929
2^{5-1}_{V}	4.1.2.1	Hadamard+All	(16,5,3,2)	1	0.917	0.934
2^{5-1}_{V}	4.1.2.3	Hadamard+All	(16,5,4_{1+3},2)	1	0.841	0.952
2^{16-11}_{IV}	4.2.1	Doubling	(32,16,3,2)	1	0.917	0.970
2^{16-11}_{IV}	4.2.2	Mirror image	(32,16,3,2)	1	0.917	0.971
2^{16-11}_{IV}	4.2.3	Mirror image	(32,16,3,4)	1	0.834	0.908
2^{6-1}_{VI}	4.2.4.1	Mirror image	(32,6,3*,2)	1	0.943	0.971
2^{6-1}_{VI}	4.2.4.2	Mirror image	(32,6,4,2)	0.982	0.917	0.959
2^{6-1}_{VI}	4.2.4.4	Mirror image	(32,6,5_3,2)	0.963	0.948	0.958
2^{6-1}_{VI}	4.2.4.5	Mirror image	(32,6,5_{3+2},2)	0.966	0.906	0.939
2^{6-1}_{VI}	4.2.5.1	Mirror image	(32,6,3*,4)	0.949	0.875	0.908
2^{6-1}_{VI}	4.2.5.2	Mirror image	(32,6,4,4)	0.924	0.826	0.870
2^{6-1}_{VI}	4.2.5.3	Mirror image	(32,6,5_3,4)	0.866	0.866	0.866
2^{7-2}_{IV}	4.2.6.1	Hadamard	(32,7,3,2)	1	0.917	0.982
2^{7-2}_{IV}	4.2.6.2	Hadamard	(32,7,3,4)	1	0.917	0.939
2^{8-3}_{IV}	4.2.6.3	Hadamard	(32,8,3,2)	1	0.917	0.981
2^{8-3}_{IV}	4.2.6.4	Hadamard	(32,8,3,4)	1	0.853	0.929
2^{9-4}_{IV}	4.2.6.5	Hadamard	(32,9,3,2)	1	0.917	0.982
2^{9-4}_{IV}	4.2.6.6	Hadamard	(32,9,3,4)	1	0.853	0.925
2^{32-26}_{IV}	4.3.1	Doubling	(64,32,3,2)	1	0.917	0.987
2^{32-26}_{IV}	4.3.2	Doubling	(64,32,3,4)	1	0.913	0.960
2^{8-2}_{V}	4.3.3.1	Hadamard	(64,8,3*,2)	1	0.965	0.992
2^{8-2}_{V}	4.3.3.2	Hadamard	(64,8,4,2)	1	0.958	0.986
2^{8-2}_{V}	4.3.3.3	Hadamard	(64,8,3*,4)	1	0.931	0.982
2^{8-2}_{V}	4.3.3.4	Hadamard	(64,8,4,4)	1	0.917	0.966
2^{8-2}_{V}	4.3.3.5	Hadamard	(64,8,3*,8)	1	0.853	0.918
2^{8-2}_{V}	4.3.3.6	Hadamard	(64,8,4,8)	0.917	0.808	0.889

Table 4.49 is used to assess the screening properties of the designs. The column "Screen" shows the screening properties when using the recommended blocking in the corresponding section. The column "Recommended blocks generator(s)" shows the block generator(s) recommended by Wu and Hamada [7]. For the designs with more than nine factors, recommended block generators were not found in the literature. The column "Screen 2" shows the screening properties when using the recommended block generator(s).

Note that the extended screening definition $(n,k,P_{\alpha+a},b)$ was not explored for Screen 2, so the (n,k,P,b) definition is better suited for comparison. It should also be noted that the projectivity is a rather strict measure when using the recommended blocks by confounding of interactions. The projectivity is for example $P = 1$ even if only one among many two-factor interactions cannot be estimated. To exemplify, the only two-factor interaction which cannot be estimated when using the recommended block by confounding of interactions for the 2_V^{5-1} design is AB, as it is the block generator. Having that in mind, it can be seen that the projectivity is increased for all designs when using the approaches tested in this thesis instead of the recommended blocking by confounding of interactions.

Table 4.49: Summary of the results for all the designs tested. The column "Screen" shows the screening property when using the recommended blocking in the corresponding section. The column "Screen 2" shows the screening property when using the recommended block generator(s).

Design	Section	Strategy	Screen	Recommended block generator(s)	Screen 2
2_{IV}^{8-4}	4.1.1	Mirror image	(16,8,3,2)	AB	(16,8,1,2)
2_{V}^{5-1}	4.1.2.1	Hadamard+All	(16,5,3,2)	AB	(16,5,1,2)
2_{V}^{5-1}	4.1.2.3	Hadamard+All	(16,5,4_{1+3},2)	AB	(16,5,1,2)
2_{IV}^{16-11}	4.2.1	Doubling	(32,16,3,2)	Unknown	Unknown
2_{IV}^{16-11}	4.2.2	Mirror image	(32,16,3,2)	Unknown	Unknown
2_{IV}^{16-11}	4.2.3	Mirror image	(32,16,3,4)	Unknown	Unknown
2_{VI}^{6-1}	4.2.4.1	Mirror image	(32,6,3*,2)	ABC	(32,6,2,2)
2_{VI}^{6-1}	4.2.4.2	Mirror image	(32,6,4,2)	ABC	(32,6,2,2)
2_{VI}^{6-1}	4.2.4.4	Mirror image	(32,6,5_3,2)	ABC	(32,6,2,2)
2_{VI}^{6-1}	4.2.4.5	Mirror image	(32,6,5_{3+2},2)	ABC	(32,6,2,2)
2_{VI}^{6-1}	4.2.5.1	Mirror image	(32,6,3*,4)	B1=ACD, B2=BCD	(32,6,1,4)
2_{VI}^{6-1}	4.2.5.2	Mirror image	(32,6,4,4)	B1=ACD, B2=BCD	(32,6,1,4)
2_{VI}^{6-1}	4.2.5.3	Mirror image	(32,6,5_3,4)	B1=ACD, B2=BCD,	(32,6,1,4)
2_{IV}^{7-2}	4.2.6.1	Hadamard	(32,7,3,2)	ACD	(32,7,2,2)
2_{IV}^{7-2}	4.2.6.2	Hadamard	(32,7,3,4)	B1=ACD, B2=BCD	(36,7,1,4)
2_{IV}^{8-3}	4.2.6.3	Hadamard	(32,8,3,2)	ABE	(32,8,2,2)
2_{IV}^{8-3}	4.2.6.4	Hadamard	(32,8,3,4)	B1=AC, B2=AD	(36,8,1,4)
2_{IV}^{9-4}	4.2.6.5	Hadamard	(32,9,3,2)	AB	(32,9,1,2)
2_{IV}^{9-4}	4.2.6.6	Hadamard	(32,9,3,4)	B1=AB, B2=AC	(36,9,1,4)
2_{IV}^{32-26}	4.3.1	Doubling	(64,32,3,2)	Unknown	Unknown
2_{IV}^{32-26}	4.3.2	Doubling	(64,32,3,4)	Unknown	Unknown
2_{V}^{8-2}	4.3.3.1	Hadamard	(64,8,3*,2)	ACE	(64,8,2,2)
2_{V}^{8-2}	4.3.3.2	Hadamard	(64,8,4,2)	ACE	(64,8,2,2)
2_{V}^{8-2}	4.3.3.3	Hadamard	(64,8,3*,4)	B1=ACE, B2=BDF	(64,8,2,4)
2_{V}^{8-2}	4.3.3.4	Hadamard	(64,8,4,4)	B1=ACE, B2=BDF	(64,8,2,4)
2_{V}^{8-2}	4.3.3.5	Hadamard	(64,8,3*,8)	B1=ADF, B2=BDF, B3=ACDEF	(64,8,1,8)
2_{V}^{8-2}	4.3.3.6	Hadamard	(64,8,4,8)	B1=ADF, B2=BDF, B3=ACDEF	(64,8,1,8)

So which designs are preferable for different run sizes? In most cases, it depends on how many factors are believed to be active. If it is important to get low standard deviations for the estimates, a high minimum D_s-efficiency may be prioritised over being able to screen many factors. A short evaluation of the designs with equal run sizes capable of estimating the same number of active factors is given below.

4.4.1 16 runs, three active factors, two blocks

The 16-run designs tested were a 2_{IV}^{8-4} and a 2_{V}^{5-1} design, blocked using the mirror image pairs approach and rearranging a Hadamard matrix, respectively. The 2_{IV}^{8-4} design could be used to screen eight factors, and for three active factors, all effects were estimable. The 2_{V}^{5-1} design could be used to screen five factors, and as for the 2_{IV}^{8-4} design, all effects were estimable when there were three active factors. The minimum D_s-values were equal, as were the maximum D_s-values. The 2_{V}^{5-1} design obtained a slightly higher average D_s-efficiency, making it preferable if it is sufficient to screen five factors.

4.4.2 32 runs, three active factors, two and four blocks

Division into two blocks

The 32-run designs tested which could be used to estimate all effects for three active factors were a 2_{VI}^{6-1}, a 2_{IV}^{7-2}, a 2_{IV}^{8-3}, a 2_{IV}^{9-4} and a 2_{IV}^{16-11} design. The 2_{IV}^{16-11} design was blocked using the mirror image pairs approach and doubling of the blockings that were preferred for the 2_{IV}^{8-4} design. The best result was obtained using the mirror image-strategy, yielding a (32,16,3,2) screen with a minimum D_s-efficiency of 0.917, maximum of 1 and average of 0.971. The 2_{VI}^{6-1} design was blocked using the mirror image approach, resulting in a (32,6,4,2) screen. The minimum D_s efficiency when estimating all effects for three active factors was 0.943, the maximum was 1 and the average 0.971. The 2_{IV}^{7-2}, 2_{IV}^{8-3} and 2_{IV}^{9-4} designs were blocked using a Hadamard matrix. They all achieved projectivity $P = 3$, with a minimum D_s-efficiency of 0.917, maximum of 1, and very similar averages of 0.982 for the 2_{IV}^{7-2} and 2_{IV}^{9-4} designs and 0.981 for the 2_{IV}^{8-3} design.

As the average and maximum D_s-efficiencies were quite similar for all five designs, the 2_{VI}^{6-1} design is preferable if six or fewer factors are to be screened, as it has a higher minimum D_s-efficiency. In addition, it can be used to screen for up to five active factors when up to three-factor interactions are to be estimated, thus enabling discovery of more active factors than the other designs. If seven, eight or nine factors are to be screened, a slightly higher average is obtained using the 2_{IV}^{7-2}, 2_{IV}^{8-3} or 2_{IV}^{9-4} design instead of a subset of columns from the 2_{IV}^{16-11} design, but the difference is negligible.

Division into four blocks
The same 32-run designs which were divided into two blocks were also divided into four blocks. The 2_{IV}^{16-11} design divided into the recommended blocks resulted in a (32,16,3,4) screen, obtaining a minimum D_s-efficiency of 0.824, a maximum of 1 and an average of 0.908,. When the 2_{VI}^{6-1} design was divided into four blocks and used to screen for three active factors, the average D_s-efficiency was also 0.908, but the minimum was 0.875 and the maximum was 0.949. The corresponding results for dividing the 2_{IV}^{7-2} design into four blocks was a minimum of 0.917, a maximum of 1 and an average of 0.939. The 2_{IV}^{8-3} design obtained a minimum of 0.853, a maximum of 1 and an average of 0.929. The 2_{IV}^{8-3} divided into four blocks also obtained a minimum of 0.853 and a maximum of 1, but a slightly lower average of 0.925.

As the 2_{IV}^{7-2} design obtained the highest average and minimum value, and a maximum D_s-efficiency of 1, it is preferable if it is sufficient to screen up to seven factors. If more than seven factors are to be screened, the smallest design accommodating all factors is recommendable, as the 2_{IV}^{8-3} design is slightly better than the 2_{IV}^{9-4} design, which is better than the 2_{IV}^{16-11} design.

4.4.3 64 runs, three active factors, two and four blocks

Division into two blocks
The two 64-run designs tested were a 2_{V}^{8-2} design blocked using columns orthogonal to the main effects found by rearranging a Hadamard matrix, and a 2_{IV}^{32-26} design blocked by utilising the doubling of a 2_{IV}^{16-11} design blocked using the mir-

ror image pairs approach. The 2_{IV}^{32-26} design can include 32 screening factors, and the preferred block yielded a minimum D_s-efficiency of 0.917, a maximum of 1, and an average of 0.987. The 2_V^{8-2} design on the other hand, can only be used to screen eight factors, and the preferred block yielded a minimum value of 0.965, a maximum of 1 and an average of 0.992. Thus a higher minimum value is ensured using the 2_V^{8-2} design, at the cost of being able to screen only eight factors instead of 32. An advantage of the 2_V^{8-2} design is that it enables screening of four active factors using the same blocks as for three active factors.

Division into four blocks

The 64-run designs which were divided into two blocks were also divided into four blocks. The 2_{IV}^{32-26} design obtained a minimum of 0.913, a maximum of 1 and an average D_s-efficiency of 0.960 when three factors were active. The 2_V^{8-2} design obtained a minimum of 0.931, a maximum of 1 and an average of 0.982 for threee active factors, making it preferable if it is sufficient to screen eight factors. Another advantage of the 2_V^{8-2} design is that some of the preferred blockings in the case of three active factors also enables estimation of all effects for four active factors.

Chapter 5

Evaluation of D_s-efficiencies

Obtaining high D_s-efficiencies for as many active factors as possible was the aim of the results section. But how much is the estimation of effects affected by using blocks with different D_s-efficiencies? To investigate this, an example with reactor data from Box, Hunter and Hunter [22] is used to illustrate the differences in parameter estimates and standard deviations when using a block with $D_s < 1$ compared to a block with $D_s = 1$. The results are presented in section 5.1. In section 5.2, the standard deviations are found and compared when the active factors yield the lowest and highest D_s-efficiencies for the preferred block for all designs.

5.1 Comparison using reactor data example

Box, Hunter and Hunter [22] includes a widely used example of a 2^5 factorial design used to analyse reactor data, and the corresponding 2_V^{5-1} design using ABCDE as the design generator. The focus in this section will be on the 2_V^{5-1} design, whose five factors and corresponding levels can be found in table 5.1. The response was the percentage which had reacted.

In section 4.1.2.1, it was shown how the 2_V^{5-1} design could be divided into two blocks resulting in a (16,5,3,2) screen. To check how the results would have been affected if the reactor experiment had been blocked, one block yielding a D_s-

Table 5.1: The factors and levels used in the reactor example.

Letter	Variable	-1	1
A	Feed rate (liters/min)	10	15
B	Catalyst (%)	1	2
C	Agitation rate (rpm)	100	120
D	Temperature (°C)	140	180
E	Concentration (%)	3	6

efficiency of 1 and one block yielding a D_s-efficiency of 0.917 when B, D and E are the active factors were tested. The factor columns and the blocks can be found in table 5.2. The table also includes the responses. To introduce a block effect, the response in the block with 1's was increased by five and the response in the block with -1's was decreased by five when analysing the data. The resulting responses can be found in the columns "D_s=1 response" and "D_s=0.917 response".

The models tested had the same explanatory variables except for the block effect, namely B, D, E, BD, BE, DE, BDE. The model including the block with D_s=1 had the "D_s=1 response", and the model including the block with D_s=0.917 had the "D_s=0.917 response". The resulting parameter estimates and corresponding standard deviations can be found in table 5.3. As the block with D_s=1 was orthogonal to all effects, the parameter estimates were equal to the estimates in the book (except for the estimate for BDE, which was not included there). All the parameter estimates had the same standard deviation when using the D_s=1 block. When using the block with D_s=0.917, the parameter estimates were different from the original ones for the interactions which were partially confounded with the block, namely BE and DE. The parameter estimates for these effects were slightly lower than the original estimates, while the estimated block effect was higher than for the orthogonal block. As the estimators for the effects of BE an DE are unbiased despite the partial confounding, this result happened by chance. The standard deviations were higher for all parameter estimates when using the $D_s = 0.917$-block than when using the $D_s = 1$ block. This is because the design with the $D_s = 0.917$-block utilised less information due to the partially confounded columns. For the

Table 5.2: The factors, the different blocks being tested and the response from the reactor example. The columns "D_s=1 response" and "D_s=0.917 response" shows the responses when a block effect is introduced.

B	D	E	D_s=1 block	D_s=0.917 block	Response	D_s=1 response	D_s=0.917 response
-1	-1	1	1	1	56	61	61
-1	-1	-1	-1	-1	53	48	48
1	-1	-1	1	1	63	68	68
1	-1	1	-1	-1	65	60	60
-1	-1	-1	1	1	53	58	58
-1	-1	1	-1	-1	55	50	50
1	-1	1	1	-1	67	72	62
1	-1	-1	-1	1	61	56	66
-1	1	-1	-1	-1	69	64	64
-1	1	1	1	1	45	50	50
1	1	1	1	1	78	83	83
1	1	-1	-1	-1	93	88	88
-1	1	1	-1	1	49	44	54
-1	1	-1	1	-1	60	65	55
1	1	-1	1	1	95	100	100
1	1	1	-1	-1	82	77	77

Table 5.3: Parameter estimates when using blocks with different D_s-efficiencies.

Effect	D_s=1 estimate	D_s=0.917 Estimate	D_s=1 Std.dev	D_s=0.917 Std.dev
K	65.2500	65.2500	0.71183	0.74926
B	10.2500	10.2500	0.71183	0.74926
D	6.1250	6.1250	0.71183	0.74926
E	-3.1250	-3.1250	0.71183	0.74926
BD	5.3750	5.3750	0.71183	0.74926
BE	0.6250	0.5625	0.71183	0.91765
DE	-4.7500	-4.6875	0.71183	0.91765
BDE	0.2500	0.2500	0.71183	0.74926
Block	4.3750	4.8750	0.71183	1.05961

$D_s = 0.917$-block, interactions orthogonal to the block had equal standard deviations, while interactions with the same degree of confounding with the block, BE and DE, had the same higher standard deviation. The standard deviation of the corresponding estimates was $\frac{0.91765}{0.74926} = 1.22474$ times as large as the standard deviation of the estimates for the orthogonal effects. The block effect had the highest standard deviation, $\frac{1.05961}{0.74923} = 1.41427$ times as large as the standard deviation of the orthogonal effects.

This ratio can also be found by inspecting $(\mathbf{X}^T\mathbf{X})^{-1}$, where $\mathbf{X}$ was the design matrix with the $D_s = 0.917$ block. $(\mathbf{X}^T\mathbf{X})^{-1}$ can be found in table 5.4. As explained in section 2.1.1, the covariance matrix of the regression coefficients is given by $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$, and the variance of coefficient nr. i can be found as $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ii}$. The ratio of the standard deviation of coefficient i divided by the standard deviation of coefficient j is thereby $\frac{\sqrt{\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ii}}}{\sqrt{\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{jj}}} = \frac{\sqrt{(\mathbf{X}^T\mathbf{X})^{-1}_{ii}}}{\sqrt{(\mathbf{X}^T\mathbf{X})^{-1}_{jj}}}$. Note that for the estimated standard deviations, $\hat{\sigma}^2$ replaces σ^2, but as it cancels in the expression, it does not affect the final result. The ratio of the standard deviation of the partially confounded interactions divided by the standard deviation of the orthogonal interactions was $\frac{\sqrt{0.09375}}{\sqrt{0.06250}}$=1.22474, as found previously.

Table 5.4: $(\mathbf{X}^T\mathbf{X})^{-1}$ for the example design, with the D_s=0.917 block.

	K	**B**	**D**	**E**	**BD**	**BE**	**DE**	**BDE**	**Block**
K	0.0625	0	0	0	0	0	0	0	0
B	0	0.0625	0	0	0	0	0	0	0
D	0	0	0.0625	0	0	0	0	0	0
E	0	0	0	0.0625	0	0	0	0	0
BD	0	0	0	0	0.0625	0	0	0	0
BE	0	0	0	0	0	0.0938	-0.0313	0	0.0625
DE	0	0	0	0	0	-0.0313	0.0938	0	-0.0625
BDE	0	0	0	0	0	0	0	0.0625	0
Block	0	0	0	0	0	0.0625	-0.0625	0	0.1250

Failing to block when needed

The estimates for BE and DE became slightly altered when a block with which they were partially confounded was used. But what would have happened to the estimates if there was a block effect present, but the experimenter had failed to adjust for it by introducing a blocking column? To test this, the model with "D_s=0.917 response" as response was fitted without block as an explanatory variables. Then the block effect of size five was no longer accounted for. The estimates for the effects in orthogonal columns were the same as before, but BE and DE changed. The estimate of BE was now -1.8750, and the estimate of DE was -2.2500. The total change in estimates was $|(0.5625-(-1.8750))|+|(-4.6875-(-2.2500))| = 4.8750$, exactly the size of the previously estimated block effect. Thus when the block is not taken into account, its effect is captured by the partially confounded interactions instead, making their estimates wrong. The standard deviations of all the estimates for the model not including the block effect was 1.406, almost twice as high as the standard deviations of the estimates for the orthogonal effects for the model including the block effect. Failing to block thereby also decreases the chances of discovering significant effects.

5.2 Evaluation of the preferred blockings for all designs

Having seen in the previous section that partial confounding between interactions and blocks leads to higher standard deviations, a closer look at the effect of partial confounding is interesting. For all designs which were tested in section 4, the D_s-efficiency reflects the degree of partial confounding between interaction effects and blocks, as only blocks which were orthogonal to the main effects columns were chosen. The preferred blockings yielded different D_s-efficiencies for different combinations of active factors. The question of interest is then to quantify how much different D_s-efficiencies, and thereby different partial confoundings, affect the standard deviation of the estimates. To do so, the preferred blockings of all designs were tested for the combinations of active factors yielding the highest and lowest D_s-efficiency. The results can be found in table 5.5.

The column "Max D_s" shows the highest obtained D_s-efficiency when using the preferred block in the corresponding section, while the column "Min D_s" shows the lowest obtained D_s-efficiency. To the right of these columns, "Max eff." shows the highest standard deviation of any main or interaction effect i divided by the smallest standard deviation of any main or interaction effect j, $\frac{\sqrt{\max((X^TX)_{ii}^{-1})}}{\sqrt{\min((X^TX)_{jj}^{-1})}}$ for the combinations of active factors yielding the maximum and minimum D_s-efficiencies, respectively.

For many blocks, there are interactions with different degrees of partial confounding with the block. The maximum value of the standard deviation of a block effect is then obtained by the block column that has the strongest partial confounding with an interaction effect column. "Max block" shows the highest standard deviation of any block effect b divided by the smallest standard deviation of any main or interaction effect j, $\frac{\sqrt{\max((X^TX)_{bb}^{-1})}}{\sqrt{\min((X^TX)_{jj}^{-1})}}$. As for "Max eff.", the "Max block" to the right of "Max D_s" shows the ratio when using the combination yielding the highest obtained D_s-efficiency, and the "Max block" to the right of "Min D_s" shows the ratio when using the combination yielding the lowest obtained D_s-efficiency.

Table 5.5: The ratios "Max D_s"= $\frac{\sqrt{\max((X^TX)^{-1}_{ii})}}{\sqrt{\min((X^TX)^{-1}_{jj})}}$ and "Max block"= $\frac{\sqrt{\max((X^TX)^{-1}_{bb})}}{\sqrt{\min((X^TX)^{-1}_{jj})}}$ for the combination yielding the highest and lowest D_s-efficiencies, for all preferred blockings.

Design	Section	Proj.	Max D_s	Max eff.	Max block	Min D_s	Max eff.	Max block
2^{8-4}_{IV}	4.1.1	(16,8,3,2)	1	1	1	0.917	1.2247	1.4142
2^{5-1}_{V}	4.1.2.1	(16,5,3,2)	1	1	1	0.917	1.2247	1.4142
2^{5-1}_{V}	4.1.2.3	(16,5,4_{1+3},2)	1	1	1	0.841	1.4142	2
2^{16-11}_{IV}	4.2.1	(32,16,3,2)	1	1	1	0.917	1.2247	1.4142
2^{16-11}_{IV}	4.2.2	(32,16,3,2)	1	1	1	0.917	1.2247	1.4142
2^{16-11}_{IV}	4.2.3	(32,16,3,4)	1	1	1	0.834	1.3904	1.5275
2^{6-1}_{VI}	4.2.4.1	(32,6,3*,2)	1	1	1	0.943	1.1832	1.2649
2^{6-1}_{VI}	4.2.4.2	(32,6,4,2)	0.982	1.0408	1.1547	0.917	1.4142	2
2^{6-1}_{VI}	4.2.4.4	(32,6,5_3,2)	0.963	1.2910	1.6330	0.948	1.4142	2
2^{6-1}_{VI}	4.2.4.5	(32,6,5_{3+2},2)	0.966	1.2910	1.6330	0.906	2.2361	4
2^{6-1}_{VI}	4.2.5.1	(32,6,3*,4)	0.949	1.0742	1.0742	0.875	1.4771	1.3817
2^{6-1}_{VI}	4.2.5.2	(32,6,4,4)	0.924	1.2910	1.4142	0.826	1.7321	2.2361
2^{6-1}_{VI}	4.2.5.3	(32,6,5_3,4)	0.866	1.8257	2.4495	0.866	1.8257	2.4495
2^{7-2}_{IV}	4.2.6.1	(32,7,3,2)	1	1	1	0.917	1.2247	1.4142
2^{7-2}_{IV}	4.2.6.2	(32,7,3,4)	1	1	1	0.917	1.2247	1.4142
2^{8-3}_{IV}	4.2.6.3	(32,8,3,2)	1	1	1	0.917	1.2247	1.4142
2^{8-3}_{IV}	4.2.6.4	(32,8,3,4)	1	1	1	0.853	1.2247	1.4142
2^{9-4}_{IV}	4.2.6.5	(32,9,3,2)	1	1	1	0.917	1.2247	1.4142
2^{9-4}_{IV}	4.2.6.6	(32,9,3,4)	1	1	1	0.853	1.2247	1.4142
2^{32-26}_{IV}	4.3.1	(64,32,3,2)	1	1	1	0.917	1.2247	1.4142
2^{32-26}_{IV}	4.3.2	(64,32,3,4)	1	1	1	0.913	1.1704	1.2163
2^{8-2}_{V}	4.3.3.1	(64,8,3*,2)	1	1	1	0.965	1.1547	1.1547
2^{8-2}_{V}	4.3.3.2	(64,8,4,2)	1	1	1	0.958	1.2247	1.4142
2^{8-2}_{V}	4.3.3.3	(64,8,3*,4)	1	1	1	0.931	1.1547	1.1547
2^{8-2}_{V}	4.3.3.4	(64,8,4,4)	1	1	1	0.917	1.2247	1.4142
2^{8-2}_{V}	4.3.3.5	(64,8,3*,8)	1	1	1	0.853	1.2247	1.4142
2^{8-2}_{V}	4.3.3.6	(64,8,4,8)	0.917	1.2247	1.4142	0.808	1.5492	1.5492

The ratios "Max eff." and "Max block" for the combination yielding the lowest D_s-efficiency, as shown in column eight and nine, can be interpreted as the magnitude of the increase in standard deviation for the block and the interaction with the strongest partial confounding. For example considering the 2_{IV}^{8-4} design with a minimum D_s-efficiency of 0.917, the standard deviation was 22% higher for the strongest partially confounded interaction than for the orthogonal ones. An increase of 22% makes the confidence interval of the parameter estimate 22% wider. Clearly, checking the maximum ratio of the standard deviations is useful when being sure to determine the significant interactions is important.

An interesting observation is that the highest "Max eff." for the 2_{VI}^{6-1} design when using it as a (32,6,5_{3+2},2) screen was 2.236, thus the standard deviation of the interaction with the strongest partial confounding was 123.6% higher than the standard deviation of the orthogonal ones. This is a surprisingly large difference, as the minimum D_s-efficiency was 0.906, just below the D_s-efficiency which lead to a 22% increase for the 2_{IV}^{8-4} design. The reason is that the (32,6,5_{3+2},2) screen has a lot more columns. Thus the effect of one strongly partially confounded effect on the overall D_s-efficiency is not very large, but the ability to determine whether the effect is significant is nevertheless very limited due to the increased standard deviation. Hence it might be useful to investigate how much the standard deviations increases when using blocks which are partially confounded with interaction effects even when the D_s-efficiency is high, particularly for large designs.

Despite the D_s-efficiency decreasing with the number and strength of partial confoundings between interactions and blocks, there was no one-to-one correspondence between the D_s-efficiency and the standard deviation ratios. The standard deviation ratios were for example equal for some designs for the combination of active factors yielding a D_s-efficiency of 0.917, among others the 2_{IV}^{8-4} design used as a (16,8,3,2) screen, the 2_{IV}^{16-11} design used as a (32,16,3,2) screen and the 2_{IV}^{7-2} design used as a (32,7,3,2) screen. For these designs, the $(\mathbf{X}^T\mathbf{X})$ matrices had the same structure: Two, and only two, columns were partially confounded with the block column and had a dot product with the block column whose absolute value was half the value of the dot product of the block with itself. $D_s = 0.917$ was however also obtained for the 2_{VI}^{6-1} design when used as a (32,6,4,2) screen.

This design did not yield the same standard deviation ratios as the others, as it had several interactions with weak partial confounding with the block.

In addition to the existence of designs with different standard deviation ratios yielding the same minimum D_s-efficiency, there were also designs with the same standard deviation ratios and different minimum D_s-efficiencies. The 2_{VI}^{6-1} design did for example have a "Max eff." of 1.4142 for the minimum D_s-combination both when estimating all effects for four active factors and when estimating up to three-factor interactions for five active factors. The minimum D_s-efficiencies were however 0.917 and 0.948, respectively.

The overall impression after evaluating the standard deviation ratios for all designs is that the D_s-efficiency is a good measure of the efficiency with which the parameter estimates are found. Its limitation lies in not indicating whether standard deviations become very large for the partially confounded interactions. This highlights the importance of choosing blocks which are orthogonal to the main effects, as their estimates are then guaranteed the lowest possible standard deviation. The experimenter is recommended to check the $(\mathbf{X}^T\mathbf{X})^{-1}$ matrix herself if being able to determine the significance of all interactions is important.

Chapter 6

Concluding remarks

In this thesis, the main idea was to allow for partial confounding between blocks and interaction effects to possibly achieve better projectivity properties than when using orthogonal blocks based on confounding of interactions. The methods used to find candidate blocks were utilising mirror image pairs and rearranging Hadamard matrices. These were tested on a number of 16-, and 32- and 64-run designs which have good projectivity properties and accommodate a high number of screening factors. Both blocking approaches were found to give better projectivity properties for the blocked designs than the confounding approach, without substantially decreasing the D_s-efficiencies.

Utilising mirror image pairs proved to be an interesting approach as the blocks made up of mirror image pairs amounted for only a small fraction of all the possible blocks for the different run sizes, but many of them nevertheless yielded good projectivity properties and D_s-efficiencies for the designs tested. The method is easily scaled for designs of any run size, as long as the designs consist of mirror image pairs. The number of candidate blocks does however increase substantially with the run size, so testing all possible blocks based on mirror image pairs is not feasible for large designs. Then one might consider using the approach of doubling a smaller design which has been divided into blocks, and using them to define the new blocks. A disadvantage of basing the division into blocks on

mirror image pairs is that it limits the number of screening factors which may be included in the design, as all design generators must have an odd number of letters.

Rearranging Hadamard matrices imposes no such limitation on the number of screening factors, and also yielded good projectivity properties and D_s-efficiencies for the designs tested. An advantage of rearranging Hadamard matrices is that the number of candidate blocks generated from one matrix is relatively low, yet most of the candidate blocks tested yielded the desired projectivity properties and high D_s-efficiences. The disadvantage of this method is that the number of possible ways to rearrange a Hadamard matrix strongly increases with its size, making it impossible to know whether the best possible block has been found. As some of the matrices tested were not arranged into the desired design even when testing different ways of ordering the rows for a long time, it would have been interesting to investigate which Hadamard matrices can be rearranged to match different designs, and if the Hadamard matrix columns used as design columns could have been chosen in a more planned manner. These questions are particularly important for large designs, as the number of Hadamard matrices drastically increases for order 32 and above. Note also that the Hadamard matrix approach was only tested when the designs did not consist of mirror image pairs. Another interesting issue is therefore whether rearranging Hadamard matrices would have yielded better results for the designs which were blocked using mirror image pairs.

After testing the blocking methods, the preferred blocks were evaluated in terms of their impact on the standard deviations of the partially confounded effects. The most interesting observation was that the amount by which the standard deviations increased could not be found merely by considering the D_s-efficiency. This was particularly relevant for one of the large designs when more than three factors were allowed to be active, as the total number of screening factor in that case masked the effect of a strongly partially confounded effect on the D_s-efficiency. A general advise when choosing blocks is therefore to investigate both the D_s-efficiencies and the diagonal of $(\mathbf{X}^T\mathbf{X})^{-1}$ when obtaining low standard deviations for the estimated effects is important for the screening. A question for further research could be which blocks to prefer if an upper bound is defined for the increase in the standard deviations of the estimated effects.

Bibliography

[1] Bodmer, W., "RA Fisher, statistician and geneticist extraordinary: a personal view," *International Journal of Epidemiology*, vol. 32, no. 6, pp. 938–942, 2003.

[2] Fisher, R., *The Design of Experiments. 1935*. Edinburgh: Oliver and Boyd, 1935.

[3] Box, G., "An Accidental Statistician," 6 2010, speech held at the 50th anniversay of the Statistics Department of the University of Wisconsin. [Online]. Available: http://www.stat.wisc.edu/~yandell/stat/50-year/Box_George.html

[4] Box, G. and Tyssedal, J., "Projective Properties of Certain Orthogonal Arrays," *Biometrika*, vol. 83, no. 4, pp. 950–955, 1996.

[5] Tyssedal, J., "Design of Experiments," NTNU.

[6] Montgomery, D., *Design and Analysis of Experiments, 7th edition*.

[7] Hamada, M. and Wu, J., *Planning, Analysis, and Parameter Design Optimization*. Wiley, 2000.

[8] Evangelaras, H. and Koukouvinos, C., "On generalized projectivity of two-level screening designs," *Statistics & Probability Letters*, vol. 68, no. 4, pp. 429–434, 2004.

[9] Hussain, S. and Tyssedal, J., "Projection Properties of Blocked Non-regular Two-level Designs," *Quality and Reliability Engineering International*, vol. 32, no. 8, pp. 3011–3021.

[10] Tyssedal, J., "Projectivity in Experimental Designs," *Encyclopedia of Statistics in Quality and Reliability*, 2008.

[11] Samset, O. and Tyssedal, J., "Two-Level Designs with Good Projection Properties," 1999.

[12] Atkinson, A., Donev, A. and Tobias, R., *Optimum Experimental Designs, with SAS*. Oxford Statistical Science Series, 2007.

[13] Gupta, A., "Generalized Variance," *Encyclopedia of Statistical Sciences*, 2006.

[14] Bibby, J., Kent, J. and Mardia, K., *Multivariate Analysis*. Academic press, 1979.

[15] Jacroux, M., "Blocking in two-level non-regular fractional factorial designs," *Journal of Statistical Planning and Inference*, vol. 139, no. 3, pp. 1215–1220, 2009.

[16] Kllogjeri, A. and Kllogjeri, P, "Partition of a Set with N Elements into K Blocks with Number of Elements in Accordance with the Composition of Number N As a Sum of Any K Natural Summands (Another Representation of Stirling Number)," *International Journal of Advanced Computing*, vol. 46, pp. 2051–845, 08 2013.

[17] Cameron, P., "Hadamard matrices," http://www.maths.qmul.ac.uk/~lsoicher/designtheory.org/library/encyc/topics/had.pdf, accessed: 2018-10-05.

[18] Hadi, K. and Tayfeh-Rezaie, B., “Hadamard Matrices of Order 32,” *Journal of Combinatorial Designs*, vol. 21, no. 5, pp. 212–221.

[19] Sloane, N., “A Library of Hadamard Matrices,” http://neilsloane.com/hadamard/, accessed: 2018-10-05.

[20] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2018. [Online]. Available: https://www.R-project.org/

[21] Box, G. and Tyssedal, J, “Sixteen Run Designs of High Projectivity for Factor Screening,” *Communications in Statistics - Simulation and Computation*, vol. 30, no. 2, pp. 217–228, 2001.

[22] Box, G., Hunter, J. and Hunter, W., *Statistics for experimenters*. John Wiley Sons, 1978.

Appendix A

Table 6.1: Blocks 1-14 of the candidates for blocking the 2^{8-4}_{IV} design in section 4.1.1.

$b1$	$b2$	$b3$	$b4$	$b5$	$b6$	$b7$	$b8$	$b9$	$b10$	$b11$	$b12$	$b13$	$b14$
1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	-1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	-1
-1	1	-1	-1	-1	1	-1	-1	-1	-1	1	1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1	1	-1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1	1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1	-1	1	1	-1	1
1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	-1
-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.2: Blocks 15-28 of the candidates for blocking the 2_{IV}^{8-4} design in section 4.1.1.

$b15$	$b16$	$b17$	$b18$	$b19$	$b20$	$b21$	$b22$	$b23$	$b24$	$b25$	$b26$	$b27$	$b28$
1	1	1	1	1	1	1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	-1
-1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1
1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	1
1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	1
-1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1
-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.3: The submatrix **B1** used to define the design matrix in section 4.3.1.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.4: The submatrix **B2** used to define the design matrix in section 4.3.1.

-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	1	1
1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.5: Upper left submatrix of the design matrix in section 4.3.1 arranged in the preferred order.

Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
3	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
4	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
5	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
6	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
7	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
8	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
9	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
10	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
11	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
12	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
13	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
14	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
15	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
18	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
19	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
20	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
21	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
22	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
23	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
24	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
25	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
26	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
27	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
28	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
29	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
30	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
31	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
32	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.6: Lower left submatrix of the design matrix in section 4.3.1 arranged in the preferred order.

Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
33	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
34	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
33	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
36	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
37	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
38	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
39	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
40	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
41	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
42	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
43	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
44	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
45	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
46	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
47	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
48	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
49	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
50	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
51	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
52	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
53	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
54	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
55	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
56	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
57	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
58	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
59	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
60	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
61	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
62	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
63	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
64	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.7: Upper right submatrix of the design matrix in section 4.3.1 arranged in the preferred order.

Row	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
3	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
4	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
5	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
6	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
7	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
8	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
9	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
10	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
11	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
12	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
13	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
14	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
15	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
18	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
19	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
20	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
21	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
22	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
23	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
24	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
25	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
26	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
27	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
28	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
29	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
30	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
31	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
32	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1

Table 6.8: Lower right submatrix of the design matrix in section 4.3.1 arranged in the preferred order.

Row	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2
33	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
34	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
35	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
36	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
37	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
38	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
39	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
4	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
9	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
10	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
11	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
12	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
13	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
14	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
15	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
18	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
19	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
20	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
21	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
22	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
23	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
24	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
25	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
26	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
27	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
28	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
29	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
30	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
31	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
32	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.9: Upper left submatrix of the design matrix in section 4.3.2 arranged in the preferred order.

Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
2	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
3	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
4	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
5	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
6	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
7	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
8	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
9	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
10	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
11	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
12	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
13	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
14	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
15	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
16	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
19	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
20	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
21	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
22	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
23	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
25	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
26	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
27	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
28	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
29	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
30	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
31	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
32	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.10: Lower left submatrix of the design matrix in section 4.3.2 arranged in the preferred order.

Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
33	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
34	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
35	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
36	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
37	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
38	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
39	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
40	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
41	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
42	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
43	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
44	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
45	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
46	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
47	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
48	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
49	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
50	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
51	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
52	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
53	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
54	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
55	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
56	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
57	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
58	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
59	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
60	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
61	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
62	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
63	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
64	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1

Table 6.11: Upper right submatrix of the design matrix in section 4.3.2 arranged in the preferred order.

Row	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2
1	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
2	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
3	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
4	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
5	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
6	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
7	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
8	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
9	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
10	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
11	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
12	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
13	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
14	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
15	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
16	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
19	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
20	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
21	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
22	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
23	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
25	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
26	-1	-1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	1	1
27	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1
28	1	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
29	-1	1	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	1	-1
30	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
31	1	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
32	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1

Table 6.12: Lower right submatrix of the design matrix in section 4.3.2 arranged in the preferred order.

Row	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2
33	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
34	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
35	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	1
36	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1
37	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
38	1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
39	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1
40	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
41	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
42	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
43	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
44	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
45	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
46	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
47	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
48	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
49	-1	-1	-1	1	1	-1	1	1	1	1	-1	1	1	-1	-1	-1
50	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
51	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
52	-1	-1	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1
53	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
54	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
55	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
56	1	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
57	1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
58	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
59	-1	1	1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	1
60	1	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
61	-1	-1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	1	1	1
62	1	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
63	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
64	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1

Table 6.13: D_s-efficiencies when estimating all interactions for four active factors when using the preferred division into two blocks for the 2_{VI}^{6-1} design in section 4.2.4.2.

Active factors	D_s-efficiency
ABCD	0.982
ABCE	0.958
ABCF	0.958
ABDE	0.958
ABDF	0.958
ABEF	0.982
ACDE	0.917
ACDF	0.982
ACEF	0.958
ADEF	0.958
BCDE	0.917
BCDF	0.982
BCEF	0.958
BDEF	0.958
CDEF	0.958

Table 6.14: D_s-efficiencies for the blocks in table 4.21 for all combinations of four active factors for the 2^{6-1}_{VI} design in section 4.2.5.2.

Active factors	D_s-efficiency
ABCD	0.924
ABCE	0.890
ABCF	0.871
ABDE	0.826
ABDF	0.862
ABEF	0.862
ACDE	0.890
ACDF	0.853
ACEF	0.853
ADEF	0.862
BCDE	0.826
BCDF	0.890
BCEF	0.924
BDEF	0.826
CDEF	0.890

Table 6.15: First 16 columns of the matrix **M** in section 4.2.6

Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
5	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
7	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	-1	1	1	-1
8	1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
11	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
12	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
13	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
14	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
15	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	1
16	1	-1	-1	1	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
19	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
20	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
21	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
22	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
23	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	-1	1	1	-1
24	1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1
25	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
26	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
27	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
28	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
29	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
30	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
31	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	1
32	1	-1	-1	1	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1

Table 6.16: Last 16 columns of the matrix **M** in section 4.2.6

Row	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
5	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
7	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	-1	1	1	-1
8	1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
11	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
12	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
13	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
14	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
15	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	1
16	1	-1	-1	1	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
19	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
20	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
21	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1
22	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
23	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1	1	-1	-1	1
24	-1	1	1	-1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1
25	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1
26	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
27	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1
28	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
29	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1
30	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1
31	-1	-1	1	1	1	1	-1	-1	1	-1	-1	1	-1	1	1	-1
32	-1	1	1	-1	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1

Table 6.17: First 16 columns of the matrix **M2** in section 4.2.6

Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	1	1	1	-1	-1	-1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1
5	1	1	1	1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	1	1	-1
7	1	1	-1	-1	-1	-1	1	1	1	-1	1	-1	-1	1	-1	1
8	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1	1	1	1	1
11	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
12	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1
13	1	1	1	1	-1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1
14	1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1
15	1	1	-1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	-1
16	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	-1	1	-1	1	-1	1	-1	1	1	1	1	-1	-1	-1	-1
19	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
20	1	-1	-1	1	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1
21	1	1	1	1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1	-1
22	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	1	1	-1
23	1	1	-1	-1	-1	-1	1	1	1	-1	1	-1	-1	1	-1	1
24	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
25	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
26	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1	1	1	1	1
27	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
28	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1
29	1	1	1	1	-1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1
30	1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1
31	1	1	-1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	-1
32	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1

Table 6.18: Last 16 columns of the matrix **M2** in section 4.2.6

Row	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	1	1	1	-1	-1	-1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	1	-1	-1	-1	-1	1	1
5	1	1	1	1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	1	1	-1
7	1	1	-1	-1	-1	-1	1	1	1	-1	1	-1	-1	1	-1	1
8	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1	1	1	1	1
11	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
12	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1
13	1	1	1	1	-1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1
14	1	-1	1	-1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1
15	1	1	-1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	-1
16	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	1	1	1	1
19	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
20	-1	1	1	-1	-1	1	1	-1	-1	-1	1	1	1	1	-1	-1
21	-1	-1	-1	-1	1	1	1	1	-1	1	-1	1	-1	1	-1	1
22	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	1	-1	-1	1
23	-1	-1	1	1	1	1	-1	-1	-1	1	-1	1	1	-1	1	-1
24	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
25	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1
26	-1	1	-1	1	-1	1	-1	1	1	1	1	1	-1	-1	-1	-1
27	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1
28	-1	1	1	-1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1
29	-1	-1	-1	-1	1	1	1	1	1	-1	1	-1	1	-1	1	-1
30	-1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1
31	-1	-1	1	1	1	1	-1	-1	1	-1	1	-1	-1	1	-1	1
32	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1

Table 6.19: First 16 columns of the matrix **M3** in section 4.2.6

Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
5	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
7	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
8	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	1	1	1
11	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1	1	-1
12	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1	1
13	1	-1	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1
14	1	-1	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1
15	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1
16	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1	1	-1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
19	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
20	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
21	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
22	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
23	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
24	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
25	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
26	1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	1	1	1
27	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1	1	-1
28	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1	1
29	1	-1	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1
30	1	-1	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1
31	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1
32	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1	1	-1

Table 6.20: Last 16 columns of the matrix **M3** in section 4.2.6

Row	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
3	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
5	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
6	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
7	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
8	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
9	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
10	1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	1	1	1
11	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1	1	-1
12	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1	1
13	1	-1	1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	1	1
14	1	-1	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1
15	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1
16	1	-1	-1	-1	1	1	-1	1	-1	1	1	1	-1	-1	1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
19	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
20	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
21	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1
22	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
23	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1
24	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
25	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1
26	-1	-1	-1	1	-1	1	1	1	1	1	1	-1	1	-1	-1	-1
27	-1	-1	1	-1	1	1	1	-1	1	1	-1	1	-1	-1	-1	1
28	-1	-1	1	1	1	-1	-1	1	1	1	-1	-1	-1	1	1	-1
29	-1	1	-1	-1	1	-1	1	1	1	-1	1	1	-1	1	-1	-1
30	-1	1	-1	1	1	1	-1	-1	1	-1	1	-1	-1	-1	1	1
31	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1
32	-1	1	1	1	-1	-1	1	-1	1	-1	-1	-1	1	1	-1	1

Table 6.21: The first 32 rows of the 2_V^{8-2} design and the preferred blocks from section 4.3.3.

Row	A	B	C	D	E	F	G	H	$b1$	$b2$	$b3$	$b4$	$b5$	$b6$	$b7$	$b8$	$b9$	$b10$	$b11$	$b12$
1	-1	-1	-1	-1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
2	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1
3	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1
5	-1	-1	1	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
6	1	-1	1	-1	-1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	1	-1	-1
7	-1	1	1	-1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	1	1
8	1	1	1	-1	-1	-1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	1
9	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
10	1	-1	-1	1	-1	-1	1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
11	-1	1	-1	1	-1	-1	1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1
12	1	1	-1	1	-1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	1	-1
13	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	-1	-1	1
14	1	-1	1	1	-1	-1	-1	-1	-1	1	-1	1	1	-1	1	-1	1	1	-1	-1
15	-1	1	1	1	-1	-1	-1	-1	-1	1	-1	1	-1	1	-1	1	1	1	1	1
16	1	1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
17	-1	-1	-1	-1	1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1
18	1	-1	-1	-1	1	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1
19	-1	1	-1	-1	1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1
20	1	1	-1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
21	-1	-1	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	-1
22	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1
23	-1	1	1	-1	1	-1	1	1	-1	1	-1	1	1	-1	1	-1	-1	-1	1	1
24	1	1	1	-1	1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
25	-1	-1	-1	1	1	-1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	-1	1
26	1	-1	-1	1	1	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
27	-1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
28	1	1	-1	1	1	-1	-1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
29	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
30	1	-1	1	1	1	-1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1
31	-1	1	1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	-1	1	1
32	1	1	1	1	1	-1	1	-1	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1

Table 6.22: The last 32 rows of the 2_V^{8-2} design and the preferred block from section 4.3.3.

Row	A	B	C	D	E	F	G	H	$b1$	$b2$	$b3$	$b4$	$b5$	$b6$	$b7$	$b8$	$b9$	$b10$	$b11$	$b12$
33	-1	-1	-1	-1	-1	1	1	-1	-1	1	-1	1	-1	1	-1	1	1	1	1	1
34	1	-1	-1	-1	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1
35	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	1	-1	-1	1	-1	-1	1
36	1	1	-1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	1	-1	1	1	-1	-1
37	-1	-1	1	-1	-1	1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1
38	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	-1	1	1	-1	1	-1	1	-1
39	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
40	1	1	1	-1	-1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
41	-1	-1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	1	1	1
42	1	-1	-1	1	-1	1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	1
43	-1	1	-1	1	-1	1	1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
44	1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	-1	1	-1	1	1	1	-1	-1
45	-1	-1	1	1	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
46	1	-1	1	1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1
47	-1	1	1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
48	1	1	1	1	-1	1	1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1
49	-1	-1	-1	-1	1	1	1	1	1	-1	1	-1	-1	1	-1	1	-1	-1	1	1
50	1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1
51	-1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
52	1	1	-1	-1	1	1	1	1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1	-1
53	-1	-1	1	-1	1	1	-1	1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
54	1	-1	1	-1	1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
55	-1	1	1	-1	1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	-1	1
56	1	1	1	-1	1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
57	-1	-1	-1	1	1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	-1	1	1
58	1	-1	-1	1	1	1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
59	-1	1	-1	1	1	1	1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	-1
60	1	1	-1	1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	-1	-1
61	-1	-1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1
62	1	-1	1	1	1	1	-1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
63	-1	1	1	1	1	1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1
64	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.23: The first 32 rows of the columns used to define the preferred division into four blocks for the 2_V^{8-2} design in sections 4.3.3.3 and 4.3.3.4.

Row	13	14	15	16	26	33	36	37	40	47	48	59	60
1	-1	1	1	-1	1	1	1	-1	-1	1	-1	1	-1
2	1	1	1	1	-1	1	-1	1	-1	1	1	-1	-1
3	-1	-1	-1	-1	1	-1	-1	-1	-1	1	1	-1	-1
4	1	-1	-1	1	-1	1	-1	-1	1	-1	1	-1	1
5	-1	1	1	-1	1	-1	-1	1	1	-1	1	-1	1
6	1	1	1	1	-1	-1	1	-1	1	-1	-1	1	1
7	-1	-1	-1	-1	1	1	1	1	1	-1	-1	1	1
8	1	-1	-1	1	-1	-1	1	1	-1	1	-1	1	-1
9	1	-1	-1	1	1	1	-1	-1	1	-1	1	1	-1
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
11	1	1	1	1	1	1	-1	1	-1	1	1	1	1
12	-1	1	1	-1	-1	1	1	-1	-1	1	-1	-1	1
13	1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	1
14	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
15	1	1	1	1	1	-1	1	-1	1	-1	-1	-1	-1
16	-1	1	1	-1	-1	-1	-1	1	1	-1	1	1	-1
17	1	-1	1	-1	-1	-1	-1	1	1	-1	1	-1	1
18	1	1	-1	-1	1	-1	1	-1	1	1	1	1	1
19	-1	-1	1	1	-1	1	1	1	1	1	1	1	1
20	-1	1	-1	1	1	-1	1	1	-1	1	-1	1	-1
21	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	-1
22	1	1	-1	-1	1	1	-1	1	-1	-1	-1	-1	-1
23	-1	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
24	-1	1	-1	1	1	1	-1	-1	1	-1	1	-1	1
25	-1	1	-1	1	-1	-1	1	1	-1	1	-1	-1	1
26	-1	-1	1	1	1	1	1	1	1	1	1	-1	-1
27	1	1	-1	-1	-1	-1	1	-1	1	1	1	-1	-1
28	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	-1
29	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1
30	-1	-1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
31	1	1	-1	-1	-1	1	-1	1	-1	-1	-1	1	1
32	1	-1	1	-1	1	1	1	-1	-1	1	-1	-1	1

Table 6.24: The last 32 rows of the columns used to define the preferred division into four blocks for the 2_V^{8-2} design in sections 4.3.3.3 and 4.3.3.4.

Row	13	14	15	16	26	33	36	37	40	47	48	59	60
33	1	1	-1	-1	1	1	1	1	1	-1	-1	-1	-1
34	1	-1	1	-1	-1	-1	1	1	-1	-1	1	-1	1
35	-1	1	-1	1	1	-1	-1	1	1	1	-1	1	-1
36	-1	-1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1
37	1	1	-1	-1	1	-1	-1	-1	-1	1	1	1	1
38	1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	-1
39	-1	1	-1	1	1	1	1	-1	-1	-1	1	-1	1
40	-1	-1	1	1	-1	1	-1	1	-1	1	1	1	1
41	-1	-1	1	1	1	-1	1	-1	1	-1	-1	1	1
42	-1	1	-1	1	-1	-1	-1	1	1	1	-1	-1	1
43	1	-1	1	-1	1	-1	1	1	-1	-1	1	1	-1
44	1	1	-1	-1	-1	1	1	1	1	-1	-1	1	1
45	-1	-1	1	1	1	1	-1	1	-1	1	1	-1	-1
46	-1	1	-1	1	-1	1	1	-1	-1	-1	1	1	-1
47	1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1
48	1	1	-1	-1	-1	-1	-1	-1	-1	1	1	-1	-1
49	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	1
50	-1	1	1	-1	1	1	-1	-1	1	1	-1	1	-1
51	1	-1	-1	1	-1	1	1	-1	-1	-1	1	-1	1
52	-1	-1	-1	-1	1	1	-1	1	-1	-1	-1	1	1
53	1	1	1	1	-1	1	1	1	1	1	1	-1	-1
54	-1	1	1	-1	1	-1	1	1	-1	-1	1	-1	1
55	1	-1	-1	1	-1	-1	-1	1	1	1	-1	1	-1
56	-1	-1	-1	-1	1	-1	1	-1	1	1	1	-1	-1
57	-1	-1	-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1
58	1	-1	-1	1	1	1	1	-1	-1	-1	1	1	-1
59	-1	1	1	-1	-1	1	-1	-1	1	1	-1	-1	1
60	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
61	-1	-1	-1	-1	-1	-1	1	-1	1	1	1	1	1
62	1	-1	-1	1	1	-1	-1	1	1	1	-1	-1	1
63	-1	1	1	-1	-1	-1	1	1	-1	-1	1	1	-1
64	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.25: The first 32 rows of the columns used to define the preferred division into eight blocks for the 2_V^{8-2} design in sections 4.3.3.5 and 4.3.3.6.

Row	18	20	26	28	51	59
1	-1	1	1	-1	-1	1
2	1	-1	1	-1	1	1
3	-1	-1	1	1	-1	1
4	1	-1	1	-1	-1	-1
5	1	1	-1	-1	1	-1
6	1	1	1	1	-1	-1
7	1	-1	-1	1	1	-1
8	1	1	1	1	1	1
9	1	1	1	1	-1	-1
10	1	-1	-1	1	-1	1
11	1	1	1	1	1	1
12	1	1	-1	-1	-1	1
13	1	-1	1	-1	1	1
14	-1	-1	1	1	1	-1
15	1	-1	1	-1	-1	-1
16	-1	1	1	-1	1	-1
17	-1	1	-1	1	-1	-1
18	1	-1	-1	1	1	-1
19	-1	1	-1	1	1	1
20	1	1	-1	-1	1	-1
21	-1	-1	-1	-1	1	1
22	-1	-1	1	1	-1	1
23	-1	-1	-1	-1	-1	-1
24	-1	1	1	-1	-1	1
25	-1	1	1	-1	1	-1
26	-1	-1	-1	-1	1	1
27	-1	-1	1	1	1	-1
28	-1	-1	-1	-1	-1	-1
29	1	1	-1	-1	-1	1
30	-1	1	-1	1	-1	-1
31	1	-1	-1	1	-1	1
32	-1	1	-1	1	1	1

Table 6.26: The last 32 rows of the columns used to define the preferred division into eight blocks for the 2_V^{8-2} design in sections 4.3.3.3 and 4.3.3.4.

Row	18	20	26	28	51	59
33	-1	-1	1	1	1	-1
34	-1	-1	-1	-1	1	1
35	-1	1	1	-1	1	-1
36	-1	-1	-1	-1	-1	-1
37	1	-1	-1	1	-1	1
38	-1	1	-1	1	-1	-1
39	1	1	-1	-1	-1	1
40	-1	1	-1	1	1	1
41	-1	1	-1	1	-1	-1
42	1	1	-1	-1	1	-1
43	-1	1	-1	1	1	1
44	1	-1	-1	1	1	-1
45	-1	-1	-1	-1	1	1
46	-1	1	1	-1	-1	1
47	-1	-1	-1	-1	-1	-1
48	-1	-1	1	1	-1	1
49	1	1	1	1	-1	-1
50	1	1	-1	-1	-1	1
51	1	1	1	1	1	1
52	1	-1	-1	1	-1	1
53	1	-1	1	-1	1	1
54	-1	1	1	-1	1	-1
55	1	-1	1	-1	-1	-1
56	-1	-1	1	1	1	-1
57	-1	-1	1	1	-1	1
58	1	-1	1	-1	1	1
59	-1	1	1	-1	-1	1
60	1	-1	1	-1	-1	-1
61	1	-1	-1	1	1	-1
62	1	1	1	1	-1	-1
63	1	1	-1	-1	1	-1
64	1	1	1	1	1	1

Appendix B: R code

Code for section 4.1.1: Blocking a 2_{IV}^{8-4} design using MIP

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=2*factors)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  int=factors-1
  combins=combn(factors,int)
  for(j in 1:ncol(combins)){
design[,(4+j)]=design[,combins[1,j]]*design[,combins[2,j]]*
design[,combins[3,j]]
  }
  design
}

#fac=#number of factors in design
fac=4
#fac2=#number of factors in design, w. combos
fac2=2*fac
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F","G","H")
```

```
#Function which takes in n, the number of columns,
#and m, the number of ones.
#It makes all possible rows with m ones and n-m zeroes
#It is later used to make all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

#Generate the blocks
perm=t(combinator(8, 4))
perm1=2*perm[,1:35]-1

#num=number of possible divisions into two blocks
num=ncol(combn(m,(m/2)))/2

#Generate all possible blocks
allblocks=matrix(data=NA,nrow=n,ncol=num)
 #colnames(blocks)<-cbind("b1","b2","b3")
 for(i in 1:num){
   for(j in 1:m){
       allblocks[j,i]=perm1[j,i]
     }
   for(k in (m+1):n){
     allblocks[k,i]=allblocks[n-k+1,i]
   }
 }

#Function which generates all possible interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  #Make column for constant
  # inter=t(t(rep(1,nrow(mat))))
  resc=matrix(data=NA,nrow=nrow(mat),ncol=92)
  colnam=numeric(92)
  for(i in 1:ncol(mat)){
```

```
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+8]=resc[,interact2[1,j]]*resc[,interact2[2,j]]
    colnam[j+8]=(paste(colnam[interact2[1,j]],colnam[interact2[2,
        j]],
    collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+8+ncol(interact2)]=resc[,interact3[1,j]]*
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+8+ncol(interact2)]=(paste(colnam[interact3[1,j]],
    colnam[interact3[2,j]],colnam[interact3[3,j]], collapse = '')
        )
  }
  colnames(resc)=colnam
  resc
}


#test: All possible interactions from the design
test=Generator2(design)
#test2=cbind(test,allblocks)
#testinter: The two-factor interactions
testinter=test[,9:36]
#unitestinter: The unique two-factor interactions
unitestinter=unique(testinter,MARGIN=2)
#Indexes of the blocks which are equal to two-factor interactions
badindex=which(duplicated(cbind(unitestinter,allblocks),MARGIN=2)
    )
-ncol(unitestinter)

#Removes the blocks found to be equal with two-factor
    interactions
goodblocks=allblocks[,-badindex]

#Combins: Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(fac2,int)
```

```
#Generate design matrix identity column and all the interactions
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  #The two- and three-factor interactions
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  #Make column for constant
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(7)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+4]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]],
    collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]
    *res1[,interact3[3,j]])
    colnam[j+4+ncol(interact2)]=(paste(colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],colnam[1+interact3[3,j]], collapse =
        ''))
  }
  colnames(res)=colnam
  res
}

#Function to calculate Ds-efficiency when there is one block
Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
```

```
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Does different blocks yield different Ds
#for the different combinations of active factors?
#num: The number of blocks tested
num=ncol(goodblocks)
#results1: vector with all Ds-efficiences obtained
#results1=numeric(num*ncol(combins))
results1=matrix(nrow = ncol(combins),ncol=num)
#Iterating over combinations
for(i in 1:ncol(combins)){
  #Iterating over blocks
  for(j in 1:num){
    matrise=cbind(combGenerator(design,combins[,i]),goodblocks[,j
        ])
    results1[i,j]=Ds(matrise)
  }
}
min(results1)
max(results1)

#Which unique Ds-values are obtained?
uni=unique(as.vector(results1))
#Print the values and how many blocks obtained
#them for each combination
for(i in 1:length(uni)){
  print(uni[i])
print(unique((rowSums(abs(results1-uni[i])<0.000001))))
}

#Finding the Ds-values for each block, testing all combinations
#results2=numeric(num*ncol(combins))
results2=matrix(nrow = num,ncol=ncol(combins))
#Iterating over combinations
for(i in 1:num){
  #Iterating over blocks
  for(j in 1:ncol(combins)){
```

```
    matrise=cbind(combGenerator(design,combins[,j]),goodblocks[,i
        ])
    results2[i,j]=Ds(matrise)
    }
  }
min(results2)
max(results2)


#Print the values and how many combinations
#for which each block obtained them
for(i in 1:length(uni)){
  print(uni[i])
  print(unique((rowSums(abs(results2-uni[i])<0.000001))))
}

#Print the unique average Ds-values
print(unique(rowMeans(results2)))

#The preferred block
prefblock=goodblocks[,1]

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,1]),prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0917=cbind(combGenerator(design,combins[,2]),prefblock)
diagonal2=diag(solve(t(matrise0917)%*%matrise0917))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.1.2.1: Dividing a 2_V^{5-1} design into two blocks using HM, three active factors

```
#The factors in the design
A=c(-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1)
P2=c(1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1)
C=c(-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1)
D=c(-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1)
H=c(-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1)

#fac=#number of factors in design
fac=5
#n=number of rows in total design
n=16
design=cbind(A,P2,C,D,H)
colnames(design)<-cbind("A","P2","C","D","H")

#The blocks suggested in the article
J2=c(1,-1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1)
K2=c(-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1)
L2=c(-1,-1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,-1,-1)
M2=c(-1,-1,-1,1,-1,1,1,1,1,1,1,-1,1,-1,-1,-1)

originalblocks=unique(cbind(J2,K2,L2,M2),MARGIN=2)
#Checking that they are valid blocks
colSums(originalblocks)


#Function which takes in n, the number of columns,
#and m, the number of ones.
#It makes all possible rows with m ones and n-m zeroes
#It is later used to make all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

#Make the blocks
perm=t(combinator(16, 8))
perm1=2*perm-1
blocks=perm1
```

```
#makes all effects up to three-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=
  (ncol(mat)+ncol(interact2)+ncol(interact3)))
  colnam=numeric((ncol(mat)+ncol(interact2)+ncol(interact3)))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=
    (paste(colnam[interact3[1,j]],
    colnam[interact3[2,j]],colnam[interact3[3,j]], collapse = '')
        )
  }
  colnames(resc)=colnam
  resc
}

#Make these, and use to check for orthogonality
test=Generator2(design)
sums=numeric(ncol(blocks))

#Choose the blocks which are orthogonal on the design columns
#And have the least amount of partial confounding
for(r in 1:ncol(blocks)){
  sums[r]=sum(t(blocks[,r]%*%design==0))
}
indeks=which((abs(sums-5)<0.00001))
blocks=blocks[,c(indeks)]
```

```
testinter=test[,6:15]
testsum=t(testinter)%*%blocks
totalsum=colSums(abs(testsum)==8)
indekser=which(totalsum==4)
blocks=blocks[,c(indekser)]

#Remove blocks that are equal (signs switched)
nullern=numeric(60)
for(t in 1:(0.5*ncol(blocks))){
  lookat=blocks[,t]
  rest=blocks[,(t+1):ncol(blocks)]
  #  print(lookat)
  rest=rest+lookat
  if(any(colSums(abs(rest))==0)==TRUE){
    nullern[t]=(t+which(colSums(abs(rest))==0))
  }
}
blocks=blocks[,-nullern]

#new number of blocks
num=ncol(blocks)
#Function to make the design matrix
combGeneratorBasic=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(7)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+length(interest)+1]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
```

```
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=    (paste(
        colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}


#Vector with interesting combinations
combins=combn(5,3)
results=matrix(nrow = num,ncol = ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterer over combinations
  for(j in 1:ncol(combins)){
    matrise=cbind(combGeneratorBasic(design,
    combins[,j]),blocks[,i])
   results[i,j]=Ds(matrise)
  }
}


#Which unique Ds-values are obtained?
uni=unique(as.vector(results))
#Print the values and how many blocks obtained them for each
    combination
```

```
for(i in 1:length(uni)){
  print(uni[i])
  print(unique((colSums(abs(results-uni[i])<0.000001))))
}

#Print the values and how many combinations for which
#each block obtained them
for(i in 1:length(uni)){
  print(uni[i])
  print(unique((rowSums(abs(results-uni[i])<0.000001))))
}

#Print the unique average Ds-values
print(unique(rowMeans(results)))

#The preferred block
prefblock=blocks[,2]

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGeneratorBasic(design,combins[,1]),prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0917=cbind(combGeneratorBasic(design,combins[,2]),
    prefblock)
diagonal2=diag(solve(t(matrise0917)%*%matrise0917))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))

#Are the original blocks preserved?
which(apply(blocks, 2, identical, originalblocks[,1]))
which(apply(blocks, 2, identical, -originalblocks[,2]))
which(apply(blocks, 2, identical, -originalblocks[,3]))
which(apply(blocks, 2, identical, -originalblocks[,4]))
```

```
#Check pref block for 4 active factors; which is equal?
which(apply(blocks, 2, identical, c(1,-1, 1, 1,-1,-1, 1,-1,
-1,-1,-1, 1,-1, 1, 1, 1)))
#Find the factors for which Ds=1
results[28,]
cbind(combGeneratorBasic(design,combins[,5]),blocks[,28])
cbind(combGeneratorBasic(design,combins[,9]),blocks[,28])
```

Code for section 4.1.2.2 and 4.1.2.3: Dividing a 2_V^{5-1} design into two blocks using HM, four active factors

```
#The factors in the design
A=c(-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1)
P2=c(1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1)
C=c(-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1)
D=c(-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1)
H=c(-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1)

#fac=#number of factors in design
fac=5
#n=number of rows in total design
n=16
design=cbind(A,P2,C,D,H)
colnames(design)<-cbind("A","P2","C","D","H")

#The blocks suggested in the article
J2=c(1,-1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1)
K2=c(-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1)
L2=c(-1,-1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,-1,-1)
M2=c(-1,-1,-1,1,-1,1,1,1,1,1,1,-1,1,-1,-1,-1)

originalblocks=unique(cbind(J2,K2,L2,M2),MARGIN=2)
#Checking that they are valid blocks
colSums(originalblocks)

#The different combinations of four active factors
combins=combn(5,4)

#Function which takes in n, the number of columns, and m,
#the number of ones.
#It makes all possible rows with m ones and n-m zeroes
```

```
#It is later used to make all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

#Makes the blocks
perm=t(combinator(16, 8))
perm1=2*perm-1
blocks=perm1

#Makes all possible effects up to three-factor effects
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)+ncol(interact3)))
  colnam=numeric((ncol(mat)+ncol(interact2)+ncol(interact3)))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=(paste(colnam[interact3
        [1,j]],
    colnam[interact3[2,j]],colnam[interact3[3,j]], collapse = '')
        )
  }
  colnames(resc)=colnam
  resc
```

```
}

#Check orthogonality
test=Generator2(design)
sums=numeric(ncol(blocks))
for(r in 1:ncol(blocks)){
  sums[r]=sum(t(blocks[,r]%*%design==0))
}
indeks=which((abs(sums-5)<0.00001))
blocks=blocks[,c(indeks)]
#Choose to ones with the least partial confounding
testinter=test[,6:15]
testsum=t(testinter)%*%blocks
totalsum=colSums(abs(testsum)==8)
indekser=which(totalsum==4)
blocks=blocks[,c(indekser)]

#Fjerner blokker som er like, bare at alle fortegn er motsatt
nullern=numeric(60)
for(t in 1:(0.5*ncol(blocks))){
  lookat=blocks[,t]
  rest=blocks[,(t+1):ncol(blocks)]
  rest=rest+lookat
if(any(colSums(abs(rest))==0)==TRUE){
  nullern[t]=(t+which(colSums(abs(rest))==0))
}
}
blocks=blocks[,-nullern]

#Number of blocks
num=ncol(blocks)


#Makes design matrix
combGenerator=function(mat,interest,combfac){
  interact2=combn(length(interest),2)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(7)
  colnam[1]="K"
```

```
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  if(combfac[1]>0){
  res=res[,-(1+length(interest)+combfac)]
  colnam=colnam[-(1+length(interest)+combfac)]
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Combfac: Which two-factor interactions should be removed?
combfac=combn(6,1)

#Testing how it goes for all combinations of 4 out of 5 factors,
#estimating all two-factor interactions
results=matrix(nrow=num,ncol=ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations of four active factors
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j],0),blocks[,i])
    results[i,j]=Ds(matrise)
```

```
    }
  }

unique(as.vector(results))
#How many are zero for each block?
rowSums(results==0)
#How many are 0.9389 for each block?
rowSums(results>0.9389)

#Testing how it goes for all combinations of 4 out of 5 factors,
#estimating 5 two-factor interactions
results3t=numeric(num*ncol(combins))
rest<- rep(0, num)
k=0
#res0[j,i] is two if combination j (f.ex. ACDH) gives DS=0
#for 2 out of six possible combinations of
#5 two-factor interactions for block i
#The column sum of res0 gives the number of combinations with Ds
    =0
#for this block, for all combinatons of factors
res0t=matrix(data=0,nrow=ncol(combins),ncol=num)
res87t=matrix(data=0,nrow=ncol(combins),ncol=num)
res93t=matrix(data=0,nrow=ncol(combins),ncol=num)
res97t=matrix(data=0,nrow=ncol(combins),ncol=num)
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations of four active factors
  for(j in 1:ncol(combins)){
    #Iterating over combinations of two-factor interactions
        included
  for(f in 1:ncol(combfac)){
    k=k+1
    matriset=cbind(combGenerator(design,combins[,j],combfac[,f]),
    blocks[,i])
    results3t[k]=Ds(matriset)
    if((abs(results3t[k]-0)<0.0000001)==TRUE){
      res0t[j,i]=res0t[j,i]+1
    }
    if((abs(results3t[k]-0.8705506)<0.0000001)==TRUE){
      res87t[j,i]=res87t[j,i]+1
    }
```

```
    if((abs(results3t[k]-0.9716417)<0.0000001)==TRUE){
      res97t[j,i]=res97t[j,i]+1
    }
    if((abs(results3t[k]-0.9330330)<0.0000001)==TRUE){
      res93t[j,i]=res93t[j,i]+1
    }
  }
  }
}
#Observe: All get trouble for one combination of four active
    factors,
#then 2 of 6 combinations of 5 two-factor interactions
#cannot be estimated
colSums(res0t)
colSums(res0t==2)
#For the combos with Ds=0 for two, the other 4 yield Ds=0.87:
colSums(res0t+res87t)

#make result for thesis with the example 0.5(AD+CD+AP-CP)
blokka=0.5*(design[,1]*design[,4]+design[,3]*design[,4]+
design[,1]*design[,2]-
design[,2]*design[,3])
results3t2=numeric(length(combfac)*ncol(combins))
rest2<- rep(0, num)
k=0
restm=matrix(data=0,nrow=ncol(combins),ncol=ncol(combfac))
  for(j in 1:ncol(combins)){
    for(f in 1:ncol(combfac)){
      k=k+1
      matriset2=cbind(combGenerator(design,combins[,j],
      combfac[,f]),blokka)
      results3t2[k]=Ds(matriset2)
      restm[j,f]=Ds(matriset2)
    }
  }

print(restm)

#Test for only three present two-factor interactions
results3tt=numeric(num*ncol(combins))
restt<- rep(0, num)
```

```
k=0
combfac2=combn(6,3)
#res0[j,i] is two if combination j (f.ex. ACDH) gives DS=0 for 2
    out
#of six possible combinations of 5 two-factor interactions for
    block i
#The column sum of res0 gives the number of combinations with Ds
    =0 for
#this block, for all combinatons of factors
res1tt=matrix(data=0,nrow=ncol(combins),ncol=num)
res96tt=matrix(data=0,nrow=ncol(combins),ncol=num)
res84tt=matrix(data=0,nrow=ncol(combins),ncol=num)
res92tt=matrix(data=0,nrow=ncol(combins),ncol=num)
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations of four active factors
  for(j in 1:ncol(combins)){
    #Iterating over combinations of two-factor interactions
        included
    for(f in 1:ncol(combfac2)){
      k=k+1
      matrisett=cbind(combGenerator(design,combins[,j],
      combfac2[,f]),blocks[,i])
      results3tt[k]=Ds(matrisett)
      if((abs(results3tt[k]-1)<0.0000001)==TRUE){
        res1tt[j,i]=res1tt[j,i]+1
      }
      if((abs(results3tt[k]-0.9646786)<0.0000001)==TRUE){
        res96tt[j,i]=res96tt[j,i]+1
      }
      if((abs(results3tt[k]-0.8408964)<0.0000001)==TRUE){
        res84tt[j,i]=res84tt[j,i]+1
      }
      if((abs(results3tt[k]-0.9170040)<0.0000001)==TRUE){
        res92tt[j,i]=res92tt[j,i]+1
      }
    }
  }
}

colSums(res84tt)
```

```
colSums(res1tt)
colSums(res96tt)
colSums(res92tt)
#Average Ds
(0.9170040*28+0.8408964*4+52*0.9646786+16*1.0000000)/100

#Example for thesis of Ds=1 combinations for the preferred block
cbind(combGenerator(design,combins[,2],combfac2[,2]),blokka)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,2],combfac2[,2]),
    blokka)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrisemin=cbind(combGenerator(design,combins[,1],combfac2[,3]),
    blokka)
diagonal2=diag(solve(t(matrisemin)%*%matrisemin))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))

#Are the original blocks preserved?
which(apply(blocks, 2, identical, originalblocks[,1]))
which(apply(blocks, 2, identical, -originalblocks[,2]))
which(apply(blocks, 2, identical, -originalblocks[,3]))
which(apply(blocks, 2, identical, -originalblocks[,4]))
```

Code for section 4.2.1: Dividing a 2_{IV}^{16-11} design into two blocks using the blocking of the 2^{8-4} design

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=2*factors)
  for(i in 1:(factors)){
    vect=numeric(2^i)
```

```
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  int=factors-1
  combins=combn(factors,int)
  for(j in 1:ncol(combins)){
    design[,(4+j)]=design[,combins[1,j]]*design[,combins[2,j]]*
    design[,combins[3,j]]
    #  colnam=colnames(mat)[interact]
  }
  design
}

#fac=#number of factors in design
fac=4
#fac2=#number of factors in design, w. combos
fac2=2*fac
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F","G","H")

#Function which takes in n, the number of columns, and m, the
    number of ones.
#It makes all possible rows with m ones and n-m zeroes
#It is later used to make all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

#Generate the blocks
perm=t(combinator(8, 4))
perm1=2*perm[,1:35]-1
```

```
#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Generate all possible blocks
 allblocks=matrix(data=NA,nrow=n,ncol=num)
 for(i in 1:num){
   for(j in 1:m){
       allblocks[j,i]=perm1[j,i]
     }
   for(k in (m+1):n){
     allblocks[k,i]=allblocks[n-k+1,i]
   }
 }

 #Remove the blocks equal to two-factor interactions,
 #as found in the 16-run script
goodblocks=allblocks[,-c(1,10,15,21,24,28,29)]

#Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(fac2,int)

#Generate design matrix
#Include up to three-factor interactions
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  #Make column for constant
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(7)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
```

```
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+4]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]],
    collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+4+ncol(interact2)]=(paste(colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],colnam[1+interact3[3,j]], collapse =
        '' ))
  }
  colnames(res)=colnam
  res
}


Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#combin2: Which three factors are active
combin2=combn(16,3)
num=ncol(goodblocks)
results3=matrix(data=NA,nrow=num,ncol=ncol(combin2))
#Iterating over blocks
for(i in 1:num){
  #Iterer over combinations
  for(j in 1:ncol(combin2)){
    B1=design[which(goodblocks[,i]==1),]
    B2=design[which(goodblocks[,i]==-1),]
    #Sjekk en kombinasjon av B1B1 osv at a time,
    #by commenting out the other
   stormatrise=rbind(cbind(B1,B1),cbind(B1,-B1),
```

```
   cbind(B2,B2),cbind(B2,-B2))
  #  stormatrise=rbind(cbind(B1,B1),cbind(B2,B2),
  cbind(B1,-B1),cbind(B2,-B2))
  # stormatrise=rbind(cbind(B1,B1),cbind(B2,-B2),
  cbind(B1,-B1),cbind(B2,B2))
   colnames(stormatrise)<-cbind("A","B","C","D","E","F","G",
   "H","I","J","K","L","M","N","O","P")
    blokk=c(rep(1,16),rep(-1,16))
    StorDs=cbind(combGenerator(stormatrise,combin2[,j]),blokk)
    results3[i,j]=Ds(StorDs)
  }
}

#Find which blocks yield Ds>0 for all combinations
indekser=which(rowSums(results3==0)==0)
#Find min and max for each block
mini=numeric(length(which(rowSums(results3==0)==0)))
maxi=numeric(length(which(rowSums(results3==0)==0)))
#Find frequencies of Ds-values for all blocks
a=rep(1,3)
for(r in 1:length(which(rowSums(results3==0)==0))){
  mini[r]=min(results3[indekser[r],])
  maxi[r]=max(results3[indekser[r],])
 a=rbind(a,(table(results3[indekser[r],])))
}
a=a[-1,]

#Use block nr. 28 as example in thesis:
B1=design[which(goodblocks[,28]==1),]
B2=design[which(goodblocks[,28]==-1),]
stormatrise=rbind(cbind(B1,B1),cbind(B1,-B1),
cbind(B2,B2),cbind(B2,-B2))
colnames(stormatrise)<-cbind("A","B","C","D","E","F","G","H","I",
"J","K","L","M","N","O","P")
blokk=c(rep(1,16),rep(-1,16))
#Look at results3[,28] to see that combin2[,1]
#yields Ds=0.917, combin2[,2] yields Ds=1
Ds1=cbind(combGenerator(stormatrise,combin2[,2]),blokk)
Ds0917=cbind(combGenerator(stormatrise,combin2[,1]),blokk)

#Find SD ratios: For best Ds
```

```
diagonal=diag(solve(t(Ds1)%*%Ds1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
diagonal2=diag(solve(t(Ds0917)%*%Ds0917))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.2: Dividing a 2_{IV}^{16-11} design into two blocks using MIP

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=16)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  int=fac-2
  combins=combn(factors,int)
  for(j in 1:ncol(combins)){
    design[,(5+j)]=design[,combins[1,j]]*design[,combins[2,j]]*
    design[,combins[3,j]]
  }
  design[,5+ncol(combins)+1]=design[,1]*design[,2]*design[,3]*
  design[,4]*design[,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
```

```
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F","G","H","I","J",
"K","L","M","N","O","P")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(fac2,int)

#Function that generates all main effects and two-factor
    interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
```

```
  resc=matrix(data=NA,nrow=nrow(mat),ncol=136)
  colnam=numeric(136)
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+16]=resc[,interact2[1,j]]*resc[,interact2[2,j]]
    colnam[j+16]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]],
    collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

#Remove bad blocks, see thesis for argumentation
doubletrouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==16)==32))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  #Make column for constant
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
```

```
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+4]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*
    res1[,interact3[2,j]]*res1[,interact3[3,j]])
    colnam[j+7]=
    (paste(colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Checking for each combination
num=ncol(allblocks)
results=matrix(data=NA,ncol=num,nrow=ncol(combins))
#Iterating over combinations
for(i in 1:ncol(combins)){
  #Iterer over blocks
  for(j in 1:num){
    matrise=cbind(combGenerator(design,combins[,i]),allblocks[,j
        ])
    results[i,j]=Ds(matrise)
  }
}
min(results)
```

```
max(results)

#Which unique Ds-values are obtained?
uni=unique(as.vector(results))
#Print the values and how many blocks obtained
#them for each combination
for(i in 1:length(uni)){
  print(uni[i])
  print(unique((rowSums(abs(results-uni[i])<0.000001))))
}

#Checking for each block
num=ncol(allblocks)
results3=matrix(data=NA,nrow=num,ncol=ncol(combins))
#Iterating over combinations
for(i in 1:num){
  #Iterer over blocks
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j]),allblocks[,i
        ])
    results3[i,j]=Ds(matrise)
  }
}
min(results3)
max(results3)


#Print the values and how many combinations
#for which each block obtained them
for(i in 1:length(uni)){
  print(uni[i])
  print(unique((rowSums(abs(results3-uni[i])<0.000001))))
}

#Print the unique average Ds-values
print(unique(rowMeans(results3)))


#The preferred block
prefblock=allblocks[,1]
#Find min and max Ds
```

```
resr=numeric(ncol(combins))
for(r in 1:ncol(combins)){
  matriset=cbind(combGenerator(design,combins[,r]),prefblock)
  resr[r]=Ds(matriset)
}

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,which.max(resr)]),
prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/
sqrt(min(diagonal[1:(len-1)])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0917=cbind(combGenerator(design
,combins[,which.min(resr)]),prefblock)
diagonal2=diag(solve(t(matrise0917)%*%matrise0917))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.3: Dividing a 2_{IV}^{16-11} design into four blocks using MIP

```
library(svMisc)
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=16)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  int=fac-2
  combins=combn(factors,int)
  for(j in 1:ncol(combins)){
```

```
    design[,(5+j)]=design[,combins[1,j]]*design[,combins[2,j]]*
    design[,combins[3,j]]
  }
  design[,5+ncol(combins)+1]=design[,1]*design[,2]*design[,3]*
  design[,4]*design[,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F","G","H",
"I","J","K","L","M","N","O","P")

#Make all possible blocks
whole=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
choice1=combn(15,3)
choice=combn(11,3)
second=matrix(data=NA,nrow=8,ncol=ncol(choice)*ncol(choice1))
for(i in 1:ncol(choice1)){
  first=c(1,1+choice1[,i])
  rest=setdiff(whole,first)
  for(j in 1:ncol(choice))
    second[,(i-1)*ncol(choice)+j]=c(first,c(rest[12],
    rest[c(choice[,j])]))
}
choice3=combn(7,3)
third=matrix(data=NA,nrow=12,ncol=ncol(choice)*
ncol(choice1)*ncol(choice3))
for(k in 1:ncol(second)){
  resten=setdiff(whole,second[,k])
  for(r in 1:ncol(choice3)){
    third[,(k-1)*ncol(choice3)+r]=c(second[,k],c(resten[8],
    resten[c(choice3[,r])]))
  }
```

```
}
final=matrix(data=NA,nrow = 16,ncol=ncol(third))
for(t in 1:ncol(third)){
  final[,t]=c(third[,t],setdiff(whole,third[,t]))
}


b1=matrix(data=NA,nrow=32,ncol(final))
b2=matrix(data=NA,nrow=32,ncol(final))
for(j in 1:ncol(final)){
  b1[final[1:4,j],j]=c(-1,-1,-1,-1)
  b2[final[1:4,j],j]=c(-1,-1,-1,-1)
  b1[final[5:8,j],j]=c(-1,-1,-1,-1)
  b2[final[5:8,j],j]=c(1,1,1,1)
  b1[final[9:12,j],j]=c(1,1,1,1)
  b2[final[9:12,j],j]=c(-1,-1,-1,-1)
  b1[final[13:16,j],j]=c(1,1,1,1)
  b2[final[13:16,j],j]=c(1,1,1,1)
  for(k in 17:32){
    b1[k,j]=b1[32-k+1,j]
    b2[k,j]=b2[32-k+1,j]
  }
}


#Let int be the number of factors of interest
int=3
combins=combn(ncol(design),int)

Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
```

```
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  #Make column for constant
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+4]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+7]=(paste(colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  #Note: s=b-3 as there are three columns corresponding to blocks
```

```
  s=b-3
  n=nrow(comb)
  detX=abs(det(t(comb)%*%comb))
  detXb=abs(det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)]))
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

blok1=b1
blok2=b2
blok12=blok1*blok2
num=ncol(b1)
nullindeks=numeric(num)
maxnull=numeric(num)
minnull=numeric(num)
aver=numeric(num)
for(i in 1:num){
  progress(i,max.value = num)
  results3=ncol(combins)
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j]),
    blok1[,i],blok2[,i],blok12[,i])
    results3[j]=Ds(matrise)
  }
  if((length(which(results3==0))==0)==TRUE){
    nullindeks[i]=1
    aver[i]=mean(results3)
    maxnull[i]=max(results3)
    minnull[i]=min(results3)
  }
}


save.image("d32alle4blokker.RData")
#Overview of the minimums obtained
sort(unique(minnull))

#Finding blocks with min Ds above 0
godeindekser=which(minnull>0)
#How many have min Ds above 0?
length(godeindekser)
```

```
#How many achieve the highest possible min Ds?
indekser=which(minnull>0.834)

#Overview of unique max Ds
sort(unique(maxnull))

#Overview of unique averages
sort(unique(aver))
#Averages obtained for the ones with the highest min Ds
sort(unique(aver[indekser]))

#How many with the highest min Ds also obtains the max Ds
#for one combination
length(indekser[which(maxnull[indekser]>0.99999999)])
besteindekser=indekser[which(maxnull[indekser]>0.99999999)]
#Finding the ones with highest min, highest max and highest
    average
besteindekser[which(aver[indekser[which(maxnull[indekser]>
0.99999999)]]>0.908206)]
#block 6064 is chosen as the preferred

#Finding Ds-values for the preferred block
bestres=numeric(ncol(combins))
for( r in 1:ncol(combins)){
  matrisei=cbind(combGenerator(design,combins[,r]),blok1[,6064],
  blok2[,6064],blok12[,6064])
  bestres[r]=Ds(matrisei)
}
#Frequency table
table(bestres)

#Examples with min and max Ds
Dsmax=cbind(combGenerator(design,combins[,which.max(bestres)]),
blok1[,6064],blok2[,6064],blok12[,6064])
Dsmin=cbind(combGenerator(design,combins[,which.min(bestres)]),
blok1[,6064],blok2[,6064],blok12[,6064])

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
diagonal=diag(solve(t(Dsmax)%*%Dsmax))
```

```
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/
sqrt(min(diagonal[1:(len-3)])))

#For one of the combinations yielding the lowest Ds
diagonal2=diag(solve(t(Dsmin)%*%Dsmin))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))
```

Code for section 4.2.4.1: Dividing a 2_{VI}^{6-1} design into two blocks using MIP, three active factors

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")
```

```
#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Make all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
```

```
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

#Makes design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}
```

```
Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Remove confounded and strongly partially confounded blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]

#Checking for each block
#num is now number of blocks
num=ncol(allblocks)
results33=matrix(data=NA,nrow = ncol(allblocks),ncol=ncol(combins
    ))
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j]),allblocks[,i
        ])
    results33[i,j]=Ds(matrise)
  }
}

#Which minimums and maximums are obtained?
unique(apply(results33, 1, FUN=min))
unique(apply(results33, 1, FUN=max))
#What are the different averages?
unique(rowMeans(results33))
```

```
goodmin=which(apply(results33, 1, FUN=min)>min(results33))
goodmax=which(apply(results33, 1, FUN=max)==max(results33))
goodmean=which(rowMeans(results33)>0.97159)
#None of the blocks with the best means have the best max
sum((duplicated(c(goodmean,goodmax))))
sum((duplicated(c(goodmin,goodmax))))


res=numeric(20)
#Number 12 is chosen as example to artice
prefblock=allblocks[,12]
for(r in 1:ncol(combins)){
  matriset=cbind(combGenerator(design,combins[,r]),prefblock)
  res[r]=(Ds(matriset))
}
#Find average
mean(res)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,which.max(res)]),
    prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/
sqrt(min(diagonal[1:(len-1)])))
print(sqrt(diagonal[len])/
sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(design,combins[,which.min(res)]),
    prefblock)
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/
sqrt(min(diagonal2[1:(len-1)])))
print(sqrt(diagonal2[len])/
sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.4.2: Dividing a 2^{6-1}_{VI} design into two blocks using MIP, four active factors

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1
```

```
allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=4
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]],collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


#Remove confounded and strongly partially confounded blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
```

```
Generator2(design))==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
```

```
  colnames(res)=colnam
  res
}


Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Checking for each block
#num is now number of blocks
num=ncol(allblocks)
results33=matrix(data=NA,nrow = ncol(allblocks),ncol=ncol(combins
    ))
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j]),allblocks[,i
        ])
    results33[i,j]=Ds(matrise)
  }
}

#Check frequencies for each combination
frek=table(results33[,1])
for(i in 2:15){
  frek=cbind(frek,table(results33[,i]))
}
#+1 because the first is not duplicated per definition
sum(duplicated(frek,MARGIN=2))+1-15

#Check frequencies for each block
frek2=table(results33[1,])
for(i in 2:num){
```

```
  frek2=cbind(frek2,table(results33[i,]))
}
sum(duplicated(frek2,MARGIN=2))+1-5040

#Which minimums and maximums are obtained?
unique(apply(results33, 1, FUN=min))
unique(apply(results33, 1, FUN=max))
#What are the different averages?
unique(rowMeans(results33))

#Example for article
res=numeric(15)
prefblock=allblocks[,1]
for(r in 1:ncol(combins)){
  matriset=cbind(combGenerator(design,combins[,r]),prefblock)
  res[r]=(Ds(matriset))
}

#Find average
mean(res)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,which.max(res)]),
    prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/
sqrt(min(diagonal[1:(len-1)])))
print(sqrt(diagonal[len])/
sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(design,combins[,which.min(res)]),
    prefblock)
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.4.4: Dividing a 2^{6-1}_{VI} design into two blocks using MIP, five active factors, estimating up to three-factor interactions

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
```

```
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=5
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


#Remove confounded and strongly partially confounded blocks
trouble=Generator2(design)
```

```
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]

#Make design matrix
combGenerator=function(mat,interest,combfac){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
```

```
  }
  #Removing the four-factor interactions not to be estimated
  res=res[,-(1+length(interest)+ncol(interact2)+
  ncol(interact3)+combfac)]
  colnam=colnam[-(1+length(interest)+ncol(interact2)+
  ncol(interact3)+combfac)]
  colnames(res)=colnam
  res
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}


#Combfac: Remove all four-factor interactions
combfac=combn(5,5)

num=ncol(allblocks)
meanres=matrix(data=NA,nrow=ncol(allblocks),
ncol=ncol(combfac)*ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterer over combinations
  for(j in 1:ncol(combins)){
    for(f in 1:ncol(combfac)){
    matrise=cbind(combGenerator(design,combins[,j],
    combfac[,f]),allblocks[,i])
    meanres[i,((j-1)*ncol(combfac)+f)]=Ds(matrise)
  }
  }
}
```

```
#How many blocks yield ds>0 for all combinations?
length(which(rowSums(meanres==0)==0))
unique(rowMeans(meanres[which(rowSums(meanres==0)==0),]))

mini=numeric(length(which(rowSums(meanres==0)==0)))
maxi=numeric(length(which(rowSums(meanres==0)==0)))
for(w in 1:length(which(rowSums(meanres==0)==0))){
  mini[w]=min(meanres[which(rowSums(meanres==0)==0)[w],])
  maxi[w]=max(meanres[which(rowSums(meanres==0)==0)[w],])
}

#Results for the blocks with the highest min Ds
bestmean=meanres[which(mini>0.94),]
#They all have the same mean
unique(rowMeans(bestmean))

#compare best max and max for max min Ds
max(maxi)
max(maxi[which(mini>0.94)])


#Check frequencies for each block
frek2=table(bestmean[1,])
for(i in 2:nrow(bestmean)){
  frek2=cbind(frek2,table(bestmean[i,]))
}
sum(duplicated(frek2,MARGIN=2))+1-nrow(bestmean)


#Choosing example block, block number 6 in allblocks
prefblock=allblocks[,6]
#Frequency table
table(meanres[6,])
#Finding the combos yielding the highest Ds
which(meanres[6,]>0.962)


#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(design,combins[,1],combfac[,1]),
```

```
    prefblock)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(design,combins[,5],combfac[,1]),
    prefblock)
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.4.5: Dividing a 2_{VI}^{6-1} design into two blocks using MIP, five active factors, estimating two four-factor interactions

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
```

```
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=5
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
```

```
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


#Remove confounded and strongly partially confounded blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]

#Make design matrix
combGenerator=function(mat,interest,combfac){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
```

```
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
  #Removing the four-factor interactions not to be estimated
  res=res[,-(1+length(interest)+ncol(interact2)+
  ncol(interact3)+combfac)]
  colnam=colnam[-(1+length(interest)++ncol(interact2)+
  ncol(interact3)+combfac)]
  colnames(res)=colnam
  res
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}


#Combfac: Removing 3 out of 5 four-factor interactions
combfac=combn(5,3)
```

```
num=ncol(allblocks)
meanres=matrix(data=NA,nrow=ncol(allblocks),ncol=ncol(combfac)*
ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    for(f in 1:ncol(combfac)){
    k=k+1
    matrise=cbind(combGenerator(design,combins[,j],combfac[,f]),
    allblocks[,i])
    meanres[i,((j-1)*ncol(combfac)+f)]=Ds(matrise)
  }
  }
}

#How many blocks handle all combos of two four-factor
    interactions?
length(which(rowSums(meanres==0)==0))
unique(rowMeans(meanres[which(rowSums(meanres==0)==0),]))

#Finding max and min values
mini=numeric(length(which(rowSums(meanres==0)==0)))
maxi=numeric(length(which(rowSums(meanres==0)==0)))
for(w in 1:length(which(rowSums(meanres==0)==0))){
  mini[w]=min(meanres[which(rowSums(meanres==0)==0)[w],])
  maxi[w]=max(meanres[which(rowSums(meanres==0)==0)[w],])
}

unique(mini)
unique(maxi)

bestindeks=which(rowSums(meanres==0)==0)
bestblocks=allblocks[,bestindeks]

#Choosing example block, block number 6 in allblocks
prefblock=bestblocks[,1]
#Frequency table
table(meanres[6,])
#Finding the combos yielding the highest Ds
```

```
which(meanres[6,]>0.9655)
#Ensuring results
matrise1=cbind(combGenerator(design,combins[,1],combfac[,10]),
    prefblock)
matrise2=cbind(combGenerator(design,combins[,2],combfac[,7]),
    prefblock)
matrise3=cbind(combGenerator(design,combins[,3],combfac[,7]),
    prefblock)
matrise4=cbind(combGenerator(design,combins[,6],combfac[,6]),
    prefblock)
Ds(matrise1)
Ds(matrise2)
Ds(matrise3)
Ds(matrise4)
#Print four-factor interactions yielding the highest Ds
print(matrise1[1,27:28])
print(matrise2[1,27:28])
print(matrise3[1,27:28])
print(matrise4[1,27:28])


#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(design,combins[,1],combfac[,1]),
    prefblock)
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.5.1: Dividing a 2_{VI}^{6-1} design into four blocks using MIP, three active factors

```
library(svMisc)
```

```
#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2


#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1
```

```
allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(6,3)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

#Remove bad blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==24)>0))]
```

```
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==32)>0))]

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
```

```
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Make all possible blocks
whole=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
choice1=combn(15,3)
choice=combn(11,3)
second=matrix(data=NA,nrow=8,ncol=ncol(choice)*ncol(choice1))
for(i in 1:ncol(choice1)){
  first=c(1,1+choice1[,i])
  rest=setdiff(whole,first)
  for(j in 1:ncol(choice))
    second[,(i-1)*ncol(choice)+j]=c(first,c(rest[12],
    rest[c(choice[,j])]))
}
choice3=combn(7,3)
third=matrix(data=NA,nrow=12,ncol=ncol(choice)*ncol(choice1)*
ncol(choice3))
for(k in 1:ncol(second)){
  resten=setdiff(whole,second[,k])
  for(r in 1:ncol(choice3)){
    third[,(k-1)*ncol(choice3)+r]=
    c(second[,k],c(resten[8],resten[c(choice3[,r])]))
  }
}
final=matrix(data=NA,nrow = 16,ncol=ncol(third))
for(t in 1:ncol(third)){
  final[,t]=c(third[,t],setdiff(whole,third[,t]))
}


b1=matrix(data=NA,nrow=32,ncol(final))
b2=matrix(data=NA,nrow=32,ncol(final))
for(j in 1:ncol(final)){
  b1[final[1:4,j],j]=c(-1,-1,-1,-1)
  b2[final[1:4,j],j]=c(-1,-1,-1,-1)
  b1[final[5:8,j],j]=c(-1,-1,-1,-1)
  b2[final[5:8,j],j]=c(1,1,1,1)
```

```
  b1[final[9:12,j],j]=c(1,1,1,1)
  b2[final[9:12,j],j]=c(-1,-1,-1,-1)
  b1[final[13:16,j],j]=c(1,1,1,1)
  b2[final[13:16,j],j]=c(1,1,1,1)
  for(k in 17:32){
    b1[k,j]=b1[32-k+1,j]
    b2[k,j]=b2[32-k+1,j]
  }
}

blok1=b1
blok2=b2
blok12=blok1*blok2
num=ncol(blok1)
allres=matrix(data=NA,nrow=num,ncol=ncol(combins))
nullindeks=numeric(num)
maxnull=numeric(num)
minnull=numeric(num)
aver=numeric(num)
k=0
  for(i in 1:num){
    results3=ncol(combins)
    for(j in 1:ncol(combins)){
     k=k+1
     progress(k,max.value = num*ncol(combins))
      matrise=cbind(combGenerator(design,combins[,j]),
      blok1[,i],blok2[,i],blok12[,i])
      allres[i,j]=Ds(matrise)
    }
  }

sort(unique(rowSums(allres[which(rowSums(allres==0)==0),])/20))
meanres=rowSums(allres[which(rowSums(allres==0)==0),])/20
#How many have Ds>0 for all combinations?
length(meanres)

mini=apply(allres,1,min)
#Which is the largest min obtained?
sort(unique(mini))
#How many obtained that?
meanresmini=rowSums(allres[which(mini>0.87),])/20
```

```
length(meanresmini)
#What is the max obtained among these, and in general?
maxi=apply(allres,1,max)
sort(unique(maxi))
maximini=apply(allres[which(mini>0.87),],1,max)
unique(maximini)
#What is the best and worst average obtained?
max(meanres)
min(meanres)

minimeanres=rowSums(allres[which(mini>0.87),])/20
unique(minimeanres)

#Example for thesis
resr=numeric(ncol(combins))
for(r in 1:ncol(combins)){
  matriset=cbind(combGenerator(design,combins[,r]),
  blok1[,16244],blok2[,16244],blok12[,16244])
  resr[r]=Ds(matriset)
}
table(resr)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
Dsmax=cbind(combGenerator(design,combins[,which.max(resr)]),
blok1[,16244],blok2[,16244],blok12[,16244])
diagonal=diag(solve(t(Dsmax)%*%Dsmax))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/
sqrt(min(diagonal[1:(len-3)])))
print(sqrt(max(diagonal[(len-3):len]))/
sqrt(min(diagonal[1:(len-3)])))

#For one of the combinations yielding the lowest Ds
Dsmin=cbind(combGenerator(design,combins[,which.min(resr)]),
blok1[,16244],blok2[,16244],blok12[,16244])
diagonal2=diag(solve(t(Dsmin)%*%Dsmin))
print(sqrt(max(diagonal2[1:(len-3)]))/
sqrt(min(diagonal2[1:(len-3)])))
print(sqrt(max(diagonal2[(len-3):len]))/
sqrt(min(diagonal2[1:(len-3)])))
```

```
save.image("testeAlle4blokker3ct.RData")
```

Code for section 4.2.5.1: Dividing a 2_{VI}^{6-1} design into four blocks using MIP, four active factors

```
library(svMisc)

#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2


#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
```

```
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}


#Vector with interesting combinations
#Let int be the number of factors of interest
int=4
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
```

```
  colnames(resc)=colnam
  resc
}

#Remove bad blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==32)>0))]

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
```

```
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
  colnames(res)=colnam
  res
}


Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Make all possible blocks
whole=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
choice1=combn(15,3)
choice=combn(11,3)
second=matrix(data=NA,nrow=8,ncol=ncol(choice)*ncol(choice1))
for(i in 1:ncol(choice1)){
  first=c(1,1+choice1[,i])
  rest=setdiff(whole,first)
  for(j in 1:ncol(choice))
    second[,(i-1)*ncol(choice)+j]=c(first,c(rest[12],
    rest[c(choice[,j])]))
}
choice3=combn(7,3)
third=matrix(data=NA,nrow=12,ncol=ncol(choice)*ncol(choice1)*
ncol(choice3))
for(k in 1:ncol(second)){
  resten=setdiff(whole,second[,k])
```

```
  for(r in 1:ncol(choice3)){
    third[,(k-1)*ncol(choice3)+r]=c(second[,k],c(resten[8],
    resten[c(choice3[,r])]))
  }
}
final=matrix(data=NA,nrow = 16,ncol=ncol(third))
for(t in 1:ncol(third)){
  final[,t]=c(third[,t],setdiff(whole,third[,t]))
}


b1=matrix(data=NA,nrow=32,ncol(final))
b2=matrix(data=NA,nrow=32,ncol(final))
for(j in 1:ncol(final)){
  b1[final[1:4,j],j]=c(-1,-1,-1,-1)
  b2[final[1:4,j],j]=c(-1,-1,-1,-1)
  b1[final[5:8,j],j]=c(-1,-1,-1,-1)
  b2[final[5:8,j],j]=c(1,1,1,1)
  b1[final[9:12,j],j]=c(1,1,1,1)
  b2[final[9:12,j],j]=c(-1,-1,-1,-1)
  b1[final[13:16,j],j]=c(1,1,1,1)
  b2[final[13:16,j],j]=c(1,1,1,1)
  for(k in 17:32){
    b1[k,j]=b1[32-k+1,j]
    b2[k,j]=b2[32-k+1,j]
  }
}



blok1=b1
blok2=b2
blok12=blok1*blok2
num=ncol(blok1)
allres=matrix(data=NA,nrow=num,ncol=ncol(combins))
nullindeks=numeric(num)
maxnull=numeric(num)
minnull=numeric(num)
aver=numeric(num)
k=0
  for(i in 1:num){
```

```
    results3=ncol(combins)
    for(j in 1:ncol(combins)){
     k=k+1
     progress(k,max.value = num*ncol(combins))
      matrise=cbind(combGenerator(design,combins[,j]),
      blok1[,i],blok2[,i],blok12[,i])
      allres[i,j]=Ds(matrise)
    }
  }


sort(unique(rowSums(allres[which(rowSums(allres==0)==0),])/15))
meanres=rowSums(allres[which(rowSums(allres==0)==0),])/15
#How many Ds>0 for all combs?
length(meanres)
#What are they?
sort(unique(meanres))
#look at all good results
allgoodres=allres[which(rowSums(allres==0)==0),]
#What are the max and mins for all blocks?
mini=apply(allgoodres,1,min)
unique(mini)
maxi=apply(allgoodres,1,max)
max(meanres)
unique(maxi)

#How many obtain the highest min?
length(which(mini>0.82))
#What averages did they obtain?
table(meanres[which(mini>0.82)])


#Example for thesis
resr=numeric(ncol(combins))
for(r in 1:ncol(combins)){
  matriset=cbind(combGenerator(design,combins[,r]),
  blok1[,5855],blok2[,5855],blok12[,5855])
  resr[r]=Ds(matriset)
}
table(resr)
```

```
#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
Dsmax=cbind(combGenerator(design,combins[,which.max(resr)]),
blok1[,5855],blok2[,5855],blok12[,5855])
diagonal=diag(solve(t(Dsmax)%*%Dsmax))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))

#For one of the combinations yielding the lowest Ds
Dsmin=cbind(combGenerator(design,combins[,which.min(resr)]),
blok1[,5855],blok2[,5855],blok12[,5855])
diagonal2=diag(solve(t(Dsmin)%*%Dsmin))
print(sqrt(max(diagonal2[1:(len-3)]))/
sqrt(min(diagonal2[1:(len-3)])))
print(sqrt(max(diagonal2[(len-3):len]))/
sqrt(min(diagonal2[1:(len-3)])))

save.image("testeAlle4blokker.RData")
```

Code for section 4.2.5.1: Dividing a 2_{VI}^{6-1} design into four blocks using MIP, five active factors

```
library(svMisc)

#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}
```

```
#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2


#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=5
```

```
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]],
    collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

#Remove bad blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==32)>0))]

#Make design matrix
combGenerator=function(mat,interest,combfac){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
```

```
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
  #Removing the four-factor interactions not to be estimated
  res=res[,-(1+length(interest)+ncol(interact2)+
  ncol(interact3)+combfac)]
  colnam=colnam[-(1+length(interest)+ncol(interact2)+
  ncol(interact3)+combfac)]
  colnames(res)=colnam
  res
  colnames(res)=colnam
  res
}


#Combfac: Remove all four-factor interactions
combfac=combn(5,5)
```

```
Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Make all possible blocks
whole=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
choice1=combn(15,3)
choice=combn(11,3)
second=matrix(data=NA,nrow=8,ncol=ncol(choice)*ncol(choice1))
for(i in 1:ncol(choice1)){
  first=c(1,1+choice1[,i])
  rest=setdiff(whole,first)
  for(j in 1:ncol(choice))
    second[,(i-1)*ncol(choice)+j]=c(first,c(rest[12],
    rest[c(choice[,j])]))
}
choice3=combn(7,3)
third=matrix(data=NA,nrow=12,ncol=ncol(choice)*
ncol(choice1)*ncol(choice3))
for(k in 1:ncol(second)){
  resten=setdiff(whole,second[,k])
  for(r in 1:ncol(choice3)){
    third[,(k-1)*ncol(choice3)+r]=c(second[,k],c(resten[8],
    resten[c(choice3[,r])]))
  }
}
final=matrix(data=NA,nrow = 16,ncol=ncol(third))
for(t in 1:ncol(third)){
  final[,t]=c(third[,t],setdiff(whole,third[,t]))
}


b1=matrix(data=NA,nrow=32,ncol(final))
b2=matrix(data=NA,nrow=32,ncol(final))
```

```
for(j in 1:ncol(final)){
  b1[final[1:4,j],j]=c(-1,-1,-1,-1)
  b2[final[1:4,j],j]=c(-1,-1,-1,-1)
  b1[final[5:8,j],j]=c(-1,-1,-1,-1)
  b2[final[5:8,j],j]=c(1,1,1,1)
  b1[final[9:12,j],j]=c(1,1,1,1)
  b2[final[9:12,j],j]=c(-1,-1,-1,-1)
  b1[final[13:16,j],j]=c(1,1,1,1)
  b2[final[13:16,j],j]=c(1,1,1,1)
  for(k in 17:32){
    b1[k,j]=b1[32-k+1,j]
    b2[k,j]=b2[32-k+1,j]
  }
}

blok1=b1
blok2=b2
blok12=blok1*blok2
num=ncol(blok1)
allres=matrix(data=NA,nrow=num,ncol=ncol(combins))
nullindeks=numeric(num)
maxnull=numeric(num)
minnull=numeric(num)
aver=numeric(num)
k=0
  for(i in 1:num){
    results3=ncol(combins)
    for(j in 1:ncol(combins)){
      for(f in 1:ncol(combfac)){
     k=k+1
     progress(k,max.value = num*ncol(combins))
      matrise=cbind(combGenerator(design,combins[,j],combfac[,f])
          ,
      blok1[,i],blok2[,i],blok12[,i])
      allres[i,j]=Ds(matrise)
    }
  }
  }

save.image("testeAlle4blokker5fak3int.RData")
```

```
sort(unique(rowSums(allres[which(rowSums(allres==0)==0),])/6))
goodres=allres[which(rowSums(allres==0)==0),]
meanres=rowSums(allres[which(rowSums(allres==0)==0),])/6
#How many have Ds>0 for all combinations?
length(meanres)

mini=apply(allres,1,min)
#Which is the largest min obtained?
sort(unique(mini))
#How many obtained that?
meanresmini=rowSums(allres[which(mini>0.86),])/6
length(meanresmini)
#What is the max obtained among these, and in general?
maxi=apply(goodres,1,max)
sort(unique(maxi))
maximini=apply(allres[which(mini>0.86),],1,max)
unique(maximini)
#Best results
bestres=allres[which(mini>0.86),]
#Are they all equal?
sum(duplicated(t(bestres),MARGIN=2))
#How are the results when the highest max is obtained?
apply(goodres[which(maxi>0.953),],1,table)

#Example for thesis
resr=numeric(ncol(combins))
for(r in 1:ncol(combins)){
  for(j in 1:ncol(combfac)){
  matriset=cbind(combGenerator(design,combins[,r],combfac[,j]),
  blok1[,13695],blok2[,13695],blok12[,13695])
  resr[r]=Ds(matriset)
}
}
table(resr)

#Finding the SD-ratios
#As all are equal, one is enough
Dsmax=cbind(combGenerator(design,combins[,which.max(resr)],
combfac[,1]),blok1[,13695],blok2[,13695],blok12[,13695])
diagonal=diag(solve(t(Dsmax)%*%Dsmax))
len=length(diagonal)
```

```
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))
```

Code for section 4.2.5.4: Utilising the division into two blocks for division into fours blocks, three active factors

```
library(svMisc)

#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
```

```
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1

allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=4
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]],collapse = ''))
  }
  colnames(resc)=colnam
```

```
  resc
}

#Remove bad blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==32)>0))]

#Int=number of factors of interest
#Combins: All combinations of them
int=3
combins=combn(6,int)

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
```

```
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Make all possible blocks based on one block
blokker4<-function(indeks){
  kombiner=combn(8,4)[,1:35]
  blokk1=matrix(nrow = 8,ncol=ncol(kombiner))
  for(i in 1:ncol(kombiner)){
    blokk1[1:4,i]=indeks[c(kombiner[,i])]
    blokk1[5:8,i]=setdiff(indeks,blokk1[1:4,i])
  }
  blokk1
}

blokker4_2<-function(blokk){
  test=which(blokk==-1)[1:8]
  test2=which(blokk==1)[1:8]
  bl1=blokker4(test)
  bl2=blokker4(test2)
  k=0
  bl3=matrix(data=NA,nrow = 16,ncol=ncol(bl1)*ncol(bl2))
  for( i in 1:ncol(bl1)){
    for(j in 1:ncol(bl2)){
      k=k+1
      bl3[,k]=rbind(bl1[,i],bl2[,j])
    }
```

```
  }

  b1=matrix(data=NA,nrow=32,ncol(bl3))
  b2=matrix(data=NA,nrow=32,ncol(bl3))
  for(j in 1:ncol(bl3)){
    b1[bl3[1:4,j],j]=c(-1,-1,-1,-1)
    b2[bl3[1:4,j],j]=c(-1,-1,-1,-1)
    b1[bl3[5:8,j],j]=c(-1,-1,-1,-1)
    b2[bl3[5:8,j],j]=c(1,1,1,1)
    b1[bl3[9:12,j],j]=c(1,1,1,1)
    b2[bl3[9:12,j],j]=c(-1,-1,-1,-1)
    b1[bl3[13:16,j],j]=c(1,1,1,1)
    b2[bl3[13:16,j],j]=c(1,1,1,1)
    for(k in 17:32){
      b1[k,j]=b1[32-k+1,j]
      b2[k,j]=b2[32-k+1,j]
    }
  }
  return(rbind(b1,b2))
}

numbr=ncol(allblocks)
notrouble=matrix(data=NA,nrow=numbr,ncol=1225)
meanres=matrix(data=NA,nrow=numbr,ncol=21)
meanres2=matrix(data=NA,nrow=numbr,ncol=21)
for(r in 1:ncol(allblocks)){
  progress(r,max.value =ncol(allblocks))
  blok1=blokker4_2(allblocks[,r])[1:32,]
  blok2=blokker4_2(allblocks[,r])[33:64,]
  blok12=blok1*blok2
  num=ncol(blok1)
  results3_2=matrix(data=NA,nrow =20,ncol=num)
  res=numeric(num)
  #Iterating over blocks
  for(i in 1:num){
    #Iterating over combinations
    for(j in 1:ncol(combins)){
      matrise=cbind(combGenerator(design,combins[,j]),blok1[,i],
      blok2[,i],blok12[,i])
      results3_2[j,i]=Ds(matrise)
    }
```

```
  }
  #meanres: store largest mean
  meanres[r,]=c((results3_2[,which(colSums(results3_2==0)==0)])
  [,which.max(colSums(results3_2[,which(colSums(results3_2==0)
      ==0)]))],
  which.max(colSums(results3_2[,which(colSums(results3_2==0)==0)
      ])))
  #meanres2: largest min
  meanres2[r,]=c((results3_2[,which(colSums(results3_2==0)==0)])
  [,which.max(apply(results3_2[,which(colSums(results3_2==0)==0)
      ],2,min))],
  which.max(apply(results3_2[,which(colSums(results3_2==0)==0)
      ],2,min)))
}

mini=numeric(nrow(meanres))
maxi=numeric(nrow(meanres))
mini2=numeric(nrow(meanres))
maxi2=numeric(nrow(meanres))
for(k in 1:nrow(meanres)){
  mini[k]=min(meanres[k,1:20])
  maxi[k]=max(meanres[k,1:20])
  mini2[k]=min(meanres2[k,1:20])
  maxi2[k]=max(meanres2[k,1:20])
}

#How many have each max and min?
table(maxi)
table(mini)
table(maxi2)
table(mini2)

#How are the mean values distributed?
means=rowMeans(meanres[,1:20])
table(means)
minmeans=rowMeans(meanres2[,1:20])
table(minmeans)
save.image("factorialDesign3fourblocksUsing2blocking3act.RData")
```

Code for section 4.2.5.4: Dividing a 2_{VI}^{6-1} design into four blocks using the division into two blocks, four active factors

```
library(svMisc)

#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=6)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  design[,6]=design[,1]*design[,2]*design[,3]*design[,4]*design
      [,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2


#Making all possible blocks
combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
```

```
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}


#Vector with interesting combinations
#Let int be the number of factors of interest
int=4
combins=combn(6,int)

#Make all effects up to two-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],colnam[
        interact2[2,j]],
    collapse = ''))
  }
  colnames(resc)=colnam
  resc
```

```
}

#Remove bad blocks
trouble=Generator2(design)
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==16)==4))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
trouble)==32)>0))]

#Int=number of factors of interest
#Combins: All combinations of them
int=4
combins=combn(6,int)

#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(16)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
```

```
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Make all possible blocks based on one block
blokker4<-function(indeks){
  kombiner=combn(8,4)[,1:35]
  blokk1=matrix(nrow = 8,ncol=ncol(kombiner))
  for(i in 1:ncol(kombiner)){
    blokk1[1:4,i]=indeks[c(kombiner[,i])]
    blokk1[5:8,i]=setdiff(indeks,blokk1[1:4,i])
  }
  blokk1
}

blokker4_2<-function(blokk){
test=which(blokk==-1)[1:8]
test2=which(blokk==1)[1:8]
bl1=blokker4(test)
```

```
bl2=blokker4(test2)
k=0
bl3=matrix(data=NA,nrow = 16,ncol=ncol(bl1)*ncol(bl2))
for( i in 1:ncol(bl1)){
  for(j in 1:ncol(bl2)){
    k=k+1
    bl3[,k]=rbind(bl1[,i],bl2[,j])
  }
}

b1=matrix(data=NA,nrow=32,ncol(bl3))
b2=matrix(data=NA,nrow=32,ncol(bl3))
for(j in 1:ncol(bl3)){
  b1[bl3[1:4,j],j]=c(-1,-1,-1,-1)
  b2[bl3[1:4,j],j]=c(-1,-1,-1,-1)
  b1[bl3[5:8,j],j]=c(-1,-1,-1,-1)
  b2[bl3[5:8,j],j]=c(1,1,1,1)
  b1[bl3[9:12,j],j]=c(1,1,1,1)
  b2[bl3[9:12,j],j]=c(-1,-1,-1,-1)
  b1[bl3[13:16,j],j]=c(1,1,1,1)
  b2[bl3[13:16,j],j]=c(1,1,1,1)
  for(k in 17:32){
    b1[k,j]=b1[32-k+1,j]
    b2[k,j]=b2[32-k+1,j]
  }
}
return(rbind(b1,b2))
}

numbr=ncol(allblocks)
notrouble=matrix(data=NA,nrow=numbr,ncol=1225)
meanres=matrix(data=NA,nrow=numbr,ncol=16)
meanres2=matrix(data=NA,nrow=numbr,ncol=16)
for(r in 1:ncol(allblocks)){
  blok1=blokker4_2(allblocks[,r])[1:32,]
  blok2=blokker4_2(allblocks[,r])[33:64,]
  blok12=blok1*blok2
num=ncol(blok1)
results3_2=matrix(data=NA,nrow = 15,ncol=num)
#Iterating over blocks
for(i in 1:num){
```

```
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    matrise=cbind(combGenerator(design,combins[,j]),blok1[,i],
    blok2[,i],blok12[,i])
   results3_2[j,i]=results3[k]
}
}
#meanres: store largest mean
meanres[r,]=c((results3_2[,which(colSums(results3_2==0)==0)])
[,which.max(colSums(results3_2[,which(colSums(results3_2==0)==0)
    ]))],
which.max(colSums(results3_2[,which(colSums(results3_2==0)==0)]))
    )
#meanres2: largest min
meanres2[r,]=c((results3_2[,which(colSums(results3_2==0)==0)])
[,which.max(apply(results3_2[,which(colSums(results3_2==0)==0)],
2,min))],
which.max(apply(results3_2[,which(colSums(results3_2==0)==0)],
2,min)))
}

mini=numeric(nrow(meanres))
maxi=numeric(nrow(meanres))
mini2=numeric(nrow(meanres))
maxi2=numeric(nrow(meanres))
for(k in 1:nrow(meanres)){
  mini[k]=min(meanres[k,1:15])
  maxi[k]=max(meanres[k,1:15])
  mini2[k]=min(meanres2[k,1:15])
  maxi2[k]=max(meanres2[k,1:15])
}

#How many have each max and min?
table(maxi)
table(mini)
table(maxi2)
table(mini2)

#How are the mean values distributed?
means=rowMeans(meanres[,1:15])
table(means)
```

```
minmeans=rowMeans(meanres2[,1:15])
table(minmeans)
save.image("factorialDesign3fourblocksUsing2blocking.RData")
```

Code for section 4.2.6.1: A 2^{7-2}_{IV} design, two blocks

```
install.packages("xtable")
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
install.packages("pracma")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
```

```
  res
}

#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*
    resc[,interact2[2,j]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=
    (paste(colnam[interact3[1,j]],colnam[interact3[2,j]],
    colnam[interact3[3,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}
```

```
hadam1=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
-1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,
1,1,-1,-1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,1,
-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,-1,
1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,
1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,
1,1,-1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,
1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,-1,
-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,
1,1,1,-1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,
1,-1,1,-1,1,1,-1,1,-1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,

-1,1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,

-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,-1,

1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
1,-1,-1,1,-1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,-1,1,1,

-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,

1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,
```

```
-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,1,

-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,1,

1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,

-1,-1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,

1,1,-1,1,-1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,
-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,

1,-1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,
1),nrow=32,ncol=32)
dmatrise=hadam1[,2:32]
int=3
fac=7
combins=combn(factors,int)

best=0
it=0
#while(best<0.1){
valid2=FALSE
while(valid2==FALSE){
 it=it+1
  valid=FALSE
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=
    ncol(sortmatrise))
    indekser=c(32,16,8,4,2)
    #Shuff saves all possible columns indices
    shuff=seq(from = 1, to = 32, by = 1)
    #Iterate through all indices (=make columns for the 6 factors
        )
    for(i in 1:5){
      #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
          the
      #subcolumns which should be sorted in a certain way
      #Shuff is used to save the indices of the columns where all
```

```
#the subcolumns sum to zero
shuff=seq(from = 1, to = 31, by = 1)
designsplit=numeric(indekser[i])
#Badshuff saves the indices for the columns where
#the subcolmnsum is not zero
badshuff=rep(0,31)
badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
#V iterates over the same numbers as k
for(v in 1:(32/indekser[i])){
  #designsplit is a matrix with the subcolumns which are
  sorted by.
  #Here taken directly from sortmatrise
  #It contains all subcolumns of size  64/indekser[i]
  designsplit=cbind(designsplit,sortmatrise[((v-1)*
  indekser[i]+1):(v*indekser[i]),])
}
#Remove the column with zeroes used when initialising
designsplit=designsplit[,-1]
#Go through all columns in the original setup,
#to check if all subcolumns in each column sum to zero
for(t in 1:31){
  if((sum(colSums(designsplit)[seq(from=t,
  to=(31*(-1+(32/indekser[i]))+t),by=31)]>0)>0)){
    #save the trouble-making indices
    badshuff[t]=t
  }
}
badindeks=badshuff
#Remove the bad columns from the vector to be tested
if(sum(badindeks)>0){
  shuff=shuff[-badindeks]
}
shuff=sample(shuff)
first=shuff[1]
iter=1
#Find row indices based on the first ok shuff
for(k in 1:(32/indekser[i])){
  #Radindekser is the indices of the rows to be
  included in the subcolumn,
  #Sort sortmatrises indices to make -1 come first, then 1
  radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
```

```
      indekser[i]+1):(k*indekser[i]),shuff[iter]],
      decreasing= FALSE)
      #Iterate to everything is ok
      while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
      shuff[iter]]))>0){
        iter=iter+1
        radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
        indekser[i]+1):(k*indekser[i]),shuff[iter]],
        decreasing = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=
        sortmatrise[radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
      nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
  print(indeksliste)
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
designmatrise[,4]*designmatrise[,5]*designmatrise[,1])
colnames(designmatrise)=c("E","D","C","B","A","F","G")
blokker=sortmatrise[,-indeksliste]
ending=ncol(blokker)
trouble=Generator3(designmatrise)
print(rowSums(abs(t(blokker[,])%*%trouble)==32))
valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
if(valid2>0){
goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%
trouble)==32)==0)]
num=ncol(goodblocks)
results3=matrix(data=NA,nrow=num,ncol=ncol(combins))
k=0
#Iterating over blocks
```

```
  for(i in 1:num){
    #Iterating over combinations
    for(j in 1:ncol(combins)){
      k=k+1
      stormatrise=cbind(combGenerator(designmatrise,combins[,j]),
      goodblocks[,i])
   #   print(stormatrise)
      results3[i,j]=Ds(stormatrise)
    }
  }

  braindekser=which(rowSums(results3==0)==0)
  best=max(rowSums(results3[braindekser,])/ncol(combins))
  }
}

rowMeans(results3[braindekser,])
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
max(maxi)
max(mini)

#In what order was the matrix sorted?
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)
sort(unique(rowMeans(results3[braindekser,])))
#Which columns were good blocks?
which(duplicated(cbind(goodblocks[,braindekser],
sortmatrise),MARGIN=2))-length(braindekser)+1
#(+1 for the removed column)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
for(i in 1:length(braindekser)){
```

```
  print(table(results3[braindekser[i],]))
}
printmat=cbind(designmatrise[,5],designmatrise[,4],designmatrise
    [,3],
designmatrise[,2],designmatrise[,1],designmatrise[,6],
designmatrise[,7],goodblocks[,braindekser])
colnames(printmat)=c("A","B","C","D","E","F","G","b1","b2","b3","
    b4")
xtable(printmat, digits = 0)
#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,
combins[,which.max(results3[5,])]),goodblocks[,5])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,
combins[,which.min(results3[5,])]),goodblocks[,5])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.6.4: A 2_{IV}^{7-2} design, four blocks

```
install.packages("xtable")
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
install.packages("pracma")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
```

```
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],colnam[
```

```
        interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*resc
        [,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=(paste(colnam[interact3
        [1,j]],colnam[interact3[2,j]], colnam[interact3[3,j]],
        collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam2=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
1,1,-1,-1,-1,-1, 1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,
-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,
1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,
1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,
-1,1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,-1,
1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,
-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,
```

```
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,1,
1,-1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,
1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,
1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,
1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,
-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,
-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,
-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,
1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,-1,-1,-1,1,1,1,
-1,1,-1,1,1,-1,-1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,
1,-1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,
-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,
1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,1,1,-1,1,1,1,-1,-1,1,
-1,-1,1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,-1,1,
1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1),nrow=32,ncol=32)
dmatrise=hadam2[,2:32]
int=3
fac=7
combins=combn(fac,int)

best=0
it=0
#while(best<0.1){
braindekser=NULL
while(length(braindekser)==0){
  it=it+1
  valid=FALSE
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
        sortmatrise))
```

```
indekser=c(32,16,8,4,2)
#Shuff saves all possible columns indices
shuff=seq(from = 1, to = 32, by = 1)
#Iterate through all indices (=make columns for the 6 factors
    )
for(i in 1:5){
  #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
       the subcolumns which should be sorted in a certain way
  #Shuff is used to save the indices of the columns where all
       the subcolumns sum to zero
  shuff=seq(from = 1, to = 31, by = 1)
  designsplit=numeric(indekser[i])
  #Badshuff saves the indices for the columns where the
      subcolmnsum is not zero
  badshuff=rep(0,31)
  badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
  #V iterates over the same numbers as k
  for(v in 1:(32/indekser[i])){
    #designsplit is a matrix with the subcolumns which are
        sorted by. Here taken directly from sortmatrise
    #It contains all subcolumns of size  64/indekser[i]
    designsplit=cbind(designsplit,sortmatrise[((v-1)*indekser
        [i]+1):(v*indekser[i]),])
  }
  #Remove the column with zeroes used when initialising
  designsplit=designsplit[,-1]
  #Go through all columns in the original setup, to check if
      all subcolumns in each column sum to zero
  for(t in 1:31){
    if((sum(colSums(designsplit)[seq(from=t,to=(31*(-1+(32/
        indekser[i])))+t),by=31)]>0)>0)){
      #ace the trouble-making indices
      badshuff[t]=t
    }
  }
  badindeks=badshuff
  #Remove the bad columns from the vector to be tested
  if(sum(badindeks)>0){
    shuff=shuff[-badindeks]
  }
  shuff=sample(shuff)
```

```
    first=shuff[1]
    iter=1
    #Find row indices based on the first ok shuff
    for(k in 1:(32/indekser[i])){
      #Radindekser is the indices of the rows to be included in
          the subcolumn, Sort sortmatrises indices to make -1
          come first, then 1
      radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
          indekser[i]+1):(k*indekser[i]),shuff[iter]],
          decreasing = FALSE)
      #Iterate to everything is ok
      while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
          shuff[iter]]))>0){
        iter=iter+1
        radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
            indekser[i]+1):(k*indekser[i]),shuff[iter]],
            decreasing = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[
            radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
          nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
  print(indeksliste)
  #Add one to make it correspond to matrix in thesis (because
      of removed column)
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
    designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
    designmatrise[,4]*designmatrise[,5]*designmatrise[,1])
colnames(designmatrise)=c("E","D","C","B","A","F","G")
blokker=sortmatrise[,-indeksliste]
ending=ncol(blokker)
```

```
  trouble=Generator3(designmatrise)
  #print(rowSums(abs(t(blokker[,])%*%trouble)==32))
  valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
  if(valid2>0){
    goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%trouble
        )==32)==0)]
    num=ncol(goodblocks)
    blockcomb=combn(num,2)
    results3=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(
        combins))
    k=0
    #Iterating over blocks
    for(i in 1:ncol(blockcomb)){
      #Iterating over combinations
      for(j in 1:ncol(combins)){
        k=k+1
        stormatrise=cbind(combGenerator(designmatrise,combins[,j
            ]),goodblocks[,blockcomb[1,i]],goodblocks[,blockcomb
            [2,i]],goodblocks[,blockcomb[1,i]]*goodblocks[,
            blockcomb[2,i]])
        #   print(stormatrise)
        results3[i,j]=Ds(stormatrise)
      }
    }

    braindekser=which(rowSums(results3==0)==0)
  }
}

bestres=results3[braindekser,]
bestmeans=rowMeans(results3[braindekser,])
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
max(maxi)
max(mini)
which(bestmeans>0.939)
which(min>0.917)
bestcomb=which(bestmeans>0.939)
#Which combins were good blocks? +1 because of removed row of
    matrix
bestcombreal=braindekser[bestcomb]
```

```
which(duplicated(cbind(goodblocks[,blockcomb[1,bestcombreal[1]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[2,bestcombreal[1]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[1,bestcombreal[2]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[2,bestcombreal[2]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[1,bestcombreal[3]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[2,bestcombreal[3]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[1,bestcombreal[4]]],
    sortmatrise),MARGIN=2))
which(duplicated(cbind(goodblocks[,blockcomb[2,bestcombreal[4]]],
    sortmatrise),MARGIN=2))


#In what order was the matrix sorted?
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)
sort(unique(rowMeans(results3[braindekser,])))
#Which columns make up the design matrix?
print(indeksliste+1)

printmat=cbind(designmatrise[,5],designmatrise[,4],designmatrise
    [,3],designmatrise[,2],designmatrise[,1],designmatrise[,6],
    designmatrise[,7],sortmatrise[,c(10,11,24,25)])
colnames(printmat)=c("A","B","C","D","E","F","G","11","12","25","
    26")
xtable(printmat, digits = 0)
#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=stormatrise=cbind(combGenerator(designmatrise,combins
```

```
    [,5]),goodblocks[,blockcomb[1,14]],goodblocks[,blockcomb
    [2,14]],goodblocks[,blockcomb[1,14]]*goodblocks[,blockcomb
    [2,14]])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))

#For one of the combinations yielding the lowest Ds
matrise0=stormatrise=cbind(combGenerator(designmatrise,combins
    [,2]),goodblocks[,blockcomb[1,14]],goodblocks[,blockcomb
    [2,14]],goodblocks[,blockcomb[1,14]]*goodblocks[,blockcomb
    [2,14]])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))

save.image("cyclic64_hadamard_72_4blocks.RData")
```

Code for section 4.2.6.1: A 2^{8-3}_{IV} design, two blocks

```
install.packages("xtable")
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
install.packages("pracma")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
```

```
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]
    *res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*
```

```
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=
    (paste(colnam[interact3[1,j]],colnam[interact3[2,j]],
    colnam[interact3[3,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam1=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,
-1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,
-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,-1,
1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,1,
1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,
1,1,-1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,
1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,-1,
-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,
1,1,1,-1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,
1,-1,1,-1,1,1,-1,1,-1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,
```

```
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,

-1,1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,
-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,
1,-1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,
-1,1,1,1,1,-1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,
1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,-1,
1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,
-1,1,-1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,
1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,
-1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
1,1,1,1,1,1,-1,-1,-1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,
-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,-1,
-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,-1,
-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,
1,-1,1,-1,1),nrow=32,ncol=32)
dmatrise=hadam1[,2:32]

int=3
fac=8
combins=combn(factors,int)

best=0
it=0
#Valid2 is true if a blocking yielding Ds>0 for all combinations
#is found
valid2=FALSE
while(valid2==FALSE){
  it=it+1
  valid=FALSE
  #valid is true if the 2^5 can be made
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),
    ncol=ncol(sortmatrise))
```

```
indekser=c(32,16,8,4,2)
#Shuff saves all possible columns indices
shuff=seq(from = 1, to = 32, by = 1)
#Iterate through all indices (=make columns for the 6 factors
    )
for(i in 1:5){
  #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
       the
  #subcolumns which should be sorted in a certain way
  #Shuff is used to save the indices of the columns where all
  #the subcolumns sum to zero
  shuff=seq(from = 1, to = 31, by = 1)
  designsplit=numeric(indekser[i])
  #Badshuff saves the indices for the columns where the
  #subcolumnsum is not zero
  badshuff=rep(0,31)
  badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
  #V iterates over the same numbers as k
  for(v in 1:(32/indekser[i])){
    #designsplit is a matrix with the subcolumns which are
    #sorted by. Here taken directly from sortmatrise
    #It contains all subcolumns of size  64/indekser[i]
    designsplit=cbind(designsplit,
    sortmatrise[((v-1)*indekser[i]+1):(v*indekser[i]),])
  }
  #Remove the column with zeroes used when initialising
  designsplit=designsplit[,-1]
  #Go through all columns in the original setup,
  #to check if all
  #subcolumns in each column sum to zero
  for(t in 1:31){
    if((sum(colSums(designsplit)[seq(from=t,
    to=(31*(-1+(32/indekser[i])))+t),by=31)]>0)>0)){
      #Save the trouble-making indices
      badshuff[t]=t
    }
  }
  badindeks=badshuff
  #Remove the bad columns from the vector to be tested
  if(sum(badindeks)>0){
    shuff=shuff[-badindeks]
```

```
    }
    shuff=sample(shuff)
    first=shuff[1]
    iter=1
    #Find row indices based on the first ok shuff
    for(k in 1:(32/indekser[i])){
      #Radindekser is the indices of the rows to be included in
      #the subcolumn, Sort sortmatrises indices to make -1 come
           first, then 1
      radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
      indekser[i]+1):(k*indekser[i]),shuff[iter]], decreasing =
           FALSE)
      #Iterate to everything is ok
      while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
      shuff[iter]]))>0){
        iter=iter+1
        radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
        indekser[i]+1):(k*indekser[i]),shuff[iter]], decreasing
             = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=
        sortmatrise[radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
      nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,5]*
designmatrise[,3]*designmatrise[,2]*designmatrise[,1])
colnames(designmatrise)=c("E","D","C","B","A","F","G","H")
blokker=sortmatrise[,-indeksliste]
```

```
  ending=ncol(blokker)
  trouble=Generator3(designmatrise)
  valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
  if(valid2>0){
    goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%
    trouble)==32)==0)]
    num=ncol(goodblocks)
    results3=matrix(data=NA,nrow=num,ncol=ncol(combins))
    k=0
    #Iterating over blocks
    for(i in 1:num){
      #Iterating over combinations
      for(j in 1:ncol(combins)){
        k=k+1
        stormatrise=cbind(combGenerator(designmatrise,combins[,j
            ]),
        goodblocks[,i])
        results3[i,j]=Ds(stormatrise)
      }
    }

    braindekser=which(rowSums(results3==0)==0)
    best=max(rowSums(results3[braindekser,])/ncol(combins))
  }
}
#Find the order in which the matrix was sorted
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}

#In what order was the matrix sorted?
print(rekkef)
unique(rowMeans(results3[braindekser,]))
#Which columns were good blocks?
which(duplicated(cbind(goodblocks[,braindekser],
sortmatrise),MARGIN=2))-
```

```
length(braindekser)+1 #(+1 for the removed column)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
for(i in 1:length(braindekser)){
print(table(results3[braindekser[i],]))
}
printmat=cbind(designmatrise[,5],designmatrise[,4],
designmatrise[,3],
designmatrise[,2],designmatrise[,1],designmatrise[,6],
designmatrise[,7],
designmatrise[,8],goodblocks[,braindekser])
colnames(printmat)=c("A","B","C","D","E","F","G","H",
"b1","b2","b3","b4")
xtable(printmat, digits = 0)
smallprintmat=cbind(designmatrise[,6],designmatrise[,5],
designmatrise[,4],designmatrise[,3],designmatrise[,2],
designmatrise[,1])
colnames(smallprintmat)=c("A","B","C","D","E","F")
xtable(smallprintmat, digits = 0)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,
combins[,which.max(results3[1,])]),goodblocks[,1])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,
combins[,which.min(results3[1,])]),goodblocks[,1])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.1: A 2_{IV}^{8-3} design, four blocks

```
install.packages("xtable")
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
install.packages("pracma")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
```

```
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],colnam[
        interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*resc
        [,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=(paste(colnam[interact3
        [1,j]],colnam[interact3[2,j]], colnam[interact3[3,j]],
        collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam2=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
```

```
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
1,1,-1,-1,-1,-1, 1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,
-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,
1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,
-1,1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,-1,
1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,
-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,
1,-1,-1,-1,1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,
1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,-1,1,1,-1,1,1,-1,
1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,
1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,
1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,
-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,
-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,
-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,
1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,-1,-1,-1,1,1,1,
-1,1,-1,1,1,-1,-1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,
1,-1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,
-1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,
1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,1,1,-1,-1,-1,1,1,1,
-1,-1,1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,-1,1,
1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1),nrow=32,ncol=32)
dmatrise=hadam2[,2:32]
int=3
fac=8
combins=combn(fac,int)
```

```
best=0
it=0
#while(best<0.1){
braindekser=NULL
while(length(braindekser)==0){
  it=it+1
  valid=FALSE
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
        sortmatrise))
    indekser=c(32,16,8,4,2)
    #Shuff saves all possible columns indices
    shuff=seq(from = 1, to = 32, by = 1)
    #Iterate through all indices (=make columns for the 6 factors
        )
    for(i in 1:5){
      #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
          the subcolumns which should be sorted in a certain way
      #Shuff is used to save the indices of the columns where all
          the subcolumns sum to zero
      shuff=seq(from = 1, to = 31, by = 1)
      designsplit=numeric(indekser[i])
      #Badshuff saves the indices for the columns where the
          subcolmnsum is not zero
      badshuff=rep(0,31)
      badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
      #V iterates over the same numbers as k
      for(v in 1:(32/indekser[i])){
        #designsplit is a matrix with the subcolumns which are
            sorted by. Here taken directly from sortmatrise
        #It contains all subcolumns of size  64/indekser[i]
        designsplit=cbind(designsplit,sortmatrise[((v-1)*indekser
            [i]+1):(v*indekser[i]),])
      }
      #Remove the column with zeroes used when initialising
      designsplit=designsplit[,-1]
      #Go through all columns in the original setup, to check if
          all subcolumns in each column sum to zero
      for(t in 1:31){
```

```
    if((sum(colSums(designsplit)[seq(from=t,to=(31*(-1+(32/
        indekser[i])))+t),by=31)]>0)>0)){
      #ace the trouble-making indices
      badshuff[t]=t
    }
  }
  badindeks=badshuff
  #Remove the bad columns from the vector to be tested
  if(sum(badindeks)>0){
    shuff=shuff[-badindeks]
  }
  shuff=sample(shuff)
  first=shuff[1]
  iter=1
  #Find row indices based on the first ok shuff
  for(k in 1:(32/indekser[i])){
    #Radindekser is the indices of the rows to be included in
         the subcolumn, Sort sortmatrises indices to make -1
        come first, then 1
    radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
        indekser[i]+1):(k*indekser[i]),shuff[iter]],
        decreasing = FALSE)
    #Iterate to everything is ok
    while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
        shuff[iter]]))>0){
      iter=iter+1
      radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
          indekser[i]+1):(k*indekser[i]),shuff[iter]],
          decreasing = FALSE)
    }
    indeksliste[i]=shuff[iter]
    for(r in 1:length(radindekser)){
      nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[
          radindekser[r],]
    }
    sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
        nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
  }
}
valid=sum(is.na(indeksliste))<1
print(indeksliste)
```

```
  #Add one to make it correspond to matrix in thesis (because
      of removed column)
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
    designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
    designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,5]*
    designmatrise[,3]*designmatrise[,2]*designmatrise[,1])
colnames(designmatrise)=c("E","D","C","B","A","F","G","H")
blokker=sortmatrise[,-indeksliste]
ending=ncol(blokker)
trouble=Generator3(designmatrise)
#print(rowSums(abs(t(blokker[,])%*%trouble)==32))
valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
if(valid2>0){
  goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%trouble
      )==32)==0)]
  num=ncol(goodblocks)
  blockcomb=combn(num,2)
  results3=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(
      combins))
  k=0
  #Iterating over blocks
  for(i in 1:ncol(blockcomb)){
    #Iterating over combinations
    for(j in 1:ncol(combins)){
      k=k+1
      stormatrise=cbind(combGenerator(designmatrise,combins[,j
          ]),goodblocks[,blockcomb[1,i]],goodblocks[,blockcomb
          [2,i]],goodblocks[,blockcomb[1,i]]*goodblocks[,
          blockcomb[2,i]])
      #   print(stormatrise)
      results3[i,j]=Ds(stormatrise)
    }
  }

  braindekser=which(rowSums(results3==0)==0)
}
```

```
}

bestres=results3[braindekser,]
bestmeans=rowMeans(results3[braindekser,])
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
max(maxi)
max(mini)
print(sort(unique(mini)))
print(sort(unique(bestmeans)))
print(sort(unique(maxi)))
which(bestmeans>0.9288)
which(mini>0.853)
bestcomb=which(bestmeans>0.9288)
#Which combins were good blocks? +1 because of removed row of
    matrix
bestcombreal=braindekser[bestcomb]
for(i in 1:length(bestcombreal)){
print(cat("Current␣comb:␣", i))
print(which(duplicated(cbind(goodblocks[,blockcomb[1,bestcombreal
    [i]]],sortmatrise),MARGIN=2)))
print(which(duplicated(cbind(goodblocks[,blockcomb[2,bestcombreal
    [i]]],sortmatrise),MARGIN=2)))
}

#Print frequencies
for(i in 1:nrow(bestres[bestcomb,])){
print(table(bestres[bestcomb,][i,]))
}

#In what order was the matrix sorted?
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)
sort(unique(rowMeans(results3[braindekser,])))
```

```
#Which columns make up the design matrix?
print(indeksliste+1)

printmat=cbind(designmatrise[,5],designmatrise[,4],designmatrise
    [,3],designmatrise[,2],designmatrise[,1],designmatrise[,6],
    designmatrise[,7],designmatrise[,8],sortmatrise[,(c
    (9,10,11,12,13,14,18,19,22,23,25,26,31,32)-1)])
colnames(printmat)=c("A","B","C","D","E","F","G","H","9","10","11
    ","12","13","14","18","19","22","23","25","26","31","32")
xtable(printmat, digits = 0)
#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=stormatrise=cbind(combGenerator(designmatrise,combins
    [,27]),goodblocks[,blockcomb[1,8]],goodblocks[,blockcomb
    [2,8]],goodblocks[,blockcomb[1,8]]*goodblocks[,blockcomb
    [2,8]])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))

#For one of the combinations yielding the lowest Ds
matrise0=stormatrise=cbind(combGenerator(designmatrise,combins
    [,1]),goodblocks[,blockcomb[1,8]],goodblocks[,blockcomb[2,8]],
    goodblocks[,blockcomb[1,8]]*goodblocks[,blockcomb[2,8]])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))

save.image("cyclic32_hadamard_83_4blocks.RData")
```

Code for section 4.2.6.5: A 2_{IV}^{9-4} design, two blocks

```
install.packages("pracma")
library(pracma)
#Defining some functions at first
```

```
#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make effects up to four-factor interactions
combGenerator2=function(mat,interest){
  interact2=combn(length(interest),2)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(7)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
```

```
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}


#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+
  ncol(interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*
    resc[,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=
    (paste(colnam[interact3[1,j]],colnam[interact3[2,j]],
    colnam[interact3[3,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}
```

```
#Make all effects up to three-factor interactions
Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)))
  colnam=numeric(ncol(mat)+ncol(interact2))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],
    colnam[interact2[2,j]], collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam1=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,
-1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,
-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
```

-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,-1,
1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,
1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,
1,1,-1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,
1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,-1,
-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,
1,1,1,-1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,
1,-1,1,-1,1,1,-1,1,-1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,

-1,1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,

-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,-1,

1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,-1,1,1,

-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,

1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,

-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,1,

-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,1,

1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,

-1,-1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,-1,-1,

1,1,-1,1,-1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,

```
-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,

1,-1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,
1),nrow=32,ncol=32)
dmatrise=hadam1[,2:32]

int=3
fac=9
combins=combn(factors,int)

best=0
it=0
#while(best<0.1){
valid2=FALSE
while(valid2==FALSE){
 it=it+1
  valid=FALSE
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),
    ncol=ncol(sortmatrise))
    indekser=c(32,16,8,4,2)
    #Shuff saves all possible columns indices
    shuff=seq(from = 1, to = 32, by = 1)
    #Iterate through all indices (=make columns for the 6 factors
        )
    for(i in 1:5){
      #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
          the
      #subcolumns which should be sorted in a certain way
      #Shuff is used to save the indices of the columns where all
      #the subcolumns sum to zero
      shuff=seq(from = 1, to = 31, by = 1)
      designsplit=numeric(indekser[i])
      #Badshuff saves the indices for the columns where
      #the subcolumnsum is not zero
      badshuff=rep(0,31)
      badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
      #V iterates over the same numbers as k
      for(v in 1:(32/indekser[i])){
```

```
  #designsplit is a matrix with the subcolumns which
  #are sorted by. Here taken directly from sortmatrise
  #It contains all subcolumns of size  64/indekser[i]
  designsplit=cbind(designsplit,sortmatrise[((v-1)*
  indekser[i]+1):(v*indekser[i]),])
}
#Remove the column with zeroes used when initialising
designsplit=designsplit[,-1]
#Go through all columns in the original setup,
#to check if all subcolumns in each column sum to zero
for(t in 1:31){
  if((sum(colSums(designsplit)[seq(from=t,
  to=(31*(-1+(32/indekser[i])))+t),by=31)]>0)>0)){
    #save the trouble-making indices
    badshuff[t]=t
  }
}
badindeks=badshuff
#Remove the bad columns from the vector to be tested
if(sum(badindeks)>0){
  shuff=shuff[-badindeks]
}
shuff=sample(shuff)
first=shuff[1]
iter=1
#Find row indices based on the first ok shuff
for(k in 1:(32/indekser[i])){
  #Radindekser is the indices of
  #the rows to be included
  #in the subcolumn.
  #Sort sortmatrises indices to make -1
  #come first
  radindekser=(k-1)*indekser[i]+
  order(sortmatrise[((k-1)*
  indekser[i]+1):(k*indekser[i]),
  shuff[iter]], decreasing = FALSE)
  #Iterate to everything is ok
  while(length(shuff)>iter&abs(sum(sortmatrise[
  radindekser,shuff[iter]]))>0){
    iter=iter+1
    radindekser=(k-1)*indekser[i]+order(
```

```
        sortmatrise[((k-1)*
        indekser[i]+1):(k*indekser[i]),
        shuff[iter]],decreasing = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=
        sortmatrise[radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
      nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
  print(indeksliste)
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,5]*
designmatrise[,3]*designmatrise[,2])
designmatrise=cbind(designmatrise,designmatrise[,4]*
designmatrise[,3]*designmatrise[,2]*designmatrise[,1])
colnames(designmatrise)=c("E","D","C","B","A","F","G","H","I")
blokker=sortmatrise[,-indeksliste]
ending=ncol(blokker)
trouble=Generator3(designmatrise)
valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
if(valid2>0){
goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%
trouble)==32)==0)]
num=ncol(goodblocks)
results3=matrix(data=NA,nrow=num,ncol=ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
```

```
      k=k+1
      stormatrise=cbind(combGenerator(designmatrise,combins[,j]),
      goodblocks[,i])
      results3[i,j]=Ds(stormatrise)
    }
  }

  braindekser=which(rowSums(results3==0)==0)
  best=max(rowSums(results3[braindekser,])/ncol(combins))
  }
}


rowMeans(results3[braindekser,])
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
max(maxi)
max(mini)

#In what order was the matrix sorted?
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)
sort(unique(rowMeans(results3[braindekser,])))
#Which columns were good blocks?
which(duplicated(cbind(goodblocks[,braindekser],sortmatrise),
MARGIN=2))-
length(braindekser)+1 #(+1 for the removed column)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
for(i in 1:length(braindekser)){
  print(table(results3[braindekser[i],]))
}
printmat=cbind(designmatrise[,5],designmatrise[,4],
```

```
designmatrise[,3],
designmatrise[,2],designmatrise[,1],designmatrise[,6],
designmatrise[,7],
designmatrise[,8],designmatrise[,9],goodblocks[,braindekser])
colnames(printmat)=c("A","B","C","D","E","F","G","H","I","b1",
"b2","b3","b4","b5","b6","b7","b8")
xtable(printmat, digits = 0)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,
combins[,which.max(results3[1,])]),goodblocks[,1])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,
combins[,which.min(results3[1,])]),goodblocks[,1])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))
```

Code for section 4.2.6.6: A 2^{9-4}_{IV} design, four blocks

```
install.packages("xtable")
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
install.packages("pracma")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
```

```
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make all effects up to three-factor interactions
Generator3=function(mat){
  interact2=combn(ncol(mat),2)
  interact3=combn(ncol(mat),3)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=(ncol(mat)+ncol(
      interact2)+ncol(interact3)))
  colnam=numeric(ncol(mat)+ncol(interact2)+ncol(interact3))
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+ncol(mat)]=resc[,interact2[1,j]]*resc[,interact2[2,j
        ]]
    colnam[j+ncol(mat)]=(paste(colnam[interact2[1,j]],colnam[
        interact2[2,j]], collapse = ''))
```

```
  }
  for(j in 1:ncol(interact3)){
    resc[,j+ncol(mat)+ncol(interact2)]=resc[,interact3[1,j]]*resc
        [,interact3[2,j]]*resc[,interact3[3,j]]
    colnam[j+ncol(mat)+ncol(interact2)]=(paste(colnam[interact3
        [1,j]],colnam[interact3[2,j]], colnam[interact3[3,j]],
        collapse = ''))
  }
  colnames(resc)=colnam
  resc
}


Ds=function(comb){
  b=ncol(comb)
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam2=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
1,1,-1,-1,-1,-1, 1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,
-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,
1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,
1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,
-1,1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,-1,
1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,
-1,1,-1,1,-1,1,1,1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,1,
```

```
1,-1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,
1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,
1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,
1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,
-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,
-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,
-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,1,-1,-1,
1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,-1,-1,-1,1,1,1,
-1,1,-1,1,1,-1,-1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,
1,-1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,
-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,
1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,1,1,-1,-1,-1,1,1,1,
-1,-1,1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,-1,1,
1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1),nrow=32,ncol=32)
dmatrise=hadam2[,2:32]
int=3
fac=9
combins=combn(fac,int)

best=0
it=0
#while(best<0.1){
braindekser=NULL
while(length(braindekser)==0){
  it=it+1
  valid=FALSE
  while(valid==FALSE){
    indeksliste=numeric(5)
    sortmatrise=dmatrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
        sortmatrise))
    indekser=c(32,16,8,4,2)
```

```
#Shuff saves all possible columns indices
shuff=seq(from = 1, to = 32, by = 1)
#Iterate through all indices (=make columns for the 6 factors
    )
for(i in 1:5){
  #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
       the subcolumns which should be sorted in a certain way
  #Shuff is used to save the indices of the columns where all
       the subcolumns sum to zero
  shuff=seq(from = 1, to = 31, by = 1)
  designsplit=numeric(indekser[i])
  #Badshuff saves the indices for the columns where the
      subcolmnsum is not zero
  badshuff=rep(0,31)
  badshuffmat=matrix(data=NA,nrow =(32/indekser[i]),ncol=31)
  #V iterates over the same numbers as k
  for(v in 1:(32/indekser[i])){
    #designsplit is a matrix with the subcolumns which are
        sorted by. Here taken directly from sortmatrise
    #It contains all subcolumns of size  64/indekser[i]
    designsplit=cbind(designsplit,sortmatrise[((v-1)*indekser
        [i]+1):(v*indekser[i]),])
  }
  #Remove the column with zeroes used when initialising
  designsplit=designsplit[,-1]
  #Go through all columns in the original setup, to check if
      all subcolumns in each column sum to zero
  for(t in 1:31){
    if((sum(colSums(designsplit)[seq(from=t,to=(31*(-1+(32/
        indekser[i]))+t),by=31)]>0)>0)){
      #ace the trouble-making indices
      badshuff[t]=t
    }
  }
  badindeks=badshuff
  #Remove the bad columns from the vector to be tested
  if(sum(badindeks)>0){
    shuff=shuff[-badindeks]
  }
  shuff=sample(shuff)
  first=shuff[1]
```

```
    iter=1
    #Find row indices based on the first ok shuff
    for(k in 1:(32/indekser[i])){
      #Radindekser is the indices of the rows to be included in
          the subcolumn, Sort sortmatrises indices to make -1
          come first, then 1
      radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
          indekser[i]+1):(k*indekser[i]),shuff[iter]],
          decreasing = FALSE)
      #Iterate to everything is ok
      while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
          shuff[iter]]))>0){
        iter=iter+1
        radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
            indekser[i]+1):(k*indekser[i]),shuff[iter]],
            decreasing = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[
            radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
          nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
  print(indeksliste)
  #Add one to make it correspond to matrix in thesis (because
      of removed column)
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
    designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,2]*
    designmatrise[,4]*designmatrise[,5])
designmatrise=cbind(designmatrise,designmatrise[,5]*
    designmatrise[,3]*designmatrise[,2])
designmatrise=cbind(designmatrise,designmatrise[,4]*
    designmatrise[,3]*designmatrise[,2]*designmatrise[,1])
```

```
  colnames(designmatrise)=c("E","D","C","B","A","F","G","H","I")
  blokker=sortmatrise[,-indeksliste]
  ending=ncol(blokker)
  trouble=Generator3(designmatrise)
  #print(rowSums(abs(t(blokker[,])%*%trouble)==32))
  valid2=sum(rowSums(abs(t(blokker[,])%*%trouble)==32)==0)
  if(valid2>0){
    goodblocks=blokker[,which(rowSums(abs(t(blokker[,])%*%trouble
        )==32)==0)]
    num=ncol(goodblocks)
    blockcomb=combn(num,2)
    results3=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(
        combins))
    k=0
    #Iterating over blocks
    for(i in 1:ncol(blockcomb)){
      #Iterating over combinations
      for(j in 1:ncol(combins)){
        k=k+1
        stormatrise=cbind(combGenerator(designmatrise,combins[,j
            ]),goodblocks[,blockcomb[1,i]],goodblocks[,blockcomb
            [2,i]],goodblocks[,blockcomb[1,i]]*goodblocks[,
            blockcomb[2,i]])
        #   print(stormatrise)
        results3[i,j]=Ds(stormatrise)
      }
    }

    braindekser=which(rowSums(results3==0)==0)
  }
}

bestres=results3[braindekser,]
bestmeans=rowMeans(results3[braindekser,])
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
print(sort(unique(mini)))
print(sort(unique(bestmeans)))
print(sort(unique(maxi)))
which(bestmeans>0.925)
which(mini>0.853)
```

```
bestcomb=which(bestmeans>0.925)
#Which combins were good blocks? +1 because of removed row of
    matrix
bestcombreal=braindekser[bestcomb]
for(i in 1:length(bestcombreal)){
  print(cat("Current␣comb:␣", i))
  print(which(duplicated(cbind(goodblocks[,blockcomb[1,
      bestcombreal[i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(goodblocks[,blockcomb[2,
      bestcombreal[i]]],sortmatrise),MARGIN=2)))
}

#Print frequencies
for(i in 1:nrow(bestres[bestcomb,])){
  print(table(bestres[bestcomb,][i,]))
}

#In what order was the matrix sorted?
rekkef=numeric(nrow(dmatrise))
for(i in 1:nrow(dmatrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(dmatrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)
sort(unique(rowMeans(results3[braindekser,])))
#Which columns make up the design matrix?
print(indeksliste+1)

printmat=cbind(designmatrise[,5],designmatrise[,4],designmatrise
    [,3],designmatrise[,2],designmatrise[,1],designmatrise[,6],
    designmatrise[,7],designmatrise[,8],designmatrise[,9],
    sortmatrise[,(c(11,12,25,26)-1)])
colnames(printmat)=c("A","B","C","D","E","F","G","H","I","11","12
    ","25","26")
xtable(printmat, digits = 0)
#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=stormatrise=cbind(combGenerator(designmatrise,combins
```

```
    [,7]),goodblocks[,blockcomb[1,25]],goodblocks[,blockcomb
    [2,25]],goodblocks[,blockcomb[1,25]]*goodblocks[,blockcomb
    [2,25]])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))

#For one of the combinations yielding the lowest Ds
matrise0=stormatrise=cbind(combGenerator(designmatrise,combins
    [,58]),goodblocks[,blockcomb[1,25]],goodblocks[,blockcomb
    [2,25]],goodblocks[,blockcomb[1,25]]*goodblocks[,blockcomb
    [2,25]])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))

save.image("cyclic32_hadamard_94_4blocks.RData")
```

Code for section 4.3.1: Dividing a 2_{IV}^{32-26} design into two blocks using the blocking of the 2^{16-11} design

```
library(svMisc)

#Function making the design columns
designGenerator<-function(factors,n){
  design=matrix(data=NA,nrow=n,ncol=16)
  for(i in 1:(factors)){
    vect=numeric(2^i)
    vect[1:(2^(i-1))]=-1
    vect[((2^(i-1))+1):(2^i)]=1
    design[,i]=rep(vect,times=(n/(2^i)))
  }
  int=fac-2
  combins=combn(factors,int)
  for(j in 1:ncol(combins)){
```

```
    design[,(5+j)]=design[,combins[1,j]]*design[,combins[2,j]]*
    design[,combins[3,j]]
  }
  design[,5+ncol(combins)+1]=design[,1]*design[,2]*design[,3]*
  design[,4]*design[,5]
  design
}

#fac=#number of factors in design
fac=5
#fac2=#number of factors in design, w. combos
fac2=16
#n=number of rows in total design
n=2^fac
#m=mirror image pairs
m=n/2
design=designGenerator(fac,n)
colnames(design)<-cbind("A","B","C","D","E","F","G","H","I","J",
"K","L","M","N","O","P")

#num=number of combinations
num=ncol(combn(m,(m/2)))/2

combinator <- function(n, m) {
  index <- combn(seq_len(n), m)
  index <- t(index) + (seq_len(ncol(index)) - 1) * n
  result <- rep(0, nrow(index) * n)
  result[index] <- 1
  matrix(result, ncol = n, nrow = nrow(index), byrow = TRUE)
}

perm=t(combinator(16, 8))
perm1=2*perm[,1:(ncol(perm)/2)]-1


allblocks=matrix(data=NA,nrow=n,ncol=num)
for(i in 1:num){
  for(j in 1:m){
    allblocks[j,i]=perm1[j,i]
  }
  for(k in (m+1):n){
```

```
    allblocks[k,i]=allblocks[n-k+1,i]
  }
}

#Vector with interesting combinations
#Let int be the number of factors of interest
int=3
combins=combn(fac2,int)

Generator2=function(mat){
  interact2=combn(ncol(mat),2)
  resc=matrix(data=NA,nrow=nrow(mat),ncol=136)
  colnam=numeric(136)
  for(i in 1:ncol(mat)){
    resc[,i]=mat[,i]
    colnam[i]=colnames(mat)[i]
  }
  for(j in 1:ncol(interact2)){
    resc[,j+16]=resc[,interact2[1,j]]*resc[,interact2[2,j]]
    colnam[j+16]=(paste(colnam[interact2[1,j]],colnam[interact2
        [2,j]],
    collapse = ''))
  }
  colnames(resc)=colnam
  resc
}

#Remove bad blocks
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==24)>0))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==16)==32))]
allblocks=allblocks[,-c(which(rowSums(abs(t(allblocks[,])%*%
Generator2(design))==32)>0))]
#Generate design matrix
#Let interest be a vector with the factors of interest
#Let mat be the design matrix
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  #Make column for constant
```

```
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+4]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]],collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+7]=(paste(colnam[1+interact3[1,j]],
    colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}

Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

#Testing pattern D2
goodblocks=allblocks
combin2=combn(32,3)
num=ncol(goodblocks)
results3=matrix(data=NA,nrow=num,ncol=ncol(combin2))
```

```
k=0
blokk=c(rep(1,32),rep(-1,32))
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combin2)){
    k=k+1
    B1=design[which(goodblocks[,i]==1),]
    B2=design[which(goodblocks[,i]==-1),]
    stormatrise=rbind(cbind(B1,B1),cbind(B1,-B1),cbind(B2,B2),
        cbind(B2,-B2))
    colnames(stormatrise)<-cbind("A","B","C","D","E","F",
    "G","H","I","J",
    "K","L","M","N","O","P","A2","B2","C2","D2","E2","F2","G2",
    "H2","I2","J2","K2","L2","M2","N2","O2","P2")
    StorDs=cbind(combGenerator(stormatrise,combin2[,j]),blokk)
    results3[i,j]=Ds(StorDs)
  }
}

indekser1=which(rowSums(results3==0)==0)
mini1=numeric(length(which(rowSums(results3==0)==0)))
maxi1=numeric(length(which(rowSums(results3==0)==0)))
a1=rep(1,6)
for(r in 1:length(which(rowSums(results3==0)==0))){
  mini1[r]=min(results3[indekser1[r],])
  maxi1[r]=max(results3[indekser1[r],])
  a1=rbind(a1,table(results3[indekser1[r],]))
}
a1=a1[-1,]
#Finding average Ds
unique(rowMeans(results3))
#Finding minimums
unique(apply(results3, 1, FUN=min))
#Finding maximums
unique(apply(results3, 1, FUN=max))

#Testing pattern D3
results33=matrix(data=NA,nrow=num,ncol=ncol(combin2))
k=0
#Iterating over combinations
```

```
for(i in 1:num){
  #Iterating over blocks
  for(j in 1:ncol(combin2)){
    progress(k,max.value = num*ncol(combin2))
    B1=design[which(goodblocks[,i]==1),]
    B2=design[which(goodblocks[,i]==-1),]
    stormatrise=rbind(cbind(B1,B1),cbind(B2,-B2),
    cbind(B1,-B1),cbind(B2,B2))
    colnames(stormatrise)<-cbind("A","B","C","D","E","F","G",
    "H","I","J", "K","L","M","N","O","P","A2","B2","C2","D2",
    "E2","F2","G2","H2","I2","J2","K2","L2","M2","N2","O2","P2")
    StorDs=cbind(combGenerator(stormatrise,combin2[,j]),blokk)
    (StorDs[1,])
    results33[i,j]=Ds(StorDs)
  }
}


indekser3=which(rowSums(results33==0)==0)
a3=rep(1,6)
mini3=numeric(length(which(rowSums(results33==0)==0)))
maxi3=numeric(length(which(rowSums(results33==0)==0)))
for(r in 1:length(which(rowSums(results33==0)==0))){
  mini3[r]=min(results33[indekser3[r],])
  maxi3[r]=max(results33[indekser3[r],])
  a3=rbind(a3,table(results33[indekser3[r],]))
}
a3=a3[-1,]
#Finding average Ds
unique(rowMeans(results33))
#Finding minimums
unique(apply(results33, 1, FUN=min))
#Finding maximums
unique(apply(results33, 1, FUN=max))
#Finding frequencies
unique(t(a3),MARGIN=2)

#Use block nr 1 as example in thesis!

#Example to thesis
B1=design[which(goodblocks[,1]==1),]
```

```
B2=design[which(goodblocks[,1]==-1),]
stormatrise=rbind(cbind(B1,B1),cbind(B2,-B2),
cbind(B1,-B1),cbind(B2,B2))
colnames(stormatrise)<-cbind("A","B","C","D","E","F","G","H","I",
    "J",
"K","L","M","N","O","P","A2","B2","C2","D2","E2","F2","G2","H2","
    I2",
"J2","K2","L2","M2","N2","O2","P2")

Ds1=cbind(combGenerator(stormatrise,
combin2[,which.max(results33[1,])]),blokk)
Ds0917=cbind(combGenerator(stormatrise,
combin2[,which.min(results33[1,])]),blokk)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
diagonal=diag(solve(t(Ds1)%*%Ds1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/sqrt(min(diagonal[1:(len-1)
    ])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
diagonal2=diag(solve(t(Ds0917)%*%Ds0917))
print(sqrt(max(diagonal2[1:(len-1)]))/sqrt(min(diagonal2[1:(len
    -1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))

#Save results
save.image("design32testermetode.RData")
```

Code for section 4.3.2: Dividing a 2_{IV}^{32-26} design into four blocks using the blocking of the 2_{IV}^{16-11} design

```
load("~/compulsory_3/Markov/d32alle4blokker.RData")

#finner gode blokker: h[U+FFFD]y mulig min ds)
indekser=which(minnull>0.834)
#Finner indeks til de som har h[U+FFFD]y min og maks
testindeks=indekser[which(maxnull[indekser]>0.99999999)]
#Finner de med h[U+FFFD]y gj.snitt (NB: ikke riktig indeks her, ref.
```

```
     til indeks i indeks....)
#Lagre de med h[U+FFFD]y snitt
testindeks=testindeks[which(aver[indekser[which(maxnull[indekser
    ]>0.99999999)]]>0.908206)]

#Lager alle mulige blokker
whole2=c(1,2,3,4,5,6,7,8)
#first=cbind(1,combn(15,3)+1)
choice12=combn(7,1)
choice22=combn(5,1)
second2=matrix(data=NA,nrow=4,ncol=ncol(choice22)*ncol(choice12))
#first=matris(data=na)
for(i in 1:ncol(choice12)){
  first2=c(1,1+choice12[,i])
  rest2=setdiff(whole2,first2)
  for(j in 1:ncol(choice22))
    second2[,(i-1)*ncol(choice22)+j]=c(first2,c(rest2[6],rest2[c(
        choice22[,j])]))
}
choice32=combn(3,1)
third2=matrix(data=NA,nrow=6,ncol=ncol(choice22)*ncol(choice12)*
    ncol(choice32))
for(k in 1:ncol(second2)){
  resten2=setdiff(whole2,second2[,k])
  for(r in 1:ncol(choice32)){
    third2[,(k-1)*ncol(choice32)+r]=c(second2[,k],c(resten2[4],
        resten2[c(choice32[,r])]))
    #  print(third[,(k-1)*ncol(choice3)+r])
  }
}
final2=matrix(data=NA,nrow = 8,ncol=ncol(third2))
for(t in 1:ncol(third2)){
  final2[,t]=c(third2[,t],setdiff(whole2,third2[,t]))
}

library(svMisc)
meanvals=numeric(ncol(final2))
goodblok1=blok1[,testindeks]
goodblok2=blok2[,testindeks]
goodblok12=goodblok1*goodblok2
combin2=combn(32,3)
```

```
num=10
results3=matrix(data=NA,nrow=num,ncol=ncol(combin2))
k=0
blokk1=c(rep(1,32),rep(-1,32))
blokk2=c(rep(1,16),rep(-1,16),rep(1,16),rep(-1,16))
blokk12=blokk1*blokk2
minimat=matrix(nrow=num,ncol = ncol(final2))
maximat=matrix(nrow=num,ncol = ncol(final2))
avermat=matrix(nrow=num,ncol = ncol(final2))
#Iterer over kombinasjoner
for(r in 1:ncol(final2)){
  progress(r,max.value = ncol(final2))
  #r=27
for(i in 1:num){
  #Iterer over blokker
  B1=design[which(goodblok1[,i]==1 & goodblok2[,i]==1),]
  B2=design[which(goodblok1[,i]==1 & goodblok2[,i]==-1),]
  B3=design[which(goodblok1[,i]==-1 & goodblok2[,i]==1),]
  B4=design[which(goodblok1[,i]==-1 & goodblok2[,i]==-1),]
  allrows=rbind(cbind(B1,B1),cbind(B1,-B1),cbind(B2,B2),cbind(B2
      ,-B2),cbind(B3,B3),cbind(B3,-B3),cbind(B4,B4),cbind(B4,-B4))
  stormatrise=rbind(allrows[((final2[1,r]-1)*8+1):(8*final2[1,r])
      ,],allrows[((final2[2,r]-1)*8+1):(8*final2[2,r]),],allrows
      [((final2[3,r]-1)*8+1):(8*final2[3,r]),],allrows[((final2[4,
      r]-1)*8+1):(8*final2[4,r]),],allrows[((final2[5,r]-1)*8+1)
      :(8*final2[5,r]),],allrows[((final2[6,r]-1)*8+1):(8*final2
      [6,r]),],allrows[((final2[7,r]-1)*8+1):(8*final2[7,r]),],
      allrows[((final2[8,r]-1)*8+1):(8*final2[8,r]),])
  colnames(stormatrise)<-cbind("A","B","C","D","E","F","G","H","I
      ","J","K","L","M","N","O","P","A2","B2","C2","D2","E2","F2",
      "G2","H2","I2","J2","K2","L2","M2","N2","O2","P2")
  for(j in 1:ncol(combin2)){
    StorDs=cbind(combGenerator(stormatrise,combin2[,j]),blokk1,
        blokk2,blokk12)
    results3[i,j]=Ds(StorDs)
  }
  minimat[i,r]=min(results3[i,])
  avermat[i,r]=mean(results3[i,])
  maximat[i,r=max(results3[i,])]
}
}
```

```
#How many yields Ds>0 for all combinations?
length(which(minimat[1,]>0))
table(minimat[1,])
#How are the averages then?
table(avermat[1,which(minimat[2,]>0)])
which(avermat[1,]>0.95966)
which(minimat[1,]>0.9126)

#Find the good results


r=36
i=1
results36=numeric(ncol(combin2))
  #Iterer over blokker
  B1=design[which(goodblok1[,i]==1 & goodblok2[,i]==1),]
  B2=design[which(goodblok1[,i]==1 & goodblok2[,i]==-1),]
  B3=design[which(goodblok1[,i]==-1 & goodblok2[,i]==1),]
  B4=design[which(goodblok1[,i]==-1 & goodblok2[,i]==-1),]
  allrows=rbind(cbind(B1,B1),cbind(B1,-B1),cbind(B2,B2),cbind(B2
      ,-B2),cbind(B3,B3),cbind(B3,-B3),cbind(B4,B4),cbind(B4,-B4))
  stormatrise=rbind(allrows[((final2[1,r]-1)*8+1):(8*final2[1,r])
      ,],allrows[((final2[2,r]-1)*8+1):(8*final2[2,r]),],allrows
      [((final2[3,r]-1)*8+1):(8*final2[3,r]),],allrows[((final2[4,
      r]-1)*8+1):(8*final2[4,r]),],allrows[((final2[5,r]-1)*8+1)
      :(8*final2[5,r]),],allrows[((final2[6,r]-1)*8+1):(8*final2
      [6,r]),],allrows[((final2[7,r]-1)*8+1):(8*final2[7,r]),],
      allrows[((final2[8,r]-1)*8+1):(8*final2[8,r]),])
  colnames(stormatrise)<-cbind("A","B","C","D","E","F","G","H","I
      ","J","K","L","M","N","O","P","A2","B2","C2","D2","E2","F2",
      "G2","H2","I2","J2","K2","L2","M2","N2","O2","P2")
  for(j in 1:ncol(combin2)){
    StorDs=cbind(combGenerator(stormatrise,combin2[,j]),blokk1,
       blokk2,blokk12)
    results36[j]=Ds(StorDs)
  }


#Til artikkel:
table(results36)
```

```
Dsmax=cbind(combGenerator(stormatrise,combin2[,which.max(
    results36)]),blokk1,blokk2,blokk12)
Dsmin=cbind(combGenerator(stormatrise,combin2[,which.min(
    results36)]),blokk1,blokk2,blokk12)

#For one of the combinations yielding the highest Ds
diagonal=diag(solve(t(Dsmax)%*%Dsmax))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))

#For one of the combinations yielding the lowest Ds
diagonal2=diag(solve(t(Dsmin)%*%Dsmin))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))

save.image("d32alle4blokkertestermetode.RData")
```

Code for section 4.3.3.1 and 4.3.3.2: Dividing a 2_V^{8-2} design into two blocks using HM

```
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
```

```
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
    res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)]=
    (paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j]],
    colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],collapse =
        ''))
  }
  colnames(res)=colnam
  res
}

#Make effects up to three-factor interactions
combGenerator3=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
```

```
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
    colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
    res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=
    (paste(colnam[1+interact3[1,j]],colnam[1+interact3[2,j]],
    colnam[1+interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}


Ds=function(comb){
  b=ncol(comb)
  s=b-1
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,b])%*%comb[,b])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}


#Generate the design
hadam1=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,
-1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,
-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,
```

-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,-1,
1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,
1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,
1,1,-1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,
1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,-1,
-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,
1,1,1,-1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,
1,-1,1,-1,1,1,-1,1,-1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,

-1,1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,

-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,-1,

1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,-1,1,1,

-1,1,-1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

-1,-1,-1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,

1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,

-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,1,

-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,1,

1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,

-1,-1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,-1,-1,

1,1,-1,1,-1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,-1,

```
-1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,

1,-1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,1,-1,
1),nrow=32,ncol=32)
dmatrise=hadam1

matrise=rbind(cbind(dmatrise,dmatrise),cbind(dmatrise,-dmatrise))
matrise=matrise[,2:64]
int=4
fac=8
combins=combn(fac,int)
combins3=combn(fac,3)

#In the loop, a design yielding good results for four active
    factors is found
best=0
it=0
while(best<0.95){
  it=it+1
valid=FALSE
#print(it)
while(valid==FALSE){
indeksliste=numeric(6)
sortmatrise=matrise
nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
    sortmatrise))
indekser=c(64,32,16,8,4,2)
#Shuff saves all possible columns indices
shuff=seq(from = 1, to = 64, by = 1)
#Iterate through all indices (=make columns for the 6 factors)
for(i in 1:6){
  #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
  #the subcolumns which should be sorted in a certain way
  #Shuff is used to save the indices of the columns where all
  #the subcolumns sum to zero
    shuff=seq(from = 1, to = 63, by = 1)
    designsplit=numeric(indekser[i])
    #Badshuff saves the indices for the columns where
    #the subcolumnsum is not zero
    badshuff=rep(0,63)
    badshuffmat=matrix(data=NA,nrow =(64/indekser[i]),ncol=63)
```

```
#V iterates over the same numbers as k
for(v in 1:(64/indekser[i])){
#designsplit is a matrix with the subcolumns which are sorted
     by.
#Here taken directly from sortmatrise
#It contains all subcolumns of size  64/indekser[i]
designsplit=cbind(designsplit,sortmatrise[((v-1)*
indekser[i]+1):(v*indekser[i]),])
}
#Remove the column with zeroes used when initialising
designsplit=designsplit[,-1]
#Go through all columns in the original setup,
#to check if all subcolumns in each column sum to zero
for(t in 1:63){
 if((sum(colSums(designsplit)[seq(from=t,
 to=(63*(-1+(64/indekser[i]))+t),by=63)]>0)>0)){
   #save the trouble-making indices
  badshuff[t]=t
  }
}
badindeks=badshuff
#Remove the bad columns from the vector to be tested
if(sum(badindeks)>0){
shuff=shuff[-badindeks]
}
shuff=sample(shuff)
first=shuff[1]
iter=1
#Find row indices based on the first ok shuff
for(k in 1:(64/indekser[i])){
  #Radindekser is the indices of the rows to be included in
      the
  #subcolumn, Sort sortmatrises indices to make -1 come first
radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
indekser[i]+1):(k*indekser[i]),shuff[iter]], decreasing =
    FALSE)
#Iterate to everything is ok
while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
shuff[iter]]))>0){
  iter=iter+1
  radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
```

```
      indekser[i]+1):(k*indekser[i]),shuff[iter]], decreasing =
          FALSE)
    }
    indeksliste[i]=shuff[iter]
    for(r in 1:length(radindekser)){
      nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[radindekser[r
          ],]
    }
    sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
    nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
  }
}
valid=sum(is.na(indeksliste))<1
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
designmatrise[,4]*designmatrise[,5]*designmatrise[,6])
designmatrise=cbind(designmatrise,designmatrise[,1]*
designmatrise[,2]*designmatrise[,5]*designmatrise[,6])
colnames(designmatrise)=c("F","E","D","C","B","A","G","H")
blokker=sortmatrise[,-indeksliste]

num=ncol(blokker)
results3=matrix(data=NA,nrow=num,ncol=ncol(combins))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    k=k+1
    stormatrise=cbind(combGenerator(designmatrise,combins[,j]),
    blokker[,i])
    results3[i,j]=Ds(stormatrise)
  }
}

braindekser=which(rowSums(results3==0)==0)
mini=apply(results3, 1, min)
best=max(mini)
print(best)
```

```
}

#Finding min and max obtained for each block
mini=apply(results3, 1, min)
maxi=apply(results3, 1, max)
table(mini)
table(maxi)
#Find unique means for the blocks
rmeans=(rowMeans(results3))
print(sort(unique(rowMeans(results3[braindekser,]))))

#Find the order in which the matrix was sorted
rekkef=numeric(nrow(matrise))
for(i in 1:nrow(matrise)){
  for(j in 1: nrow(sortmatrise)){
   if(identical(matrise[i,],sortmatrise[j,])==TRUE){
    rekkef[i]=j
   }
  }
}
print(rekkef)

sort(unique(rowMeans(results3[braindekser,])))
table(rowMeans(results3[braindekser,]))
bestindekser=braindekser[which(rowMeans(results3[braindekser,])>
0.986)]
#Which columns were good blocks?
which(duplicated(cbind(blokker[,bestindekser],sortmatrise),
MARGIN=2))-length(bestindekser)+1 #(+1 for the removed column)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results3[bestindekser,],1,table)
printmat=cbind(designmatrise[,6],designmatrise[,5],
designmatrise[,4],
designmatrise[,3],designmatrise[,2],designmatrise[,1],
designmatrise[,7],
designmatrise[,8],blokker[,bestindekser])
colnames(printmat)=c("A","B","C","D","E","F","G","H","b1","b2",
"b3","b4","b5","b6","b7","b8","b9","b10","b11","b12")
xtable(printmat[1:32,], digits = 0)
```

```
xtable(printmat[33:64,], digits = 0)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,
combins[,which.max(results3[bestindekser[1],])]),
blokker[,bestindekser[1]])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-1)]))/
sqrt(min(diagonal[1:(len-1)])))
print(sqrt(diagonal[len])/sqrt(min(diagonal[1:(len-1)])))

#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,
combins[,which.min(results3[bestindekser[1],])]),
blokker[,bestindekser[1]])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-1)]))/
sqrt(min(diagonal2[1:(len-1)])))
print(sqrt(diagonal2[len])/sqrt(min(diagonal2[1:(len-1)])))

#Checking for three active factors
results33=matrix(data=NA,nrow=num,ncol=ncol(combins3))
k=0
#Iterating over blocks
for(i in 1:num){
  #Iterating over combinations
  for(j in 1:ncol(combins3)){
    k=k+1
    stormatrise3=cbind(combGenerator3(designmatrise,
    combins3[,j]),blokker[,i])
    results33[i,j]=Ds(stormatrise3)
  }
}

#Finding min and max obtained for each block
mini3=apply(results33[braindekser3,], 1, min)
maxi3=apply(results33[braindekser3,], 1, max)
table(mini3)
table(maxi3)
```

```
bestindekser3=braindekser3[which(mini3>0.92)]
sort(unique(rowMeans(results33[braindekser3,])))
table(rowMeans(results33[bestindekser3,]))
allerbestindekser3=bestindekser3[which(rowMeans(
results33[bestindekser3,])>0.992)]

apply(results33[allerbestindekser3,],1,table)
#Which columns were good blocks?
which(duplicated(cbind(blokker[,allerbestindekser3],
sortmatrise),MARGIN=2))-
length(allerbestindekser3)+1 #(+1 for the removed column)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results33[allerbestindekser3,],1,table)


#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise13=cbind(combGenerator3(designmatrise,
combins3[,which.max(results33[allerbestindekser3[1],])]),
blokker[,allerbestindekser3[1]])
diagonal3=diag(solve(t(matrise13)%*%matrise13))
len3=length(diagonal3)
print(sqrt(max(diagonal3[1:(len3-1)]))/
sqrt(min(diagonal3[1:(len3-1)])))
print(sqrt(diagonal3[len3])/
sqrt(min(diagonal3[1:(len3-1)])))

#For one of the combinations yielding the lowest Ds
matrise03=cbind(combGenerator3(designmatrise,
combins3[,which.min(results33[allerbestindekser3[1],])]),
blokker[,allerbestindekser3[1]])
diagonal23=diag(solve(t(matrise03)%*%
matrise03))
print(sqrt(max(diagonal23[1:(len3-1)]))/
sqrt(min(diagonal23[1:(len3-1)])))
print(sqrt(diagonal23[len3])/
sqrt(min(diagonal23[1:(len3-1)])))
```

Code for section 4.3.3.3 and 4.3.3.4: Dividing a 2_V^{8-2} design into four blocks using HM

```
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
        res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)
        ]=(paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j
```

```
        ]],colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],
        collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make effects up to three-factor interactions
combGenerator3=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}



Ds=function(comb){
  b=ncol(comb)
```

```
  s=b-3
  n=nrow(comb)
  detX=det(t(comb)%*%comb)
  detXb=det(t(comb[,c(b-2,b-1,b)])%*%comb[,c(b-2,b-1,b)])
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam2=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,
1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
1,1,-1,-1,-1,-1, 1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,
-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,
1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,
-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,
-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,
1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,
-1,1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,-1,
1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,
-1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,
-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,
-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,1,
1,-1,-1,-1,1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,
1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,-1,1,1,1,-1,1,1,-1,
1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,
1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,-1,-1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,1,1,
1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,
1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,-1,1,-1,1,1,1,-1,1,-1,
-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,
-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,
-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,
-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,
```

```
-1,1,-1,1,-1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,1,1,1,-1,-1,
1,-1,1,-1,1,1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,1,-1,-1,-1,1,1,1,
-1,1,-1,1,1,-1,-1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,-1,1,
1,-1,-1,1,-1,1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,
-1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,
1,-1,-1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,1,1,-1,-1,-1,1,1,1,
-1,-1,1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1,-1,1,
1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1),nrow=32,ncol=32)
dmatrise=hadam2

matrise=rbind(cbind(dmatrise,dmatrise),cbind(dmatrise,-dmatrise))
matrise=matrise[,2:64]
int=4
fac=8
combins=combn(fac,int)
combins3=combn(fac,3)

#In the loop, a design yielding good results for four active
    factors is found
best=0
it=0
braindekser=NULL
while(length(braindekser)<1){
  it=it+1
  valid=FALSE
  #print(it)
  while(valid==FALSE){
    indeksliste=numeric(6)
    sortmatrise=matrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
        sortmatrise))
    indekser=c(64,32,16,8,4,2)
    #Shuff saves all possible columns indices
    shuff=seq(from = 1, to = 64, by = 1)
    #Iterate through all indices (=make columns for the 6 factors
        )
    for(i in 1:6){
      #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
          the subcolumns which should be sorted in a certain way
      #Shuff is used to save the indices of the columns where all
          the subcolumns sum to zero
```

```
shuff=seq(from = 1, to = 63, by = 1)
designsplit=numeric(indekser[i])
#Badshuff saves the indices for the columns where the
    subcolmnsum is not zero
badshuff=rep(0,63)
badshuffmat=matrix(data=NA,nrow =(64/indekser[i]),ncol=63)
#V iterates over the same numbers as k
for(v in 1:(64/indekser[i])){
  #designsplit is a matrix with the subcolumns which are
      sorted by. Here taken directly from sortmatrise
  #It contains all subcolumns of size  64/indekser[i]
  designsplit=cbind(designsplit,sortmatrise[((v-1)*indekser
      [i]+1):(v*indekser[i]),])
}
#Remove the column with zeroes used when initialising
designsplit=designsplit[,-1]
#Go through all columns in the original setup, to check if
    all subcolumns in each column sum to zero
for(t in 1:63){
  if((sum(colSums(designsplit)[seq(from=t,to=(63*(-1+(64/
      indekser[i])))+t),by=63)]>0)>0)){
    #ace the trouble-making indices
    badshuff[t]=t
  }
}
badindeks=badshuff
#Remove the bad columns from the vector to be tested
if(sum(badindeks)>0){
  shuff=shuff[-badindeks]
}
shuff=sample(shuff)
first=shuff[1]
iter=1
#Find row indices based on the first ok shuff
for(k in 1:(64/indekser[i])){
  #Radindekser is the indices of the rows to be included in
       the subcolumn, Sort sortmatrises indices to make -1
      come first, then 1
  radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
      indekser[i]+1):(k*indekser[i]),shuff[iter]],
      decreasing = FALSE)
```

```
      #Iterate to everything is ok
      while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
          shuff[iter]]))>0){
        iter=iter+1
        radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
            indekser[i]+1):(k*indekser[i]),shuff[iter]],
            decreasing = FALSE)
      }
      indeksliste[i]=shuff[iter]
      for(r in 1:length(radindekser)){
        nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[
            radindekser[r],]
      }
      sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
          nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
    }
  }
  valid=sum(is.na(indeksliste))<1
}

designmatrise=sortmatrise[,indeksliste]
designmatrise=cbind(designmatrise,designmatrise[,3]*
    designmatrise[,4]*designmatrise[,5]*designmatrise[,6])
designmatrise=cbind(designmatrise,designmatrise[,1]*
    designmatrise[,2]*designmatrise[,5]*designmatrise[,6])
colnames(designmatrise)=c("F","E","D","C","B","A","G","H")
blokker=sortmatrise[,-indeksliste]

num=ncol(blokker)
blockcomb=combn(num,2)
results3=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(combins)
    )
#Iterating over blocks
for(i in 1:ncol(blockcomb)){
  #Iterating over combinations
  for(j in 1:ncol(combins)){
    stormatrise=cbind(combGenerator(designmatrise,combins[,j]),
        blokker[,blockcomb[1,i]],blokker[,blockcomb[2,i]],
        blokker[,blockcomb[1,i]]*blokker[,blockcomb[2,i]])
    results3[i,j]=Ds(stormatrise)
  }
```

```
  }

  braindekser=which(rowSums(results3==0)==0)
}

#Finding min and max obtained for each block
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
sort(unique(mini))
sort(unique(maxi))
table(mini)
table(maxi)
#Find unique means for the blocks
print(sort(unique(rowMeans(results3[braindekser,]))))

#Find the order in which the matrix was sorted
rekkef=numeric(nrow(matrise))
for(i in 1:nrow(matrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(matrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)

sort(unique(rowMeans(results3[braindekser,])))
rmeans=rowMeans(results3[braindekser,])
which(rmeans>0.965)
godmini=braindekser[which(mini>0.917)]
table(rowMeans(results3[braindekser,]))
bestindekser=godmini[which(rowMeans(results3[godmini,])>0.965)]


for(i in 1:length(bestindekser)){
  print(cat("Current␣comb:␣", i))
  print(which(duplicated(cbind(blokker[,blockcomb[1,bestindekser[
      i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(blokker[,blockcomb[2,bestindekser[
      i]]],sortmatrise),MARGIN=2)))
}
```

```
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results3[bestindekser,],1,table)

printmat=sortmatrise[,(c(13,14,15,16,26,33,36,37,40,47,48,59,60)
    -1)]
colnames(printmat)=c("13","14","15","16","26","33","36","37","40"
    ,"47","48","59","60")
xtable(printmat[1:32,], digits = 0)
xtable(printmat[33:64,], digits = 0)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,combins[,5]),blokker[,
    blockcomb[1,487]],blokker[,blockcomb[2,487]],blokker[,
    blockcomb[1,487]]*blokker[,blockcomb[2,487]])
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-3)]))/sqrt(min(diagonal[1:(len-3)
    ])))
print(sqrt(max(diagonal[(len-3):len]))/sqrt(min(diagonal[1:(len
    -3)])))


#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,combins[,9]),blokker[,
    blockcomb[1,487]],blokker[,blockcomb[2,487]],blokker[,
    blockcomb[1,487]]*blokker[,blockcomb[2,487]])
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-3)]))/sqrt(min(diagonal2[1:(len
    -3)])))
print(sqrt(max(diagonal2[(len-3):len]))/sqrt(min(diagonal2[1:(len
    -3)])))


#Checking for three active factors
results33=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(combins3)
    )
#Iterating over blocks
```

```
for(i in 1:ncol(blockcomb)){
  #Iterating over combinations
  for(j in 1:ncol(combins3)){
    stormatrise3=cbind(combGenerator3(designmatrise,combins3[,j])
        ,blokker[,blockcomb[1,i]],blokker[,blockcomb[2,i]],blokker
        [,blockcomb[1,i]]*blokker[,blockcomb[2,i]])
    results33[i,j]=Ds(stormatrise3)
  }
}
braindekser3=which(rowSums(results33==0)==0)

#Finding min and max obtained for each block
mini3=apply(results33[braindekser3,], 1, min)
maxi3=apply(results33[braindekser3,], 1, max)
sort(unique(mini3))
table(mini3)
table(maxi3)
bestindekser3=braindekser3[which(mini3>0.93)]
sort(unique(rowMeans(results33[braindekser3,])))
table(rowMeans(results33[bestindekser3,]))
allerbestindekser3=bestindekser3[which(rowMeans(results33[
    bestindekser3,])>0.98)]

for(i in 1:length(allerbestindekser3)){
  print(cat("Current␣comb:␣", i))
  print(which(duplicated(cbind(blokker[,blockcomb[1,
      allerbestindekser3[i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(blokker[,blockcomb[2,
      allerbestindekser3[i]]],sortmatrise),MARGIN=2)))
}

apply(results33[allerbestindekser3,],1,table)
#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results33[allerbestindekser3,],1,table)

#Finding the SD-ratios
#For one of the combinations yielding the highest Ds
matrise13=cbind(combGenerator3(designmatrise,combins3[,1]),
    blokker[,blockcomb[1,487]],blokker[,blockcomb[2,487]],blokker
```

```
    [,blockcomb[1,487]]*blokker[,blockcomb[2,487]])
diagonal3=diag(solve(t(matrise13)%*%matrise13))
len3=length(diagonal3)
print(sqrt(max(diagonal3[1:(len3-3)]))/sqrt(min(diagonal3[1:(len3
    -3)])))
print(sqrt(max(diagonal3[(len3-3):len3]))/sqrt(min(diagonal3[1:(
    len3-3)])))

#For one of the combinations yielding the lowest Ds
matrise03=cbind(combGenerator3(designmatrise,combins3[,2]),
    blokker[,blockcomb[1,487]],blokker[,blockcomb[2,487]],blokker
    [,blockcomb[1,487]]*blokker[,blockcomb[2,487]])
diagonal23=diag(solve(t(matrise03)%*%matrise03))
print(sqrt(max(diagonal23[1:(len3-3)]))/sqrt(min(diagonal23[1:(
    len3-3)])))
print(sqrt(max(diagonal23[(len3-3):len3]))/sqrt(min(diagonal23
    [1:(len3-3)])))

save.image("cyclic64_hadamard_4blocks.RData")
```

Code for section 4.3.3.5 and 4.3.3.6: Dividing a 2_V^{8-2} design into eight blocks using HM

```
library(xtable)
options(xtable.floating = FALSE)
options(xtable.timestamp = "")
library(pracma)
#Defining some functions at first

#Make effects up to four-factor interactions
combGenerator=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  interact4=combn(length(interest),4)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
```

```
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  for(j in 1:ncol(interact4)){
    res=cbind(res,res1[,interact4[1,j]]*res1[,interact4[2,j]]*
        res1[,interact4[3,j]]*res1[,interact4[4,j]])
    colnam[j+1+length(interest)+ncol(interact2)+ncol(interact3)
        ]=(paste(colnam[1+interact4[1,j]],colnam[1+interact4[2,j
        ]],colnam[1+interact4[3,j]],colnam[1+interact4[4,j]],
        collapse = ''))
  }
  colnames(res)=colnam
  res
}

#Make effects up to three-factor interactions
combGenerator3=function(mat,interest){
  interact2=combn(length(interest),2)
  interact3=combn(length(interest),3)
  inter=t(t(rep(1,nrow(mat))))
  res=inter
  res1=matrix(data=NA,nrow=nrow(mat),ncol=length(interest))
  colnam=numeric(8)
  colnam[1]="K"
  for(i in 1:length(interest)){
    res=cbind(res,mat[,interest[i]])
    res1[,i]=mat[,interest[i]]
    colnam[i+1]=colnames(mat)[interest[i]]
  }
```

```
  for(j in 1:ncol(interact2)){
    res=cbind(res,res1[,interact2[1,j]]*res1[,interact2[2,j]])
    colnam[j+1+length(interest)]=(paste(colnam[1+interact2[1,j]],
        colnam[1+interact2[2,j]], collapse = ''))
  }
  for(j in 1:ncol(interact3)){
    res=cbind(res,res1[,interact3[1,j]]*res1[,interact3[2,j]]*
        res1[,interact3[3,j]])
    colnam[j+1+length(interest)+ncol(interact2)]=(paste(colnam[1+
        interact3[1,j]],colnam[1+interact3[2,j]],colnam[1+
        interact3[3,j]], collapse = ''))
  }
  colnames(res)=colnam
  res
}



Ds8=function(comb){
  b=ncol(comb)
  #Note: s=b-3 as there are three columns corresponding to blocks
  s=b-7
  n=nrow(comb)
  detX=abs(det(t(comb)%*%comb))
  detXb=abs(det(t(comb[,c(b-6,b-5,b-4,b-3,b-2,b-1,b)])%*%comb[,c(
      b-6,b-5,b-4,b-3,b-2,b-1,b)]))
  Ds=((detX/detXb)^(1/s))/n
  Ds
}

hadam3=matrix(data=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,
1,-1,1,-1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,
1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,
-1,1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,-1,1,1,-1,1,
-1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,-1,-1,1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,
1,-1,-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,
-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,
-1,1,1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,1,1,-1,-1,1,
-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,
```

```
-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,
-1,1,-1,-1,-1,-1,-1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,
1,1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,-1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,-1,1,
-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,1,-1,-1,-1,
-1,-1,-1,1,1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,
-1,1,-1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,
1,-1,1,1,-1,-1,1,1,-1,1,1,-1,-1,-1,-1,1,1,-1,1,1,-1,1,-1,-1,
1,1,1,-1,-1,-1,-1,1,1,-1,1,1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,
-1,1,-1,1,1,-1,1,-1,1,-1,-1,1,-1,1,1,-1,-1,1,-1,1,1,-1,1,-1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,
1,-1,1,-1,1,-1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,1,-1,
-1,1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,1,
-1,1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,
-1,-1,1,1,1,1,-1,-1,1,1,1,1,-1,-1,1,-1,1,-1,1,-1,-1,1,-1,1,1,-1,
-1,1,1,-1,-1,1,-1,1,-1,1,-1,1,-1,-1,1,1,-1,-1,-1,1,1,-1,1,-1,1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,-1,1,
-1,1,-1,1,-1,-1,-1,-1,-1,1,1,1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,1,
-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,1,1,
-1,-1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,
-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,-1,1,-1,1,-1,1,1,1,1,-1,
-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,
-1,1,1,1,-1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,
1,-1,1,-1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,-1,1,1,-1,1,1,-1,1,-1,
-1,1,-1,-1,1,1,1,1,-1,-1,1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,-1,1,
1,-1,-1,1,-1,1,1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,
-1,1)
,nrow=32,ncol=32)
dmatrise=hadam3

matrise=rbind(cbind(dmatrise,dmatrise),cbind(dmatrise,-dmatrise))
matrise=matrise[,2:64]
int=4
fac=8
combins=combn(fac,int)
combins3=combn(fac,3)
```

```
#In the loop, a design yielding good results for four active
    factors is found
best=0
it=0
braindekser=NULL
while(length(braindekser)<1){
  it=it+1
  valid=FALSE
  #print(it)
  while(valid==FALSE){
    indeksliste=numeric(6)
    sortmatrise=matrise
    nymatrise=matrix(data=,nrow=nrow(sortmatrise),ncol=ncol(
        sortmatrise))
    indekser=c(64,32,16,8,4,2)
    #Shuff saves all possible columns indices
    shuff=seq(from = 1, to = 64, by = 1)
    #Iterate through all indices (=make columns for the 6 factors
        )
    for(i in 1:6){
      #k goes from 1 to 1, 1 to 2, 1 to 4 and so on, divides into
          the subcolumns which should be sorted in a certain way
      #Shuff is used to save the indices of the columns where all
          the subcolumns sum to zero
      shuff=seq(from = 1, to = 63, by = 1)
      designsplit=numeric(indekser[i])
      #Badshuff saves the indices for the columns where the
          subcolmnsum is not zero
      badshuff=rep(0,63)
      badshuffmat=matrix(data=NA,nrow =(64/indekser[i]),ncol=63)
      #V iterates over the same numbers as k
      for(v in 1:(64/indekser[i])){
        #designsplit is a matrix with the subcolumns which are
            sorted by. Here taken directly from sortmatrise
        #It contains all subcolumns of size  64/indekser[i]
        designsplit=cbind(designsplit,sortmatrise[((v-1)*indekser
            [i]+1):(v*indekser[i]),])
      }
      #Remove the column with zeroes used when initialising
      designsplit=designsplit[,-1]
      #Go through all columns in the original setup, to check if
```

```
      all subcolumns in each column sum to zero
  for(t in 1:63){
    if((sum(colSums(designsplit)[seq(from=t,to=(63*(-1+(64/
        indekser[i]))+t),by=63)]>0)>0)){
      #ace the trouble-making indices
      badshuff[t]=t
    }
  }
  badindeks=badshuff
  #Remove the bad columns from the vector to be tested
  if(sum(badindeks)>0){
    shuff=shuff[-badindeks]
  }
  shuff=sample(shuff)
  first=shuff[1]
  iter=1
  #Find row indices based on the first ok shuff
  for(k in 1:(64/indekser[i])){
    #Radindekser is the indices of the rows to be included in
        the subcolumn, Sort sortmatrises indices to make -1
        come first, then 1
    radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
        indekser[i]+1):(k*indekser[i]),shuff[iter]],
        decreasing = FALSE)
    #Iterate to everything is ok
    while(length(shuff)>iter&abs(sum(sortmatrise[radindekser,
        shuff[iter]]))>0){
      iter=iter+1
      radindekser=(k-1)*indekser[i]+order(sortmatrise[((k-1)*
          indekser[i]+1):(k*indekser[i]),shuff[iter]],
          decreasing = FALSE)
    }
    indeksliste[i]=shuff[iter]
    for(r in 1:length(radindekser)){
      nymatrise[((k-1)*indekser[i]+r),]=sortmatrise[
          radindekser[r],]
    }
    sortmatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]=
        nymatrise[((k-1)*indekser[i]+1):(k*indekser[i]),]
  }
}
```

```
    valid=sum(is.na(indeksliste))<1
  }

  designmatrise=sortmatrise[,indeksliste]
  designmatrise=cbind(designmatrise,designmatrise[,3]*
      designmatrise[,4]*designmatrise[,5]*designmatrise[,6])
  designmatrise=cbind(designmatrise,designmatrise[,1]*
      designmatrise[,2]*designmatrise[,5]*designmatrise[,6])
  colnames(designmatrise)=c("F","E","D","C","B","A","G","H")
  blokker=sortmatrise[,-indeksliste]

  num=ncol(blokker)
  blockcomb=combn(num,3)
  results3=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(combins)
      )
  #Iterating over blocks
  for(i in 1:ncol(blockcomb)){
    #Iterating over combinations
    for(j in 1:ncol(combins)){
      blokk1=blokker[,blockcomb[1,i]]
      blokk2=blokker[,blockcomb[2,i]]
      blokk3=blokker[,blockcomb[3,i]]
      blokk12=blokk1*blokk2
      blokk13=blokk1*blokk3
      blokk23=blokk2*blokk3
      blokk123=blokk1*blokk2*blokk3
      stormatrise=cbind(combGenerator(designmatrise,combins[,j]),
          blokk1,blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
      results3[i,j]=Ds8(stormatrise)
    }
  }

  braindekser=which(rowSums(results3==0)==0)
  print(indeksliste)
}

#Finding min and max obtained for each block
mini=apply(results3[braindekser,], 1, min)
maxi=apply(results3[braindekser,], 1, max)
sort(unique(mini))
sort(unique(maxi))
```

```
table(mini)
table(maxi)
#Find unique means for the blocks
print(sort(unique(rowMeans(results3[braindekser,]))))

#Find the order in which the matrix was sorted
rekkef=numeric(nrow(matrise))
for(i in 1:nrow(matrise)){
  for(j in 1: nrow(sortmatrise)){
    if(identical(matrise[i,],sortmatrise[j,])==TRUE){
      rekkef[i]=j
    }
  }
}
print(rekkef)

sort(unique(rowMeans(results3[braindekser,])))
godmini=braindekser[which(mini>0.807)]
table(rowMeans(results3[godmini,]))
bestindekser=godmini[which(rowMeans(results3[godmini,])>0.888)]


for(i in 1:length(bestindekser)){
  print(cat("Current comb: ", i))
  print(which(duplicated(cbind(blokker[,blockcomb[1,bestindekser[
      i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(blokker[,blockcomb[2,bestindekser[
      i]]],sortmatrise),MARGIN=2)))
}

#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results3[bestindekser,],1,table)

#Print columns used for blocks
printmat=sortmatrise[,(c(18,20,26,28,51,59)-1)]
colnames(printmat)=c("18","20","26","28","51","59")
xtable(printmat[1:32,], digits = 0)
xtable(printmat[33:64,], digits = 0)
```

```
#Finding the SD-ratios
blokker1=blokker[,blockcomb[1,bestindekser[1]]]
blokker2=blokker[,blockcomb[2,bestindekser[1]]]
blokker3=blokker[,blockcomb[3,bestindekser[1]]]
blokk12=blokk1*blokk2
blokk13=blokk1*blokk3
blokk23=blokk2*blokk3
blokk123=blokk1*blokk2*blokk3
#For one of the combinations yielding the highest Ds
matrise1=cbind(combGenerator(designmatrise,combins[,3]),blokk1,
    blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
diagonal=diag(solve(t(matrise1)%*%matrise1))
len=length(diagonal)
print(sqrt(max(diagonal[1:(len-7)]))/sqrt(min(diagonal[1:(len-7)
    ])))
print(sqrt(max(diagonal[(len-7):len]))/sqrt(min(diagonal[1:(len
    -7)])))


#For one of the combinations yielding the lowest Ds
matrise0=cbind(combGenerator(designmatrise,combins[,67]),blokk1,
    blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
diagonal2=diag(solve(t(matrise0)%*%matrise0))
print(sqrt(max(diagonal2[1:(len-7)]))/sqrt(min(diagonal2[1:(len
    -7)])))
print(sqrt(max(diagonal2[(len-7):len]))/sqrt(min(diagonal2[1:(len
    -7)])))

#Checking for 3 active factors
results33=matrix(data=NA,nrow=ncol(blockcomb),ncol=ncol(combins3)
    )
#Iterating over blocks
for(i in 1:ncol(blockcomb)){
  #Iterating over combinations
  for(j in 1:ncol(combins3)){
    blokk1=blokker[,blockcomb[1,i]]
    blokk2=blokker[,blockcomb[2,i]]
    blokk3=blokker[,blockcomb[3,i]]
    blokk12=blokk1*blokk2
    blokk13=blokk1*blokk3
    blokk23=blokk2*blokk3
```

```
    blokk123=blokk1*blokk2*blokk3
    stormatrise3=cbind(combGenerator3(designmatrise,combins3[,j])
        ,blokk1,blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
    results33[i,j]=Ds8(stormatrise3)
  }
}

braindekser3=which(rowSums(results33==0)==0)

#Finding min and max obtained for each block
mini3=apply(results33[braindekser3,], 1, min)
maxi3=apply(results33[braindekser3,], 1, max)
sort(unique(mini3))
table(mini3)
table(maxi3)
bestindekser3=braindekser3[which(mini3>0.853)]
sort(unique(rowMeans(results33[braindekser3,])))
table(rowMeans(results33[bestindekser3,]))
allerbestindekser3=bestindekser3[which(rowMeans(results33[
    bestindekser3,])>0.918)]

for(i in 1:length(allerbestindekser3)){
  print(cat("Current comb: ", i))
  print(which(duplicated(cbind(blokker[,blockcomb[1,
      allerbestindekser3[i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(blokker[,blockcomb[2,
      allerbestindekser3[i]]],sortmatrise),MARGIN=2)))
  print(which(duplicated(cbind(blokker[,blockcomb[3,
      allerbestindekser3[i]]],sortmatrise),MARGIN=2)))
}

#Which columns make up the design matrix?
print(indeksliste+1)
#Frequencies
apply(results33[allerbestindekser3,],1,table)

#Finding the SD-ratios
blokker1=blokker[,blockcomb[1,allerbestindekser3[1]]]
blokker2=blokker[,blockcomb[2,allerbestindekser3[1]]]
blokker3=blokker[,blockcomb[3,allerbestindekser3[1]]]
blokk12=blokk1*blokk2
```

```
blokk13=blokk1*blokk3
blokk23=blokk2*blokk3
blokk123=blokk1*blokk2*blokk3
#For one of the combinations yielding the highest Ds
matrise13=cbind(combGenerator3(designmatrise,combins3[,24]),
    blokk1,blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
diagonal3=diag(solve(t(matrise13)%*%matrise13))
len3=length(diagonal3)
print(sqrt(max(diagonal3[1:(len3-7)]))/sqrt(min(diagonal3[1:(len3
    -7)])))
print(sqrt(max(diagonal3[(len3-7):len3]))/sqrt(min(diagonal3[1:(
    len3-7)])))

#For one of the combinations yielding the lowest Ds
matrise03=cbind(combGenerator3(designmatrise,combins3[,3]),blokk1
    ,blokk2,blokk3,blokk12,blokk23,blokk13,blokk123)
diagonal23=diag(solve(t(matrise03)%*%matrise03))
print(sqrt(max(diagonal23[1:(len3-7)]))/sqrt(min(diagonal23[1:(
    len3-7)])))
print(sqrt(max(diagonal23[(len3-7):len3]))/sqrt(min(diagonal23
    [1:(len3-7)])))


save.image("cyclic64_hadamard_8blocks_4act")
```